Journal of Artificial Intelligence Research 38 (2010) 687-755 Submitted 01/10; published 08/10

Automatic Induction of Bellman-Error Features for Probabilistic Planning

Jia-Hong Wu JW@ALUMNI.PURDUE.EDU
Robert Givan GIVAN@PURDUE.EDU
Electrical and Computer Engineering
Purdue University, W. Lafayette, IN 47907 USA

Abstract

Domain-specific features are important in representing problem structure throughout machine learning and decision-theoretic planning. In planning, once state features are provided, domain-independent algorithms such as approximate value iteration can learn weighted combinations of those features that often perform well as heuristic estimates of state value (e.g., distance to the goal). Successful applications in real-world domains often require features crafted by human experts. Here, we propose automatic processes for learning useful domain-specific feature sets with little or no human intervention. Our methods select and add features that describe state-space regions of high inconsistency in the Bellman equation (statewise Bellman error) during approximate value iteration. Our method can be applied using any real-valued-feature hypothesis space and corresponding learning method for selecting features from training sets of state-value pairs. We evaluate the method with hypothesis spaces defined by both relational and propositional feature languages, using nine probabilistic planning domains. We show that approximate value iteration using a relational feature space performs at the state-of-the-art in domain-independent stochastic relational planning. Our method provides the first domain-independent approach that plays Tetris successfully (without human-engineered features).

1. Introduction

There is a substantial gap in performance between domain-independent planners and domain-specific planners. Domain-specific human input is able to produce very effective planners in all competition planning domains as well as many game applications such as backgammon, chess, and Tetris. In deterministic planning, work on TLPLAN (Bacchus & Kabanza, 2000) has shown that simple depth-first search with domain-specific human input, in the form of temporal logic formulas describing acceptable paths, yields an effective planner for a wide variety of competition domains. In stochastic planning, feature-based value-function representations have been used with human-selected features with great success in applications such as backgammon (Sutton & Barto, 1998; Tesauro, 1995) and Tetris (Bertsekas & Tsitsiklis, 1996). The usage of features provided by human experts is often critical to the success of systems using such value-function approximations. Here, we consider the problem of automating the transition from domain-independent planning to domain-specific performance, replacing the human input with automatically learned domain properties. We thus study a style of planner that learns from encountering problem instances to improve performance on subsequently encountered problem instances from the same domain.

We focus on stochastic planning using machine-learned value functions represented as linear combinations of state-space features. Our goal then is to augment the state-space representation during planning with new machine-discovered features that facilitate accurate representation of the value function. The resulting learned features can be used in representing the value function for other problem instances from the same domain, allowing amortization of the learning costs across solution of multiple problem instances. Note that this property is in contrast to most competition planners, especially in deterministic planning, which retain no useful information between problem instances. Thus, our approach to solving planning problems can be regarded as automatically constructing domain-specific planners, using domain-independent techniques.

We learn features that correlate well to the statewise Bellman error of value functions encountered during planning, using any provided feature language with a corresponding learner to select features from the space. We evaluate this approach using both relational and propositional feature spaces. There are other recent approaches to acquiring features in stochastic planning with substantial differences from our approach which we discuss in detail in Section 5 (Patrascu, Poupart, Schuurmans, Boutilier, & Guestrin, 2002; Gretton & Thiébaux, 2004; Sanner & Boutilier, 2009; Keller, Mannor, & Precup, 2006; Parr, Painter-Wakefield, Li, & Littman, 2007). No previous work has evaluated the selection of relational features by correlation to statewise Bellman error.

Recent theoretical results (Parr et al., 2007) for uncontrolled Markov processes show that exactly capturing statewise Bellman error in new features, repeatedly, will lead to convergence to the uncontrolled optimal value for the value function selected by linear-fixed-point methods for weight training. Unfortunately for machine-learning approaches to selecting features, these results have not been transferred to approximations of statewise Bellman-error features: for this case, the results in the work of Parr et al. (2007) are weaker and do not imply convergence. Also, none of this theory has been transferred to the controlled case of interest here, where the analysis is much more difficult because the effective (greedy) policy under consideration during value-function training is changing.

We consider the controlled case, where no known theoretical properties similar to those of Parr et al. (2007) have been shown. Lacking such theory, our purpose is to demonstrate the capability of statewise Bellman error features empirically, and with rich representations that require machine learning techniques that lack approximation guarantees. Next, we give an overview of our approach, introducing Markov decision processes, value functions, Bellman error, feature hypothesis languages and our feature learning methods.

We use Markov decision processes (MDPs) to model stochastic planning problems. An MDP is a formal model of a single agent facing a sequence of action choices from a pre-defined action space, and transitioning within a pre-defined state space. We assume there is an underlying stationary stochastic transition model for each available action from which state transitions occur according to the agent's action choices. The agent receives reward after each action choice according to the state visited (and possibly the action chosen), and has the objective of accumulating as much reward as possible (possibly favoring reward received sooner, using discounting, or averaging over time, or requiring that the reward be received by a finite horizon).

MDP solutions can be represented as state-value functions assigning real numbers to states. Informally, in MDP solution techniques, we desire a value function that respects the action transitions in that "good" states will either have large immediate rewards or have actions available that lead to other "good" states; this well-known property is formalized in *Bellman equations* that recursively characterize the optimal value function (see Section 2). The degree to which a given value function fails to respect action transitions in this way, to be formalized in the next section, is referred to as the *Bellman error* of that value function, and can be computed at each state.

Intuitively, statewise Bellman error has high magnitude in regions of the state space which appear to be undervalued (or overvalued) relative to the action choices available. A state with high Bellman error has a locally inconsistent value function; for example, a state is inconsistently labeled with a low value if it has an action available that leads only to high-value states. Our approach is to use machine learning to fit new features to such regions of local inconsistency in the current value function. If the fit is perfect, the new features guarantee we can represent the "Bellman update" of the current value function. Repeated Bellman updates, called "value iteration", are known to converge to the optimal value function. We add the learned features to our representation and then train an improved value function, adding the new features to the available feature set.

Our method for learning new features and using them to approximate the value function here can be regarded as a *boosting-style* learning approach. A linear combination of features can be viewed as a weighted combination of an ensemble of simple hypotheses. Each new feature learned can be viewed as a simple hypothesis selected to match a training distribution focused on regions that the previous ensemble is getting wrong (as reflected in high statewise Bellman error throughout the region). Growth of an ensemble by sequentially adding simple hypotheses selected to correct the error of the ensemble so far is what we refer to as "boosting style" learning.

It is important to note that our method scores candidate features by correlation to the statewise Bellman error of the current value function, *not* by minimizing the statewise Bellman error of some value function found using the new candidate feature. This *pre-feature-addition* scoring is much less expensive than scoring that involves retraining weights with the new feature, especially when being repeated many times for different candidates, relative to the same current value function. Our use of *pre-feature-addition* scoring to select features for the controlled setting enables a much more aggressive search for new features than the previously evaluated post-feature-addition approach discussed in the work of Patrascu et al. (2002).

Our approach can be considered for selecting features in any feature-description language for which a learning method exists to effectively select features that match state-value training data. We consider two very different feature languages in our empirical evaluation. Human-constructed features are typically compactly described using a relational language (such as English) wherein the feature value is determined by the relations between objects in the domain. Likewise, we consider a relational feature language, based on domain predicates from the basic domain description. (The domain description may be written, for example, in a standard planning language such as PPDDL in Younes, Littman, Weissman, & Asmuth, 2005.) Here, we take logical formulas of one free variable to represent features that count the number of true instantiations of the formula in the state being evaluated. For example, the "number of holes" feature that is used in many Tetris experiments (Bertsekas & Tsitsiklis, 1996; Driessens, Ramon, & Gärtner, 2006) can be interpreted as counting the number of empty squares on the board that have some other filled squares above them. Such numeric features provide a mapping from states to natural numbers.

In addition to this relational feature language, we consider using a propositional feature representation in our learning structure. Although a propositional representation is less expressive than a relational one, there exist very effective off-the-shelf learning packages that utilize propositional representations. Indeed, we show that we can reformulate our feature learning task as a related classification problem, and use a standard classification tool, the decision-tree learner C4.5 (Quinlan, 1993), to create binary-valued features. Our reformulation to classification considers only the sign, not the magnitude, of the statewise Bellman error, attempting to learn features that characterize the positive-sign regions of the state space (or likewise the negative-sign regions). A standard supervised classification problem is thus formulated and C4.5 is then applied to generate a decision-tree feature, which we use as a new feature in our value-function representation. This propositional approach is easier to implement and may be more attractive than the relational one when there is no obvious advantage in using relational representation, or when computing the exact statewise Bellman error for each state is significantly more expensive than estimating its sign. In our experiments, however, we find that our relational approach produces superior results than our propositional learner. The relational approach also demonstrates the ability to generalize features between problem sizes in the same domain, an asset unavailable in propositional representations.

We present experiments in nine domains. Each experiment starts with a single, constant feature, mapping all states to the same number, forcing also a constant value function that makes no distinctions between states. We then learn domain-specific features and weights from automatically generated sampled state trajectories, adjusting the weights after each new feature is added. We evaluate the performance of policies that select their actions greedily relative to the learned value functions. We evaluate our learners using the stochastic computer-game Tetris and seven planning domains from the two international probabilistic planning competitions (Younes et al., 2005; Bonet & Givan, 2006). Our method provides the first domain-independent approach to playing Tetris successfully (without human-engineered features). Our relational learner also demonstrates superior success ratio in the probabilistic planning-competition domains as compared both to our propositional approach and to the probabilistic planners FF-Replan (Yoon, Fern, & Givan, 2007) and FOALP (Sanner & Boutilier, 2006, 2009). Additionally, we show that our propositional learner outperforms the work of Patrascu et al. (2002) on the same SysAdmin domain evaluated there.

2. Background

Here we present relevant background on the use of Markov Decision Processes in planning.

2.1 Markov Decision Processes

We define here our terminology for Markov decision processes. For a more thorough discussion of Markov decision processes, see the books by Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998). A Markov decision process (MDP) M is a tuple (S, A, R, T, s_0). Here, S is a finite state space containing initial state s_0, and A selects a non-empty finite available action set $A(s)$ for each state s in S. The reward function R assigns a real reward to each state-action-state triple (s, a, s') where action a is enabled in state s, i.e., a is in $A(s)$. The transition probability function T maps state-action pairs (s, a) to probability distributions over S, $\mathcal{P}(S)$, where a is in $A(s)$.

Given discount factor $0 \leq \gamma < 1$ and *policy* π mapping each state $s \in S$ to an action in $A(s)$, the value function $V^{\pi}(s)$ gives the expected discounted reward obtained from state s selecting action $\pi(s)$ at each state encountered and discounting future rewards by a factor of γ per time step. There is at least one optimal policy π^* for which $V^{\pi^*}(s)$, abbreviated $V^*(s)$, is no less than $V^{\pi}(s)$ at every state s, for any policy π. The following "Q function" evaluates an action a with respect to a future-value function V,

$$Q(s, a, V) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')].$$

Recursive Bellman equations use $Q()$ to describe V^* and V^{π} as follows. First, $V^{\pi}(s) = Q(s, \pi(s), V^{\pi})$. Then, $V^*(s) = \max_{a \in A(s)} Q(s, a, V^*)$. Also using $Q()$, we can select an action greedily relative to any value function. The policy Greedy(V) selects, at any state s, the action $\arg\max_{a \in A(s)} Q(s, a, V)$.

Value iteration iterates the operation

$$\mathcal{U}(V)(s) = \max_{a \in A(s)} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')],$$

computing the "Bellman update" $\mathcal{U}(V)$ from V, producing a sequence of value functions converging in the sup-norm to V^*, regardless of the initial V used.

We define the *statewise Bellman error* $B(V, s)$ for a value function V at a state s to be $\mathcal{U}(V)(s) - V(s)$. We will be inducing new features based on their correlation to the statewise Bellman error, or based on the sign of the statewise Bellman error. The sup-norm distance of a value function V from the optimal value function V^* can be bounded using the Bellman error magnitude, which is defined as $\max_{s \in S} |B(V, s)|$ (e.g., see Williams & Baird, 1993). We use the term "statewise Bellman error" to emphasize the distinction from the widely used "sup-norm Bellman error".

We note that computing $\mathcal{U}(V)$, and thus statewise Bellman error, can involve a summation over the entire state space, whereas our fundamental motivations require avoiding such summations. In many MDP problems of interest, the transition matrix T is sparse in a way that set of states reachable in one step with non-zero probability is small, for any current state. In such problems, statewise Bellman error can be computed effectively using an appropriate representation of T. More generally, when T is not sparse in this manner, the sum can be effectively approximately evaluated by sampling next states according to the distribution represented by T.

2.2 Modeling Goal-oriented Problems

Stochastic planning problems can be goal-oriented, where the objective of solving the problem is to guide the agent toward a designated state region (i.e., the goal region). We model such problems by structuring the reward and transition functions R and T so that any action in a goal state leads with positive reward to a zero-reward absorbing state, and reward is zero everywhere else. We retain discounting to represent our preference for shorter paths to the goal. Alternatively, such problems can be modeled as stochastic shortest path MDPs without discounting (Bertsekas, 1995). Our techniques can easily be generalized to formalisms which allow varying action costs as well, but we do not model such variation in this work.

More formally, we define a goal-oriented MDP to be any MDP meeting the following constraints. Here, we use the variables s and s' for states in S and a for actions in $A(s)$. We require that S contain a zero-reward absorbing state $\perp$, i.e., such that $R(\perp, a, s) = 0$ and $T(\perp, a, \perp) = 1$ for all s and a. The transition function T must assign either one or zero to triples $(s, a, \perp)$, and we call the region of states s for which $T(s, a, \perp)$ is one *the goal region*. The reward function is constrained so that $R(s, a, s')$ is zero unless $s' = \perp$. In constructing goal-oriented MDPs from other problem representations, we may introduce dummy actions to carry out the transitions involving $\perp$ described here.

2.3 Compactly Represented MDPs

In this work, we consider both propositional and relational state representations.

In relational MDPs, the spaces S and $A(s)$ for each s are relationally represented, i.e., there is a finite set of objects O, state predicates P, and action names N used to define these spaces as follows. A *state fact* is an application $p(o_1, \ldots, o_n)$ of an n-argument state predicate p to object arguments o_i, for any n[1]. A state is any set of state facts, representing exactly the true facts in that state. An *action instance* $a(o_1, \ldots, o_n)$ is an application of an n-argument action name to n objects o_i, for any n. The action space $A = \bigcup_{s \in S} A(s)$ is the set of all action instances.

MDPs with compactly represented state and action spaces also use compact representations for the transition and reward functions. One such compact representation is the PPDDL planning language, informally discussed in the next subsection and formally presented in the work of Younes et al. (2005).

In propositional problems, the action space is explicitly specified and the state space is compactly specified by providing a finite sequence of basic state properties called *state attributes*, with Boolean, integer, or real values. A propositional state is then any vector of values for the state attributes.

Given a relational MDP, an equivalent propositional MDP can be easily constructed by "grounding," in which an explicit action space is constructed by forming all action-name applications and a set of state attributes is computed by forming all state-predicate applications, thus removing the use of the set of objects in the representation.

2.4 Representing PPDDL Planning Problems using MDPs

We discuss how to represent goal-oriented stochastic planning problems defined in standardized planning languages such as PPDDL (Younes et al., 2005) as goal-oriented MDPs. We limit our focus to problems in which the goal regions can be described as (conjunctive) sets of state facts. We reference and follow the approach used in the work of Fern, Yoon, and Givan (2006) here regarding converting from planning problems to compactly represented MDPs in a manner that facilitates generalization between problem instances. We first discuss several difficult representational issues and then finally pull that discussion together in a formal definition of the MDP we analyze to represent any given PPDDL problem instance. We do not consider quantified and/or disjunctive goals, but handling such goals would be an interesting and useful extension of this work.

2.4.1 Planning Domains and Problems

A *planning domain* is a distribution over problem instances sharing the same state predicates P_W, action names N, and action definitions. Actions can take objects as parameters, and are defined by giving discrete finite probability distributions over action outcomes, each of which is specified using add and delete lists of state facts about the action parameters.

Given a domain definition, each problem instance in the domain specifies a finite object set O, initial state s_i and goal condition G. The initial state is given as a set of state facts and the goal condition is given as a conjunction of state facts, each constructed from the predicates in P_W.

1. Each state predicate has an associated "arity" indicating the number of objects it relates. The state predicate can be "applied" to that number of objects from the domain to form a ground state fact that can be either true or false in each state; states are then the different possible ways to select the true state facts. Likewise, each action name has an associated "arity" that is a natural number indicating the number of objects the action will act upon. The action name can then be "applied" to that number of objects to form a "grounded action".

2.4.2 PPDDL REPRESENTATION

PPDDL is the standard planning language for the international probabilistic planning competitions. In PPDDL, a planning domain syntax and a planning problem syntax is defined. To completely define a planning instance, one has to specify a domain definition and a problem definition using the respective syntax. Conditional effects and quantified preconditions are allowed in the domain definition.

In planning competitions, it has been customary to specify planning domains by providing *problem generators* that accept size parameters as input and then output PPDDL problem instances. These generators thus specify size-parameterized planning domains. It is important to note, however, that not all problem generators provided in the recent planning competitions specify planning domains according to the definition used here. In particular, some problem generators vary the action set or the state predicates between the instances generated. The relationship between the different problem instances generated by such generators is much looser than that required by our definition, and as such these "domains" are somewhat more like arbitrary collections of planning problems.

Because our logical language allows generalization between problems only if those problems share the same state and action language, we limit our empirical evaluation in Section 7 to domains that were provided with problem generators that specify planning domains as just defined here, i.e., without varying the action definitions between instances (or for which we can easily code such a generator). We refer to domains with such generators as *planning domains with fixed action definitions*.

2.4.3 GENERALIZATION BETWEEN PROBLEMS OF VARYING SIZE

Because the object set varies in size, without bound, across the problem instances of a domain, there are infinitely many possible states within the different instances of a single domain. Each MDP we analyze has a finite state space, and so we model a planning domain as an infinite set of MDPs for which we are seeking a good policy (in the form of a good value function), one MDP for each problem instance[2].

A value function for an infinite set of MDPs is a mapping from the disjoint union of the state spaces of the MDPs to the real numbers. Such a value function can be used greedily as a policy in any of the MDPs in the set. However, explicit representation of such a value function would have infinite size. Here, we will use knowledge representation techniques to compactly represent value functions over the infinite set of problem instance MDPs for any given planning domain. The compact representation derives from generalization across the domains, and our approach is fundamentally about finding good generalizations between the MDPs within a single planning domain. Our representation for value functions over planning domains is given below in Sections 2.5 and 4.

