

An Online Learning Method for Improving Over-subscription Planning

Sungwook Yoon

CSE Dept.
Arizona State Univ.
Tempe, AZ 85281
Sungwook.Yoon@asu.edu

J. Benton

CSE Dept.
Arizona State Univ.
Tempe, AZ 85281
J.Benton@asu.edu

Subbarao Kambhampati

CSE Dept.
Arizona State Univ.
Tempe, AZ 85281
rao@asu.edu

Abstract

Despite the recent resurgence of interest in learning methods for planning, most such efforts are still focused exclusively on classical planning problems. In this work, we investigate the effectiveness of learning approaches for improving over-subscription planning, a problem that has received significant recent interest. Viewing over-subscription planning as a domain-independent optimization problem, we adapt the STAGE (Boyan and Moore 2000) approach to learn and improve the plan search. The key challenge in our study is how to automate the feature generation process. In our case, we developed and experimented with a relational feature set, based on Taxonomic syntax as well as a propositional feature set, based on ground-facts. The feature generation process and training data generation process are all automatic, making it a completely domain-independent optimization process that takes advantage of online learning. In empirical studies, our proposed approach improved upon the baseline planner for over-subscription planning on many of the benchmark problems.

Introduction

Use of learning techniques to improve the performance of automated planners was a flourishing enterprise in the late eighties and early nineties, but has however dropped off the radar in the recent years (Zimmerman and Kambhampati 2003). One apparent reason for this is the tremendous scale-up of plan synthesis algorithms in the last decade fueled by powerful domain-independent heuristics. While early planners needed learning to solve even toy problems, the orthogonal approach of improved heuristics proved sufficiently powerful to reduce the need for learning as a crutch.

This situation is however again changing, with learning becoming an integral part of planning, as automated planners move from restrictive classical planning problems to focus on increasingly complex classes of problems.¹ One such class is “over-subscription” (aka “partial satisfaction”) plan-

ning. Unlike classical planning where the focus is on achieving a set of goals starting from an initial state and all solution plans are considered equally valid, in over-subscription planning (OSP) actions have costs and goals have utilities. Among many classes of OSP problems, we focus our attention on maximizing “net benefit” which is the cumulative utility of goals achieved minus the cumulative cost of actions used in the plan. The problem is further complicated by the fact that the goals have both “cost” and “utility” interactions (the former because achievement of one goal may make it cheaper or costlier to achieve another goal; and the latter because the achievement of one goal may make it more or less useful to achieve another goal).

Like other planning problems, the dominant approach for OSP problems is forward state space search and one challenge in improving over-subscription planners has been in developing effective heuristics that take cost and utility interactions into account (c.f. (Do et al. 2007)). Our research aims to investigate if it is possible to boost the heuristic search with the help of learning techniques. Given the optimizing nature of OSP, we were drawn in particular to STAGE (Boyan and Moore 2000) which had shown significant promise for improving search in optimization contexts.

STAGE is an online learning approach that was originally invented to improve the performance of random-restart hill-climbing techniques on optimization problems. Rather than resort to random restarts which may or may not help the base-level search escape local minimum, STAGE aims to learn a policy that can be used to intelligently generate restart states that are likely to lead the hill-climbing search towards significantly better local optima. The algorithm proceeds in two iterated stages. In the first stage, the base-level hill-climbing search is run until it reaches a local minimum. This is followed by a learning phase where STAGE trains on the sequence of states that the hill-climbing search passed through in order to learn a function that predicts for any given state s the value v of the optima that will be reached from s by hill climbing. This learned function is then used in the second stage (alternative) local search to scout for a state s' (that has the highest promise of reaching a better state). If the learner is effective, s' is expected to be a good restart point for the base-level search. The stages are then repeated starting with s' as the initial point.

The main challenge in adapting the STAGE approach to

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

¹One sign of this renewed interest is the fact that for the first time, in 2008, the International Planning Competition will have a track devoted to planners that employ learning techniques.

OSP involves finding appropriate state features to drive the learner. In their original work, Boyan and Moore used *hand-crafted* state features to drive learning. While this may be reasonable for the applications they considered, it is infeasible for us to hand-generate features for every planning domain and problem. Moreover, such manual intervention runs counter to the basic tenets of domain-independent planning. Rather, we would like the features to be generated automatically from the problem and domain specifications. To this end, we developed two techniques for generating features. The first uses “facts” of the states and the actions leading to those states as features. The second, more sophisticated idea uses a Taxonomic syntax to generate higher level features (McAllester and Givan 1993). We are not aware of any other work that used the STAGE approach in the context of automatically generated features. We implemented both these feature generation techniques and used them to adapt a variant of the STAGE approach to support online learning in solving OSP problems. We compared the performance of our online learning system to a baseline heuristic search approach for solving these planning problems (c.f. (Do et al. 2007)). Our results convincingly demonstrate the promise of our learning approach. Particularly, our online learning system outperforms the baseline system including the learning time, which is typically ignored in prior studies in learning and planning.

