Learning Heuristic Functions from Relaxed Plans

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 present a novel approach to learning heuristic functions for AI planning domains. Given a state, we view a relaxed plan (RP) found from that state as a relational database, which includes the current state and goal facts, the actions in the RP, and the actions' add and delete lists. We represent heuristic functions as linear combinations of generic features of the database, selecting features and weights using training data from solved problems in the target planning domain. Many recent competitive planners use RP-based heuristics, but focus exclusively on the length of the RP, ignoring other RP features. Since RP construction ignores delete lists, for many domains, RP length dramatically under-estimates the distance to a goal, providing poor guidance. By using features that depend on deleted facts and other RP properties, our learned heuristics can potentially capture patterns that describe where such under-estimation occurs. Experiments in the STRIPS domains of IPC 3 and 4 show that best-first search using the learned heuristic can outperform FF (Hoffmann & Nebel 2001), which provided our training data, and frequently outperforms the top performances in IPC 4.

Introduction

It is somewhat surprising that a number of today's state-of-the-art planners are based on the old idea of forward state-space heuristic search (Bonet & Geffner 1999; Hoffmann & Nebel 2001; Nguyen, Kambhampati, & Nigenda 2002). These performances are primarily due to the recent progress in defining domain-independent heuristic functions that provide good guidance across a wide range of domains. However, there remain many domains where these heuristics are deficient, leading to planning failure. In this work, we consider an inductive approach to correcting such deficiencies. Given a set of solved problems from a target domain, our technique learns to augment a domain-independent heuristic in order to improve planning performance in that domain.

Central to our approach is the use of relaxed planning problems. A relaxed planning problem is a simplified version of a target problem where all delete lists on actions have been removed. Good plans for such problems (or relaxed plans) can be found quickly, and relaxed-plan length often provides an accurate heuristic for many planning domains (Hoffmann & Nebel 2001). However, in a number of domains (e.g. Blocksworld, Gridworld), relaxed-plan length severely underestimates the distance to goal, leading to poor guidance.

One way to improve on relaxed-plan length as a heuristic is to partially incorporate delete list information. Pairwise mutex relationships (Blum & Furst 1995) are one form of partial information and can be included in heuristic computations at polynomial cost (Geffner 2004; Nguyen, Kambhampati, & Nigenda 2002). However, it can be difficult to balance the trade-off between searching more nodes with a weaker heuristic and using a more accurate heuristic that takes longer to compute.

In this work, we consider an inductive way to leverage the delete lists for the actions in a relaxed plan, along with additional properties of relaxed plans, in order to improve upon the basic relaxed-plan length heuristic. The resulting learned heuristics can be computed with little additional cost beyond that of computing a relaxed plans.

Our learning approach is very simple. We attempt to learn a linear regression function that approximates the difference between relaxed-plan length and the observed distance-to-goal of states in the available training plans. We then take the sum of the relaxed-plan length and regression function as our new heurstic. A main challenge is to define a generic feature space that allows learning of good regression functions across a wide range of planning domains. Our contribution here is to describe such a feature space. Given a current state and goal, our feature space is based on constructing a relational database that contains the actions in the relaxed plan, the actions' add and delete lists, along with the current state and goal. Our features then correspond to taxonomic syntax expressions (McAllester & Givan 1993), each describing a set of objects in the database, taking the feature values to be the sizes of these sets.

While there has been a substantial body of work on learning heuristics or value functions for search, e.g. (Boyan & Moore 2000; Zhang & Dietterich 1995; Buro 1998), virtually all such work uses human engineered features for each domain considered. Here, such engineering is not required on a per domain basis. Our feature space is defined on any STRIPS planning domain, and our results show that the features lead to good performance in a range of domains.

We implemented our ideas in a planning system and evaluated the performance in all of the STRIPS domains of the two most recent planning competitions. We train on the first 15 competition problem in each domain and test on the remaining problems. The results show that best first search with the learned heuristic typically outperforms FF, the planner used to generate our training trajectories. In addition, in a number of domains the learned heuristic outperforms the top performances in the competition results of IPC4.

Problem Description

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

Given a planning domain, a planning problem is a tuple (s, A, g), where $A \subseteq \mathcal{A}$ is a set of actions, $s \in \mathcal{S}$ is the initial state, and g is a set of state facts representing the goal. A solution plan for a planning problem is a sequence of actions $(a_1, \ldots, a_l)$, where the sequential application of the sequence starting in state s leads to a goal state s' where $g \subseteq s'$.

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

In particular, here we focus on developing learning capabilities within the simple, but highly successful, framework of heuristic state-space search planning. Given a problem set from a particular planning domain, our system first uses a domain-independent planner, in our case FF, to solve the easier planning problems in the set. The solutions are then used to learn an augmentation to FF's heuristic function that is better suited to the domain, facilitating the solution of harder problems. It would be natural then to learn from those newly solved harder problems, though we do not consider such iterative learning in this paper.