In this section, we discuss how to represent as a single finite MDP any single planning problem instance. However, we note that our objective in this work is to find good value functions for the infinite collections of such MDPs that represent planning domains. Throughout this paper, we assume that each planning domain is provided along with a means for sampling example problems from the domain, and that the sampling is parameterized by difficulty (generally, problem size) so

2. In this paper we consider two candidate representations for features; only one of these, the relational representation, is capable of generalizing between problem sizes. For the propositional representation, we restrict all training and testing to problem instances of the same size.

that easy example problems can be selected. Although, PPDDL does not provide any such problem distributions, benchmark planning domains are often provided with problem generators defining such distributions: where such generators are available, we use them, and otherwise we code our own distributions over problem instances.

2.4.4 GENERALIZING BETWEEN PROBLEMS WITH VARYING GOALS

To facilitate generalization between problem instances with different goals, and following the work of Martin and Geffner (2004) and Fern et al. (2006), we translate a PPDDL instance description into an MDP where each state specifies not only what is true in the state but also what the goal is. Action transitions in this MDP will never change the "goal", but the presence of that goal within the state description allows value functions (that are defined as conditioning only on the state) to depend on the goal as well. The goal region of the MDP will simply be those MDP states where the specified current state information matches the specified goal information.

Formally, in translating PPDDL problem instances into compact MDPs, we enrich the given set of world-state predicates P_W by adding a copy of each predicate indicating the desired state of that predicate. We name the goal-description copy of a predicate p by prepending the word "goal-" to the name. The set of all goal-description copies of the predicates in P_W is denoted P_G, and we take $P_W \cup P_G$ to be the state predicates for the MDP corresponding to the planning instance. Intuitively, the presence of goal-$p(a,b)$ in a state indicates that the goal condition requires the fact $p(a, b)$ to be part of the world state. The only use of the goal predicates in constructing a compact MDP from a PPDDL description is in constructing the initial state, which will have the goal conditions true for the goal predicates.

We use the domain Blocksworld as an example here to illustrate the reformulation (the same domain is also used as an example in Fern et al., 2006). The goal condition in a Blocksworld problem can be described as a conjunction of ground **on-top-of** facts. The world-state predicate **on-top-of** is in P_W. As discussed above, this implies that the predicate **goal-on-top-of** is in P_G. Intuitively, one ground instance of that predicate, **goal-on-top-of(b1,b2)**, means that for a state in the goal region, the block **b1** has to be directly on the top of the block **b2**.

2.4.5 STATES WITH NO AVAILABLE ACTIONS

PPDDL allows the definition of domains where some states do not meet the preconditions for any action to be applied. However, our MDP formalism requires at least one available action in every state. In translating a PPDDL problem instance to an MDP we define the action transitions so that any action taken in such a "dead" state transitions deterministically to the absorbing $\perp$ state. Because we consider such states undesirable in plan trajectories, we give these added transitions a reward of negative one unless the source state is a goal state.

2.4.6 THE RESULTING MDP

We now pull together the above elements to formally describe an MDP $M = (S, A, R, T, s_0)$ given a PPDDL planning problem instance. As discussed in Section 2.3, the set S is defined by specifying the predicates and objects available. The PPDDL description specifies the sets N of action names and O of objects, as well as a set P_W of world predicates. We construct the enriched set $P = P_W \cup P_G$ of state predicates and define the state space as all sets of applications of these predicates to the objects in O. The set $A(s)$ for any state s is the set of PPDDL action instances built from N and O for which s satisfies the preconditions, except that if this set is empty, $A(s)$ is the set of all PPDDL action instances built from N and O. In the latter case, we say the state is "dead." The reward function R is defined as discussed previously in Section 2.2; i.e., $R(s, a, s') = 1$ when the goal condition G is true in s, $R(s, a, s') = -1$ when s is a non-goal dead state, and zero otherwise. We define $T(s, a, s')$ according to the semantics of PPDDL augmented with the semantics of $\perp$ from Section 2.2—$T(s, a, \perp)$ will be one if s satisfies G, s is dead, or $s = \perp$, and zero otherwise.[3] Transiting from one state to another never changes the goal condition description in the states given by predicates in P_G. The MDP initial state s_0 is just the PPDDL problem initial state s_i augmented by the goal condition G using the goal predicates from P_G. If a propositional representation is desired, it can be easily constructed directly from this relational representation by grounding.

2.5 Linear Approximation of Value Functions

As many previous authors have done (Patrascu et al., 2002; Sanner & Boutilier, 2009; Bertsekas & Tsitsiklis, 1996; Tesauro, 1995; Tsitsiklis & Roy, 1997), we address very large compactly represented S and/or A by implicitly representing value functions in terms of state-space *features* $f : S \to \mathbb{R}$. Our features f must select a real value for each state. We describe two approaches to representing and selecting such features in Section 4.

Recall from Section 1 that our goal is to learn a value function for a family of related MDP problems. We assume that our state-space features are defined across the union of the state spaces in the family.

We represent value functions using a linear combination of l features extracted from s, i.e., as $\tilde{V}(s) = \sum_{i=0}^{l} w_i f_i(s)$, where $f_0(s) = 1$. Our goal is to find features f_i (each mapping states to real values) and weights w_i so that $\tilde{V}$ closely approximates V^*. Note that a single set of features and weight vector defines a value function for all MDPs in which those features are defined.

Various methods have been proposed to select weights w_i for linear approximations (see, e.g., Sutton, 1988 or Widrow & Hoff, 1960). Here, we review and use a trajectory-based approximate value iteration (AVI) approach. Other training methods can easily be substituted. AVI constructs a finite sequence of value functions V^1, V^2, ..., V^T, and returns the last one. Each value function is represented as $V^\beta(s) = \sum_{i=0}^{l} w_i^\beta f_i(s)$. To determine weights $w_i^{\beta+1}$ from V^β, we draw a set of training states $s_1, s_2, \ldots, s_n$ by following policy Greedy(V^β) in different example problems sampled from the provided problem distribution at the current level of problem difficulty. (See Section 3 for discussion of the control of problem difficulty.) The number of trajectories drawn and the maximum length of each trajectory are parameters of this AVI method. For each training state s, we compute the Bellman update $\mathcal{U}(V^\beta)(s)$ from the MDP model of the problem instance. We can then compute $w_i^{\beta+1}$ from the training states using

$$w_i^{\beta+1} = w_i^\beta + \frac{1}{n_i} \sum_j \alpha f_i(s_j)(\mathcal{U}(V^\beta)(s_j) - V^\beta(s_j)), \tag{1}$$

where α is the learning rate and n_i is the number of states s in $s_1, s_2, \ldots, s_n$ for which $f_i(s)$ is non-zero. Weight updates using this weight-update formula descend the gradient of the L_2 distance between V^β and $\mathcal{U}(V^\beta)$ on the training states, with the features first rescaled to normalize the

3. Note that according to our definitions in Section 2.2, the dead states are now technically "goal states", but have negative rewards.

effective learning rate to correct for feature values with rare occurrence in the training set.[4] Pseudo-code for our AVI method and for drawing training sets by following a policy is available in Online Appendix 1 (available on JAIR website), on page 2.

Here, we use the greedy policy to draw training examples in order to focus improvement on the most relevant states. Other state distributions can be generated that are not biased by the current policy; in particular, another option worth considering, especially if feature learning is stuck, would be the long random walk distribution discussed in the work of Fern, Yoon, and Givan (2004). We leave detailed exploration of this issue for future work. For a more substantial discussion of the issues that arise in selecting the training distribution, please see the book by Sutton and Barto (1998). It is worth noting that on-policy training has been shown to converge to the optimal value function in the closely related reinforcement learning setting using the SARSA algorithm (Singh, Jaakkola, Littman, & Szepesvari, 2000).

In general, while AVI often gives excellent practical results, it is a greedy gradient-descent method in an environment that is not convex due to the maximization operation in the Bellman error function. As such, there is no guarantee on the quality of the weight vector found, even in the case of convergence. Convergence itself is not guaranteed, and, in our experiments, divergent weight training was in fact a problem that required handling. We note that our feature-discovery methods can be used with other weight-selection algorithms such as approximate linear programming, should the properties of AVI be undesirable for some application.

We have implemented small modifications to the basic weight update rule in order to use AVI effectively in our setting; these are described in Section 5 in Online Appendix 1 (available on JAIR website).

3. Feature-Discovering Value-function Construction

In planning, once state features are provided, domain-independent algorithms such as AVI can learn weighted combinations of those features that often perform well as heuristic estimates of state value (e.g., distance to the goal). We now describe methods to select and add features that describe state-space regions of high inconsistency in the Bellman equation (statewise Bellman error) during approximate value iteration. Our methods can be applied using any real-valued-feature hypothesis space with a corresponding learning method for selecting features to match a real-valued function on a training set of states. Here, we will use the learner to select features that match the statewise Bellman error function.

As noted above, we use a "boosting style" learning approach in finding value functions, iterating between selecting weights and generating new features by focusing on the Bellman error in the current value function. Our value function representation can be viewed as a weighted ensemble of single-feature hypotheses. We start with a value function that has only a trivial feature, a constant feature always returning the value one, with initial weight zero. We iteratively both retrain the weights and select new features matching regions of states for which the current weighted ensemble has high statewise Bellman error.

We take a "learning from small problems" approach and learn features first in problems with relatively lower difficulty, and increase problem difficulty over time, as discussed below. Lower difficulty problems are typically those with smaller state spaces and/or shorter paths to positive

4. In deriving this gradient-descent weight-update formula, each feature f_i is scaled by $r_i = \sqrt{\frac{n}{n_i}}$, giving $f'_i = r_i f_i$.

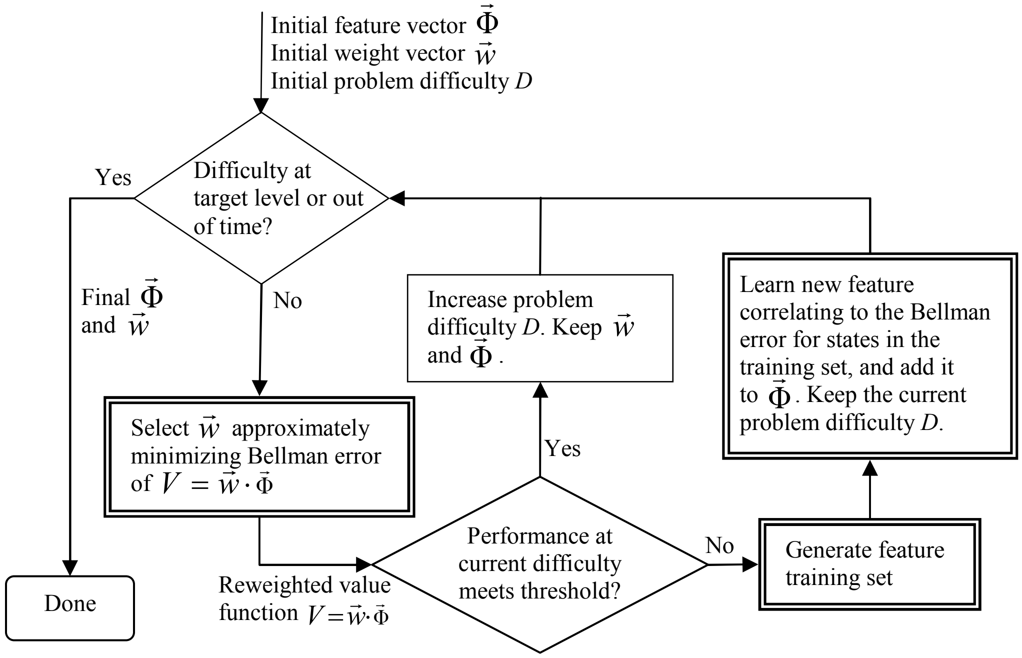


Figure 1: Control flow for feature learning. Boxes with double borders represent assumed subroutines for our method. We assume that the problem distribution is parameterized by problem difficulty (such as problem size).

feedback (e.g. goal states). Learning initially in more difficult problems will typically lead to inability to find positive feedback and random-walk behavior; as a result learning first in lower difficulty problems has been found more effective (Martin & Geffner, 2004; Yoon, Fern, & Givan, 2002). We show experimentally in Section 7 that good value functions for high difficulty problems can indeed be learned in this fashion from problems of lower, increasing difficulties.

Our approach relies on two assumed subroutines, and can be instantiated in different ways by providing different algorithms for these subroutines. First, a method of weight selection is assumed; this method takes as input a problem domain and a fixed set of features, and selects a weight vector for a value function for the problem domain using the provided features. We intend this method to heuristically or approximately minimize L_∞ Bellman error in its choice of weight vector, but in practice it may be easier to adjust weights to approximate L_2 Bellman error. Second, a feature hypothesis space and corresponding learner are assumed to be provided by the system designer.

The control flow for our approach is shown in Figure 1. Each iteration at a fixed problem distribution selects weights for the current feature set (using any method attempting to minimize L_∞ Bellman error) to define a new value function V, selects a training set of states for feature learning, then learns a new feature correlating well to the statewise Bellman error of V, adding that feature to the feature set. A user-provided performance-threshold function τ detects when to increase the problem difficulty. A formalization of this control flow is given in Figure 2, in the form of pseudo-code.

Feature-discovering Value-function Construction

Inputs: Initial feature vector $\overrightarrow{\Phi}_0$, initial weight vector $\overrightarrow{w}_0$,
Sequence of problem distributions $D_1, D_2, \cdots, D_{\text{max}}$ of increasing difficulty,
Performance threshold function τ.
//$\tau(D, V)$ tests the performance of value function V in distribution D.

Outputs: Feature vector $\overrightarrow{\Phi}$, weight vector $\overrightarrow{w}$

1. $\overrightarrow{\Phi} \leftarrow \overrightarrow{\Phi}_0$, $\overrightarrow{w} \leftarrow \overrightarrow{w}_0$, $d \leftarrow 1$
2. **while** not ($d >$ max or out of time)
3.   Select $\overrightarrow{w}$ approximately minimizing Bellman error of $V = \overrightarrow{w} \cdot \overrightarrow{\Phi}$ over D_d
4.   **if** $\tau(D_d, \overrightarrow{w} \cdot \overrightarrow{\Phi})$
5.   **then** $d \leftarrow d + 1$
6.   **else**
7.   Generate a sequence of training states T using D_d
8.   Learn new feature f correlating to the Bellman error feature $B(\overrightarrow{w} \cdot \overrightarrow{\Phi}, \cdot)$ for the states in T
9. $\overrightarrow{\Phi} \leftarrow (\overrightarrow{\Phi}; f)$, $\overrightarrow{w} \leftarrow (\overrightarrow{w}; 0)$
10. **return** $\overrightarrow{\Phi}$, $\overrightarrow{w}$

Notes:

1. $B(\cdot, \cdot)$ is the statewise-Bellman error function, as defined in Section 2.1.
2. The code for approximate value iteration **AVI**, shown in Online Appendix 1 (available on JAIR website) on page 2, is an example implementation of line 3.
3. The code for **draw**(Greedy($\overrightarrow{w} \cdot \overrightarrow{\Phi}$), N_{training}), shown in Online Appendix 1 on page 2, is an example implementation of line 7. N_{training} is the number of states in the feature training set. Duplicated states are removed as specified in Section 3.1.
4. The beam-search code for learning relational features **beam-search-learn**(**score**($\cdot, T, B(\overrightarrow{w} \cdot \overrightarrow{\Phi}, \cdot)$)) is an example implementation of line 8, where **beam-search-learn** is shown in figure 3 in Section 3, and **score** is defined in Section 4.2.

Figure 2: Pseudo-code for learning a set of features.

For the experiments reported in Section 7, we evaluate the following choices for the assumed subroutines. For all experiments we use AVI to select weights for feature sets. We evaluate two choices for the feature hypothesis space and corresponding learner, one relational and one propositional, as described in Section 4.

Separate training sets are drawn for weight selection and for the feature learning; the former will depend on the weight selection method, and is described for AVI in Section 2.5, and the latter is described in this section.

Problem difficulty is increased when sampled performance of the greedy policy at the current difficulty exceeds user-specified performance thresholds. In our planning-domain experiments, the performance parameters measured are success ratio (percentage of trials that find the goal) and average successful plan length (the average number of steps to the goal among all successful trials). The non-goal-oriented domains of Tetris and SysAdmin use different performance measures: average total reward for Tetris and Bellman error for SysAdmin (to facilitate comparison with Patrascu et al., 2002).

We also assume a user-provided schedule for problem difficulty increases in problems where difficulty is parameterized by more than one parameter (e.g., size may be measured in by the number of objects of each type); further domain-independent automation of the increase in difficulty is a topic for future research. We give the difficulty-increase schedules and performance thresholds for our experiments in the section presenting the experiments, Section 7.

3.1 Training Set Generation

The training set for selection of a new feature is a set of states. The training set is constructed by repeatedly sampling an example problem instance from the problem distribution at the current level of difficulty, and applying the current greedy policy Greedy(V) to that problem instance to create a trajectory of states encountered. Every state (removing duplicates) encountered is added to the training set. The size of the feature-selection training set and the maximum length of each training trajectory are specified by the user as parameters of the algorithm.

Retaining duplicate states in the training set is another option that can be considered. Our preliminary empirical results have not favored this option, but it is certainly worth further exploration. We note that the goal of finding a near-optimal value function does not necessarily make reference to a state distribution: the most widely used notion of "near-optimal" in the theory of MDPs is the sup-norm distance to V^*. Moreover, the state distribution represented by the duplicates in our training sets is typically the distribution under a badly flawed policy; heeding this distribution can prevent correcting Bellman error in critical states that are visited by this policy, but visited only rarely. (These states may be, for instance, rarely visited "good exits" from the visited state region that are being misunderstood by the current value function.) At this point, our primary justification for removing duplicates is the empirical performance we have demonstrated in Section 7.

Similar reasoning would suggest removing duplicate states in the training sets for AVI weight training, described in Section 2.5. Because there are many large AVI training sets generated in our experiments, duplicate removal must be carefully handled to control runtime; for historical reasons, our experiments shown here do not include duplicate removal for AVI.

A possible problem occurs when the current greedy policy cannot reach enough states to complete the desired training set. If 200 consecutive trajectories are drawn without visiting a new state before the desired training set size is reached, the process is modified as follows. At that point, the method attempts to complete the training set by drawing trajectories using random walk (again using sampled example problems from the current problem distribution). If this process again leads to 200 consecutive trajectories without a new state, the method terminates training-set generation and uses the current training set even though it is smaller than the target size.

3.2 Applicability of the Method

Feature-discovering value-function construction as just described does not require complete access to the underlying MDP model. Our AVI updates and training set generation are both based on the following computations on the model:

1. Given a state s the ability to compute the action set $A(s)$.
2. Given a state s, action $a \in A(s)$, and value function V, the ability to compute the Q-value $Q(s, a, V)$.
3. Given a state s and action $a \in A(s)$, the ability to draw a state from the next state distribution defined by $T(s, a, s')$.
4. Given a state s, the ability to compute the features in the selected feature language on s and any computations on the state required for the selected feature learner. As examples,
   (a) in Section 4, we introduce a relational feature language and learner that require knowledge of a set of domain predicates (and their arities) such that each state is a conjunctive set of predicate facts (see Section 2.3),
   (b) and, also in Section 4, we describe a propositional feature language and learner that require knowledge of a set of propositional state attributes such that each state is a truth assignment to the attributes.

