Learning Measures of Progress for Planning Domains

SungWook Yoon
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
sy@purdue.edu

Alan Fern
Computer Science Department
Oregon State University
Corvallis, OR 97331
afern@cs.orst.edu

Robert Givan
Electrical & Computer Engineering
Purdue University
West Lafayette, IN 47907
givan@purdue.edu

Abstract

We study an approach to learning heuristics for planning domains from example solutions. There has been little work on learning heuristics for the types of domains used in deterministic and stochastic planning competitions. Perhaps one reason for this is the challenge of providing a compact heuristic language that facilitates learning. Here we introduce a new representation for heuristics based on lists of set expressions described using taxonomic syntax. Next, we review the idea of a *measure of progress* (Parmar 2002), which is any heuristic that is guaranteed to be improvable at every state. We take finding a measure of progress as our learning goal, and describe a simple learning algorithm for this purpose. We evaluate our approach across a range of deterministic and stochastic planning-competition domains. The results show that often greedily following the learned heuristic is highly effective. We also show our heuristic can be combined with learned rule-based policies, producing still stronger results.

Introduction

Good plans can often be understood as seeking to achieve specific subgoals en route to the goal. In an object-oriented view, these subgoals, along with the goal itself, can be seen as properties of objects, where in each case we wish to increase the number of objects with the given property. Taking this view, we consider control knowledge in the form of compact descriptions of object classes that a controller is to select actions to enlarge. For example in BLOCKSWORLD, in an optimal trajectory, the number of blocks "well placed" from the table up never decreases. Or in LOGISTICS the number of solved packages never decreases in optimal plans.

Good trajectories also often exhibit locally monotonic properties: properties that increase for local periods, but not all the time during the trajectory. In BLOCKSWORLD, consider a block "solvable" if its desired destination is clear and well placed from the table up. Then, in good trajectories, while the number of blocks well placed from the table up stays the same, the number of solvable blocks need never decrease locally; but, globally, the number of solvable blocks may decrease as the number of blocks well placed from the table up increases. Sequences of such properties can be used to construct a *prioritized-ensemble heuristic*, where we select the action that is able to improve the highest-priority property, while preserving higher-priority properties.

Here, we are not directly interested only in characterizing *optimal* trajectories, but more generally in learning the apparent strategy underlying a set of training trajectories. If we assume the training trajectories were themselves generated by seeking to maximize some prioritized-ensemble heuristic, we can hope to recover that heuristic. More generally, though, we seek to find heuristics that enable us to achieve performance similar to that of the training trajectories.

When a heuristic has the property that it can always be increased in one step, then it constitutes a *strong measure of progress* in the sense of Parmar (2002). This is a desirable property for any heuristic and we take finding such a heuristic as our learning goal. Our empirical results demonstrate that this learning goal results in useful heuristic.

It is a straightforward result that strong measures of progress exist for any deadlock-free domain, if one is willing to represent them as explicit state-to-predicate-extension mappings, simply encoding the distance to the goal. Here, we are interested in finding implicitly and compactly represented mappings, which may not exist in all planning domains for any given representation language. It is an assumption of our approach that such compactly represented mappings can usefully if not completely characterize the training trajectories. It is important to note that our learned heuristics may not have the strong guarantee required for Parmar's definition—however, as we demonstrate, the practical benefits for planning can still be significant.

Here we give a compact representation for heuristic functions using a class-description language to describe a prioritized sequence of class expressions, each expression denoting a set of objects in a given state. Given a state, a sequence of class expressions induces a vector of class occupancies, which we treat as lexicographically ordered; by this means, a sequence of class expressions can be treated as a heuristic for selecting actions. In our example above, the sequence ("well-placed blocks", "solvable blocks") represents the idea that as long as the number of well-placed blocks changes, we don't care about a decrease in solvable blocks, but otherwise we do. Given a set of training trajectories, we provide a machine-learning approach to acquiring a sequence of taxonomic classes, giving a prioritized-ensemble

heuristic that approximately increases along the training trajectories. We treat such a heuristic as a strong measure of progress with respect to the training trajectories, though it may only approximately satisfy Parmar's definition.