Defining Heuristics Using Relaxed Plans

A heuristic function $H(s, A, g)$ is simply a function of a state s, action set A, and goal g that estimates the cost of achieving the goal from s using actions in A. In this work we will consider learning heuristic functions that are represented as weighted linear combinations of features, i.e. $H(s, A, g) = \Sigma_i w_i \cdot f_i(s, A, g)$. In particular, for each domain we would like to learn a distinct set of features and their corresponding weights that lead to good planning performance in that domain. One of the challenges with this approach is to define a generic feature space from which features are selected. This space must be rich enough to capture important properties of a wide range of planning domains, but also be amenable to searching for those properties. The main contribution of this work is to define such a feature space, based on properties of relaxed plans. Below we first review the concept of relaxed plans and how they have previously been used in heuristic search. Next, we describe how we extend that previous work by using relaxed plans to define features for planning domains.

Relaxed Plans

Given a planning problem (s, A, g) we define the corresponding relaxed planning problem to be the problem (s, A^+, g) where the new action set A^+ is created by copying A and then removing the delete list from each of the actions. Thus, a relaxed planning problem is a version of the original planning problem where it is not necessary to worry about delete effects of actions. A relaxed plan for a planning problem (s, A, g) is simply a plan that solves the relaxed planning problem.

Relaxed planning problems have two important characteristics. First, although a relaxed plan may not necessarily solve the original planning problem, the length of the shortest relaxed plan serves as an admissible heuristic for the original planning problem. This is because preconditions and goals are defined in terms of positive state facts, and hence removing delete lists can only make it easier to achieve the goal. Second, in general, it is computationally easier to find relaxed plans compared to solving general planning problems. In the worst case, this is apparent by noting that the problem of plan exisitence can be solved in polynomial time for relaxed planning problems, but is PSPACE-complete for general problems. However, it is still NP-hard to find minimum-length relaxed plans. Nevertheless, practically speaking, there are very fast polynomial time algorithms that typically return short relaxed plans whenever they exists, and the lengths of these plans, while not admissible, often provide good heuristics. There are likely no such algorithms for unrestricted planning as indicated by the PSPACE-hardness of plan existence.

The above observations have been used to realize a number of state-of-the-art planners based on heuristic search. HSP (Bonet & Geffner 1999) uses forward state-space search guided by a heuristic that estimates the length of the optimal relaxed plan. FF (Hoffmann & Nebel 2001) also takes this approach, but unlike HSP estimates the optimal relaxed-plan length by explicitly computing a relaxed plan.

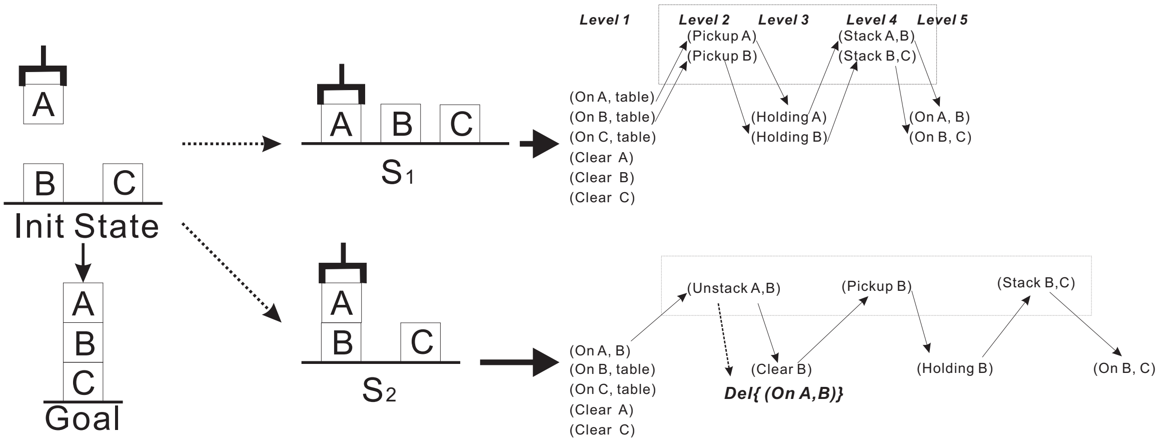

Figure 1: Blocksworld Example

Our work builds on FF, using the same relaxed-plan construction technique, but using information about the relaxed plan other than just its length to guide search. Below we briefly review FF's relaxed-plan extraction approach.

FF computes relaxed plans using a relaxed plan graph (RPG). An RPG is simply the usual plan graph created by Graphplan (Blum & Furst 1995), but for the relaxed planning problem rather than the original problem. Since there are no delete lists in the relaxed plan, there will be no mutex relations in the plan graph. The RPG is a leveled graph alternating between action levels and state-fact levels, with the first level containing the state facts in the initial state. An action level is created by including any action whose preconditions are satisfied in the preceding state-fact level. A state-fact level is created by including any fact that is the add list of an action in the preceding action level or that is in the initial state. RPG construction stops when a fixed point is reached or the goal facts are all contained in the most recent state-fact level. After constructing the RPG for a planning problem, FF starts at the last RPG level and uses a backtrack-free procedure that extracts a sequence of actions that correspond to a successful relaxed plan. All of this can be done very efficiently, allowing for fast heuristic computation.