The first three items enable the computation of the Bellman update of s and the last item enables computation of the estimated value function given the weights and features defining it as well as the selection of new features by the feature learner. These requirements amount to substantial access to the problem model; as a result our method must be considered a model-based technique.

A consequence of these requirements is that our algorithm cannot be directly applied to the standard reinforcement learning setting where the only model access is via acting in the world without the ability to reset to selected states; in this setting Bellman error computations for particular states cannot necessarily be carried out. It would be possible to construct a noisy Bellman error training set in such a model-free setting and it would be appropriate future work to explore the use of such a training set in feature learning.

While the PPDDL planning domains studied provide all the information needed to perform these computations, our method also applies to domains that are not natural to represent in PPDDL. These can be analyzed by our method once the above computations can be implemented. For instance, in our Tetris experiments in Section 7.2, the underlying model is represented by providing hand-coded routines for the above computations within the domain.

3.3 Analysis

MDP value iteration is guaranteed to converge to the optimal value function if conducted with a tabular value-function representation in the presence of discounting (Bertsekas, 1995). Although weight selection in AVI is designed to mimic value iteration, while avoiding a tabular representation, there is no general guarantee that the weight updates will track value iteration and thus converge to the optimal value function. In particular, there may be no weighted combination of features that represents the optimal value function, and likewise none that represents the Bellman update $\mathcal{U}(V)$ for some value function V produced by AVI weight training process. Our learning system introduces new features to the existing feature ensemble in response to this problem: the training set used to select the new feature pairs states with their statewise Bellman error. If the learned feature exactly captures the statewise Bellman-error concept (by exactly capturing the training set and generalizing successfully) then the new feature space will contain the Bellman update of the value function used to generate the training data.

We aim to find features that approximate the "Bellman error feature," which we take to be a function mapping states to their statewise Bellman error. Theoretical properties of Bellman error features in the uncontrolled Markov processes (i.e., without the max operator in the Bellman equation) have recently been discussed in the work of Parr et al. (2007), where the addition of such features (or close approximations thereof) is proven to reduce the weighted L_2-norm distance between the best weight setting and the the true (uncontrolled) value V^*, when linear fixed-point methods are used to train the weights before feature addition. Prior to that work (in Wu & Givan, 2005), and now in parallel to it, we have been empirically exploring the effects of selecting Bellman error features in the more complex controlled case, leading to the results reported here.

It is clear that if we were to simply add the Bellman error feature directly, and set the corresponding weight to one, the resulting value function would be the desired Bellman update $\mathcal{U}(V)$ of the current value function V. Adding such features at each iteration would thus give us a way to conduct value iteration exactly, without enumerating states. But each such added feature would describe the Bellman error of a value function defined in terms of previously added features, posing a serious computational cost issue when evaluating the added features. In particular, each Bellman error feature for a value function V can be estimated at any particular state with high confidence by evaluating the value function V at that state and at a polynomial-sized sample of next states for each action (based on Chernoff bounds).

However, if the value function V is based upon a previously added Bellman-error feature, then each evaluation of V requires further sampling (again, for each possible action) to compute. In this manner, the amount of sampling needed for high confidence grows exponentially with the number of successive added features of this type. The levels of sampling do not collapse into one expectation because of intervening choices between actions, as is often the case in decision-theoretic sampling. Our feature selection method is an attempt to tractably approximate this exact value iteration method by learning concise and efficiently computable descriptions of the Bellman-error feature at each iteration.

Our method can thus be viewed as a heuristic approximation to exact value iteration. Exact value iteration is the instance of our method obtained by using an explicit state-value table as the feature representation and generating training sets for feature learning containing all states — to obtain exact value iteration we would also omit AVI training but instead set each weight to one.

When the feature language and learner can be shown to approximate explicit features tightly enough (so that the resulting approximate Bellman update is a contraction in the L_∞ norm), then it is easy to prove that tightening approximations of V^* will result if all weights are set to one. However, for the more practical results in our experiments, we use feature representations and learners for which no such approximation bound relative to explicit features is known.

4. Two Candidate Hypothesis Spaces for Features

In this section we describe two hypothesis spaces for features, a relational feature space and a propositional feature space, along with their respective feature learning methods. For each of the two feature spaces, we assume the learner is provided with a training set of states paired with their statewise Bellman error values.

Note that these two feature-space-learner pairs lead to two instances of our general method and that others can easily be defined by defining new feature spaces and corresponding learners. In this paper we empirically evaluate the two instances presented here.

4.1 Relational Features

A relational MDP is defined in terms of a set of state predicates. These state predicates are the basic elements from which we define a feature-representation language. Below, we define a general-purpose means of enriching the basic set of state predicates. The resulting enriched predicates can be used as the predicate symbols in standard first-order predicate logic. We then consider any formula in that logic with one free variable as a feature, as follows[5].

A state in a relational MDP is a first-order interpretation. A first-order formula with one free variable is then a function from such states to natural numbers which maps each state to the number of objects in that state that satisfy the formula. We take such first-order formulas to be real-valued features by normalizing to a real number between zero and one—this normalization is done by dividing the feature value by the maximum value that the feature can take, which is typically the total number of objects in the domain, but can be smaller than this in domains where objects (and quantifiers) are typed. A similar feature representation is used in the work of Fawcett (1996).

This feature representation is used for our relational experiments, but the learner we describe in the next subsection only considers existentially quantified conjunctions of literals (with one free variable) as features. The space of such formulas is thus the effective feature space for our relational experiments.

> **Example 4.1:** Take Blocksworld with the table as an object for example, $\mathbf{on}(x, y)$ is a predicate in the domain that asserts the block x is on top of the object y, where y may be a block or the table. A possible feature for this domain can be described as $\exists y$ $\mathbf{on}(x, y)$, which is a first-order formula with x as the one free variable. This formula means that there is some other object immediately below the block object x, which essentially excludes the table object and the block being held by the arm (if any) from the object set described by the feature. For n blocks problems, the un-normalized value of this feature is n for states with no block being held by the arm, or $n - 1$ for states with a block being held by the arm.

4.1.1 The Enriched Predicate Set

More interesting examples are possible with the enriched predicate set that we now define. To enrich the set of state predicates P, we add for each binary predicate p a transitive closure form of that predicate $p+$ and predicates min-p and max-p identifying minimal and maximal elements under that predicate. In goal-based domains, recall that our problem representation (from Section 2.4) includes, for each predicate p, a goal version of the predicate called goal-p to represent the desired state of the predicate p in the goal. Here, we also add a means-ends analysis predicate correct-p to represent p facts that are present in both the current state and the goal.

So, for objects x and y, correct-$p(x,y)$ is true if and only if both $p(x, y)$ and goal-$p(x,y)$ are true. $p+(x, y)$ is true of objects x and y connected by a path in the binary relation p. The relation max-$p(x)$ is true if object x is a maximal element with respect to p, i.e., there exists no other object

5. Generalizations to allow multiple free variables are straightforward but of unclear utility at this time.

y such that $p(x, y)$ is true. The relation min-$p(x)$ is true if object x is a minimal element with respect to p, i.e., there exists no other object y such that $p(y, x)$ is true.

We formally define the feature grammar in Online Appendix 1 (available on JAIR website) on page 3.

> **Example 4.1 (cont.):** The feature $\exists y$ correct-**on**(x, y) means that x is stacked on top of some object y both in the current state and in the goal state. The feature $\exists y$ **on**$+(x, y)$ means that in the current state, x is directly above some object y, i.e., there is a sequence of **on** relations traversing a path between x and y, inclusively. The feature max-**on**$+(x)$ means that x is the table object when all block-towers are placed on the table, since the table is the only object that is not **on** any other object. The feature min-**on**$+(x)$ means that there is no other object on top of x, i.e., x is clear.

4.2 Learning Relational Features

We select first-order formulas as candidate features using a beam search with a beam width W. We present the pseudo-code for beam search in Figure 3. The search starts with basic features derived automatically from the domain description and repeatedly derives new candidate features from the best scoring W features found so far, adding the new features as candidates and keeping only the best scoring W features at all times. After new candidates have been added a fixed depth d times, the best scoring feature found overall is selected to be added to the value-function representation. Candidate features are scored for the beam search by their correlation to the Bellman error feature as formalized below.

Specifically, we score each candidate feature f with its correlation coefficient to the Bellman error feature $B(V, \cdot)$ as estimated by a training set. The correlation coefficient between functions ϕ and ϕ' is defined as **corr-coef**$(\phi, \phi') = \frac{E\{\phi(s)\phi'(s)\} - E\{\phi(s)\}E\{\phi'(s)\}}{\sigma_\phi \sigma_{\phi'}}$. Instead of using a known distribution to compute this value, we use the states in the training set Δ_s and compute a sampled version by using the following equations to approximate the true expectation E and the true standard deviation σ of any random variable X:

$$E_{\Delta_s}\{X(s)\} = \frac{1}{|\Delta_s|} \sum_{s' \in \Delta_s} X(s'),$$

$$\sigma_{X,\Delta_s} = \sqrt{\frac{1}{|\Delta_s|} \sum_{s' \in \Delta_s} (X(s') - E\{X(s)\})^2},$$

$$\textbf{corr-coef-sampled}(\phi, \phi', \Delta_s) = \frac{E_{\Delta_s}\{\phi(s)\phi'(s)\} - E_{\Delta_s}\{\phi(s)\}E_{\Delta_s}\{\phi'(s)\}}{\sigma_{\phi,\Delta_s}\sigma_{\phi',\Delta_s}}.$$

The scoring function for feature selection is then a regularized version of the correlation coefficient between the feature and the target function ϕ

$$\textbf{score}(f, \Delta_s, \phi) = |\text{corr-coef-sampled}(f, \phi, \Delta_s)|(1 - \lambda \text{depth}(f)),$$

where the "depth" of a feature is the depth in the beam search at which it first occurs, and λ is a parameter of the learner representing the degree of regularization (bias towards low-depth features).

beam-search-learn

Inputs: Feature scoring function **fscore** : features $\rightarrow [0, 1]$

Outputs: New feature f

System parameters: W: Beam width
$\max_d$: Max number of beam-search iterations
λ: Degree of regularization, as defined in Section 4.2

1. $I \leftarrow$ the set of basic features, as defined in Section 4.2.
2. $d \leftarrow 1, F \leftarrow I$.
3. **repeat**
4.   Set beam B to the highest scoring W candidates in F.
5.   Candidate feature set $F \leftarrow B$.
6.   **for** each candidate $f_1 \in B$
7.   **for** each candidate $f_2 \in (B \cup I)$, $f_2 \neq f_1$
8. $F = F \cup \text{combine}(f_1, f_2)$.
9. $d \leftarrow d + 1$.
10. **until** ($d > \max_d$) **or** (highest score so far $\geq (1 - \lambda d)$).
11. **return** the maximum scoring feature $f \in F$.

Notes:

1. Feature scoring function **fscore**(f) is used to rank candidates in lines 4 and 11. A discussion of a sample scoring function, used in our relational experiments, is given in Section 4.2.
2. Candidate scores can be cached after calls to **fscore**, so that no candidate is scored twice.
3. The value $(1 - \lambda d)$ is the largest score a feature of depth d can have.

Figure 3: Pseudo-code for beam search.

The value **score**$(f, \Delta_s, B(V, \cdot))$ is then the score of how well a feature f correlates to the Bellman error feature. Note that our features are non-negative, but can still be well correlated to the Bellman error (which can be negative), and that the presence of a constant feature in our representation allows a non-negative feature to be shifted automatically as needed.

It remains only to specify which features in the hypothesis space will be considered initial, or basic, features for the beam search, and to specify a means for constructing more complex features from simpler ones for use in extending the beam search. We first take the state predicate set P in a domain and enrich P as described in Section 4.1. After this enrichment of P, we take as basic features the existentially quantified applications of (possibly negated) state predicates to variables with zero or one free variable[6]. A grammar for basic features is defined as follows.

6. If the domain distinguishes any objects by naming them with constants, we allow these constants as arguments to the predicates here as well.

Definition: A *basic feature* is an existentially quantified $\langle$literal$\rangle$ expression with at most one free variable (see Figure 3 in Online Appendix 1, available on JAIR website, on page 3).

A feature with no free variables is treated technically as a one-free-variable feature where that variable is not used; this results in a "binary" feature value that is either zero or the total number of objects, because instantiating the free variable different ways always results in the same truth value. We assume throughout that every existential quantifier is automatically renamed away from every other variable in the system. We can also take as basic features any human-provided features that may be available, but we do not add such features in our experiments in this paper in order to clearly evaluate our method's ability to discover domain structure on its own.

At each stage in the beam search we add new candidate features (retaining the W best scoring features from the previous stage). The new candidate features are created as follows. Any feature in the beam is combined conjunctively with any other, or with any basic feature. The method of combination of two features is described in Figure 4. This figure shows non-deterministic pseudo-code for combining two input features, such that any way of making the non-deterministic choices results in a new candidate feature. The pseudo-code refers to the feature formulas f_1 and f_2 describing the two features. In some places, these formulas and others are written with their free variable exposed, as $f_1(x)$ and $f_2(y)$. Also substitution for that variable is notated by replacing it in the notation, as in $f_1(z)$.

The combination is by conjoining the feature formulas, as shown in line 2 of Figure 4; however, there is additional complexity resulting from combining the two free variables and possibly equating bound variables between the two features. The two free variables are either equated (by substitution) or one is existentially quantified before the combination is done, in line 1. Up to two pairs of variables, chosen one from each contributing feature, may also be equated, with the resulting quantifier at the front, as described in line 3. Every such combination feature is a candidate.

This beam-search construction can lead to logically redundant features that are in some cases syntactically redundant as well. We avoid syntactically redundant features at the end of the beam search by selecting the highest scoring feature that is not already in the feature set. Logical redundancy that is not syntactic redundancy is more difficult to detect. We avoid some such redundancy automatically by using ordering during the beam search to reduce the generation of symmetric expressions such as $\phi \wedge \psi$ and $\psi \wedge \phi$. However, testing logical equivalence between features in our language is NP-hard (Chandra & Merlin, 1977), so we do not deploy a complete equivalence test here.

Example 4.2: Assume we have two basic features $\exists z\, p(x, z)$ and $\exists w\, q(y, w)$. The set of the possible candidates that can be generated by combining these two features are: When line 3 in Figure 4 runs zero times,

1. $(\exists x\, \exists z\, p(x, z)) \wedge (\exists w\, q(y, w))$, from $\exists x f_1(x) \wedge f_2(y)$
2. $(\exists z\, p(x, z)) \wedge (\exists y\, \exists w\, q(y, w))$, from $f_1(x) \wedge \exists y f_2(y)$, and
3. $(\exists z\, p(x, z)) \wedge (\exists w\, q(x, w))$, from $f_1(x) \wedge f_2(x)$

and when line 3 runs one time,

4. $\exists u\, ((\exists z\, p(u, z)) \wedge (q(y, u)))$, from equating x and w in item 1 above,
5. $\exists u\, (\exists x\, p(x, u)) \wedge (q(y, u))$, from equating x and z in item 1 above,

combine

Inputs: Features $f_1(x)$, $f_2(y)$

Outputs: Set of features $\{o_1\}$

return the set of all features o_1 that can result from:

1. Perform one of
   a. $f_1 = (\exists x) f_1(x)$
   b. $f_2 = (\exists y) f_2(y)$
   c. $f_2 = f_2(x)$
2. $o_1 = f_1 \wedge f_2$
3. Perform the following variable equating step zero, one, or two times:
   a. Let v be a variable occurring in f_1 and o_1.
      Let e_1 be the expression of the form $(\exists v)\phi_1(v)$ that occurs in o_1
   b. Let w be a variable occurring in f_2 and o_1.
      Let e_2 be the expression of the form $(\exists w)\phi_2(w)$ that occurs in o_1
   c. Let u be a new variable, not used in o_1
   d. $o_2 =$ replace e_1 with $\phi_1(u)$ and replace e_2 with $\phi_2(u)$ in o_1
   e. $o_1 = (\exists u) o_2$

Notes:

1. The choice between 1a, 1b, and 1c, the choice of number of iterations of step 3, and the choices of e_1 and e_2 in steps 3a and 3b are all non-deterministic choices.
2. Any feature that can be produced by any run of this non-deterministic algorithm is included in the set of features that is returned by **combine**.
3. It is assumed that f_1 and f_2 have no variables in common, by renaming if necessary before this operation.

Figure 4: A non-deterministic algorithm for combining two feature formulas.

6. $\exists u\ (p(x,u) \wedge (\exists w\ q(u,w)))$, from equating z and y in item 2 above,
7. $\exists u\ (p(x,u) \wedge (\exists y\ q(y,u)))$, from equating z and w in item 2 above, and
8. $\exists u\ (p(x,u) \wedge (\ q(x,u)))$, from equating z and w in item 3 above.

The first three are computed using cases 1a, 1b, and 1c, respectively. The remaining five derive from the first three by equating bound variables from f_1 and f_2.

Features generated at a depth k in this language can easily require enumerating all k-tuples of domain objects. Since the cost of this evaluation grows exponentially with k, we bound the maximum number of quantifiers in scope at any point in any feature formula to q, and refuse to consider any feature violating this bound.

The values W, λ, d, and q are the parameters controlling the relational learner we evaluate in this paper. How we set these parameters is discussed further in the experimental setup description in Section 6.

We provide a brief discussion on the motivations for our feature combination method. First, we note that additive combination of features can represent disjunctions of features[7]; hence, we only consider conjunction during feature combination. Here, we have chosen to "conjoin" features in multiple ways, varying the handling/combining of the free and bound variables. We do not believe our choice to be uniquely effective, but provide it as an example realization of the proposed feature-discovery architecture.

Any choice of feature representation and combination method must trade off between the cost of evaluation of more choices and the potential gain in quality of the selected features. Here, we have chosen to limit individual features to conjunction; effectively, we have limited the features to Horn clauses over the predicates and their negations, with univariate heads.

4.3 Propositional Features

Here we discuss a second candidate hypothesis space for features, using a propositional representation. We use decision trees to represent these propositional features. A detailed discussion of classification using decision trees can be found in the book by Mitchell (1997). A decision tree is a binary tree with internal nodes labeled by binary tests on states, edges labeled "yes" and "no" representing results of the binary tests, and leaves labeled with classes (in our case, either zero or one). A path through the tree from the root to a leaf with label l identifies a labeling of some set of states—each state consistent with the state-test results on the path is viewed as labeled l by the tree. In this way, a decision tree with real number labels at the leaves is viewed as labeling all states with real numbers, and is thus a feature.

We learn decision trees from training sets of labeled states using the well known C4.5 algorithm (Quinlan, 1993). This algorithm induces a tree greedily matching the training data from the root down. We use C4.5 to induce new features—the key to our algorithm is how we construct suitable training sets for C4.5 so that the induced features are useful in reducing Bellman error.