In our empirical work, we learn from machine-generated trajectories and evaluate the usefulness of our learned measures in two ways. First, we consider the learned measures of progress directly as the sole control knowledge by choosing the action that results in the best expected value on the progress measure, breaking ties randomly. We show that this simple approach outperforms previous learned control policies for the setting where the only training data is trajectories. Previous techniques (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) which reduce control-knowledge learning to supervised classification perform poorly unless provided with the set of all "good" actions at each state. Second, we show that performance can be further enhanced by combining the learned measures of progress with policies learned via previous techniques, using the policies only to break ties in the measure of progress (in place of random tie breaking). In each case, we show results in a variety of deterministic and stochastic planning domains.

We are not aware of prior work on learning heuristic functions for planning domains such as those used in deterministic and stochastic planning competitions. The most successful previous work (Huang, Selman, & Kautz 2000; Sammut *et al.* 1992; Morales & Sammut 2004; Khardon 1996; Martin & Geffner 2000) on learning control knowledge for such domains from trajectories, uses control knowledge represented via action selection (and sometimes rejection) rules rather than as heuristic functions, conditioning only on the current state. Our approach is able to learn knowledge that selects actions in part by considering their effects explicitly, which our experiments show can be beneficial.

Problem Setup

Planning Domains. A *planning domain* is a tuple $D = \langle P, Y, O, T, G\rangle$. Here, P is a set of state predicates, Y is a set of action types, and O is a domain set of objects. A *state fact* is an instance of a predicate in P applied to objects in O (e.g., In(Box1,Truck1)), and a *state* is simply a finite set of state facts. An *action* is an instance of an action type in Y applied to objects in O (e.g., Load(Box1,Truck1)). The sets P, Y, and O implicitly define the state space S of all states and the action space A of all actions.

T is a transition function that maps $S \times A$ to "next state" distributions over S, so that $T[s,a](s')$ gives the probability of transitioning to state s' after taking action a in state s. In our system, we compactly represent T using the probabilistic STRIPS-like language PPDDL (Younes & Littman 2004). We will sometimes treat T as a stochastic function, where the call $T[s,a]$ returns a state s' with probability $T[s,a](s')$. G is a randomized problem generator that outputs planning problems according to some distribution. A *planning problem* is simply a pair $\langle s, g\rangle$ of an initial state s and a set of state facts g representing the goal region.[1]

[1]Our experiments include the COLORED BLOCKSWORLD where goals have existential variables. We handle this in an ad-hoc way by immediately grounding such variables to objects that satisfy static properties (e.g. color) of these variables in the goal.

Policies. A policy π is a (possibly stochastic) function that maps planning problems to actions. That is, a policy takes as input a current state and goal condition and selects an action, ideally an action that takes us closer to the goal. Given an initial problem $\langle s_0, g\rangle$, a policy π can generate a *trajectory* of states-goal-action triples$(\langle s_0, g, a_0\rangle, \langle s_1, g, a_1\rangle, \langle s_2, g, a_2\rangle \ldots)$, where $a_i = \pi(\langle s_i, g\rangle)$ and $s_{i+1} = T[s_i, a_i]$. When the planning domain and/or π are stochastic we get a distribution over trajectories starting from s_0. We say that a trajectory is a *solution* to a planning problem $\langle s_0, g\rangle$ iff there is an i such that $g \subseteq s_i$.

Measures of Progress

Prioritized Heuristics. Throughout we assume the context of a planning domain $D = \langle P, Y, O, T, G\rangle$. A *heuristic* is simply a mapping from planning problems (i.e., state-goal pairs) to a totally ordered set. Typically, a heuristic returns a number that is interpreted as a distance-to-goal estimate. Rather, in our work, we consider prioritized heuristics whose ranges are vectors. A prioritized-ensemble heuristic is an ordered list $F = (F_1, \ldots, F_n)$ where each component F_i is a function from planning problems to reals. The output of a prioritized heuristic is, thus, a vector of reals. We compare vectors using the *prioritized dominance* relation $\succ$, defined such that for two real-number sequences $N = (N_1, \ldots, N_k)$ and $M = (M_1, \ldots, M_k)$, $N \succ M$ iff there exists an i such that $N_i > M_i$ and for all $j < i$, $N_j = M_j$. Prioritized-ensemble heuristics provide a useful decomposition for defining heuristics in terms of local properties of a planning domain, as described in the introduction section. In addition, as described in the learning section below, there are natural techniques for learning such heuristics.