While the length of FF's relaxed plan often serves as an effective heuristic, for a number of planning domains, ignoring delete effects leads to severe underestimates of the distance to a goal. The result is poor guidance and failure on all but the smallest problems. One way to overcome this problem would be to incorporate partial information about delete lists into relaxed plan computation, e.g. by considering mutex relations. However, to date, this has not born out as a practical alternative. Another possibility is to use more information about the relaxed plan than just its length. For example, (Vidal 2004) uses relaxed plans to construct macro actions, which help the planner overcome regions of the state space where the relaxed-plan length heuristic is flat. However, that work still uses length as the sole heuristic value. In this work, we take a different approach where we use features computed from the relaxed plan in order to augment and improve the basic relaxed-plan length heuristic. In particular, this approach will allow the heuristic to be sensitive to delete lists of actions by using features that depend on those delete lists.

Example

As an example of how relaxed plans can be used to define useful features, consider a problem from the Blocksworld in Figure 1. Here we show two states S_1 and S_2 that can be reached from the initial state by applying the actions Putdown(A) and Stack(A, B) respectively. From each of these states we show the optimal relaxed plans for achieving the goal. For these states, the relaxed-plan length heuristic is 3 for S_2 and 4 for S_1, suggesting that S_2 is the better state. However, it is clear that, in fact, S_1 is better.

Notice that in the relaxed plan for S_2, on(A, B) is in the delete list of the action Unstack(A, B) and at the same time it is a goal fact. One can improve the heuristic estimation by adding together the relaxed-plan length and a term related to such deleted facts. In particular, suppose that we had a feature that computed the number of such "on" facts that were both in the delete list of some relaxed plan action and in the goal, giving a value of 0 for S_1 and 1 for S_2. We could then weight this feature by two and add it to the relaxed-plan length to get a new heuristic. This would assign a value of 4 for S_1 and 5 for S_2, correctly ranking the states. While this is an over-simplified example, it is suggestive as to the utility of utilizing features derived from relaxed plans. Below we describe a domain-independent feature space that can be instantiated for any planning domain. Our experiments show that these features are useful across a range of domains used in planning competitions.

A Relaxed-Plan Feature Space

In this work, we assume the context of a particular planning domain, noting that our feature space is defined for any planning domain. Each feature $f(s, A, g)$ is an integer valued function of a state s, goal g, and action set A. The goal is to define a space of such features that will strongly correlate with the length of the shortest plan for achieving g from s.

Each of our features is represented by a taxonomic syntax expression, as described below. Given a planning problem (s, A, g) the feature value is computed in a two step process,

1. Create a relational database that contains information about the state, goal, and relaxed plan for (s, A, g).
2. Evaluate the feature's taxonomic syntax expression with respect to the database, resulting in a set of constants from the database facts. The feature value is taken to be the size of this set.

We note that our use of taxonomic syntax here is primarily for historical reasons, following previous work that also used this language for describing properties of planning domains, e.g. (Fern, Yoon, & Givan 2004; Yoon, Fern, & Givan 2005). However, many other possibilities could be considered, e.g. description logic, or the standard syntax of first-order predicate calculus. The main novelty here is the use of a relational database that depends on the relaxed plan, which can then be exploited by any relational feature language. Below we discuss the above steps in more detail, first describing the database construction, and then presenting the syntax and semantics of taxonomic syntax.

Database Construction. Given a problem (s, A, g) let $(a_1, \ldots, a_n)$ be the relaxed plan, i.e. the one computed by FF's heurstic calculation. We denote by $\text{RDB}(s, A, g)$ the relational database that contains the following facts:

- All of the state facts in s.
- The name of each action a_i. Recall that each name is an action type applied to the appropriate number of constants, e.g. $\text{Unstack}(A, B)$.
- For each state fact in the add list of some action a_i, add a fact to the database that is the result of prepending an **a** to the fact's predicate symbol. For example, in Figure 1, for state S_2 the fact $\text{Holding}(B)$ is in the add list of $\text{Pickup}(B)$, and thus we would add the fact $\textbf{a}\text{Holding}(B)$ to the database.
- Likewise for each state fact in the delete list of some a_i, we prepend a **d** to the predicate symbol and add the resulting fact to the database. For example, in Figure 1, for S_2, we would add the fact $\textbf{d}\text{on}(A, B)$.
- For each state fact in the goal g, we prepend a **g** to the predicate symbol and add it to the database. For example, in Figure 1, we would add the facts $\textbf{g}\text{on}(A, B)$ and $\textbf{g}\text{on}(A, B)$.

Thus the database captures information about the state, goal, actions in the relaxed plan, along with their add and delete lists. Notice that the database does not capture information about the temporal structure of the relaxed plan. Such temporal information may be useful for describing heuristics, and is a natural extension of our current investigation.