The contributions of this research are thus two fold. First, we demonstrate that the performance of heuristic search planners in challenging OSP domains can be improved with the help of online learning techniques. This is interesting both because OSP is a hard planning problem and because there has been little prior work on learning techniques to improve plan quality. Second, we show that it is possible to retain the effectiveness of the STAGE approach without resorting to hand-crafted features.

In the rest of the paper, we will give preliminaries of both OSP and the STAGE approach. We follow with details of how we adapt this to OSP. Of particular interest here are the details of our automated feature generation techniques. Next we present and analyze empirical results comparing the performance of our online learning approach with the baseline heuristic search planner on a set of benchmark domains. We end the paper with a brief description of promising future directions.

Preliminaries

Over-subscription Planning

We define an over-subscription planning (OSP) problem P^o as a tuple of (O, P, Y, I, G, U, C) , where O is a set of constants, P is a set of available predicates and Y is a set of available action schema. A fact is $p \in P$ applied to appropriate set of constants in O . \mathcal{P} is a set of all facts. A state s is a set of facts and I is the initial state. Additionally, we define the set of grounded actions A , where each $a \in A$ is generated from $y \in Y$ applied to appropriate set of constants in O . As is usual in the planning literature, each action $a \in A$ consists of precondition $Pre(a)$ which must be met in the current state before applying a , $Add(a)$ describes the set of

added facts after applying a and $Del(a)$ describes the set of deleted facts after applying a . C is a cost function that maps an action a to a real valued cost, $C : a \rightarrow \mathcal{R}$.

Unlike classical planning problems where we define a set of goal facts that must be achieved, the set of goals G in a problem P^o may remain unsatisfied in the final state. Instead, utility values are defined on subsets of the goals and we have a utility function U , which evaluates the current state, $U : S \rightarrow \mathcal{R}$. More specifically, for a goal set $G \subseteq \mathcal{P}$, U consists of a set of local utility functions $\{u(\cdot)\}$ that assign real number valued utilities to subsets of the goals. In other words, with the problem goal set G , utility is given to goal subsets using local utility functions, $u(G_i) \in \mathcal{R}$, on $G_i \subseteq G$ and utility of a state s with an achieved goal subset $G' \subseteq G$ has $U(s) = \sum_{G_i \subseteq G'} u(G_i)$, following the *general additive independence model* (Bacchus and Grove 1995). Each set G_i represents a *goal dependency*, where having achieved all of the goals in G_i causes changes to the global utility of the final state. Of course, a single fact may exist in a goal dependency set, thereby allowing individual goal facts to be assigned a utility value.

Of many classes of OSP problems, we investigate maximizing “net-benefit” objective problems, one of the most popular form of OSP problems. The objective of OSP is to find the sequence of actions (and states) $A_p = \{I, a_1, s_1 \dots, a_n, s_n\}$ with the best possible *net benefit*, defined as the difference between the summed utility of s_n , $U(s_n)$, and the summed costs of the actions in the plan, $\sum_{a \in A_p} C(a)$.

OSP Planners: There are several planners developed to solve OSP problems (Smith 2004; Sanchez and Kambhampati 2005; Do et al. 2007; Benton, van den Briel, and Kambhampati 2007). Many of these planners solve a relaxed version of the original OSP problem at each search node.² Solving relaxed problems has the disadvantage that their solutions can overestimate the utilities that can be achieved.³ Despite this, forward state space heuristic search planners have shown promise and have done well in solving OSP problems in practice. In this work, our baseline planner SPUDS (Do et al. 2007) is from this class of planners, in that it finds a relaxed plan to compute heuristics for a forward state space search.

Review of STAGE approach

STAGE (Boyan and Moore 2000) aims to learn a policy for intelligently predicting restart points for a base-level random-restart hill-climbing strategy. STAGE works by alternating between two search strategies O-SEARCH and S-SEARCH. O-SEARCH is the base-level local search which aims to hill-climb with some natural objective function O for the underlying problem (e.g., number of bins used in

²Exceptions to this are *AltWlt* (Sanchez and Kambhampati 2005) and the orienteering planner (Smith 2004), both of which select goals up-front then solve for those goals in a classical planning problem.

³It is also possible that an inadmissible heuristic may sometimes underestimate achievable utilities.

the bin-packing problem). The S-SEARCH aims to scout for good restart points for the O-SEARCH.

The O-SEARCH is run first (until, for example, the hill climbing reaches a local minimum). Let $T = s_0, s_1 \dots s_n$ be the trajectory of states visited by the O-SEARCH, and let $o_*(s_i) = \text{best}_{j>i} O(s_j)$ be the objective function value of the best state found on this trajectory after s_i . STAGE now tries to learn a function V to predict that any state s' that is similar to the state s_i on the trajectory T , will lead the hill-climbing strategy to an optima of value $o_*(s_i)$.