Given a prioritized-ensemble heuristic F, we define a corresponding stochastic policy π_F that will be used for planning according to the heuristic. For any given planning problem $\langle s, g\rangle$, $\pi_F(\langle s, g\rangle)$ returns a random action chosen from the set $\text{argmax}_a E[F(\langle T[s,a], g\rangle)]$, where the maximization is relative to $\succ$ and $E[F(X)] = (E[F_1(X)], \ldots, E[F_n(X)])$. The expectation is calculated using the transition model T provided by our planning domain. Our empirical goal is to learn an F such that π_F can be executed quickly and gives good planning performance.

Measures of Progress. A desirable property for a heuristic is that at any state there should be an action that leads to an (expected) improvement in its value. Heuristics with this property were called *strong measures of progress* by Parmar (2002). Here we extend Parmar's definition, which was for deterministic domains, to stochastic planning domains.

Definition 1 (Strong Measure of Progress). *Let $F = (F_1, \ldots, F_n)$ be an ordered list where each F_i is a function from planning problems to integers. F is a strong measure of progress for planning domain D iff for any reachable problem $\langle s, g\rangle$ of D, either $g \subseteq s$ or there exists an action a such that $E[F(\langle T[s,a], g\rangle)] \succ F(\langle s, g\rangle)$.*

This definition requires that for any non-goal state there is an action that increases (in expectation) some F_i while

maintaining the preceding, higher-priority components, corresponding to Parmar's definition for deterministic domains.

A strong measure of progress exists for any deadlock-free domain[2]; e.g., taking $-F$ to be the expected solution length of an optimal policy. In addition, in deterministic domains, if one always selects actions that lead to an increase in F then a solution will always be found in at least $|S|$ actions. While this worst case bound is extremely loose, in practice useful measures of progress can be defined that lead to short solution lengths. In particular, taking the measure of progress to be the (expected) distance to the goal will result in optimal planning. Of course, using such a measure is not practical since in general it cannot be computed efficiently. Rather, we are typically interested in measures that can be computed efficiently, yet lead to "good enough" plans.

One contribution of this paper is to show that useful prioritized-ensemble heuristics can be learned by focusing on the goal of satisfying the strong-measure-of-progress property with respect to the training data. Alternatively, another approach to learning heuristics is to use a function approximator in order to learn an estimate of the distance-to-goal function. However, our experience indicates that, for benchmark planning domains, learning such estimates is difficult and so far has not led to good performance.

Representing Measures of Progress

It is often natural to measure progress in a planning domain by counting the number of objects with a certain property. For example, in LOGISTICS problems it is desirable to increase the number of objects in their final destinations, or perhaps to increase the number of objects in a plane that need to go to the plane's destination. In this section, we first describe how to represent such properties using a taxonomic syntax for first-order logic. Next, we describe how to represent prioritized measures of progress using this syntax.

Taxonomic Syntax. Following Yoon, Fern, & Givan (2002) our class language is an extension of taxonomic syntax (McAllester & Givan 1993), a first-order language for representing sets or classes of objects using class expressions. Our language is defined by

$$C ::= C_0 \mid \textbf{a-thing} \mid \neg C \mid (R\ C) \mid C \cap C \mid (\min R)$$
$$R ::= R_0 \mid R^{-1} \mid R \cap R \mid R^*.$$

Here C and R are class expressions and relations respectively. **a-thing** represents the set of all objects. C_0 is any one argument predicate from our planning domain, and R_0 is any binary predicate from the domain. Predicates in the domain of three or more arguments are represented with multiple introduced auxiliary binary predicates. Also, included in C_0 and R_0 are new copies of each predicate symbol, which are used to represent the desired goal; e.g., **gclear**(x) represent that x is clear in the goal. We also include new predicates representing that a goal fact is satisfied in the current state; e.g., **con**(x, y) represents that x is "correctly on" y, that is, x is on y in both the goal and the current states.

Given a planning problem $\langle s, g\rangle$ and class expression C, it is easy to compute the set of objects that are represented by C in $\langle s, g\rangle$, denoted by $C(s, g)$. If C is a one argument predicate then $C(s, g)$ denotes the set of objects that C is true of in $\langle s, g\rangle$. Expressions of the form $(R\ C)$ denote the image of the objects in class C under relation R—e.g., **(on gclear)** represents the set of blocks that are currently on a block that is supposed to be clear in the goal. The expression $(\min R)$ denotes the class of minimal elements under relation R (viewing R as a partial order). The expression R^* denotes the reflexive, transitive closure of the relation R. See Yoon, Fern, & Givan (2002) for the complete semantics.