Taxonomic Syntax. Taxonomic syntax is a language for writing class expressions that are used to denote sets of objects. Given a class expression C and a database D of relational facts, $C[D]$ denotes a particular set of constants (or objects) in D. Given a particular planning domain with predicate symbols P and action types Y, the possible class expressions for that domain are given by the following:

$$\begin{aligned} C \quad &= \quad C_0 \mid \textbf{a-thing} \mid C \cap C \mid \neg C \mid \\ & \quad\quad (R \;\; C_1 \ldots C_{i-1} \;\; * \;\; C_{i+1} \ldots C_{\text{n}(R)}) \end{aligned}$$

where C and the C_j are class expressions; C_0 is any one argument symbol in Y, or any one argument symbol in P possibly prepended by an **a**, **d**, or **g**; R is any multi-argument symbol in Y, or any multi-argument symbol in P possibly prepended by an **a**, **d**, or **g**. Above we denote the arity of symbol R by $\text{n}(R)$.

Given a database of facts D and a class expression we define $C[D]$ as follows. If C is a primitive class expression C_0 then $C[D]$ is the set of constants in D that C is applied to. For example, if D contains facts about a Blocksworld problem, then $\textbf{g}\text{clear}[D]$ denotes the set of blocks that are clear in the goal. The class expression **a-thing** denotes the set of all constants in D. If $C = \neg C'$ then $C[D] = \textbf{a-thing} - C'[D]$. If $C = (R \;\; C_1 \ldots C_{i-1} \;\; * \;\; C_{i+1} \ldots C_{\text{n}(R)})$ then $C[D]$ is the set of all constants c such that there exists $c_j \in C_j[D]$ such that the fact $R(c_1, \ldots, c_{i-1}, c, c_{i+1}, \ldots, c_{\text{n}(R)})$ is in D. For example, if D represents a set of facts about a Blocksworld problem, then $(\text{on} \;\; * \;\; \text{clear})[D]$ is the set of blocks that are under clear blocks.

Given the syntax and semantics of class expressions we can now define our feature space for a given planning domain. Each possible class expression C corresponds to a feature f_C. Given a planning problem (s, A, g) the value of this feature is given as $f_C(s, A, g) = |C[\text{RDB}(s, A, g)]|$. That is, the value of the feature is equal to the size of the set of constants denoted by the class expression in the database described above. This defines an infinite space of features for any given planning domain.

Learning

The input to our learning algorithm is a set of solutions to planning problems from a target domain. We do not assume that these solutions are optimal, though there is an implicit assumption that the solutions are reasonably good. Our learning objective is to learn a heuristic function that closely approximates the observed distance-to-goal for each state in the training solutions. To do this we first create a derived training set $\mathbb{J}$ that contains a training example for each state in the solution set. In particular, for each training problem (s_0, A, g) and corresponding solution trajectory $(s_0, s_1, \ldots, s_n)$ we add to $\mathbb{J}$ a set of n examples $\{\langle (s_i, A, g), n - i\rangle \mid i = 0, \ldots, n-1\}$, each example being a pair of a planning problem and the observed distance-to-goal in the training trajectory. Given the derived training set $\mathbb{J}$ we then attempt to learn a real valued function $\Delta(s, A, g)$ that closely approximates the difference between the distances recorded in $\mathbb{J}$ and the value of FF's relaxed-plan length heuristic $\text{RPL}(s, A, g)$. We then take $H(s, A, g) = \text{RPL}(s, A, g) + \Delta(s, A, g)$ to be the final heuristic function.

We represent $\Delta(s, A, g)$ as a finite linear combination of features selected from the feature space described in the previous section, i.e. $\Delta(s, A, g) = \Sigma_i w_i \cdot f_{C_i}(s, A, g)$ where

C_i is the class expression that defines feature i. Learning thus involves selecting a set of features from the above infinite space and assigning their weight values. One approach to this problem would be to impose a length bound on class expressions and then learn the weights (e.g. using least squares) for a linear combination that involves all features whose class expression are within the bound. However, the number of such features is exponential in the length bound, making this approach impractical for all but very small bounds. Such an approach will also have no chance of finding important features beyond the fixed length bound. In addition, we would prefer to use the smallest possible number of features, since the time complexity of evaluating the learned heuristic grows in proportion to the number of selected features. Thus, we consider a greedy approach where we heuristically search through the space of features, without imposing apriori length bounds.

Figure 2 gives our algorithm for learning $\Delta(s, A, g)$ from a derived training set $\mathbb{J}$. The main procedure **Learn-Delta** first creates a modified training set $\mathbb{J}'$ that is identical to $\mathbb{J}$ except that the distance-go-goal of each training example is changed to the difference between the distance-to-goal and FF's relaxed-plan length heuristic. Each iteration of **Learn-Delta** maintains a set of class expressions Φ, which represents the set of features that are currently under consideration. Initially Φ is equal to the set of primitive class expressions, that is, expressions of length one. Each iteration of the loop has two main steps. First, we use the procedure **Learn-Approximation** to select a subset of class expressions from Φ and to compute their feature weights. The result is a set of features and weights that represent the current approximaton to $\Delta(s, A, g)$. Second, we expand the candidate feature set Φ, using the selected features as seeds, via a call to the procedure **Expand-Features**. This results in a larger candidate feature set, including the seed features, which is again used by **Learn-Approximation** to find a possibly improved approximation. We continue alternating between feature space expansion and learning an approximation until the approximation accuracy does not improve. Here we measure the accuracy of the approximation by the R-square value, which is the fraction of the variance in the data that is explained by the linear approximation.