In the next phase, S-SEARCH is run using V as the objective function, to find a state s that will provide a good vantage point for restarting the O-SEARCH. S-SEARCH normally starts from s_n , the state at the end of the trajectory of the (previous) O-SEARCH (although theoretically it can start from any random state, including the initial state).⁴

This sequence of O-SEARCH, learning and S-SEARCH are iterated to provide multiple restarts for the O-SEARCH. As we go through additional iterations, the training data for the regression learner increases monotonically. For example, after the O-SEARCH goes through a second trajectory $T_2 : s_0^2 \dots s_n^2$ where the best objective value encountered in the trajectory after state s_j^2 is $o_*^2(s_j)$, in addition to the training data from the first O-SEARCH $s_i \rightarrow o_*(s_i)$, we also have the training data $s_j^2 \rightarrow o_*^2(s_j^2)$. The regression is redone to find a new V function which is then used for driving S-SEARCH in the next iteration.

Boyan and Moore showed that the STAGE approach is effective across a broad class of optimization problems. The critical indicator of STAGE's success turns out to be availability of good state features that can support effective (regression) learning. In all the problems that Boyan and Moore investigated, they provided hand-crafted state features that are customized to the problem. One of the features used for bin-packing problems, for example, is the variance of bin fullness. As we shall see, an important contribution of our work is to show that it is possible to drive STAGE with automatically generated features.

Adapting STAGE to OSP

Automated Feature Generation

One key challenge in adapting the STAGE approach to domain-independent OSP stems from the difficulty in handling the wide variety of feature space between planning domains. While task-dependent features often appear obvious in many optimization problems, domain-independent problem solvers (such as typical planning systems) generally require a different set of features for each domain. Producing such features by hand is impractical and it is undesirable to require users of a planning system to provide such a set. Instead, we use automated methods for feature construction.

⁴In fact, if we can easily find the global optimum of V , that would be the ideal restart point for the O-SEARCH. This is normally impossible because V might be learned with respect to non-linear (hand-selected) features of state. The inverse image of V on the state space forms its own complex optimization problem, thus necessitating a second local search.

In our work, we experimented with two methods for feature generation. One method derives propositional features for each problem from the ground problem facts. The other derives relational features for each domain using a Taxonomic syntax (McAllester and Givan 1993). We describe both of them below. An important difference between Taxonomic and propositional feature sets is that the former remains the same for each domain, while the latter changes from problem to problem even in the same domain. Thus, the number of propositional features grow with the size of problems while Taxonomic features do not.

Propositional Features: In a propositional feature set, each fact in the state represents a feature. Intuitively, if there is some important fact f that contributes to the achievement of some goal or a goal by itself, then states that include the fact should be valued high. In other words, a binary feature that is true with the fact f , should be weighted higher for the target value function. It is then natural to have all the potential state facts or propositions as a feature set. This intuitive idea has been tested in a probabilistic planning system (Buffet and Aberdeen 2007). In their case, the features were used to learn policies rather than value functions. Given constants O and predicates P in an OSP problem P^o , we can enumerate all the ground facts \mathcal{P} . Each ground fact is made into a binary feature, with the value of the feature being *true* when the fact is in the current state. We call the planning and learning system that uses these binary features a ‘‘Propositional’’ system.

Relational Features: Although the propositional feature set is intuitive and a simple method to implement, it cannot represent more sophisticated properties of the domain, where relations between state facts is important, e.g., conjunction or disjunction of the facts.

Our second approach involves relational (object-oriented) features. For many of the planning domains, it is natural to reason with objects in the domain. Particularly, it is reasonable to express the value of a state in terms of objects. For example, in Logisticsworld, the distance to the goal can be well represented with ‘‘number of packages not delivered’’. Here, the ‘‘packages that are not delivered yet’’ are a good set of objects that indicates the distance to the goal. If we can provide a means to represent a set of objects with such a property, then the cardinality of the set could be a good feature for the value function to learn with.

Taxonomic syntax (McAllester and Givan 1993) provides a convenient framework for these expressions. In what follows, we review Taxonomic syntax and we define our feature space with Taxonomic syntax.

Taxonomic Syntax: A relational database R is a collection of ground predicates, where ground predicates are applications of predicates $p \in P$ to the corresponding set of objects ($o \in O$). Each state in a planning problem is a good example for a relational database. We prepend a special symbol \mathbf{g} if the predicate is from the goal description and \mathbf{c} if the predicate is both true in the current state and the goal state. \mathbf{c} predicates are a syntactic convenience to express means-ends analysis (Newell and Simon 1972). Note that goal information is also part of state information with the \mathbf{g} predicates. An example relational database (a state from a

(at truck1 location1), (at package1 location1), (in-city location1 city1), (in-city location2 city1) (gat package1 location1) (cat package1 location1) (at package2 location2) (gat package2 location1)

Figure 1: Example Relational Database: A State from Logisticsworld

Logisticsworld domain) is shown in Figure 1. In this example, there are two packages *package1* and *package2*. Only *package1* is at the goal location. So there is an additional fact, (cat package1 location1). It turned out that this simple notation makes the expression compact and useful for representing means-ends analysis (Newell and Simon 1972).