Taxonomic Prioritized Measures. Recall that a prioritized measure of progress is a sequence of functions $F = (F_1, \ldots, F_n)$ from planning problems to numbers. In this work, we parameterize prioritized measures by a sequence of class expressions $C = (C_1, \ldots, C_n)$ and define the corresponding prioritized measure by $F_C(\langle s, g\rangle) = (|C_1(s, g)|, \ldots, |C_n(s, g)|)$. That is, each individual measure corresponds to a count of the objects in a certain class.

For example, in the BLOCKSWORLD consider the measure of progress described by the sequence (C_1, C_2) where $C_1 =$ (**con*** **con-table**) and $C_2 =$ (**gon** (**clear** $\cap$ **con*** **con-table**))). C_1 the higher-priority property, is the set of blocks well placed from the table up, and C_2 is the set of "solvable" blocks. This measure, prefers to increase the number of well-placed blocks. When improving C_1 is not possible, the measure will prefer to take actions that increase the number of solvable blocks, while preserving C_1.

Learning

In this section we describe a method for learning prioritized measures of progress in the taxonomic representation described in the previous section. In particular, given a planning-domain definition and a set of solution trajectories of problems from the domain, we output a list of taxonomic class expressions C, defining a prioritized measure F_C that can be used for subsequent planning in the domain.

An underlying assumption of our approach is that the action selections in the training trajectories can be explained as trying to improve an unknown prioritized measure of progress. Furthermore we assume that this measure can be compactly represented using our taxonomic syntax representation. Our learner then attempts to uncover the unknown measure of progress by analyzing the trajectories. In our experiments, we use trajectories generated by domain-independent planners and humans, and these assumptions will typically not be met exactly. Nevertheless, we show empirically that useful measures of progress can be discovered from these sources, leading to good planning performance.

Learning Algorithm. The raw input to our algorithm is a planning domain $D = \langle P, Y, O, T, G\rangle$ and a set of trajectories $J = \{J_1, \ldots, J_m\}$ where each J_i is a finite sequence of state-goal-action triples. We define $\mathbb{J}$ to be the multi-set of all state-goal-action triples in J. We say that a triple (s, g, a) is *covered* by a class expression C, if $|C(s, g)| \neq E(|C(T[s, a], g)|)$. In the case where

[2] In general a planning domain is *deadlock free* iff for all *reachable* planning problems there is a sequence of actions that can reach the goal with non-zero probability. A problem $\langle s, g\rangle$ is reachable iff G can generate a problem $\langle s_0, g\rangle$ such that there is some action sequence with non-zero probability of reaching s from s_0.

$|C(s,g)| < E(|C(T[s,a],g)|)$, we say that the triple is *positively covered*, otherwise if $|C(s,g)| > E(|C(T[s,a],g)|)$ the example is *negatively covered*. Our learner searches for sequence of class expressions $\mathcal{C} = (C_1, \ldots, C_n)$ such that the corresponding prioritized measure $F_{\mathcal{C}}$ is approximately a strong measure of progress with respect to the training data. That is, for each state-goal-action triple $\langle s, g, a\rangle$ in the training data $E[F_{\mathcal{C}}(\langle T[s,a], g\rangle)] \succ F_{\mathcal{C}}(\langle s, g\rangle)$. This requires that each training example is positively covered by some C_i and is not negatively covered by any C_j with $j < i$.

Our algorithm, shown in Figure 1, searches for class expressions in the order of highest to lowest priority, similar to Rivest-style decision-list learning (Rivest 1987). **Learn-Prioritized-Measure** starts with an empty list of classes and then searches for the first class expression C_1 by calling **Find-Best-Measure**. Intuitively, this search attempts to find a C_1 with no negative coverage that positively covers many training instances. For instances that are positively covered by C_1, the strong measure of progress condition is satisfied. For the remaining instances, we still need to find lower priority class expressions that positively cover them. For this purpose, we remove from the training set any instances that are covered by C_1 and search for C_2 again by calling **Find-Best-Measure**. The algorithm repeats this process of removing instances from $\mathbb{J}$ and finding new class expressions until all of the instances are removed, or **Find-Best-Measure** is unable to find a "good enough" class expression.

The routine **Find-Best-Measure** searches for a good class expression via a heuristically-guided beam search over a restricted space of class expressions. In particular, let $\mathbb{C}_d$ be the set of all intersection-free class expressions of depth at most d. The beam search conducts a search over longer and longer intersections of class expressions selected from $\mathbb{C}_d$, attempting to find a concept with high positive and low negative coverage. The beam width b and maximum depth d are user specified and were $b = 5$, $d = 3$ in our experiments.