We include as possible state tests for the decision trees we induce every grounded predicate application[8] from the state predicates, as well as every previously selected decision-tree feature (each of which is a binary test because all leaf labels are zero or one).

4.4 Learning Propositional Features

To construct binary features, we use only the sign of the "Bellman error feature," not the magnitude. The sign of the statewise Bellman error at each state serves as an indication of whether the state is undervalued or overvalued by the current approximation, at least with respect to exactly representing the Bellman update of the current value function. If we can identify a collection of "undervalued" states as a new feature, then assigning an appropriate positive weight to that feature

7. Representing the disjunction of overlapping features using additive combination can be done with a third feature representing the conjunction, using inclusion/exclusion and a negative weight on the conjunction.

8. A grounded predicate application is a predicate applied to the appropriate number of objects from the problem instance.

will increase their value. Similarly, identifying "overvalued" states with a new feature and assigning a negative weight will decrease their value. We note that the domains of interest are generally too large for state-space enumeration, so we will need classification learning to generalize the notions of overvalued and undervalued across the state space from training sets of sample states.

To enable our method to ignore states that are approximately converged, we discard states with statewise Bellman error near zero from either training set. Specifically, among the states with negative statewise Bellman error, we discard any state with such error closer to zero than the median within that set; we do the same among the states with positive statewise Bellman error. More sophisticated methods for discarding training data near the intended boundary can be considered in future research; these will often introduce additional parameters to the method. Here, we seek an initial and simple evaluation of our overall approach. After this discarding, we define Σ_+ to be the set of all remaining training pairs with states having positive statewise Bellman error, and Σ_- likewise those with negative statewise Bellman error.

We then use Σ_+ as the positive examples and Σ_- as the negative examples for a supervised classification algorithm; in our case, C4.5 is used. The hypothesis space for classification the space of decision trees built with tests selected from the primitive attributes defining the state space and goal; in our case, we also use previously learned features that are decision trees over these attributes. The concept resulting from supervised learning is then treated as a new feature for our linear approximation architecture, with an initial weight of zero.

Our intent, ideally, is to develop an approximately optimal value function. Such a value function can be expected to have Bellman error at many states, if not every state; however, low state-wise error in some states does not contribute to high sup-norm Bellman error. Our discarding training states with low statewise Bellman error reflects our tolerance of such low error below some threshold representing the degree of approximation sought. Note that the technical motivation for selecting features based upon Bellman error focuses on reducing the sup-norm Bellman error; given this motivation, we are not as interested in finding the exact boundary between positive and negative Bellman error as we are in identifying which states have large magnitude Bellman error (so that that large-magnitude error can be addressed by feature addition).

We observe that there is limited need to separately learn a feature matching Σ_- due to the following representability argument. Consider a binary feature F and its complement $\overline{F}$, so that exactly one of F and $\overline{F}$ is true in each state. Given the presence of a constant feature in the feature set, adding F or $\overline{F}$ to the feature set yields the same set of representable value functions (assigning weight w to F has the same effect as assigning weight $-w$ to $\overline{F}$ and adding w to the weight of the constant feature).

4.5 Discussion

We discuss below the generalization capability, learning time, and heuristic elements of our feature learning method.

4.5.1 Generalization Across Varying Domain Sizes

The propositional feature space described above varies in size as the number of objects in a relational domain is varied. As a result, features learned at one domain size are not generally meaningful (or even necessarily defined) at other domain sizes. The relational approach above is, in contrast, able to generalize naturally between different domains sizes. Our experiments report on the ability of the propositional technique to learn within each domain size directly, but do not attempt to use that approach for learning from small problems to gain performance in large problems. This is a major limitation in producing good results for large domains.

4.5.2 LEARNING TIME

The primary motivation for giving up generalization over domain sizes in order to employ a propositional approach is that the resulting learner can use highly efficient, off-the-shelf classification algorithms. The learning times reported in Section 7 show that our propositional learner learns new features orders of magnitude faster than the relational learner.

4.5.3 HEURISTIC ELEMENTS OF THE METHOD

As mentioned earlier, our algorithm heuristically approximates the repeated addition of Bellman error features to a linear value-function approximation in order to carry out value iteration. Also as mentioned earlier, value iteration itself is guaranteed to converge to the optimal value function. However, due to the scale of problems we target, heuristic approximations are required. We discuss the motivations for each heuristic approximation we employ briefly here.

First, we do not compute exact Bellman error features. Instead, we use machine learning to fit a training set of sample states and their Bellman error values. The selection of this training set is done heuristically, using trajectories drawn from the current greedy policy. Our use of on-policy selection of training data is loosely motivated by on-policy convergence results for reinforcement learning (Singh et al., 2000), and serves to focus training on relevant states. (See Section 3.1.)

Second, for the relational instance of our feature framework, the beam-search method we use to select the highest scoring relational feature (with the best fit to the Bellman error) is ad-hoc, greedy, and severely resource bounded. The fit obtained to the Bellman error is purely heuristic. We provide our heuristic method for this machine learning problem only as an example, and we intend future research to provide better relational learners and resulting better planning performance. Heuristic elements of the current method are further discussed in Appendix A.3. Our work here can be viewed as providing a reduction from stochastic planning to structured machine learning of numeric functions. (See Section 3.)

Third, for the propositional instance of our feature framework,, the learner C4.5 selects hypotheses greedily. Also, our reduction to C4.5 classification relies on an explicit tolerance of approximation in the form of the threshold used to filter training data with near-zero Bellman error. The motivation for this approximation tolerance is to focus the learner on high Bellman error states and allow the method to ignore "almost converged" states. (See Section 4.4.)

Fourth, fundamental to this work is the use of a linear approximation of the value function and gradient-descent-based weight selection (in this case AVI). These approximation methods are a key approach to handling large state spaces and create the need for feature discovery. Our AVI method includes empirically motivated heuristic methods for controlling step size and sign changes in the weights. (See Section 5 in Online Appendix 1, available on JAIR website.)

Fifth, we rely on human input to select the sequence of problem difficulties encountered during feature discovery as well as the performance thresholds at which problem difficulty increases. We believe this aspect of the algorithm can be automated in future research. (See Section 3.)

5. Related Work

Automatic learning of relational features for approximate value-function representation has surprisingly not been frequently studied until quite recently, and remains poorly understood. Here, we review recent work that is related on one or more dimensions to our contribution.

5.1 Feature Selection Based on Bellman Error Magnitude

Feature selection based on Bellman error has recently been studied in the uncontrolled (policy-evaluation) context in the work of Keller et al. (2006) and Parr et al. (2007), with attribute-value or explicit state spaces rather than relational feature representations. Feature selection based on Bellman error is further compared to other feature selection methods in the uncontrolled context both theoretically and empirically in the work of Parr, Li, Taylor, Painter-Wakefield, and Littman (2008).

Here, we extend this work to the controlled decision-making setting and study the incorporation of relational learning and the selection of appropriate knowledge representation for value functions that generalize between problems of different sizes within the same domain.

The main contribution of the work of Parr et al. (2007) is formally showing, for the uncontrolled case of policy evaluation, that using (possibly approximate) Bellman-error features "provably tightens approximation error bounds," i.e., that adding an exact Bellman error-feature provably reduces the (weighted L_2-norm) distance from the optimal value function that can be achieved by optimizing the weights in the linear combination of features. This result is extended in a weaker form to approximated Bellman-error features, again for the uncontrolled case. The limitation to the uncontrolled case is a substantial difference from the setting of our work. The limited experiments shown use explicit state-space representations, and the technique learns a completely new set of features for each policy evaluation conducted during policy iteration. In contrast, our method accumulates features during value iteration, at no point limiting the focus to a single policy. Constructing a new feature set for each policy evaluation is a procedure more amenable to formal analysis than retaining all learned features throughout value iteration because the policy being implicitly considered during value iteration (the greedy policy) is potentially changing throughout. However, when using relational feature learning, the runtime cost of feature learning is currently too high to make constructing new feature sets repeatedly practically feasible.

Parr et al. (2007) builds on the prior work by Keller et al. (2006) that also studied the uncontrolled setting. That work provides no theoretical results nor any general framework, but provides a specific approach to using Bellman error in attribute value representations (where a state is represented as a real vector) in order to select new features. The approach provides no apparent leverage on problems where the state is not a real vector, but a structured logical interpretation, as is typical in planning benchmarks.

5.2 Feature Discovery via Goal Regression

Other previous methods (Gretton & Thiébaux, 2004; Sanner & Boutilier, 2009) find useful features by first identifying goal regions (or high reward regions), then identifying additional regions by regressing through the action definitions from previously identified regions. The principle exploited is that when a given state feature indicates value in the state, then being able to achieve that feature in one step should also indicate value in a state. Regressing a feature definition through the action definitions yields a definition of the states that can achieve the feature in one step. Repeated regression can then identify many regions of states that have the possibility of transitioning under some action sequence to a high-reward region.

Because there are exponentially many action sequences relative to plan length, there can be exponentially many regions discovered in this way, as well as an exponential increase in the size of the representation of each region. Both exponentials are in terms of the number of regression steps taken. To control this exponential growth in the number of features considered, regression has been implemented with pruning optimizations that control or eliminate overlap between regions when it can be detected inexpensively as well as dropping of unlikely paths. However, without a scoring technique (such as the fit to the Bellman-error used in this paper) to select features, regression still generates a very large number of useless new features. The currently most effective regression-based first-order MDP planner, described in the work of Sanner and Boutilier (2009), is only effective when disallowing overlapping features to allow optimizations in the weight computation. Yet clearly most human-designed feature sets in fact have overlapping features.

Our inductive technique avoids these issues by considering only compactly represented features, selecting those which match sampled statewise Bellman error training data. We provide extensive empirical comparison to the First-Order Approximate Linear Programming technique (FOALP) from the work of Sanner and Boutilier (2009) in our empirical results. Our empirical evaluation yields stronger results across a wide range of probabilistic planning benchmarks than the goal-regression approach as implemented in FOALP (although aspects of the approaches other than the goal-regression candidate generation vary in the comparison as well).

Regression-based approaches to feature discovery are related to our method of fitting Bellman error in that both exploit the fact that states that can reach valuable states must themselves be valuable, i.e. both seek local consistency. In fact, regression from the goal can be viewed as a special case of iteratively fitting features to the Bellman error of the current value function. Depending on the exact problem formulation, for any k, the Bellman error for the k-step-to-go value function will be non-zero (or otherwise nontrivially structured) at the region of states that reach the goal first in $k + 1$ steps. Significant differences between our Bellman error approach and regression-based feature selection arise for states which can reach the goal with different probabilities at different horizons. Our approach fits the magnitude of the Bellman error, and so can smoothly consider the degree to which each state reaches the goal at each horizon. Our approach also immediately generalizes to the setting where a useful heuristic value function is provided before automatic feature learning, whereas the goal-regression approach appears to require goal regions to begin regression. In spite of these issues, we believe that both approaches are appropriate and valuable and should be considered as important sources of automatically derived features in future work.

Effective regression requires a compact declarative action model, which is not always available[9]. The inductive technique we present does not require even a PDDL action model, as the only deductive component is the computation of the Bellman error for individual states. Any representation from which this statewise Bellman error can be computed is sufficient for this technique. In our empirical results we show performance for our planner on Tetris, where the model is represented only by giving a program that, given any state as input, returns the explicit next state distribution for that state. FOALP is inapplicable to such representations due to dependence on logical deductive rea-

9. For example, in the Second International Probabilistic Planning Competition, the regression-based FOALP planner required human assistance in each domain in providing the needed domain information even though the standard PDDL model was provided by the competition and was sufficient for each other planner.

soning. We believe the inductive and deductive approaches to incorporating logical representation are both important and are complementary.

The goal regression approach is a special case of the more general approach of generating candidate features by transforming currently useful features. Others that have been considered include abstraction, specialization, and decomposition (Fawcett, 1996). Research on human-defined concept transformations dates back at least to the landmark AI program AM (Davis & Lenat, 1982). Our work uses only one means of generating candidate features: a beam search of logical formulas in increasing depth. This means of candidate generation has the advantage of strongly favoring concise and inexpensive features, but may miss more complex but very accurate/useful features. But our approach directly generalizes to these other means of generating candidate features. What most centrally distinguishes our approach from all previous work leveraging such feature transformations is the use of statewise Bellman error to score candidate features. FOALP (Sanner & Boutilier, 2006, 2009) uses no scoring function, but includes all non-pruned candidate features in the linear program used to find an approximately optimal value function; the Zenith system (Fawcett, 1996) uses a scoring function provided by an unspecified "critic."

5.3 Previous Scoring Functions for MDP Feature Selection

A method, from the work of Patrascu et al. (2002), selects features by estimating and minimizing the L_1 error of the value function that results from retraining the weights with the candidate feature included. L_1 error is used in that work instead of Bellman error because of the difficulty of retraining the weights to minimize Bellman error. Because our method focuses on fitting the Bellman error of the current approximation (without retraining with the new feature), it avoids this expensive retraining computation during search and is able to search a much larger feature space effectively. While the work of Patrascu et al. (2002) contains no discussion of relational representation, the L_1 scoring method could certainly be used with features represented in predicate logic; no work to date has tried this (potentially too expensive) approach.

5.4 Other Related Work

We include discussion of additional, more distantly related research directions as Appendix A, divided into the following subsections:

1. Other relevant feature selection methods (Fahlman & Lebiere, 1990; Utgoff & Precup, 1997, 1998; Rivest & Precup, 2003; Mahadevan & Maggioni, 2007; Petrik, 2007);

2. Structural model-based and model-free solution methods for Markov decision processes, including

   (a) Relational reinforcement learning (RRL) systems (Džeroski, DeRaedt, & Driessens, 2001; Driessens & Džeroski, 2004; Driessens et al., 2006),

   (b) Policy learning via boosting (Kersting & Driessens, 2008),

   (c) Fitted value iteration (Gordon, 1995), and

   (d) Exact value iteration methods in first-order MDPs (Boutilier, Reiter, & Price, 2001; Holldobler & Skvortsova, 2004; Kersting, Van Otterlo, & De Raedt, 2004);

3. Inductive logic programming algorithms (Muggleton, 1991; Quinlan, 1996; Karalic & Bratko, 1997);

4. Approximate policy iteration for relational domains (Fern et al., 2006), with a discussion on relational decision-list-policy learners (Khardon, 1999; Martin & Geffner, 2004; Yoon et al., 2002);

5. Automatic extraction of domain knowledge (Veloso, Carbonell, Perez, Borrajo, Fink, & Blythe, 1995; Kambhampati, Katukam, & Qu, 1996; Estlin & Mooney, 1997; Fox & Long, 1998; Gerevini & Schubert, 1998).

6. Experimental Setting

We present experiments in nine stochastic planning domains, including both reward-oriented and goal-oriented domains. We use Pentium 4 Xeon 2.8GHz machines with 3GB memory. In this section, we give a general overview of our experiments before giving detailed results and discussion for individual domains in Section 7. Here, first, we briefly discuss the selection of evaluation domains in Section 6.1. Second, in Section 6.2 we set up an evaluation of our relational feature learner by comparison to variants that replace key aspects of the algorithm with random choice to determine their importance. Additional details, including many experimental parameter settings, can be found in Online Appendix 1 (available on JAIR website) in Section 3.

6.1 Domains Considered

In all the evaluation domains below, it is necessary to specify a discount factor γ when modeling the domain as an MDP with discounting. The discount factor effectively specifies the tradeoff between the goals of reducing expected plan length and increasing success rate. γ is not a parameter of our method, but of the domain being studied, and our feature-learning method can be applied for any choice of γ. Here, for simplicity, we choose γ to be 0.95 throughout all our experiments. We note that this is the same discount factor used in the SYSADMIN domain formalization that we compare to from the previous work by Patrascu et al. (2002).

6.1.1 TETRIS

In Section 7.2 we evaluate the performance of both our relational and propositional learners using the stochastic computer-game TETRIS, a reward-oriented domain where the goal of a player is to maximize the accumulated reward. We compare our results to the performance of a set of hand-crafted features, and the performance of randomly selected features.

6.1.2 PLANNING COMPETITION DOMAINS

In Section 7.3, we evaluate the performance of our relational learner in seven goal-oriented planning domains from the two international probabilistic planning competitions (IPPCs) (Younes et al., 2005; Bonet & Givan, 2006). For comparison purposes, we evaluate the performance of our propositional learner on two of the seven domains (BLOCKSWORLD and a variant of BOXWORLD described below). Results from these two domains illustrate the difficulty of learning useful propositional features in complex planning domains. We also compare the results of our relational planner with two recent competition stochastic planners FF-Replan (Yoon et al., 2007) and FOALP (Sanner &

Boutilier, 2006, 2009) that have both performed well in the planning competitions. Finally, we compare our results to those obtained by randomly selecting relational features and tuning weights for them. For a complete description of, and PPDDL source for, the domains used, please see the work of Younes et al. (2005) and Bonet and Givan (2006).

Every goal-oriented domain with a problem generator from the first or second IPPC was considered for inclusion in our experiments. For inclusion, we require a planning domain with fixed action definitions, as defined in Section 2.4, that in addition has only ground conjunctive goal regions. Four domains have these properties directly, and we have adapted three more of the domains to have these properties:

1. In BOXWORLD, we modify the problem generator so that the goal region is always a ground conjunctive expression. We call the resulting domain CONJUNCTIVE-BOXWORLD.

2. In FILEWORLD, we construct the obvious lifted version, and create a problem generator restricted to three folders because in this domain the action definitions vary with the number of folders. We call the resulting domain LIFTED-FILEWORLD3.

3. In TOWERS OF HANOI, we create our own problem generator.

The resulting selection provides seven IPPC planning domains for our empirical study. We provide detailed discussions on the adapted domains in Section 2 of Online Appendix 1 (available on JAIR website), as well as discuss the reasons for the exclusion of domains.

6.1.3 SYSADMIN

We conclude our experiments by comparing our propositional learner with a previous method by Patrascu et al. (2002), using the the same SYSADMIN domain used for evaluation there. This empirical comparison on the SYSADMIN domain is shown in Section 7.4.

6.2 Randomized Variants of the Method

Our major contribution is the introduction and evaluation of a feature learning framework in the controlled setting based on scoring with Bellman-error (BE Scoring). Our empirical work instantiates this framework with a relational feature-learning algorithm of our design based on greedy beam-search. Here, we compare the performance of this instance of our framework with variants that replace key aspects with randomized choice, illustrating the relative importance of those features. In the two random-choice experiments, we adapt our method in one of the following two ways:

1. Labeling the training states with random scores instead of Bellman Error scores. The target value in our feature training set is a random number from -1 to 1. This algorithm is called "*Random Scoring*."

2. Narrowing the beam during search randomly rather than greedily. We eliminate scoring during the beam search, instead using random selection to narrow the beam; only at the end of the beam search is scoring used to select the best resulting candidate. This algorithm is called "*Random Beam Narrowing*."