Learn-Approximation uses a simple greedy procedure. Starting with an empty feature set, on each iteration the feature from Φ that can most improve the R-square value of the current feature set is included in the approximation. This continues until the R-square value can no longer be improved. Given a current feature set, the quality of a newly considered feature is measured by calling the function *lm* from the statistics tool R, which outputs the R-square value and weights for a linear approximation that includes the new feature. After observing no improvement, the procedure returns the most recent set of selected features along with their weights, yielding a linear approximation of $\Delta(s, A, g)$.

The procedure **Expand-Features** creates a new set of class expressions that includes the seed set, along with new expressions generated from the seeds. There are many possible ways to generated an expanded set of features from a given seed C. Here we consider three such expansion functions that worked well in practice. The first function **Relational-Extension** takes a seed expression C and returns all expressions of the form $(R\ \ c_0 \ldots c_{j-1}\ \ C\ \ c_{j+1} \ldots c_{i-1}\ \ *\ \ c_{i+1} \ldots c_{\mathrm{n}(R)})$, where R is a predicate symbol of arity larger than one, the c_i are all **a-thing**, and $i, j \leq \mathrm{n}(R)$. The result is all possible ways of constraining a single argument of a relation by C and placing no other constraints on the relation. The second procedure for generating new expressions from C is **Specialize**. This procedure simply generates all class expressions that can be created by replacing a single primitive expression C' of C with the intersection of C' and another primitive class expression or the result of applying **Relational-Extension** to a primitive expression. Note that all expression generated by **Specialize** will be subsets of C. Finally we add the negation of the seed concept to the expanded feature set.

Learn-Delta ($\mathbb{J}$)
$\mathbb{J}' \leftarrow \{\langle p, d - \text{RPL}(p)\rangle \mid \langle p, d\rangle \in \mathbb{J}\}$
$\Phi \leftarrow \{C \mid C$ is a primitive class expression$\}$
repeat until no R-square value improvement observed
    $(\Phi', W) \leftarrow$ **Learn-Approximation**($\mathbb{J}'$, Φ)
    $\Phi \leftarrow$ **Expand-Features**($\mathbb{D}$, Φ')
Return Φ', W

Learn-Approximation ($\mathbb{J}, \Phi$)
$\Phi' \leftarrow \{\}$ *// return features*
repeat until no improvement in R-square value
    $C \leftarrow \arg\max_{C \in \Phi}$ R-square($\Phi' \cup \{C\}$)
    $\Phi' \leftarrow \Phi' \cup \{C\}$
$W \leftarrow lm(\mathbb{J}, \Phi')$
Return Φ', W

Expand-Features (Φ')
$\Phi \leftarrow \Phi'$ *// return features*
for-each $C \in \Phi'$
    $\Phi \leftarrow \Phi \cup$ **Relational-Extension**(C) $\cup$ **Specialize**(C) $\cup \{\neg C\}$
Return Φ

Figure 2: The learning algorithm used to approximate the difference between the relaxed planning heuristic and the observed plan lengths in the the training data.

Experiments

We evaluated our learning approach in all of the STRIPS domains of the two most recent international planning competitions, IPC3 and IPC4. Each domain provided a sequence of problems, roughly ordered by increasing hardness. We used the first 15 problems in each domain for training and the remaining problems for testing. In each domain, we generated training trajectories by running FF on each training problem with a time cut-off of 30 CPU minutes. For most domains this resulted in 15 training trajectories, and never fewer than 12. Learning times varied across domains, depending on the number of predicates and actions, which generally dictates the number of class expressions that will be considered during learning. We report the learning time for each domain at the end of the corresponding figures. Except for Freecell domain, the learning finishes in less than an hour. For all the experiments, we used Linux box with 2 Gig RAM and 2.8 Ghz Intel Xeon CPU. After training in a domain, the learned heuristic was applied to each testing problem using best-first search. We recorded the time to solve each problem with a time cut-off of 30 CPU minutes per problem.

Results on IPC3 Domains IPC3 included six STRIPS domains: Zenotravel, Satellite, Rover, Depots, Driverlog, and FreeCell. FF's heuristic is very accurate for the first three domains, trivially solving the problem sets, leaving little room for learning. Thus, here we only report results on Depots, Driverlog, and FreeCell. Each domain includes 20 problems, and FF was able to generate training data for all 15 training problems. Figure 3 gives the CPU time required to solve each of the five test problems for three planners: 1) FF, 2) LEN, a baseline planner that conducts best-first search guided solely by FF's relaxed-plan length heuristic. 3) L, our planner based on best-first search using the learned heuristic, which is an augmented version of the heuristic used by LEN. Note that LEN is identical to FF except that it does not include FF's enforced hill climbing (EHC) stage or goal-ordering information. Entries in the table without numbers indicate that the problem was not solved within the cuttoff time. Note that FF was one of the top performers in IPC3. The last row in the table shows the learning time for L.