Taxonomic syntax C is defined as follows,

$$C = \mathbf{a\text{-}thing}|(p \ C_1 \ \dots \ ? \ \dots \ C_{n(p)})|C_1 \cap C_2|\neg C$$

It consists of **a-thing**, predicates with one position in the argument is left for the output of the syntax, while other positions are filled with other class expressions, intersection of class expressions and negation of a class expression. $n(p)$ is the arity of the predicate p . We define depth $d(C)$ for enumeration purpose. **a-thing** has depth 0 and class expression with one argument predicate has depth 1.

$$d((p \ C_1 \ \dots \ ? \ \dots \ C_{n(p)})) = \max d(C_i) + 1$$

Taxonomic Syntax Semantics: Taxonomic syntax $C[R]$ against a relational database R describes a set of objects. **a-thing** describes all the objects in R . In the example in Figure 1, they are (*city1*, *truck1*, *package1*, *package2*, *location1*, *location2*). $(p \ C_1 \ \dots \ ? \ \dots \ C_{n(p)})$ describes a set of objects O that make the predicate p true in R when O is placed in the $?$ position while other positions are filled with the objects that belong to the corresponding class expression. For example, consider $C = (\mathbf{cat} \ ? \ \mathbf{a\text{-}thing})$ and let R be the relational database in Figure 1. $C[R]$ is then (*package1*). Among all the objects, only *package1* can fill in the $?$ position and make the (*cat package1 location1*) predicate true. Note that **a-thing** allows any object, including *location1*. In this example, C indicates all the objects that fill in the first argument position of **cat** and make the predicate true in the Logisticsworld, referring to all the objects that are already in the goal. As another example, consider $C' = (\mathbf{at} \ ? \ \mathbf{a\text{-}thing})$. $C'[R]$ is then (*package1*, *truck1*, *package2*).

Feature Generation Function for Over-subscription Planning: We enumerate limited depth class expressions from the domain definition. **a-thing** is included in the feature set by default. Recall the planning domain definition, $P^o = (O, P, Y, I, G, U, C)$. Using P , the set of predicates, we can enumerate Taxonomic features. First, for all the predicates, except one argument position, we fill all the other argument positions with **a-thing**. This set constitutes the depth 1 Taxonomic features. For the Logisticsworld, C and C' in the

above corresponds to this set of depth 1 features. Depth n features can then be easily enumerated by allowing depth $n - 1$ Taxonomic syntax in other argument positions than the output position. For example, ($\mathbf{at} \ \neg(\mathbf{cat} \ ? \ \mathbf{a\text{-}thing}) \ ?$) is a depth 2 feature, which is constructed by using depth 1 Taxonomic feature at the first argument position. The meaning of this feature is “the location where a package is not yet in the goal location”. Enumerating the features at high depths proved computationally expensive during our testing. For this reason, we used depth 2 for our experimental tests. We call the planning and learning system that uses the class expression feature set a “Taxonomic” system. The value of the Taxonomic features is the cardinality of the Taxonomic expressions, which gives out sets of objects. This makes the features appropriate for value function learning.

In both the “Propositional” and “Taxonomic” feature sets, we also use actions involved as part of the features. Recall that each state in OSP involves actions that led the initial state to the current state. For the “Taxonomic” feature set, we union these actions with state facts for the relational database construction. The semantics of this database straightforwardly follow from Taxonomic syntax. For the “Propositional” feature set, we also enumerate all the potential ground actions and assign a binary value 1 if they appear in the actions that led to the state.

Doing O-SEARCH with SPUDS

The STAGE approach was initially developed to work in a local (aka iterative) search model where the overall merit of the current state is completely specified in terms of the state itself. While it is possible to do such local search directly in the plan space (e.g. the LPG system (Gerevini, Saetti, and Serina 2003)), the majority of current heuristic planners, including those for OSP, search in the space of world states, with the plan (the solution) being the path traversed (actions taken) between states. This difference poses several technical difficulties in directly adapting the learning approach given by STAGE for our needs.

Specifically, the baseline planner we use, SPUDS, engages in an A^* search to traverse the space of world states, with each state s evaluated by $f(s) = g(s) + h(s)$, where $g(s)$ is the net-benefit of the plan realized in traversing from the initial state to s , and $h(s)$ is an estimate of additional net-benefit that is expected to be accrued by expanding s further (Do et al. 2007). Viewing SPUDS as doing the O-SEARCH phase of STAGE, the g value corresponds to the objective function, while the h value is used in conjunction with the g value to explore a potentially more promising part of the search space.