The original algorithm, which labels training data with Bellman error and narrows the beam greedily rather than randomly, is called "*Greedy Beam Search/BE Scoring*" in our plots. For these comparisons, we only consider the relational feature representation, as that is where our beam search method is used. Experiments with the two variants introduced here, presented below in Sections 7.2.4 and 7.3.4, show that our original method selects features that perform much better than randomly selected features, and that the greediness in the beam search is often (but not always) important in achieving good performance.

7. Experimental Results

We present experimental results for TETRIS, planning competition domains, and SYSADMIN in this section, starting with an introduction on the structure of our result presentation.

7.1 How to Read Our Results

The task of evaluating a feature-learning planning system is subtle and complex. This is particularly a factor in the relational case because generalization between problem sizes and learning from small problems must be evaluated. The resulting data is extensive and highly structured, requiring some training of the reader to understand and interpret. Here we introduce to the reader the structure of our results.

In experiments with the propositional learning (or with randomly selected propositional features), the problem size never varies within one run of the learner, because the propositional representation from Section 4.3 cannot generalize between sizes. We run a separate experiment for each size considered. Each experiment is two independent trials; each trial starts with a single trivial feature and repeatedly adds features until a termination condition is met. After each feature addition, AVI is used to select the weights for combining the features to form a value function, and the performance of that value function is measured (by sampling the performance of the greedy policy). We then compute the average (of the two trials) of the performance as a function of the number of features used. Since this results in a single line plot of performance as a function of number of features, several different fixed-problem-size learners can be compared on one figure, with one line for each, as is done for example in Figures 7 and 14. The performance measure used varies appropriately with the domain as presented below.

We study the ability of relational representation from Section 4.1 to generalize between sizes. This study can only be properly understood against the backdrop of the flowchart in Figure 1. As described in this flowchart, one trial of the learner will learn a sequence of features and encounter a sequence of increasing problem difficulties. One iteration of the learner will *either* add a new feature *or* increase the problem difficulty (depending on the current performance). In either case, the weights are then retrained by AVI and a performance measurement of the resulting greedy policy is taken. Because different trials may increase the size at different points, we cannot meaningfully average the measurements from two trials. Instead, we present two independent trials separately in two tables, such as the Figures 5 and 12. For the first trial, we also present the same data a second time as a line plot showing performance as a function of number of features, where problem size changes are annotated along the line, such as the plots in Figures 6 and 13. Note that success ratio generally increases along the line when features are added, but falls when problem size is increased. (In TETRIS, however, we measure "rows erased" rather than success ratio, and "rows erased" generally increases with either the addition of a new feature or the addition of new rows to the available grid.)

To interpret the tables showing trials of the relational learner, it is useful to focus on the first two rows, labeled "# of features" and "Problem difficulty." These rows, taken together, show the progress of the learner in adding features and and increasing problem size. Each column in the table represents the result in the indicated problem size using the indicated number of learned features. From one column to the next, there will be a change in only one of these rows—if the performance of the policy shown in a column is high enough, it will be the problem difficulty that increases, and otherwise it will be the number of features that increases. Further adding to the subtlety in interpreting these tables, we note that when several adjacent columns increase the number of features, we sometimes splice out all but two of these columns to save space. Thus, if several features are added consecutively at one problem size, with slowly increasing performance, we may show only the first and last of these columns at that problem size, with a consequent jump in the number of features between these columns. We likewise sometimes splice out columns when several consecutive columns increase problem difficulty. We have found that these splicings not only save space but increase readability after some practice reading these tables.

Performance numbers shown in each column (success ratio and average plan length, or number of rows erased, for TETRIS) refer to the performance of the weight-tuned policy resulting for that feature set at that problem difficulty. We also show in each column the performance of that value function (without re-tuning weights) on the target problem size. Thus, we show quality measures for each policy found during feature learning on both the current problem size at that point and on the target problem size, to illustrate the progress of learning from small problems on the target size via generalization.

We do not study here the problem of deciding when to stop adding features. Instead, in both propositional and relational experiments, trials are stopped by experimenter judgment when additional results are too expensive for the value they are giving in evaluating the algorithm. However, we do not stop any trials when they are still improving unless unacceptable resource consumption has occurred.

Also, in each trial, the accumulated real time for the trial is measured and shown at each point during the trial. We use real time rather than CPU time to reflect non-CPU costs such as paging due to high memory usage.

7.2 Tetris

We now present experimental results for TETRIS.

7.2.1 OVERVIEW OF TETRIS

The game TETRIS is played in a rectangular board area, usually of size 10×20, that is initially empty. The program selects one of the seven shapes uniformly at random and the player rotates and drops the selected piece from the entry side of the board, which piles onto any remaining fragments of the pieces that were placed previously. In our implementation, whenever a full row of squares is occupied by fragments of pieces, that row is removed from the board and fragments on top of the removed row are moved down one row; a reward is also received when a row is removed. The process of selecting locations and rotations for randomly drawn pieces continues until the board is "full" and the new piece cannot be placed anywhere in the board. TETRIS is stochastic since the next piece to place is always randomly drawn, but this is the only stochastic element in this game. TETRIS is also used as an experimental domain in previous MDP and reinforcement learning research (Bertsekas & Tsitsiklis, 1996; Driessens et al., 2006). A set of human-selected features is described in the book by Bertsekas and Tsitsiklis (1996) that yields very good performance when used in weighted linearly approximated value functions. We cannot fairly compare our performance in this domain to probabilistic planners requiring PPDDL input because we have found no natural PPDDL definition for TETRIS.

Our performance metric for TETRIS is the number of rows erased averaged over 10,000 trial games. The reward-scaling parameter r_{scale} (defined in Section 5 in Online Appendix 1 on page 8) is selected to be 1.

7.2.2 TETRIS RELATIONAL FEATURE LEARNING RESULTS

We represent the TETRIS grid using rows and columns as objects. We use three primitive predicates: **fill**(c, r), meaning that the square on column c, row r is occupied; **below**(r_1, r_2), meaning that row r_1 is directly below row r_2; and **beside**(c_1, c_2), meaning that column c_1 is directly to the left of column c_2. The quantifiers used in our relational TETRIS hypothesis space are typed using the types "row" and "column".

There are also state predicates representing the piece about to drop; however, for efficiency reasons our planner computes state value as a function only of the grid, not the next piece. This limitation in value-function expressiveness allows a significantly cheaper Bellman-backup computation. The one-step lookahead in greedy policy execution provides implicit reasoning about the piece being dropped, as that piece will be in the grid in all the next states.

We conduct our relational TETRIS experiments on a 10-column, n-row board, with n initially set to 5 rows. Our threshold for increasing problem difficulty by adding one row is a score of at least $15 + 20(n - 5)$ rows erased. The target problem size for these experiments is 20 rows. The results for the relational TETRIS experiments are given in Figures 5 and 6 and are discussed below.

7.2.3 TETRIS PROPOSITIONAL FEATURE LEARNING RESULTS

For the propositional learner, we describe the TETRIS state with 7 binary attributes that represent which of the 7 pieces is currently being dropped, along with one additional binary attribute for each grid square representing whether that square is occupied. The adjacency relationships between the grid squares are represented only through the procedurally coded action dynamics. Note that the number of state attributes depends on the size of the TETRIS grid, and learned features will only apply to problems of the same grid size. As a result, we show separate results for selected problem sizes.

We evaluate propositional feature learning in 10-column TETRIS grids of four different sizes: 5 rows, 7 rows, 9 rows, and 20 rows. Results from these four trials are shown together in Figure 7 and the average accumulated time required to reach each point on Figure 7 is shown in Figure 8. These results are discussed below.

7.2.4 EVALUATING THE IMPORTANCE OF BELLMAN-ERROR SCORING AND GREEDY BEAM-SEARCH IN TETRIS

Figure 9 compares our original algorithm with alternatives that vary from it on either training set scoring or greediness of beam search, as discussed in Section 6.2. For the two alternatives, we use

Trial #1															
# of features	0	1	2	3	11	11	12	17	17	18	18	18	18	18	18
Problem difficulty	5	5	5	5	5	6	6	6	7	7	8	9	10	15	20
Score	0.2	0.5	1.0	3.0	18	31	32	35	55	56	80	102	121	234	316
Accumulated time (Hr.)	0.0	2	4.2	5.2	20	21	24	39	42	46	50	57	65	111	178
Target size score	0.3	1.3	1.4	1.8	178	238	261	176	198	211	217	221	220	268	317

Trial #2																				
# of features	0	1	8	8	12	12	14	14	15	15	16	16	17	26	26	27	27	28	29	33
Problem difficulty	5	5	5	6	6	7	7	11	11	12	12	13	13	13	14	14	17	17	17	17
Score	0.2	0.6	16	28	36	53	56	133	136	151	156	167	168	175	192	210	238	251	240	241
Accumulated time (Hr.)	0.0	2.4	15	15	27	29	39	66	76	87	97	103	110	211	220	236	276	295	318	408
Target size score	0.3	1.7	104	113	108	116	130	192	196	199	206	211	211	219	225	218	231	231	233	231

Figure 5: TETRIS performance (averaged over 10,000 games). Score is shown in average rows erased, and problem difficulty is shown in the number of rows on the TETRIS board. The number of columns is always 10. Difficulty increases when the average score is greater than 15+20*(n-5), where n is the number of rows in the TETRIS board. Target problem size is 20 rows. Some columns are omitted as discussed in Section 7.1.

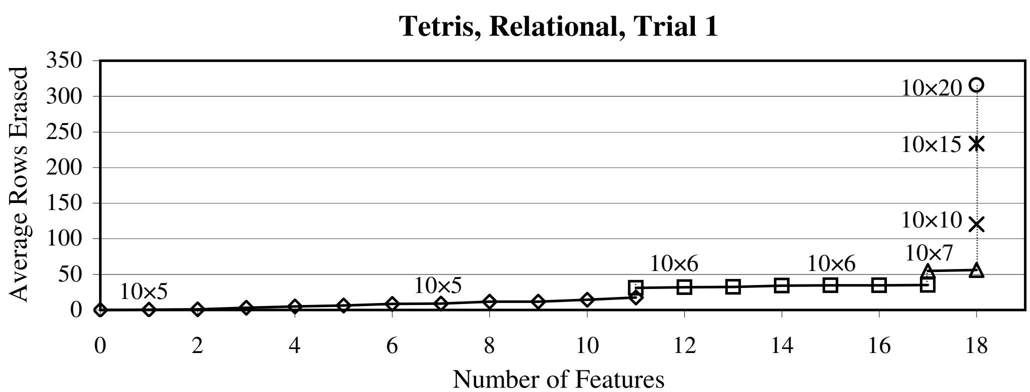


Figure 6: Plot of the average number of lines erased over 10,000 TETRIS games after each run of AVI training during the learning of relational features (trial 1). Vertical lines indicate difficulty increases (in the number of rows), as labeled along the plot.

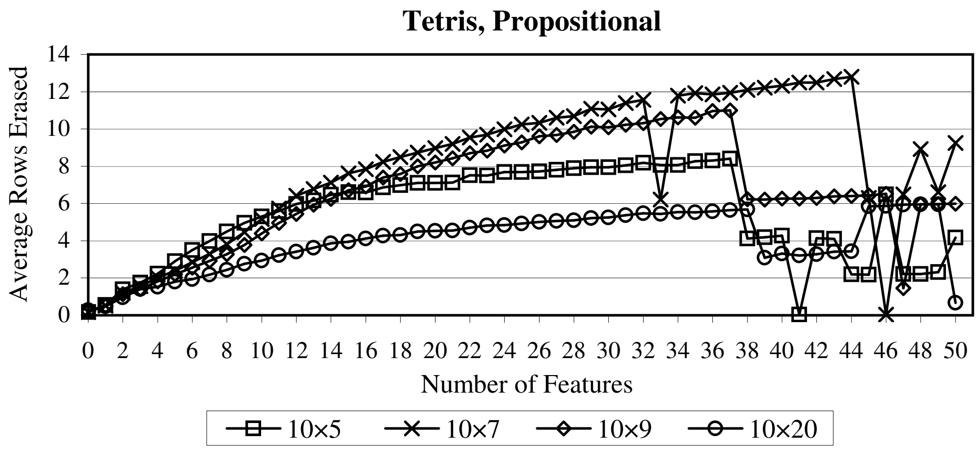


Figure 7: Plot of the average number of lines erased in 10,000 TETRIS games after each iteration of AVI training during the learning of propositional features, averaged over two trials.

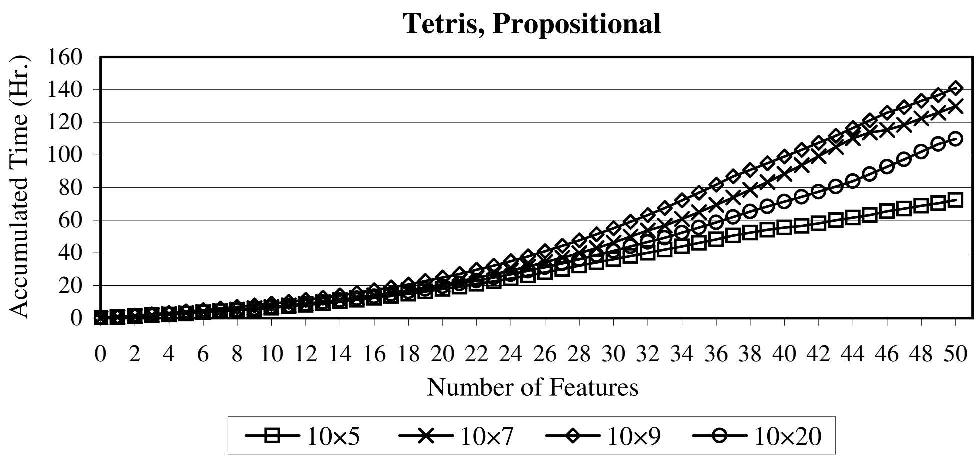


Figure 8: Plot of the accumulated time required to reach each point in Figure 7, averaged over two trials.

the same schedule used for the original Greedy Beam Search/BE Scoring algorithm in TETRIS by starting with the 10×5 problem size. However, the performance of these two alternatives is never good enough to increase the problem size.

7.2.5 EVALUATING HUMAN-DESIGNED FEATURES IN TETRIS

In addition to evaluating our relational and propositional feature learning approach, we also evaluate how the human-selected features described in the book by Bertsekas and Tsitsiklis (1996) perform in selected problem sizes. For each problem size, we start from all weights zero and use our AVI

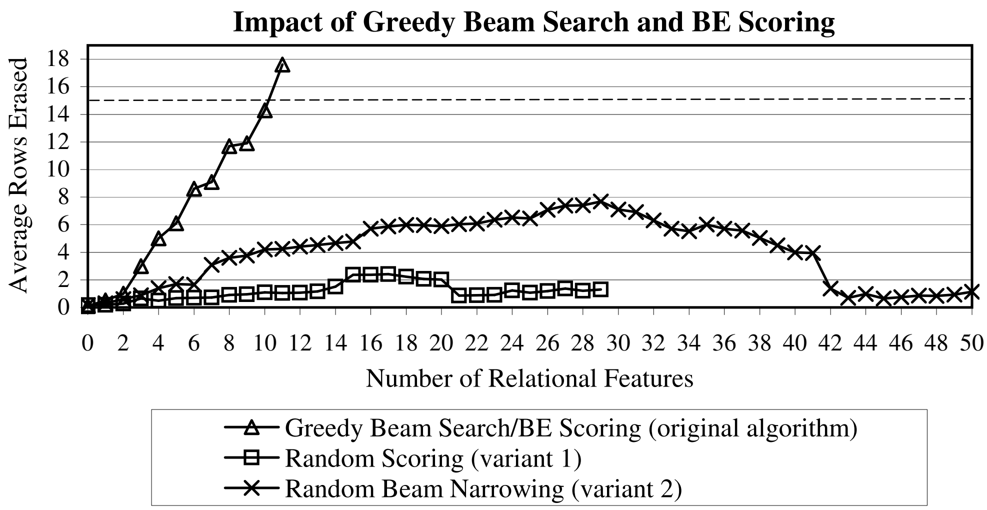


Figure 9: Plot of the average number of lines erased in 10,000 TETRIS games for relational features learned from the original algorithm and the two alternatives as discussed in Section 6.2. For Random Scoring and Random Beam Narrowing, the results are averages over two independent trials. Trials of these two variants are terminated when they fail to make progress for several feature additions. For comparison purposes, trial one of the original Greedy Beam Search/BE Scoring method is shown, reaching the threshold for difficulty increase after eleven feature additions (trial two did even better).

	10×5	10×7	10×9	10×20
Average rows erased, Trial 1	19	86	267	17,954
Average rows erased, Trial 2	19	86	266	18,125

Figure 10: The average number of lines erased in 10,000 TETRIS games for the best weighted combination of human features found in each of two trials of AVI and four problem sizes.

process described in Section 2.5 to train the weights for all 21 features until the performance appears to converge. We change the learning rate α from $\frac{3}{1+k/100}$ to $\frac{30}{1+k/100}$ as human-designed features require a larger step-size to converge rapidly. The human-designed features are normalized to a value between 0 and 1 here in our experiments. We run two independent trials for each problem size and report the performance of the best-performing weight vector found in each trial, in Figure 10.

7.2.6 PERFORMANCE COMPARISON BETWEEN DIFFERENT APPROACHES TO TETRIS

Several general trends emerge from the results on TETRIS. First of all, the addition of new learned features is almost always increasing the performance of the resulting tuned policy (on the current size and on the target size), until a best performance point is reached. This suggests we are in fact

	Relational	Prop. 10×5	Prop. 10×7	Prop. 10×9	Prop. 10×20
Average feature learning time (Min.)	167	44	52	60	44

Figure 11: Table for the average feature learning time for relational and propositional approaches.

selecting useful features. We also find clear evidence of the ability of the relational representation to usefully generalize between problem sizes: substantial performance is developed on the target problem size without ever training directly in that size.

We find that the best performance of learned propositional features is much lower than that of learned relational features in all problem sizes shown here, even though a larger feature training set size and many more learned features are used for the propositional approach. This suggests that the rich relational representation indeed is able to better capture the dynamics in TETRIS than the propositional representation.

We find that the performance of using random features in TETRIS is significantly worse than that of using learned features, demonstrating that our performance improvements in feature learning are due to useful feature selection (using Bellman error), not simply due to increasing the number of features.

Our learned relational feature performance in 10×20 TETRIS is far worse than that obtained by using the human-selected features with AVI in the same size. However, in 10×5 TETRIS the relational feature performance is close to that of the human-designed features. The human-designed features are engineered to perform well in the 10×20 TETRIS hence some concepts that are useful in performing well in smaller problem sizes may not exist in these features.

7.2.7 TIME TO LEARN EACH FEATURE

In Figure 11 we show the average time required to learn a relational feature or a propositional feature in TETRIS.