For Depots, both FF and L were able to solve all of the test problems. Here the learned heuristic shows no benefit over FF, as FF is already very effective. However, L performed better than LEN, which shows that L was able to learn an improved heuristic compared to relaxed-plan length. FF is able to improve on LEN via the use of EHC, an alternative approach to overcoming a poor heuristic or the use of goal ordering. For Freecell, each of the planners were able to solve all of the test problems. In this case, the learned heuristic leads to substantially faster solution times for the three most difficult problems. The performance of LEN shows no advantage of EHC in difficult problems of the domain. For Driverlog, FF and LEN solved just one out of five test problems, whereas L solved that problem significantly faster along with two addition problems. These results demonstrate the potential benefits of learning heuristics from previous experience.

Results on IPC4 Domains IPC4 included seven STRIPS domains: Airport, Satellite, Pipesworld, Pipesworld-with-Tankage, Optical (Promela), Process (Promela), and PSR (middle-compiled). FF's heuristic is very accurate for the first two domains, where for all of the solved problems the solution length and FF's heuristic are almost identical, leaving little room for learning. Thus, we only give results for the latter five domains. Each domain includes either 48 or 50 problems, giving a total of 33 or 35 testing problems. Figure 4 gives the CPU time required to solve each problem for LEN, FF, our system L, and the competition's best performer in the domain, labeled as B. Note that the CPU times for the best performer were taken from the official competition results, and thus are not exactly comparable to the CPU times for LEN, FF and L, which were obtained on our own system. The last row in the table shows the learning time for L.

In both of the Pipesworld domains, the performance of LEN, FF and L was weaker than the top performer. L solved more problems than FF and LEN, and often improved on their solution times. However, while L improved overall performance over FF and LEN, we see that it did fail to solve two problems that FF was able to solve, and occasionally increased the solution time. For the PSR domain, L solved more problems than FF and typically improved over FF's solution time. Furthermore, we see that L is able to solve more problems than the top performer B, noting that this comparision is only suggestive, as B and L were run on different machines. Compared to LEN the learned heuristic allows for the solution of three more problems in total. Finally, we see that for both of the Promela domains (Philosophers and Optical Telegraph), L outperfomed LEN, FF and the top performer by large margins. In these domains, we see that relaxed-plan length alone is a deficient heuristic, whereas L is able to learn other properties of RPs that apparently solve these domains. Overall, the IPC4 results show that compared to using relaxed-plan length as a heuristic, the learned heuristics are able to solve more problems and typically speed up planning time.

	Depots			Driverlog			FreeCell		
PR	LEN	FF	L	LEN	FF	L	LEN	FF	L
16	0.28	0.11	1.02	-	-	38.8	26.5	2.93	12.22
17	-	1.66	0.87	-	-	150	60.2	6.14	15.14
18	-	1.28	72.5	-	-	-	31.6	224	127
19	-	0.49	4.25	1623	1105	410	438	909	215
20	-	18	793	-	-	-	163	1730	77.1
Learning Time			325			274			4214

Figure 3: IPC 3 Results. For each domain we show the time required (in seconds) to find a solution on each of the 5 training problems for three planners: 1) LEN, a best-first search using FF's heuristic, 2) FF, and 3) L, best-first search using the learned heuristic. A dash in the table indicates that the planner was unable to find a solution within 30 minutes. The last row in the table gives the learning time in seconds for L. Note that learning only occurs once per domain before solving the test problems.

Feature Space Comparison. Our learned heuristics were based on features constructed from facts about the current state, the goals, relaxed-plan actions, and add/delete lists of those actions. Here we wish to examine which types of information contribute most to the performance of our learned heuristics. To do this, we conducted the same experiments as above, except that we removed different types of facts from consideration by our learner. As a baseline, the first column of Figure 5 records the number of test problems solved by LEN. The second column records the number of problems solved by L. Recall that L uses all of the available information (state, goal, relaxed plan) in order to augment the basic relaxed-plan length heuristic.

First, we consider the impact of using delete-list facts, which are particularly interesting as they are not used in relaxed-plan construction. The third column records the number of solved problems for a learned heuristic that can use all facts, except for the delete lists of relaxed-plan actions (denoted as L-D). Comparing the performance of L and L-D we see that the removal of the delete-list facts significantly hurts performance in the Optical Telegraph domain. Performance is also hurt, to a lesser degree in three additional domains. However, we see that the delete-lists facts alone are not solely responsible for the performance gap between relaxed-plan length (LEN) and L.

Next we consider, the impact of removing all information related to the relaxed plan. This limits the learner to using only information about the current state and goal in order to augment the relaxed-plan length heuristic. This set of information is particularly interesting since it is the same as used in the previous works on learning policies (Fern, Yoon, & Givan 2004), and measures-of-progress (Yoon, Fern, & Givan 2005) in planning domains. The performance of the learned heuristic is shown as the column labeled L-RP. Comparing L-D to L-RP we see that the removal of the remaining relaxed-plan facts results in significantly decreased performance on two of the domains. This shows that apparently the learner is able to exploit information in the relaxed plan other than just delete lists in order to usefully augment relaxed-plan length. In three domains, we see that L-RP solves a small number of additional problems over LEN, and in three others domains the number of solved problems slightly decreases.