Our beam-search heuristic is weighted accuracy. Given a set of training triples $\mathbb{J}$ and a class expression C, let p and n be the number of positively and negatively covered instances in $\mathbb{J}$ respectively. Our heuristic value is given by $H(\mathbb{J}, C) = p - \omega n$. The parameter ω is chosen by the user and trades off positive and negative coverage. In our experiments, we use $\omega = 4$. We performed limited evaluations of other more sophisticated heuristics (for example those studied in Furnkranz & Flach (2003)), but none appeared to perform significantly better than weighted accuracy.

Pitfalls. An assumption of our technique is that the training trajectories depict good planning performance and that our learner is able to uncover a measure F_C that approximately captures the behavior. One shortcoming of our current algorithm is that it can be fooled by properties that monotonically increase along all or many trajectories in a domain, even those that do not solve the planning problem.

For example, consider a domain with a class expression whose extension never decreases and frequently increases along any trajectory. Our learner will likely output this class expression as a solution, although it does not in any way distinguish good from bad trajectories. In most of our experimental domains, such properties do not seem to exist, or at least are not selected by our learner. However, in EXPLODING BLOCKSWORLD this problem did appear to arise and hurt the performance of measures alone as policies. Rule-based policy learning combined with measures overcame this problem, as shown in the experimental section.

```
Learn-Prioritized-Measures (𝕁, b, d)
// state-goal-action triples 𝕁, beam width b, depth d
C ← nil;
while |𝕁| > 0
    C' ← Find-Best-Measure(𝕁, b, d);
    if H(𝕁, C') > 0
        C = append(C, C');     // add C' at end
        𝕁 ← remove-covered(𝕁, C');
    else Return C
Return C;

Find-Best-Measure (𝕁, b, d)
B ← {a-thing}; H* ← −∞;
repeat
    G = B ∪ {C' ∩ C_b | C_b ∈ B, C' ∈ ℂ_d}
    B = beam-select(G, 𝕁, b);   // select best b concepts
    C* ← argmax_{C_b ∈ B} H(𝕁, C_b);
    if (H(𝕁, C*) = H*) then Return C*;
    H* ← H(𝕁, C*);
```

Figure 1: Pseudo-code for learning prioritized measure

There are a number of possible approaches to dealing with this pitfall, which we will pursue in future work. For example, one idea is to generate a set of random (legal) trajectories and reward class expressions that can apparently distinguish between the random and training trajectories.

Experiments

Procedure. We evaluated our learning approach in both deterministic and stochastic domains. Our deterministic domains included the three domains from test suite 1 of the 3rd International Planning Competition, DEPOTS, ZENOTRAVEL, and DRIVERLOG, along with two domains that have been used in previous work on learning rule-based policies, BLOCKSWORLD and LOGISTICS. Our stochastic domains included COLORED (and UNCOLORED) BLOCKSWORLD, EXPLODING BLOCKSWORLD, and BOXWORLD, which come from the 1st International Probabilistic Planning Competition (IPPC-1) held in 2004. For our deterministic domains, we generated training trajectories using the planner FF (Hoffmann & Nebel 2001). For stochastic domains, we generated trajectories according to a human-written policy. From the generated trajectories, we learned a prioritized-ensemble measure of progress. For comparison we also used the system described in Yoon, Fern, & Givan (2002) to learn rule-based policies from trajectories. The rule-based policies are simply lists of rules of the form "If a condition is true then take a certain action".

Figure 2 summarizes the problem sizes used for both training and testing in each domain. Each evaluation measured the success ratio (SR) (fraction of solved problems) and average solution length (AL) of *solved* problems on a

set of randomly drawn problems, using a solution time limit of 2000 cpu seconds. In addition to evaluating the learned measures and rule-based policies in isolation we also evaluated the *combined policy*, which selects an action randomly from the actions that both maximize the learned measure and are suggested by the policy. If the intersection is empty, then the combined policy selects an action according to the measure.

We conducted 20 trials in each domain, each involved drawing 20 training problems, generating the corresponding trajectories, learning a measure of progress and rule-based policy, and then evaluating the results. Figures 3 and 4 show the averages over the trials. Each row corresponds to a single planning domain. The second column gives the (SR) and (AL) of the learned rule-based policy. The third and fourth columns gives the same information for the learned measures and the combined policy respectively. The final column gives the performance of FF on the test distribution.