The time required to learn a relational feature is significantly longer than that required to learn a propositional feature, even though for the propositional approach a larger feature training set size is being used.

7.2.8 COMPARISON TO PREVIOUS TETRIS-SPECIFIC LEARNERS

In evaluating domain-independent techniques on TETRIS, we must first put aside the strong performance already shown many times in the literature for domain-dependent techniques on that domain. Then, we must face the problem that there are no published domain-independent comparison points in order to define a state-of-the-art target to surpass. For the latter problem, we provide a baseline from two different approaches to random feature selection, and show that our targeted feature selection dramatically improves on random selection. For the former problem, we include below a discussion of the domain-specific elements of key previous published results on TETRIS.

There have been many previous *domain-specific* efforts at learning to play TETRIS (Bertsekas & Tsitsiklis, 1996; Szita & Lorincz, 2006; Lagoudakis, Parr, & Littman, 2002; Farias & Van Roy, 2004; Kakade, 2001). Typically, these provide human-crafted domain-dependent features, and deploy domain-independent machine learning techniques to combine these features (often by tuning weights for a linear combination). As an example, a domain-specific feature counting the number of covered up "holes" in the board is frequently used. This feature is plausibly derived by human reasoning about the rules of the game, such as realizing that such holes are difficult to fill by later action and can lead to low scores. In all prior work, the selection of this feature is by hand, not by an automated feature-selection process (such as our scoring of correlation to Bellman error). Other frequently used domain-specific features include "column height" and "difference in height of adjacent columns", again apparently selected as relevant by human reasoning about the rules of the game.

The key research question we address, then, is whether useful features can be derived automatically, so that a decision-making situation like TETRIS can be approached by a domain-independent system without human intervention. Our method is provided only a domain-state representation using primitive horizontal and vertical positional predicates, and a single constant feature. To our knowledge, before this research there is no published evaluation on TETRIS that does not rely on domain-specific human inputs such as those just discussed. As expected, our performance on TETRIS is much weaker than that achieved by domain-specific systems such as those just cited.

7.3 Probabilistic Planning Competition Domains

Throughout the evaluations of our learners in planning domains, we use a lower plan-length cutoff of 1000 steps when evaluating success ratio during the iterative learning of features, to speed learning. We use a longer cutoff of 2000 steps for the final evaluation of policies for comparison with other planners and for all evaluations on the target problem size. The reward-scaling parameter r_{scale} (defined in Section 5 in Online Appendix 1 on page 8) is selected to be 1 throughout the planning domains.

For domains with multi-dimensional problem sizes, it remains an open research problem on how to change problem size in different dimensions automatically to increase difficulty during learning. Here, in CONJUNCTIVE-BOXWORLD and ZENOTRAVEL, we hand-design the sequence of increasing problem sizes.

As discussed in Section 6.1.2, we evaluate our feature learners in a total of seven probabilistic planning competition domains. In the following paragraphs, we provide a full discussion of BLOCKSWORLD and CONJUNCTIVE-BOXWORLD, with abbreviated results for the other five domains. We provide a full discussion of the other five domains in Appendix B.

Our relational feature learner finds useful value-function features in four of these domains (BLOCKSWORLD, CONJUNCTIVE-BOXWORLD, TIREWORLD, and LIFTED-FILEWORLD3). In the other three domains (ZENOTRAVEL, EXPLODING BLOCKSWORLD, and TOWERS OF HANOI), our relational feature learner makes progress in representing a useful fixed-size value function for the training sizes, but fails to find features that generalize well to problems of larger sizes.

7.3.1 BLOCKSWORLD

In the probabilistic, non-reward version of BLOCKSWORLD from the first IPPC, the actions **pickup** and **putdown** have a small probability of placing the handled block on the table object instead of on the selected destination.

For our relational learner, we start with 3 blocks problems. We increase from n blocks to $n + 1$ blocks whenever the success ratio exceeds 0.9 and the average successful plan length is less than $30(n - 2)$. The target problem size is 20 blocks. Results are shown in Figures 12 and 13.

Trial #1								
# of features	0	1	2	2	3	3	3	3
Problem difficulty	3	3	3	4	4	5	10	15
Success ratio	1.00	1	1	0.95	1	1	1	0.97
Plan length	89	45	20	133	19	33	173	395
Accumulated time (Hr.)	0.5	1.0	1.5	2.2	3.3	3.9	10	36
Target size SR	0	0	0	0	0.98	0.96	0.98	0.97
Target size Slen.	–	–	–	–	761	724	754	745

Trial #2								
# of features	0	1	2	2	3	3	3	3
Problem difficulty	3	3	3	4	4	5	10	15
Success ratio	1	1	1	0.94	1	1	1	0.96
Plan length	80	48	19	125	17	34	167	386
Accumulated time (Hr.)	0.5	1.0	1.4	2.0	3.3	3.8	9.4	33
Target size SR	0	0	0	0	0.97	0.98	0.98	0.98
Target size Slen.	–	–	–	–	768	750	770	741

Figure 12: BLOCKSWORLD performance (averaged over 600 problems) for relational learner. We add one feature per column until success ratio exceeds 0.9 and average successful plan length is less than $30(n-2)$, for n blocks, and then increase problem difficulty for the next column. Plan lengths shown are successful trials only. Problem difficulties are measured in number of blocks, with a target problem size of 20 blocks. Some columns are omitted as discussed in Section 7.1.

For our propositional learner, results for problem sizes of 3, 4, 5, and 10 blocks are shown in Figure 14.

Our relational learner consistently finds value functions with perfect or near-perfect success ratio up to 15 blocks. This performance compares very favorably to the recent RRL (Driessens et al., 2006) results in the deterministic BLOCKSWORLD, where goals are severely restricted to, for instance, single **ON** atoms, and the success ratio performance of around 0.9 for three to ten blocks (for the single **ON** goal) is still lower than that achieved here. Our results in BLOCKSWORLD show the average plan length is far from optimal. We have observed large plateaus in the induced value function: state regions where all states are given the same value so that the greedy policy wanders. This is a problem that merits further study to understand why feature-induction does not break such plateaus. Separately, we have studied the ability of local search to break out of such plateaus (Wu, Kalyanam, & Givan, 2008).

The performance on the target size clearly demonstrates successful generalization between sizes for the relational representation.

The propositional results demonstrate the limitations of the propositional learner regarding lack of generalization between sizes. While very good value functions can be induced for the small problem sizes (3 and 4 blocks), slightly larger sizes of 5 or 10 blocks render the method ineffective. In 10 block problems, the initial random greedy policy cannot be improved because it never finds

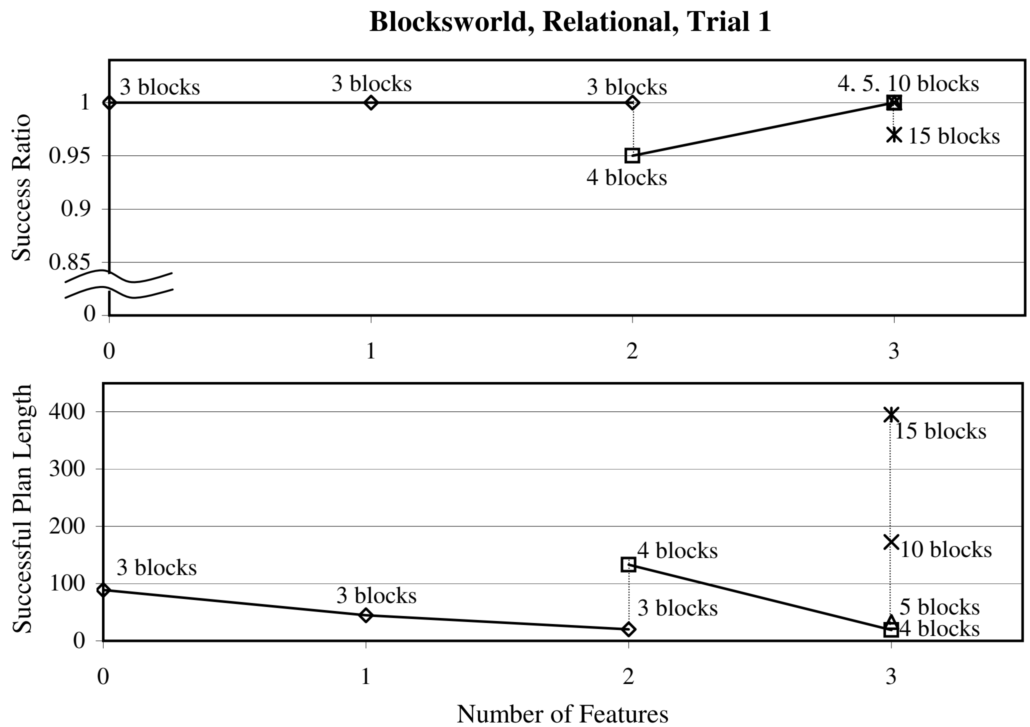


Figure 13: BLOCKSWORLD success ratio and average successful plan length (averaged over 600 problems) for the first trial from Figure 12 using our relational learner.

the goal. In addition, these results demonstrate that learning additional features once a good policy is found can degrade performance, possibly because AVI performs worse in the higher dimensional weight space that results.

7.3.2 CONJUNCTIVE-BOXWORLD

The probabilistic, non-reward version of BOXWORLD from the first IPPC is similar to the more familiar Logistics domain used in deterministic planning competitions, except that an explicit connectivity graph for the cities is defined. In Logistics, airports and aircraft play an important role since it is not possible to move trucks from one airport (and the locations adjacent to it) to another airport (and the locations adjacent to it). In BOXWORLD, it is possible to move all the boxes without using the aircraft since the cities may all be connected with truck routes. The stochastic element introduced into this domain is that when a truck is being moved from one city to another, there is a small chance of ending up in an unintended city. As described in Section 6.1, we use CONJUNCTIVE-BOXWORLD, a modified version of BOXWORLD, in our experiments.

We start with 1-box problems in our relational learner and increase from n boxes to $n+1$ boxes whenever the success ratio exceeds 0.9 and the average successful plan length is better than $30n$. All feature-learning problem difficulties use 5 cities. We use two target problem sizes: 15 boxes

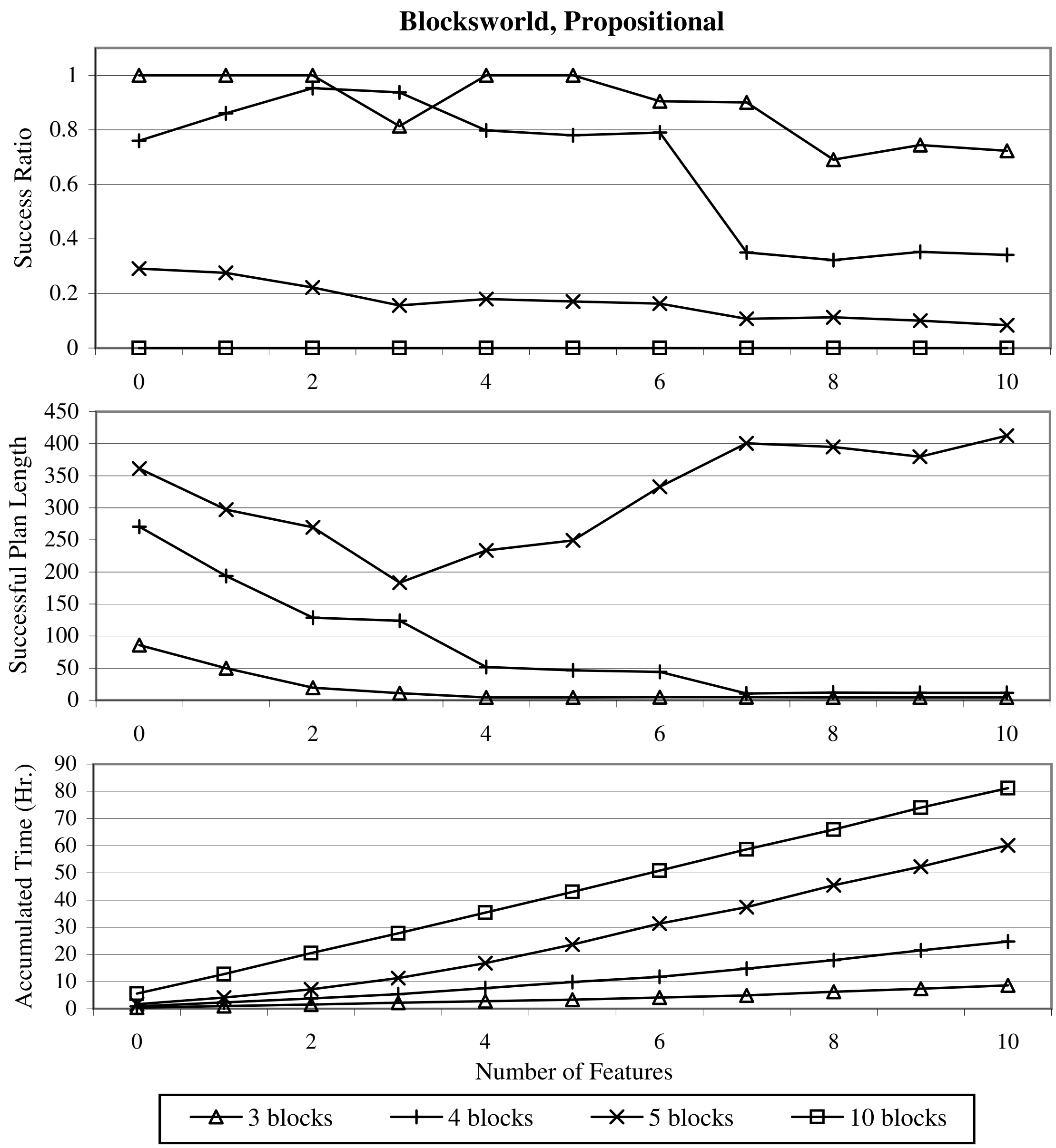


Figure 14: BLOCKSWORLD performance success ratio and average successful plan length (averaged over 600 problems), and accumulated run-time for our propositional learner, averaged over two trials.

Trial #1											
# of features	0	1	2	2	2	2	2	2	2	2	2
Problem difficulty	1	1	1	2	3	5	10	11	12	13	15
Success ratio	0.97	1	1	1	1	1	1	1	1	1	1
Plan length	226	84	23	37	44	54	77	80	313	87	92
Accumulated time (Hr.)	7.2	10	13	14	16	21	42	49	57	65	84
Target size #1 SR	0.98	1	1	1	1	1	1	1	1	1	1
Target size #1 Slen.	1056	359	93	91	90	92	90	92	355	90	91
Target size #2 SR	0.16	0.90	0.97	0.97	0.96	0.98	0.96	0.98	0.90	0.98	0.96
Target size #2 Slen.	1583	996	238	230	233	244	240	238	1024	240	239

Trial #2												
# of features	0	1	2	2	2	2	2	2	2	2	2	2
Problem difficulty	1	1	1	2	3	5	9	10	11	12	13	15
Success ratio	0.97	1	1	1	1	1	1	1	1	1.00	1	1
Plan length	235	85	24	34	43	54	72	299	80	310	84	91
Accumulated time (Hr.)	7.3	11	14	16	18	23	39	45	51	60	68	86
Target size #1 SR	0.96	1	1	1	1	1	1	1	1	1	1	1
Target size #1 Slen.	1019	365	90	91	91	92	89	359	89	363	90	90
Target size #2 SR	0.19	0.9	0.97	0.97	0.98	0.98	0.97	0.92	0.98	0.91	0.97	0.96
Target size #2 Slen.	1574	982	226	230	233	233	242	1006	231	1026	240	233

Figure 15: CONJUNCTIVE-BOXWORLD performance (averaged over 600 problems). We add one feature per column until success ratio is greater than 0.9 and average successful plan length is less than $30n$, for n boxes, and then increase problem difficulty for the next column. Problem difficulty is shown in number of boxes. Throughout the learning process the number of cities is 5. Plan lengths shown are successful trials only. Two target problem sizes are used. Target problem size #1 has 15 boxes and 5 cities. Target problem size #2 has 10 boxes and 10 cities. Some columns are omitted as discussed in Section 7.1.

and 5 cities, and 10 boxes and 10 cities. Relational learning results are shown in Figures 15 and 16, and results for the propositional learner on 5 cities with 1, 2, or 3 boxes are shown in Figures 17.

In interpreting the CONJUNCTIVE-BOXWORLD results, it is important to focus on the average successful plan-length metric. In CONJUNCTIVE-BOXWORLD problems, random walk is able to solve the problem nearly always, but often with very long plans[10]. The learned features enable more direct solutions as reflected in the average plan-length metric.

Only two relational features are required for significantly improved performance in the problems we have tested. Unlike the other domains we evaluate, for the CONJUNCTIVE-BOXWORLD domain

10. We note that, oddly, the IPPC competition domain used here has action preconditions prohibiting moving a box away from its destination. These preconditions bias the random walk automatically towards the goal. For consistency with the competition results, we retain these odd preconditions, although these preconditions are not necessary for good performance for our algorithm.

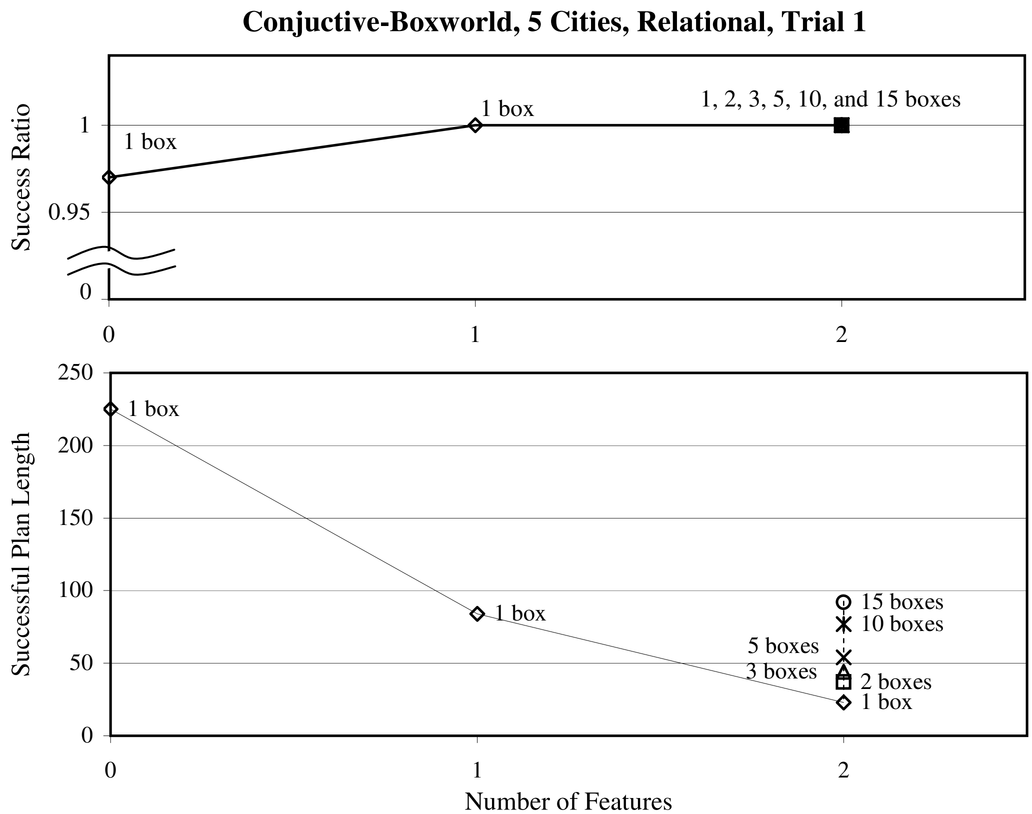


Figure 16: CONJUNCTIVE-BOXWORLD success ratio and average successful plan length (averaged over 600 problems) for the first trial using our relational learner.

the learned features are straightforwardly describable in English. The first feature counts how many boxes are correctly at their target city. The second feature counts how many boxes are on trucks.