Unlike local search which explores a single sequential trajectory, the O-SEARCH done by SPUDS essentially explores (expands) a tree of states (see Figure 2). This is because, SPUDS’s search is memory based, maintaining a priority queue of states to explore while stochastic local search in optimization does memory-less search, where no priority queue is involved. Thus we need to re-define the optima reachable for each expanded state s . We define it as the best g value of any node in the subtree rooted at s in the current search tree (as shown in the figure). Note that only the g

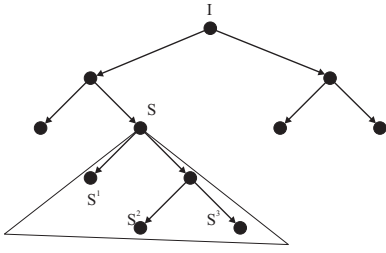


Figure 2: SPUDS Search Tree. Unlike memoryless stochastic optimization search, SPUDS maintains the search queue. Thus the best objective value that can be found after a state S , is the best value in the subtree of S . In this tree, the best expected value of S is $\max(S^1, S^2, S^3)$.

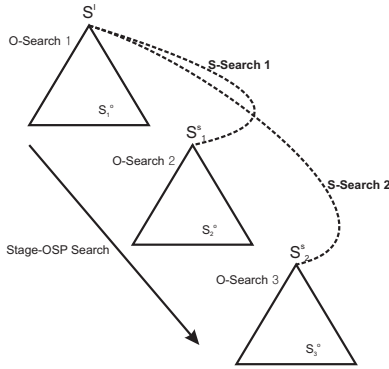


Figure 3: Stage-OSP search behavior. The triangle in the figure signifies O-SEARCH and the dotted lines signifies S-SEARCH. As Stage-OSP progresses, it restarts O-SEARCH after each end point of S-SEARCH ($S_1^s, S_2^s \dots$). Each S-SEARCH phase itself always starts at the initial state, S^I .

value is used here as it corresponds to the best actual plan encountered under s .

For example, in our baseline planner, SPUDS (Do et al. 2007), we define a policy π_{SPUDS} . As shown in Figure 2, the projected value from a state s is $\max_{s_j \in \{s, s^1, s^2, s^3\}} U(s_j) - C(s_j)$. $U(\cdot)$ (the goal utility achieved in state s_j) and $C(\cdot)$ is the cost of the actions that were used to reach s_j from I . Let $\hat{V}_n^{\pi_{SPUDS}}(s)$ be the estimated value that can be found from state s by following the heuristic search where the number of node expansions is limited by n . Given $\mathcal{S} = \{s_1, \dots, s_n\}$, we find our target value $\hat{V}_n^{\pi_{SPUDS}}(s_i)$ by $\max_{s_j \in \text{subtree of } s} U(s_j) - C(s_j)$.

Finally, to make-up for the possibility that the O-SEARCH trajectory may have been wrongly biased by the heuristic h , we initiate the S-SEARCH starting back at the initial state (rather than a state at the fringe of the expanded tree). Thus, each time the S-SEARCH will find another new start state for O-SEARCH from the initial state. Additionally, to make the S-SEARCH more ergodic, we allow it to backtrack a single step by adding an operator that rolls back the most recent action taken. We found that S-SEARCH without this backtrack operator just adds actions and could reach very local minimum state.

```

Stage-OSP ( $P^o, n$ )
  //  $P^o$  Over-subscription problem
  //  $n$  limit of evaluated states for O and S-Search

   $F \leftarrow$  Enumerate-Feature-Set( $P^o$ )
   $s_t \leftarrow I$ 
  while ( states-enumerated < MAXEVAL )
     $S = \{s_t \dots s_n\} \leftarrow$ 
      SPUDS-search( $P, s_t, n$ )
     $\mathcal{T} = \{(s_t, v_t), \dots, (s_n, v_n)\} \leftarrow$ 
      value-estimation( $S$ )
     $W \leftarrow$  linear-regression-fit( $\mathcal{T}, P, F$ )
     $\{s_t \dots s_k\} \leftarrow$ 
      stochastic-hillclimbing ( $F, W, I, n$ )
     $s_t \leftarrow s_k$ 

  Return the-best-state-found

```

Figure 4: Stage-OSP Algorithm. It repeats SPUDS search (O-SEARCH), value estimation, linear regression fit (both for learning) and stochastic hillclimbing (S-SEARCH). Also refer Figure 3 for graphical explanation.