Deterministic Domains

Success Ratio. For all deterministic domains except DRIVERLOG the learned measures lead to relatively good success ratios, indicating that our learned measures of progress are indeed capturing useful notions of progress. We believe that the primary reason for the relatively poor performance in DRIVERLOG is that our concept language (which is also used by the rule-based policy learner) is not expressive enough to capture the key concept of "path".

The success ratio of the measures are generally superior to the rule-based policies. We believe that the relatively poor success ratio of the rule-based policies is due to the fact the training examples don't completely separate good actions from bad actions. That is, each state in the training data is only labeled by a single action and the learner tries to find rules that select those actions and avoid selecting other actions. In reality a single training state may have many equally good actions, and the particular action that ends up in the training set is largely arbitrary. For example, in the BLOCKSWORLD there are often many blocks are equally good to pick-up. The rule-based policy learner will try to avoid selecting the good actions that are not in the training data, making the learning problem quite difficult.

Prior rule-based policy learners (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) have avoided this difficulty by labeling training data with all good actions. However, this information is not always available. Comparably, when learning measures of progress, as long as the trajectories exhibit monotonic properties then it is not important which action of the good actions appear in the training data. That is, measures of progress can be invariant to the specific action used to make progress since they capture information about what the action does. Rather, rule-based policies focus on information about where an action is applied not the effects. In this sense learning measures of progress appears more natural when our training data consists of only trajectories.

Average Length. The rule-based policies achieve a signficantly better AL than the learned measures. That is, when the rule-based policy solves a problem it typically finds a shorter solution, suggesting that the learned measures do not completely distinguish good actions from not so useful actions. On inspection, we found that the solution trajectories from the measures often include random actions that do not appear to make progress, yet do not destroy progress. In other words, the trajectories include bits of non-destructive wandering in between actions that make real progress.

In deterministic domains, a potential remedy for this problem could be to use FF's search technique of *enforced hill-climbing* (Hoffmann & Nebel 2001), which conducts a breadth-first search for a path to a state where the heuristic makes progress. The resulting path is then included in the plan, which may often be significantly shorter than the paths found by the non-destructive random wandering. We also note that the ability to simply find a plan quickly is valuable, as post-processing techniques could conceivably be developed to prune away wasteful action sequences—e.g. work has been done on learning plan re-write rules for simplifying sub-optimal solutions (Ambite, Knoblock, & Minton 2000).

Trajectories from the rule-based policies are qualitatively different. Typically, the policies select actions that appear to make progress, but sometimes select destructive actions (undoing progress), eventually resulting in failure. By combining the rule-based policies and measures we can filter out these destructive actions but also reduce the amount of wandering observed for the measures. As the results show, the combination typically achieves the success ratio of the measure and the average length of the rule-based policy.

Comparing to Prior Work. For benchmark planning domains, the most successful prior work on learning policies from trajectories produce rule-based policies. Khardon (1999) learns rule-based policies from only trajectories, without providing the sets of all good actions. Khardon (1999) reports experiments in BLOCKSWORLD and LOGISTICS yielding 0.56 SR for 20 blocks, and 0.59 SR for 20 packages respectively, which are similar to our rule-based policy learner and worse than our learned measures of progress. Martin & Geffner (2000) learns rule-based policies from trajectories, but requires that the set of all optimal actions be provided for each state, thus using more information than we are here. Martin & Geffner (2000) reports experiments in BLOCKSWORLD yielding 0.75 SR for 20 blocks, which is better than our rule-based learner but worse than our learned measures. Thus, our learned measures outperform the SR of the most closely related previous work in the domains they considered.

Comparing to FF. Compared to FF, the combined measures and rules win in two domains, DEPOTS and BLOCKSWORLD, and lose in three domains, LOGISTICS, DRIVERLOG, and ZENOTRAVEL. Thus, for some domains, we can learn measures from FF trajectories that allow us (in combination with rules) to outperform FF on large problems.