We note the lack of any features rewarding trucks for being in the "right" place (resulting in longer plan lengths due to wandering on value-function plateaus). Such features can easily be written in our knowledge representation (e.g. count the trucks located at cities that are the destinations for some package on the truck), but require quantification over both cities and packages. The severe limitation on quantification currently in our method for efficiency reasons prevents consideration of these features at this point. It is also worth noting that regression-based feature discovery, as studied in the work of Gretton and Thiébaux (2004) and Sanner and Boutilier (2009), can be expected to identify such features regarding trucks by regressing the goal through the action of unloading a package at the destination. Combining our Bellman-error-based method with regression-based methods is a promising future direction.

Nevertheless, our relational learner discovers two concise and useful features that dramatically reduce plan length relative to the initial policy of random walk. This is a significant success for automated domain-independent induction of problem features.

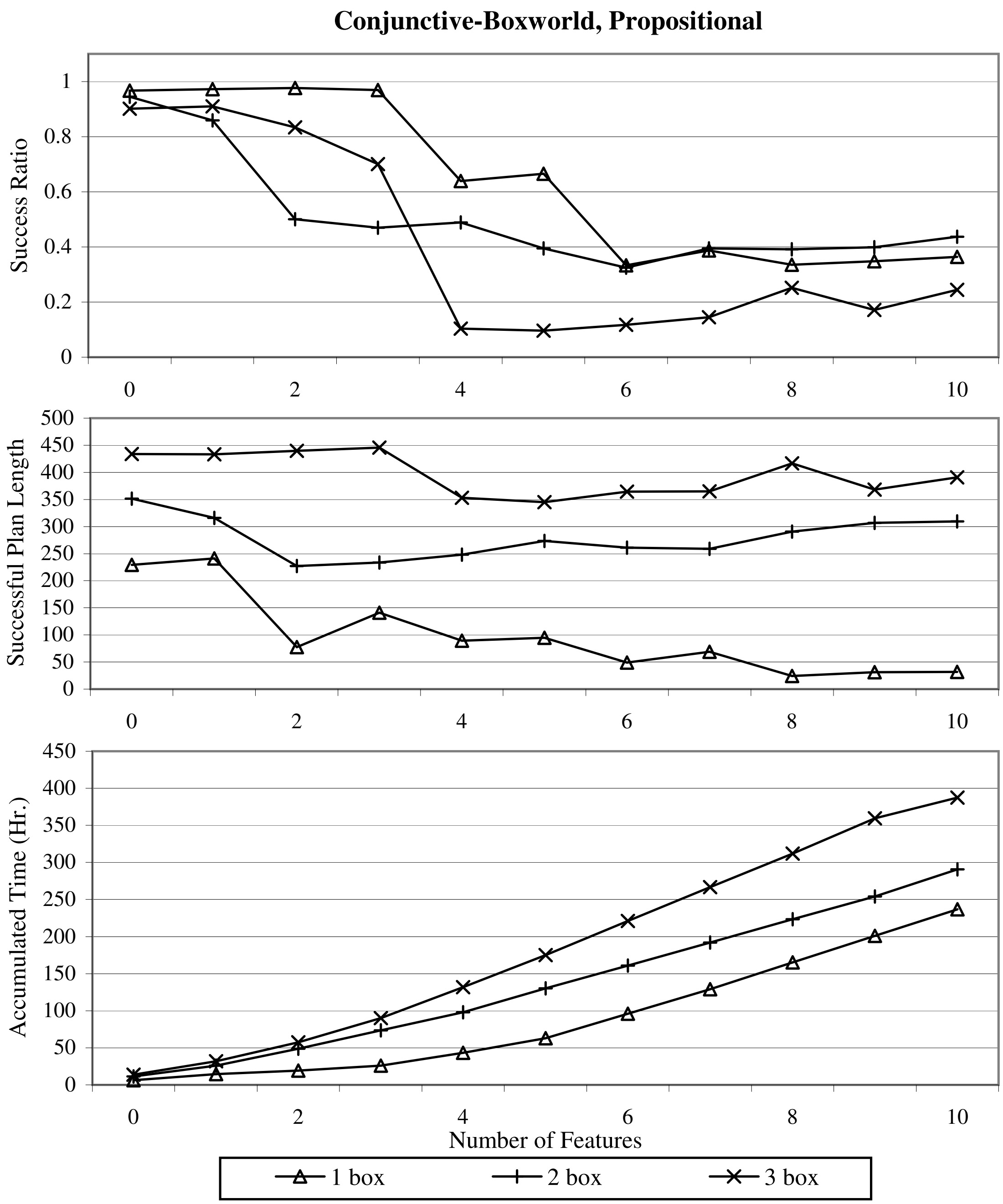


Figure 17: CONJUNCTIVE-BOXWORLD performance (averaged over 600 problems) and accumulated run-time for propositional learner, averaged over two trials. Throughout the learning process the number of cities is 5.

One trial of the relational feature learner in CONJUNCTIVE-BOXWORLD takes several days, even though we have fixed the number of cities for the training problems at five cities. New techniques are required for improving the efficiency of feature learning before we can provide results for training in larger numbers of cities. Our results here demonstrate that the current representation and learning methods adequately manage small city graphs even with larger and larger numbers of boxes to deliver, and that the resulting value functions successfully generalize to 10-city problems.

In this domain, a well known weakness of AVI is apparent. While AVI often works in practice, there is no theoretical guarantee on the quality of the weight vector found by AVI training. (Alternatively, an approximate linear programming step could replace AVI training to provide a more expensive but perhaps more robust weight selection.) In the CONJUNCTIVE-BOXWORLD results, AVI training goes astray when selecting weights in the 12 box domain size in Trial 1. As a result, the selected weights overemphasize the first feature, neglecting the second feature. This is revealed in the data shown because the plan-length performance degrades significantly for that one column of data. When AVI is repeated at the next problem size (13 boxes), good performance is restored. A similar one-column degradation of plan length occurs in trial 2 at the 10-box and 12-box sizes.

For our propositional experiments in the CONJUNCTIVE-BOXWORLD, we note that, generally, adding learned propositional features degrades the success-rate performance relative to the initial random walk policy by introducing ineffective loops into the greedy policy. The resulting greedy policies find the goal in fewer steps than random walk, but generally pay an unacceptable drop in success ratio to do so. The one exception is the policy found for one-box problems using just two propositional features, which significantly reduces plan length while preserving success ratio. Still, this result is much weaker than that for our relational feature language.

These problems get more severe as problem size increases, with 3-box problems suffering severe degradation in success rate with only modest gains in successful plan length. Also please note that accumulated runtime for these experiments is very large, especially for 3-box problems. AVI training is very expensive for policies that do not find the goal. Computing the greedy policy at each state in a long trajectory requires considering each action, and the number of available actions can be quite large in this domain. For these reasons, the propositional technique is not evaluate at sizes larger than three boxes.

7.3.3 SUMMARY RESULTS FROM ADDITIONAL DOMAINS

In Figures 18 to 20, we present summary results from five additional probabilistic planning domains. For detailed results and full discussion of these domains, please see Appendix B. From the summary results, we can see that our feature learning approach successfully finds features that perform well across increasing problem sizes in two of these five domains, TIREWORLD and LIFTED-FILEWORLD3. In the other three domains (ZENOTRAVEL, TOWERS OF HANOI, and EXPLODING BLOCKSWORLD), feature learning is able to make varying degrees of progress on fixed small problem sizes, but that progress (sometimes quite limited) does not generalize well as size increases.

7.3.4 EVALUATING THE RELATIVE IMPORTANCE OF BELLMAN-ERROR SCORING AND GREEDY BEAM-SEARCH IN GOAL-ORIENTED DOMAINS

Figure 21 compares our original algorithm with alternatives that vary from it on either training set scoring or greediness of beam search, as discussed in Section 6.2. For each trial of each variant, we generate a greedy policy for each domain using feature selection within our relational representation

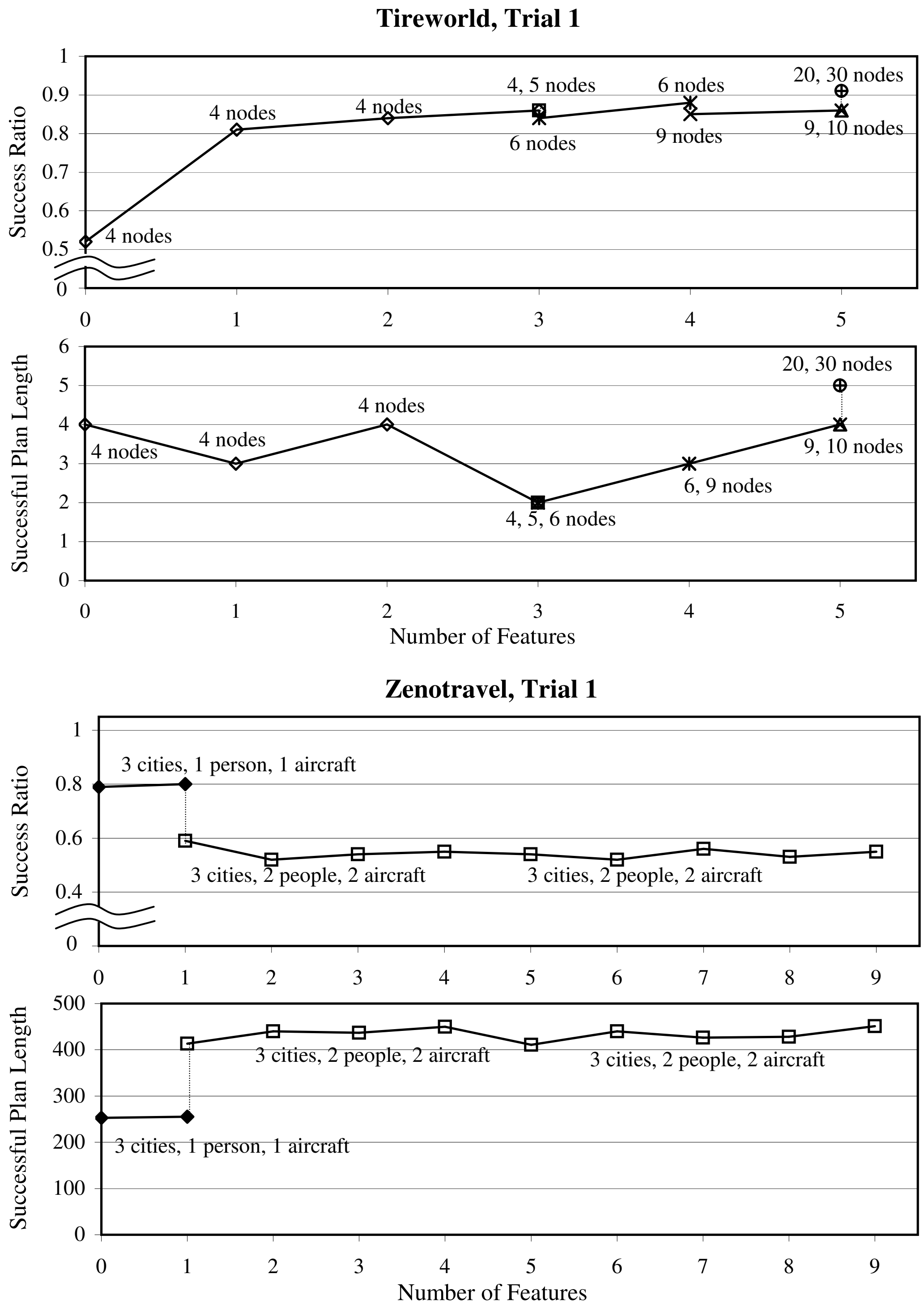


Figure 18: Summary results for TIREWORLD and ZENOTRAVEL. For full discussion and detailed results, please see Appendix B.

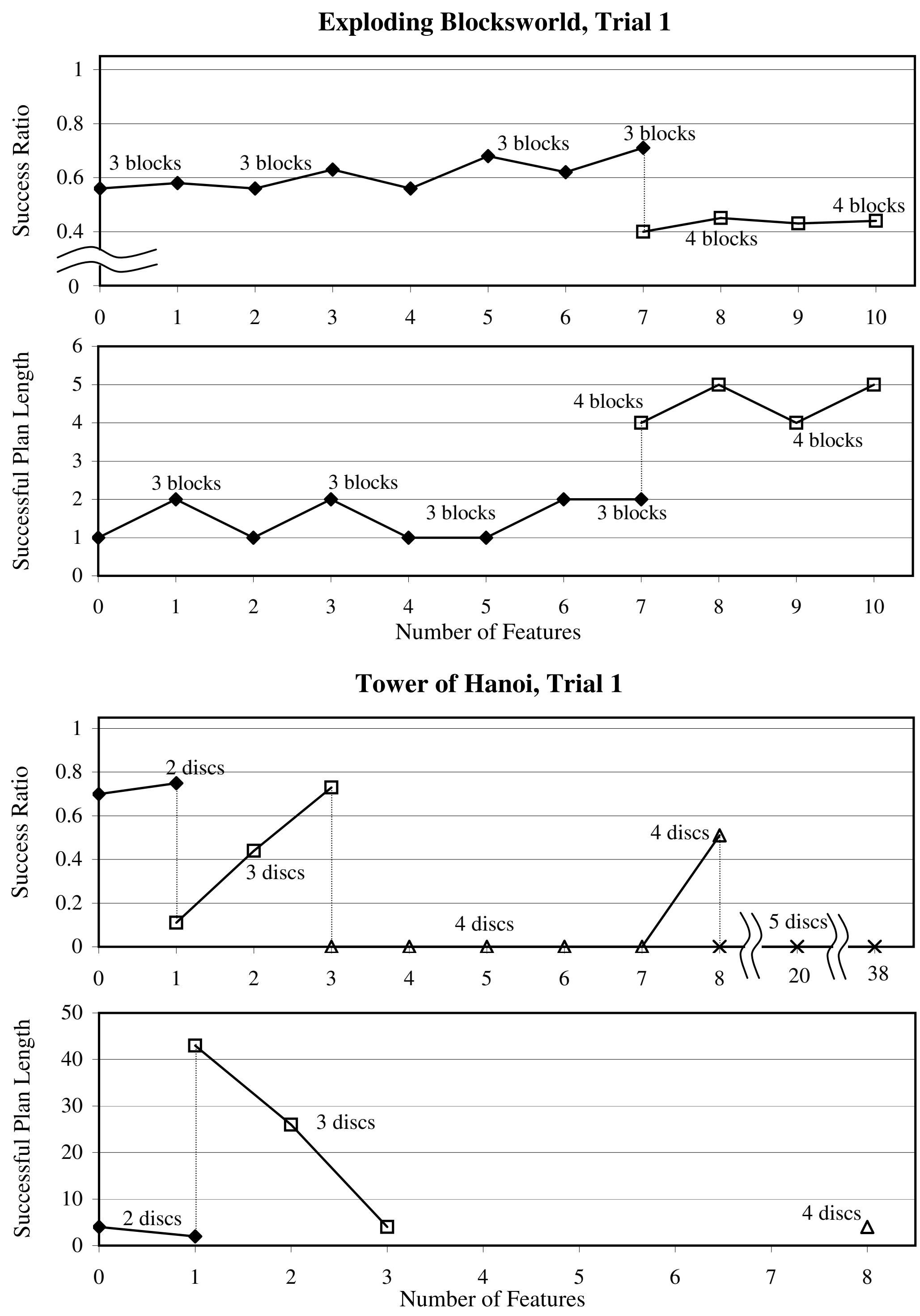


Figure 19: Summary results for EXPLODING BLOCKSWORLD and TOWERS OF HANOI. For full discussion and detailed results, please see Appendix B.

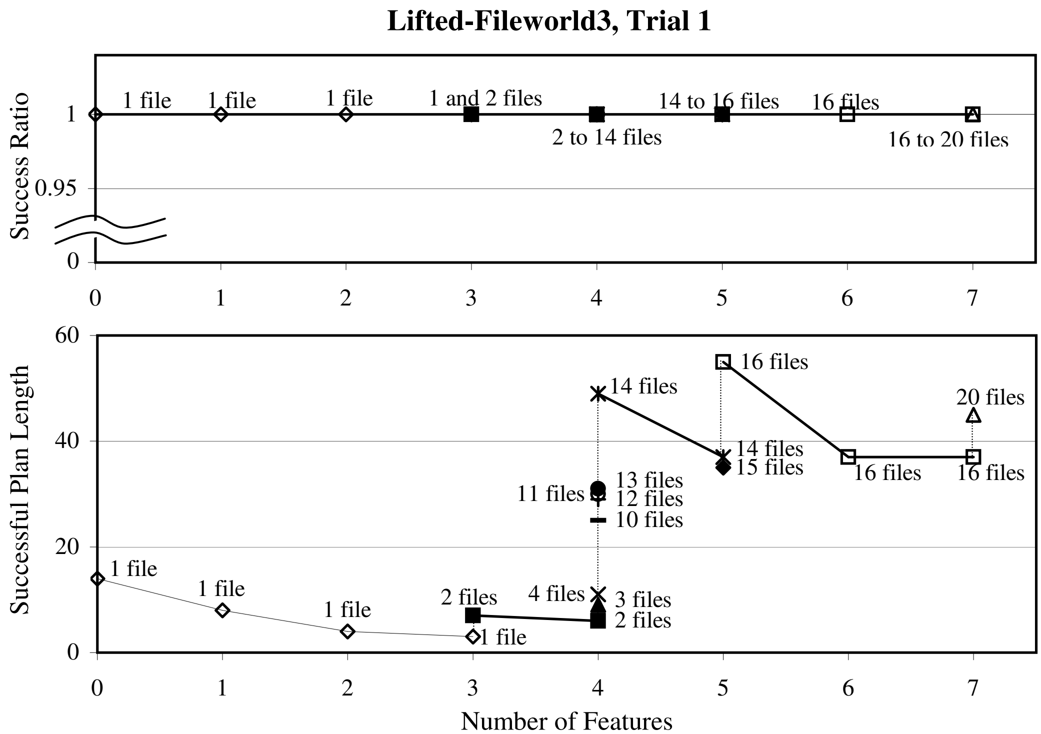


Figure 20: Summary results for LIFTED-FILEWORLD3. For full discussion and detailed results, please see Appendix B.