Overall Approach

Figure 4 summarizes our algorithm, Stage-OSP. It starts with the search provided by our baseline planner, SPUDS, where the number of evaluated states is limited by n .⁵ Afterwards, we learn a value estimate for the next search stage. Recall that the feature set is not provided as in STAGE, but is enumerated automatically by one of two different methods as described earlier. After a predefined number of node expansions n , the search returns a search tree. The procedure **value-estimation** returns training data to perform the **linear regression fit**, which maps the best values found in the subtree under the state. The learning is then finding weight vectors W for enumerated features F . Then, Stage-OSP enters S-SEARCH which uses the newly learned value function (inner product of the W and $F(\cdot)$) and restarts from the initial state I (note that I is passed into the search as against S). S-SEARCH, like that described in the STAGE approach, is memory-less and uses stochastic hill-climbing.

Figure 3 shows the over all search behavior of the Stage-OSP algorithm. Each of the triangles correspond to O-SEARCH stages, and the dotted lines correspond to S-SEARCH stages. After each O-SEARCH, we train value functions, and do an S-SEARCH to find the next restart point. The next O-SEARCH starts from the restart point identified by the previous S-SEARCH.

Experimental Results

To test our system, we conducted experiments on the Rovers, Satellite and Zenotravel benchmark domains in the context of OSP. These domains are from International Planning Competition and modified to add net-benefit features, action

⁵We can use different n for O-SEARCH and S-SEARCH. We are currently investigating this issue.

costs and goal-interaction utilities. The machine used for these tests was a 2.8 Ghz Xeon processor using a Linux operating system. For our training data, we used $n = 1000$ evaluated states and set the timeout for each problem to 30 minutes of CPU time (including learning time for State-OSP case).⁶ The planner is written in Java. Note that the learning time was not significant as will be revealed soon, as the number of automated features generated was typically less than 10,000. This effectively enables our system do on-line learning and compete against domain-independent planning systems head to head.

For the linear regression fit, we used two different libraries for our different automated feature types. The statistical package R (R-Project) was used for the Taxonomic features, but operated more slowly when learning with the binary propositional features. The Java Weka library worked better on this set, and we therefore used it when handling features of this type. For our evaluation, we address the performance of the Stage-OSP system in each domain on the baseline planner (Do et al. 2007), Stage-OSP with the Taxonomic features, and Stage-OSP with the Propositional features.

For the case of learning with “Taxonomic” features, we also used a simple wrapper method. That is, we greedily add one feature at a time until there is convergence in the approximation measure. For this purpose, we used the R-square metric, which measures the explanation for the variances. This is rather a practical algorithm design choice, since R cannot handle too many features.

Rovers Domain: The Rovers domain was developed to mimic a Mars rover vehicle problem. It models Rovers sent to a planet to perform various tasks (such as taking photos and soil samples). Vehicles can travel between pre-defined points on the planet with varying costs for each traversal. At certain locations, the rover can communicate data to earth collected. However, because of limitations on the costs for traversal, a plan needs to maximize collective *net benefit* for the mission (i.e., a plan needs to achieve high utility goals with minimum cost).

We ran on 20 problems for each domain. Figure 7 shows the results. In the graph shown, the Y axis is the best net benefit found for each problem, thus higher values are better (and we want to maximize our net benefit). The Rovers domain yielded the best results of the 3 domains we tested. Except for on a few problem instances, both feature types, the Taxonomic and Propositional outperformed SPUDS. The cumulative net benefit across the problems in each domain is available in Figure 5. The number of features enumerated is also available in the figure. In Figure 5, for Rovers domain, we can see that both of the learning systems, Propositional and Taxonomic, outperform the baseline planner, achieving twice the cumulative net benefit of SPUDS. This shows the benefit of the learning involved. Note that, in our experiments, there was no prior training. That is, in most of the recent machine learning systems for planning, they used prior

⁶We have tried alternative training data sets, by changing the “n” parameter variously between 500 to 2000, but the results were more or less the same.

```
(navigate athing (gcommunicated-soil-data ?) ?)
(take-image ? (have-rock-analysis athing ?)
athing athing athing)
```

Figure 6: Taxonomic features found for Rover domain

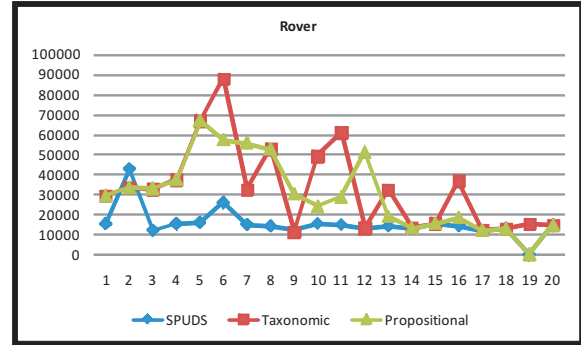


Figure 7: Results on Rovers domain: The X-axis is for the problem numbers. There were 20 problems. The Y-axis shows net-benefit obtained by each system. As can be seen in the figure, Taxonomic system significantly outperformed SPUDS for most of the problems.

training data to tune the machine learner, while our systems learn online fashion. This enables our system to compete in the domain independent competition realm.