Stochastic Domains

For stochastic domains, we compared our results to FF-Replan, the overall winner of IPPC 2004. FF-Replan determinizes a stochastic domain using the most likely outcomes for each action, and solves the resulting deterministic domain, replanning using FF whenever an "unexpected" outcome occurs. The learned measures substantially improve

Domains	Training Problem Size	Testing Problem Size
Depots	2 depots, 2 distributors 6 crates	2 depots, 2 distributors 12 crates
Driverlog	3 junctions, 3 drivers 6 packages, 3 trucks	3 junctions, 8 drivers 10 packages, 8 trucks
ZenoTravel	5 cities, 2 planes 8 persons	6 cities, 3 planes 20 persons
Block	10 blocks	20 blocks
Logistic	1 plane, 3 cities, 3 locations 10 packages	1 plane, 3 cities, 3 locations 20 packages
(Un) Colored Blocksworld	10 blocks, 5 colors	20 blocks, 5 colors
Exploding	5 blocks	10 blocks
Boxworld	5 boxes, 5 cities	5 boxes, 10 cities

Figure 2: Domain Sizes

Domains	Rules	Measures	R + M	FF
Depots	0.30 (40)	0.84 (90)	0.90 (51)	0.85 (75)
Driverlog	0.40 (58)	0.47 (188)	0.60 (67)	0.95 (35)
ZenoTravel	0.98 (35)	0.90 (85)	0.98 (34)	1.00 (27)
Block	0.58 (59)	0.94 (83)	0.87 (59)	0.84 (62)
Logistics	0.40 (110)	0.89 (168)	0.92 (109)	1.00 (91)

Figure 3: Deterministic Domains

on the SR of both FF-Replan and the rule-based policies in two of the four domains, again with an increase in plan length. In BOXWORLD, FF-Replan outperforms all of the learned knowledge. As for DRIVERLOG, above we believe that this is due to inadequate knowledge representation.

In the EXPLODING BLOCKSWORLD the rule-based policy significantly outperforms the learned measure in SR, exposing a weakness of our learning technique noted earlier. In particular, in this domain the set of "detonated" blocks never decreases and frequently increases. Thus, the number of detonated blocks is a monotonic property that is learned by our system, and thus following our learned measure will tend to detonate blocks whenever possible. However, blindly detonating blocks in this domain is dangerous as it can destroy the table. As discussed earlier there are a number of extensions to our basic approach to address this issue. We note that the EXPLODING BLOCKSWORLD was one of the most difficult domains from IPPC-1, and the combined rules and measures achieve a respectable 0.74 SR. This is encouraging given that none of the planners in IPPC-1 were able to solve the representative problem from this domain.

Conclusion

We introduced a compact representation for prioritized-ensemble heuristics using lists of set expressions described in taxonomic syntax. We also introduced the learning objective of finding heuristics that are (approximately) strong measures of progress, which suggested a simple technique for learning such heuristics. We show that the learned heuristics are useful for planning in both stochastic and deterministic benchmark domains, and when combined with learned rule-based policies can yield state-of-the-art results.

Domains	Rules	Measures	R + M	FF-Replan
Exploding	0.44 (63)	0.10 (37)	0.74 (67)	0.07 (20)
Colored	0.69 (71)	0.96 (77)	0.97 (71)	0.45 (67)
UnColored	0.70 (70)	0.96 (76)	0.97 (71)	0.46 (68)
Boxworld	0.26 (51)	0.20 (71)	0.20 (34)	1.00 (36)

Figure 4: Stochastic Domains

Acknowledgements

We thank the reviewers for helping to improve this paper. This work was supported by NSF grant 0093100-IIS.

References

Ambite, J. L.; Knoblock, C. A.; and Minton, S. 2000. Learning plan rewriting rules. In *Artificial Intelligence Planning Systems*, 3–12.

Furnkranz, J., and Flach, P. A. 2003. An analysis of rule evaluation metrics. In *ICML*.

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

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

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

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

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *Proceedings of the 7th International Conference on Knowledge Representation and Reasoning*.

McAllester, D., and Givan, R. 1993. Taxonomic syntax for first-order inference. *Journal of the ACM* 40:246–283.

Morales, E., and Sammut, C. 2004. Learning to fly by combining reinforcement learning with behavioural cloning. In *ICML*.

Parmar, A. 2002. A Logical Measure of Progress for Planning. In *AAAI/IAAI*, 498–505. AAAI Press.

Rivest, R. 1987. Learning decision lists. *Machine Learning* 2(3):229–246.

Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*. Aberdeen: Morgan Kaufmann.

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

Younes, H. L. S., and Littman, M. L. 2004. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. In *Technical Report CMU-CS-04-162*.