Overall the results indicate that across these domains the learner is not able to significantly improve on LEN using just state and goal information. Rather, the bulk of the improvement demonstrated by L can be attributed to the use of relaxed-plan–based information.

Qualitative Assessment of Learned Features. Somewhat surprisingly, across our domains, the average number of features selected for inclusion in the learned heuristic was only two. Typically the class expressions corresponding to these features were quite long, involving tens of predicate symbols. Thus, for each domain our learning approach was able to discover a small number of quite complex class expressions that resulted in good accuracy. A learning approach based on any reasonable depth bound would not have discovered such features. Unfortunately, we were not able to understand any of the discovered features at an intuitive level. One of our immediate goals is to study the learned expressions, by watching search traces, in order to better understand where they are providing leverage over the simple relaxed-plan length heuristic.

Discussion and Future Work

There has been much prior work in the area of learning to plan. Still, in the context of competition planning domains, there is no learning system that can compete with state-of-the-art "non-learning" planners. We believe that one of the key hurdles has been in selecting a knowledge representation that is expressive enough for a wide range of domains, yet facilitates robust learning. While many representations have been explored, e.g. search-control rules (Minton 1993), HTN preconditions (Ilghami *et al.* 2005), rule-based policies (Khardon 1999; Huang, Selman, & Kautz 2000; Martin & Geffner 2000; Fern, Yoon, & Givan 2004), none of them have demonstrated reliable performance across a significant number of the more recent competition benchmarks.

Domains	LEN	L	L-D	L-RP
Depots	1	5	2	3
Driverlog	1	3	2	1
FreeCell	5	5	3	3
Pipesworld	15	24	24	15
Pipesworld-tankage	4	12	8	8
PSR	21	24	24	24
Philosopher	0	33	33	0
Optical Telegraph	0	33	0	0

Figure 5: Feature Space Comparison. The table gives the total number of test problems solved across all domains for best-first search using four different heuristics: 1) LEN, FF's heuristic, 2) L, the heuristic learned based on relaxed plan, state, and goal information, 3) L-D, the heuristic learned with all information used by L except for deleted facts in the relaxed plan, and 4) L-RP, the heuristic learned from just state and goal information.

The primary contribution of this work was to introduce a novel representation, based on generic features of relaxed plans. We showed that with this representation, a very simple combination of heuristic search planning and linear function approximation led to highly competitive performance. To the best of our knowledge this is the first system that learns to improve a state-of-the-art heuristic search planner. In addition, it is the first published evaluation of a learning-to-plan system on the IPC3 and IPC4 benchmarks. Based on the experiments, we believe that our approach is a promising direction for finally developing learning systems that yield state-of-the-art domain-independent planning performance.

Our immediate future work will be to improve our system so that it can be used in the context of a planning competition. The primary hurdle is to speed-up the learning component, so that newly solved problems from a given domain can be used quickly to improve the heuristic. We believe that this can be accomplished by using more sophisticated feature-enumeration techniques and incremental weight learning.

One natural extension to the feature space we've provided in this paper is to consider properties based on the temporal structure of relaxed plans. This could be accomplished by extending our current feature language to include temporal modalities. The learning approach taken here reduces the problem of heuristic learning to one of standard function approximation. There are a number of ways in which we might further improve the quality of the learned heuristic. One approach would be to use ensemble-learning techniques such as bagging (Breiman 1996) where we learn and combine multiple heuristic functions. Another more interesting extension would be to develop a learning technique that explicitly considers the search behavior of the heuristic, focusing on parts of the state space that need improvement most. Along these lines we plan to study discriminative learning techniques for this problem, where we discriminately train the heuristic function to rank states so that it leads best-first search to the goal quickly. While in concept, existing techniques (Daume III & Marcu 2005) can be used for this purpose, there are many challenges in using them for AI planning.

References

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

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

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

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

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

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

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*.

Geffner, H. 2004. Planning graphs and knowledge compilation. In *International Conference on Principles of Knowledge Representation and Reasoning*.

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

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

Ilghami, O.; Munoz-Avila, H.; Nau, D. S.; and Aha, D. W. 2005. Learning preconditions for planning from plan traces and htn structure. In *International Conference on Machine Learning (ICML)*.

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.

Minton, S., ed. 1993. *Machine Learning Methods for Planning*. Morgan Kaufmann Publishers.

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

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *International Conference on Automated Planning and Scheduling*.