For further analysis, it is helpful to see how the the net benefit of the best incumbent plan varies as the search progresses. Note that the baseline planner, SPUDS, performs an incrementally improving “anytime” search. Figure 8 shows how the net benefit changes over time for SPUDS as well as the Stage-OSP method using the two feature sets (Taxonomic and Propositional). The effectiveness of the restarts provided by the S-SEARCH can be seen by observing the staircase pattern of the net benefit improvement in Stage-OSP. In the instances where SPUDS outperformed Stage-OSP, the learned value function could not reveal a better projected state and led the search into a local minimum of the evaluation function, while SPUDS was able to improve the plans through a systematic search.

Finally, Figure 6 lists some of the selected features by the wrapper method with the Taxonomic system. The first listed feature indicates the number of locations traveled where soil data is to be communicated is located. The second provides the number of “take image” actions with rock-analysis in hand. As can be seen in these expressions, the Taxonomic syntax can express more relationally expressive notions than ground facts and we suspect that is the reason why the Taxonomic system outperformed Propositional system. Note also that these features make sense: Moving to a location where soil data will likely move us to improved net benefit. Additionally, taking a goal image while already having finished analysis also moves us toward a goal (and therefore higher net benefit).

Domain	Measure	SPUDS	Stage-OSP (Propositional)	Stage-OSP (Taxonomic)
Rover	Net Benefit	3.0×10^5	6.0×10^5	6.5×10^5
	No. Features		14336	2874
Satellite	Net Benefit	0.89×10^6	0.92×10^6	1.06×10^6
	No. Features		6161	466
Zenotravel	Net Benefit	4.3×10^5	4.1×10^5	4.5×10^5
	No. Features		22595	971

Figure 5: Summary of the cumulative Net Benefit across the problems and number features for each domain

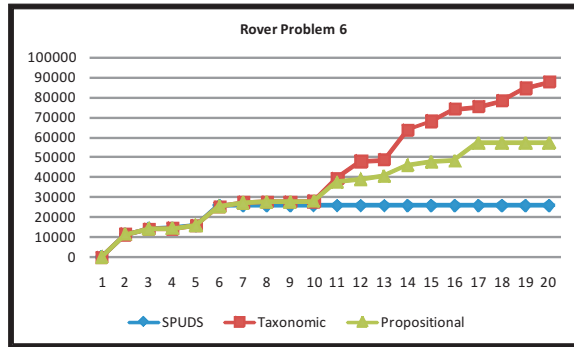


Figure 8: Anytime behavior of systems on Rover problem 6. The X axis shows the CPU time used in minutes.

(cpointing ? athing)
 (turn-to (cpointing ? athing) athing ?)

Figure 9: Taxonomic features found for Satellite domain

Satellite Domain: In the Satellite domain, the objective is to take as many pictures or measurements as possible without consuming too much energy. To perform an operation, a Satellite needs to turn to the right direction, calibrate its instruments and finally take a photo or perform a measurement. Figure 10 shows the results on Satellite domain. The performance of Stage-OSP using either of the feature sets does not dominate as strongly as seen in the Rovers domain. However, Stage-OSP still outperformed the baseline planner in cumulative net benefit measure on the problems, as can be verified through Figure 5.

Figure 9 lists the features of Taxonomic system found by the wrapper method. The first one feature expresses correctly-pointing facts (note that *c*-predicates were used) and the second one expresses the number of actions that turn to the correctly pointing areas, these features help with finding end-state “pointing” goals.

Zenotravel Domain: Figure 12 shows the results of Zenotravel domain. Zenotravel’s objective is getting as many people to their goal locations as possible. The plane involved has fuel cost associated with it and if there is no fuel left, the

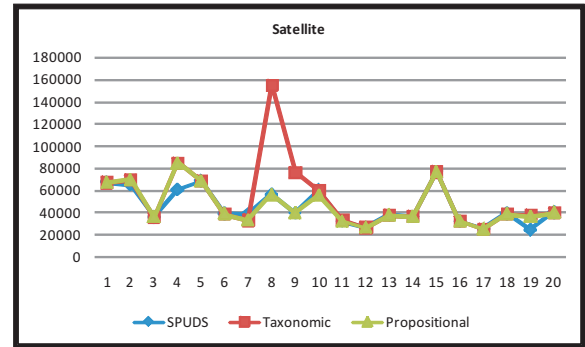


Figure 10: Results on Satellite domain

(fuel-level ? (fly athing athing athing athing ?))
 (gat ? (zoom athing athing ? athing athing athing))

Figure 11: Taxonomic features found for Zenotravel domain

plane needs to refuel. Thus, reducing refueling actions while serving as many people as possible is the key objective to achieve. The learners did not fare as well in this domain. As can be seen in Figure 12, the learning systems lost to SPUDS on the same number of problems as the number of problems they won. The cumulative net benefit across problems are shown in Figure 5. The numbers show a slight edge using the Taxonomic features. The margin is much smaller than the other domains.

Figure 11 shows the features found in the Taxonomic system. The first feature listed expresses the number of refuel actions taken (and is thus negatively weighted) and the second expresses the number of zooming actions taken to the goal location.

When the learning system fared well, for example, in the Rovers domain, we found that the learned value function led the S-SEARCH to a quite deeper state *s*, that requires many actions to reach from the initial state but achieves the key goal facts.

Although we provided the action features to take the action cost structure into account, the learned value function is not too sensitive to the actions used. One possible reason for this may be that the Taxonomic syntax uses set semantics rather than bag semantics. That is, when the partial plan

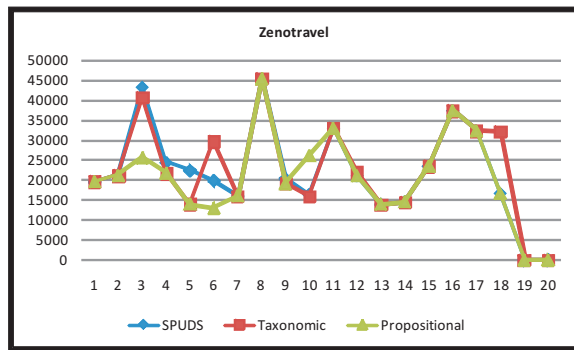


Figure 12: Results on Zenotravel domain

corresponding to a search node contains multiple instances of the same action, they are counted only once. We suspect that this is why the learning systems did not fare well in the Zenotravel domain, where the cost structure of the action space needs careful consideration of the number of actions involved. We plan to improve the semantics of Taxonomic syntax to account for bag semantics.

Related Works

There has been very little prior work focused on learning to improve plan quality. The closest learning system for planning that tried to improve the quality of plans produced was the work by Perez (Pérez 1996) almost a decade back. In contrast to our approach, theirs used explanation-based learning techniques to learn search control rules. As we discussed, one reason Stage-OSP outperforms SPUDS is that the S-SEARCH with learned evaluation function allows it to go to deeper parts of the search tree (and probe those regions with SPUDS search). Prior work, such as YAHSP (Vidal 2004) and BBOP-LP (Benton, van den Briel, and Kambhampati 2007), achieve a similar effect by using the relaxed plan as a macro-operator (and thus access deeper parts of the search space). One important difference is that while S-SEARCH uses an evaluation function that is learned online from the current problem episode (and is thus potentially more adaptive to the problem), YAHSP and BBOP-LP were guided by relaxed plan, whose effectiveness could vary with the domains, depending on the importance of deleted facts.

Conclusion

Motivated by the success of the STAGE approach in learning to improve search in optimization problems, we adapted it to OSP. The critical challenge in the adaptation was the need to provide automated features for the learning phase of STAGE. We experimented with two automated feature generation methods. One of them—the Taxonomic feature set—is especially well suited to planning problems because of its object-oriented nature. Our experiments with Stage-OSP show that it is able to provide significant improvements over SPUDS—a state of the art heuristic planning approach for solving OSP problems.

One immediate task is developing a feature space that provides a bag rather than set semantics for the actions taken.

As we have argued, this seems to be important for domains like Zenotravel. Another direction we plan to explore in the near future is to investigate the utility of features derived from relaxed plans. Recent work in learning to improve classical planners (Yoon, Fern, and Givan 2006) shows that such features are quite effective.

Acknowledgements

This work is supported in part by the DARPA Integrated Learning Program (through a sub-contract from Lockheed Martin), and by ONR grants N000140610058 and N00014-07-1-1049 (MURI sub-contract from Indiana University).

References

- Bacchus, F., and Grove, A. 1995. Graphical model for preference and utility. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, 3–10.
- Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *Proceedings of ICAPS*.
- Boyan, J., and Moore, A. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1:77–112.
- Buffet, O., and Aberdeen, D. 2007. FF+FPG: Guiding a policy-gradient planner. In *Proceedings of International Conference on Automated Planning and Scheduling*.
- Do, M.; Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. Planning with goal utility dependencies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1872–1878.
- Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in Ipg. *J. Artif. Intell. Res. (JAIR)* 20:239–290.
- McAllester, D., and Givan, R. 1993. Taxonomic syntax for first order inference. *JACM* 40(2):246–283.
- Newell, A., and Simon, H. A. 1972. *Human problem solving*. Prentice-Hall.
- Pérez, M. 1996. Representing and learning quality-improving search control knowledge. In *ICML*.
- R-Project. *The R Project for Statistical Computing*. www.r-project.org.
- Sanchez, R., and Kambhampati, S. 2005. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 192–201.
- Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *ICAPS*, 393–401.
- 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. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*.
- Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Mag.* 24(2):73–96.