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

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

	Pipesworld				Pipesworld tankage				PSR				Philosophers				Optical Telegraph			
Pr	LEN	FF	L	B	LEN	FF	L	B	LEN	FF	L	B	LEN	FF	L	B	LEN	FF	L	B
16	1.86	5.04	0.90	6.02	-	-	89.4	-	518	476	52.1	103	-	-	0.86	0.1	-	-	17.6	171
17	23.1	18.5	6.74	0.02	527	489	828	1186	176	187	41.7	34.2	-	-	0.99	0.09	-	-	22.2	228
18	45.5	0.46	1.72	0.03	-	-	142	61	1337	1206	68.5	239	-	-	1.12	0.15	-	-	27.9	316
19	-	0.08	0.53	0.03	-	-	3.97	0.41	301	288	59.9	47.7	-	-	1.29	0.14	-	-	33.9	403
20	3.52	0.10	0.67	0.11	-	-	1049	-	1594	1491	217	341	-	-	1.44	0.14	-	-	42.5	528
21	-	1.97	0.28	0.03	-	-	2.11	0.37	-	-	586	-	-	-	1.64	0.19	-	-	51.2	673
22	-	-	-	0.06	-	-	168	233	299	260	123	53	-	-	1.82	0.20	-	-	61.5	865
23	0.71	0.67	0.34	0.03	-	-	24.7	0.82	202	282	369	52.7	-	-	2.05	0.21	-	-	73.7	1068
24	5.51	4.37	0.46	0.06	-	-	39.7	3.56	102	244	35.8	26.2	-	-	2.33	0.23	-	-	87.7	1321
25	-	0.95	20.9	0.05	-	-	-	3.38	217	185	294	36.9	-	-	2.59	0.25	-	-	104	1633
26	-	-	5.16	0.08	-	-	-	2.48	575	546	1205	82.1	-	-	2.86	0.25	-	-	121	-
27	20.2	17.6	8.94	0.09	-	-	-	2.75	446	411	558	99.8	-	-	3.23	0.26	-	-	142	-
28	249	812	2.05	0.05	-	-	-	252	94.5	183	120	25.1	-	-	3.58	0.31	-	-	166	-
29	18.3	16.9	88	0.12	-	-	-	12.8	-	-	-	-	-	-	3.97	0.34	-	-	201	-
30	-	7.82	87.6	0.07	-	-	-	3.43	1642	-	465	295	-	-	4.37	-	-	-	223	-
31	-	12.4	2.15	0.05	93.4	124	42.8	16.8	213	387	148	48	-	-	4.84	-	-	-	258	-
32	4.84	7.74	1.71	0.08	26.3	26.2	79.9	4.1	-	-	-	-	-	-	5.29	-	-	-	300	-
33	99	74.7	40	0.06	-	-	-	1787	1306	1559	-	297	-	-	5.94	-	-	-	341	-
34	521	400	2.29	0.07	-	-	-	1.48	-	-	339	-	-	-	6.40	-	-	-	384	-
35	3.28	47.3	7.15	0.15	-	-	-	-	-	-	-	-	-	-	7.01	-	-	-	434	-
36	-	-	48.1	0.65	-	-	-	65.2	-	-	364	-	-	-	7.66	-	-	-	494	-
37	-	-	20.4	0.44	-	-	-	10.1	985	-	-	-	-	-	8.17	-	-	-	558	-
38	-	-	-	0.31	-	-	-	29.3	-	-	-	-	-	-	9.07	-	-	-	627	-
39	-	-	20.4	0.14	-	-	-	24.7	1037	-	1316	-	-	-	9.69	-	-	-	703	-
40	-	-	53.3	0.33	-	-	-	112	-	-	-	-	-	-	10.7	-	-	-	787	-
41	45.8	54.2	3.00	0.12	686	1488	1486	0.86	-	-	1781	-	-	-	11.7	-	-	-	865	-
42	-	-	-	6.36	-	-	-	-	1739	-	-	-	-	-	12.8	-	-	-	964	-
43	-	-	-	32.4	-	-	-	-	552	1123	813	169	-	-	13.6	-	-	-	1077	-
44	-	-	-	71.8	-	-	-	319	-	-	-	-	-	-	14.6	-	-	-	1208	-
45	-	-	-	4.76	-	-	-	60.4	-	-	-	-	-	-	15.7	-	-	-	1328	-
46	-	-	-	0.52	-	-	-	1460	531	1324	306	202	-	-	16.9	-	-	-	1457	-
47	-	-	-	0.89	-	-	-	-	-	-	-	-	-	-	18.2	-	-	-	1601	-
48	-	-	-	1.25	-	-	-	-	700	1615	481	266	-	-	19.9	-	-	-	1777	-
49	27.2	3.87	-	1.11	-	-	-	228												
50	-	7.18	-	0.59	-	-	-	330												
Learning Time			709				2091				2848				340				826	

Figure 4: IPC 4 Results. For each domain we show the time required (in seconds) to find a solution on each of the 5 training problems for four planners: 1) LEN, a best-first search using FF's heuristic, 2) FF, 3) L, best-first search using the learned heuristic, and 4) B, the best performing planner in the competitoin on the particular domain. A dash in the table indicates that the planner was unable to find a solution within 30 minutes. The last row in the table gives the learning time in seconds for L. Note that learning only occurs once per domain before solving the test problems.