(alternating AVI training, difficulty increase, and feature generation as in the original algorithm). During each trial, in each domain, we select the best performing policy, running the algorithm until the target problem difficulty is reached or there is no improvement for at least three feature additions; in the latter case generating at least nine features. We evaluate each greedy policy acquired in this manner, measuring the average target-problem-size performance in each domain, and average the results of two trials. The results are shown in Figure 21.

In no domain does the alternative Random Scoring perform comparably to the original Greedy Beam Search/BE Scoring, with the exception of three domain/size combinations where both learners perform very poorly (ZENOTRAVEL, 10-block EXPLODING BLOCKSWORLD, and 5-disc TOWERS OF HANOI). The alternative Random Beam Narrowing is sometimes adequate to replace the original approach, but in some domains, greedy beam search is critical to our performance.

7.3.5 COMPARISON TO FF-REPLAN AND FOALP

We compare the performance of our learned policies to FF-Replan and FOALP on each of the PPDDL evaluation domains used above. We use the problem generators provided by the planning competitions to generate 30 problems for each tested problem size except for TOWERS OF HANOI

Domain Size	BW 20	Box (15,5)	Box (10,10)	Tire 30	Zeno (10,2,2)	EX-BW 5	EX-BW 10	TOH 4	TOH 5	File 30
Greedy Beam/BE Scoring (orig.) SR	0.98	1	0.98	0.92	0.11	0.34	0.03	0.51	0.00	1
Greedy Beam/BE Scoring (orig.) SLen.	748	90	235	5	1137	6	23	4	14	65
Random Scoring (var. 1) SR	0	0.99	0.21	0.67	0.05	0.27	0.01	0.24	0.03	1
Random Scoring (var. 1) SLen.	–	946	1582	6	910	6	12	13	26	215
Random Beam Narrowing (var. 2) SR	0.01	1	0.99	0.91	0.13	0.35	0.02	0.25	0.01	1
Random Beam Narrowing (var. 2) SLen.	258	90	242	6	1127	8	19	38	84	250
Random walk SR	0	0.97	0.18	0.18	0.06	0.13	0	0.09	0.00	1
Random walk SLen.	–	1038	1579	6	865	4	–	14	14	251

Figure 21: Target-problem-size performance (averaged over 600 problems) for relational features learned from the original algorithm and the two alternatives as discussed in Section 6.2, and from random walk, averaged over the best results of two independent trials for each target problem size.

and LIFTED-FILEWORLD3, where there is one fixed problem for each problem size. We evaluate the performance of each planner 30 times for each problem, and report in Fig. 22 the success ratio of each planner in each problem size (averaged over all attempts). Our policies, learned from the two independent trials shown above, are indicated as RFAVI #1 and RFAVI #2. Each planner has a 30-minute time limit for each attempt. The average time required to finish a successful attempt for the largest problem size in each domain is reported in Figure 23.

For each of the two trials of our learner in each domain, we evaluate here the policy that performed the best in the trial on the (first) target problem size. (Here, a "policy" is a set of features and a corresponding weight vector learned by AVI during the trial.) Performance is measured by success rate, with ties broken by plan length. Any remaining ties are broken by taking the later policy in the trial from those that are tied. In each case, we consider that policy to be the "policy learned from the trial."

The results show that our planner's performance is incomparable with that of FF-Replan (winning in some domains, losing in others) and generally dominates that of FOALP.

RFAVI performs the best of the planners in larger BLOCKSWORLD, CONJUNCTIVE-BOXWORLD, and TIREWORLD problems. RFAVI is essentially tied with FF-Replan in performance in LIFTED-FILEWORLD3. RFAVI loses to FF-Replan in the remaining three domains, EXPLODING BLOCKSWORLD, ZENOTRAVEL, and TOWERS OF HANOI. Reasons for the difficulties in the last three domains are discussed above in the sections presenting results for those domains. We note that FOALP does not have a learned policy in ZENOTRAVEL, EXPLODING BLOCKSWORLD, TOWERS OF HANOI, and LIFTED-FILEWORLD3.

RFAVI relies on random walk to explore plateaus of states not differentiated by the selected features. This reliance frequently results in long plan lengths and at times results in failure. We have recently reported elsewhere on early results from ongoing work remedying this problem by using search in place of random walk (Wu et al., 2008).

The RFAVI learning approach is very different from the non-learning online replanning used by FF-Replan, where the problem is determinized, dropping all probability parameters. It is an

	15 blocks BW	20 blocks BW	25 blocks BW	30 blocks BW	
RFAVI #1	1 (483)	1 (584)	0.85 (1098)	0.75 (1243)	
RFAVI #2	1.00 (463)	1.00 (578)	0.85 (1099)	0.77 (1227)	
FF-Replan	0.93 (52)	0.91 (71)	0.7 (96)	0.23 (118)	
FOALP	1 (56)	0.73 (73)	0.2 (96)	0.07 (119)	
	(10BX,5CI)Box	(10BX,10CI)Box	(10BX,15CI)Box	(15BX,5CI)Box	(20BX,20CI)Box
RFAVI #1	1 (76)	0.97 (225)	0.93 (459)	1 (90)	0.82 (959)
RFAVI #2	1 (75)	0.97 (223)	0.93 (454)	1 (90)	0.82 (989)
FF-Replan	1 (70)	0.98 (256)	0.93 (507)	1 (88)	0.35 (1069)
FOALP	1 (35)	0.70 (257)	0.28 (395)	0.99 (56)	0.0 (711)
	20 nodes Tire	30 nodes Tire	40 nodes Tire	(10CI,2PR,2AT)Zeno	
RFAVI #1	0.87 (5)	0.85 (7)	0.98 (6)	0.06 (1240)	
RFAVI #2	0.85 (4)	0.84 (7)	0.97 (6)	0.07 (1252)	
FF-Replan	0.76 (2)	0.73 (3)	0.83 (3)	1 (99)	
FOALP	0.92 (4)	0.90 (5)	0.91 (5)	N/A	
	5 blocks EX-BW	10 blocks EX-BW	4 discs TOH	5 discs TOH	30 files Lifted-File
RFAVI #1	0.25 (8)	0.02 (30)	0.43 (4)	0 (–)	1 (65)
RFAVI #2	0.25 (8)	0.01 (35)	0.47 (4)	0 (–)	1 (65)
FF-Replan	0.91 (7)	0.45 (20)	0.57 (3)	0.37 (7)	1 (66)
FOALP	N/A	N/A	N/A	N/A	N/A

Figure 22: Comparison of our planner (RFAVI) against FF-Replan and FOALP. Success ratio for a total of 900 attempts (30 attempts for TOWERS OF HANOI and LIFTED-FILEWORLD3) for each problem size is reported, followed by the average successful plan length in parentheses. The two rows for RFAVI map to two learning trials shown in the paper.

	30 BW	(20,20) BX	40 Tire	(10,2,2) Zeno	10 EX-BW	5 TOH	30 Files
RFAVI #1	106s	83s	1s	51s	2s	–	1s
RFAVI #2	105s	86s	0s	51s	3s	–	1s
FF-Replan	872s	739s	0s	1s	8s	3s	10s
FOALP	16s	173s	24s	N/A	N/A	N/A	N/A

Figure 23: Average runtime of the successful attempts, from the results shown in Figure 22, on the largest problem size for each domain.

important topic for future research to try to combine the benefits obtained by these very different planners across all domains.

The dominance of RFAVI over FOALP in these results implies that RFAVI is at the state of the art among first-order techniques — those that work with the problem in lifted form and use lifted generalization. Although FOALP uses first-order structure in feature representation, the learned features are aimed at satisfying goal predicates individually, not as a whole. We believe that the goal-decomposition technique can sometimes work well in small problems but does not scale well to large problems.

In these comparisons, it should also be noted that FOALP does not read PPDDL domain descriptions directly, but requires human-written domain axioms for its learning, unlike our completely automatic technique (requiring only a few numeric parameters characterizing the domain). This requirement for human-written domain axioms is one of the reasons why FOALP did not compete in some of the competition domains and does not have a learned policy for some of the domains tested here.

In CONJUNCTIVE-BOXWORLD[11], we note that FF-Replan uses an "all outcomes" problem determinization that does not discriminate between likely and unlikely outcomes of truck-movement actions. As a result, plans are frequently selected that rely on unlikely outcomes (perhaps choosing to move a truck to an undesired location, relying on the unlikely outcome of "accidentally" moving to the desired location). These plans will usually fail, resulting in repeated replanning until FF luckily selects the high-likelihood outcome or plan execution happens to get the desired low-likelihood outcome. This behavior is in effect similar to the behavior our learned value function exhibits because, as discussed in Section 7.3.2, our learner failed to find any feature rewarding appropriate truck moves. Both planners result in long plan lengths due to many unhelpful truck moves. However, our learned policy conducts the random walk of trucks much more efficiently (and thus more successfully) than the online replanning of FF-Replan, especially in the larger problem sizes. We believe even more dramatic improvements will be available with improved knowledge representation for features.

7.4 SysAdmin

A full description of the SYSADMIN domain is provided in the work of Guestrin, Koller, and Parr (2001). Here, we summarize that description. In the SYSADMIN domain, machines are connected in different topologies. Each machine might fail at each step, and the failure probability depends on the number of failed machines connected to it. The agent works toward minimizing the number of failed machines by rebooting machines, with one machine rebooted at each time step. For a problem with n machines and a fixed topology, the dynamic state space can be sufficiently described by n propositional variables, each representing the on/off status of a certain machine.

We test this domain for the purpose of direct comparison of the performance of our propositional techniques to the published results in the work of Patrascu et al. (2002). We test exactly the topologies evaluated there and measure the performance measure reported there, sup-norm Bellman error.

We evaluate our method on the exact same problems (same MDPs) used for evaluation in the work of Patrascu et al. (2002) for testing this domain. Two different kinds of topologies are tested:

11. We hand-convert the nested universal quantifiers and conditional effects in the original BOXWORLD domain definition to an equivalent form without universal quantifiers and conditional effects to allow FF-Replan to read the domain.

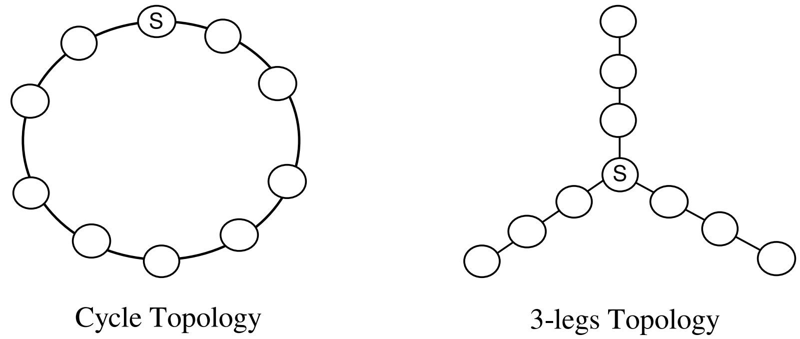


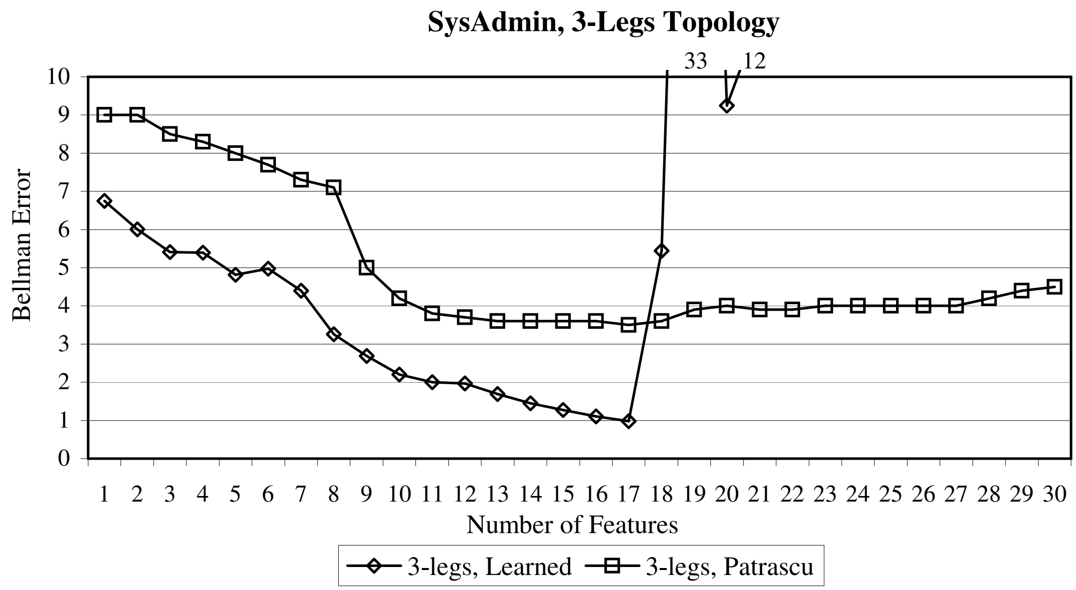
Figure 24: Illustration of the two topologies in the SYSADMIN domain (10 nodes). Each node represents a machine. The "S" label indicates a server machine, as specified in the work of Patrascu et al. (2002).

3-legs and cycle. The "3-legs" topology has three three-node legs (each a linear sequence of three connected nodes) each connected to a single central node at one end. The "cycle" topology arranges the ten nodes in one large cycle. There are 10 nodes in each topology. These two topologies are illustrated in Figure 24. The target of learning in this domain is to keep as many machines operational as possible, so the number of operating machines directly determines the reward for each step. Since there are only 10 nodes and the basic features are just the on/off statuses of the nodes, there are a total of 1024 states. The reward-scaling parameter r_{scale} (defined in Section 5 in Online Appendix 1, available on JAIR website, on page 8) is selected to be 10.

The work by Patrascu et al. (2002) uses L_∞ (sup norm) Bellman error as the performance measurement in SYSADMIN. Our technique, as described above, seeks to reduce mean Bellman error more directly than L_∞ Bellman error. We report the L_∞ Bellman error, averaged over two trials, in Figure 25. Also included in Figure 25 are the results shown in the work of Patrascu et al. (2002). We select the best result shown there (from various algorithmic approaches) from the 3-legs and cycle topologies shown in their paper. These correspond to the "d-o-s" setting for the cycle topology and the "d-x-n setting" for the 3-legs topology, in the terminology of that paper.

Both topologies show that our algorithm reduces the L_∞ Bellman error more effectively per feature as well as more effectively overall than the experiments previously reported in the work of Patrascu et al. (2002). Both topologies also show Bellman error eventually diverges as AVI cannot handle the complexity of the error function as dimensionality increases. Our algorithm can still achieve low Bellman error by remembering and restoring the best-performing weighted feature set once weakened performance is detected.

We note that our superior performance in reducing Bellman error could be due entirely or in part to the use of AVI for weight training instead of approximate linear programming (ALP), the method used by Patrascu et al. However, no such systematic superiority is known for AVI over ALP, so these results suggest superior performance of the feature learning itself.



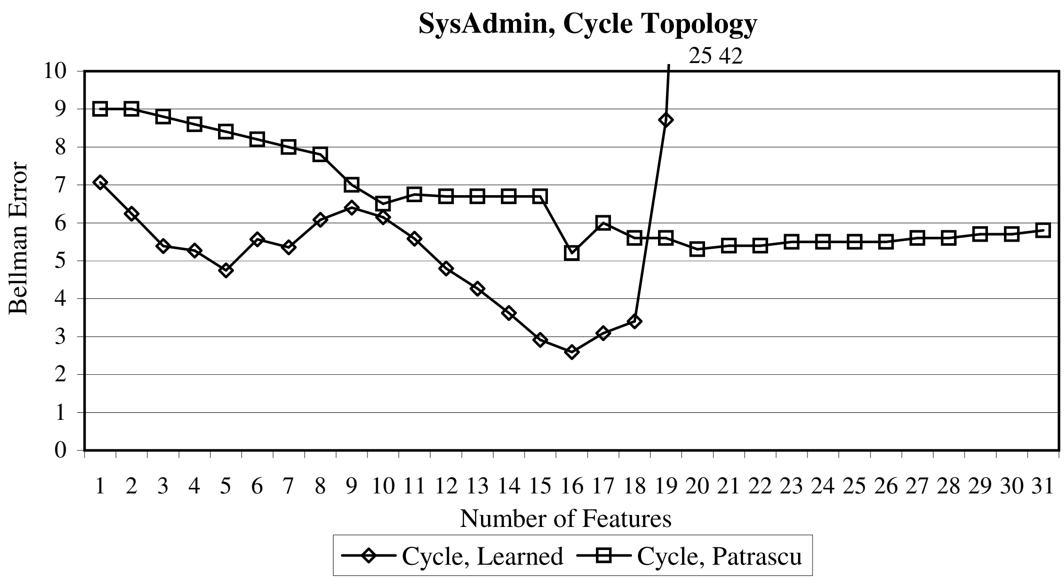


Figure 25: L_∞ Bellman error for the SYSADMIN domain (10 nodes) for two topologies. Values for the results from the work of Patrascu et al. (2002) are taken from Figure 2 and 3 of the work of Patrascu et al. (2002).

7.5 Demonstration of Generalization Across Problem Sizes

An asset of the relational feature representation presented in this paper is that learned relational features are applicable to any problem size in the same domain. In section 2.4, we have discussed

Target problem sizes Intermediate problem sizes	10 × 20 Tetris 10 × 10 Tetris	15 blocks BW 10 blocks BW	(15 box, 5 city) BX (10 box, 5 city) BX	30 nodes Tire 15 nodes Tire	30 files Lifted-File 10 files Lifted-File
Generalize from target size	55	1 (171)	1 (76)	0.88 (4)	1 (25)
Learn in intermediate size	119	1 (170)	1 (188)	0.89 (4)	1 (25)
Random walk	0.1	0 (–)	0.97 (893)	0.29 (6)	1 (88)

Figure 26: Performance in intermediate-sized problems by generalization. We show here the performance of value functions learned in target problem sizes when evaluated on intermediate-sized problems, to demonstrate generalization between sizes. For comparison, also on intermediate-sized problems, we show the performance of value functions learned directly in the intermediate size as well as the performance of random walk. Generalization results and intermediate size learning results are averages of two trials. For TETRIS, average accumulated rows erased are shown. For the goal-oriented domains, success ratio and successful plan length (in parentheses) are shown for each domain.

the modeling of a planning domain as an infinite set of MDPs, one for each problem instance in the domain. Over this infinite set of MDPs, a feature vector plus a weight vector defines a single value function that is well defined for every problem instance MDP. Here we discuss the question of whether our framework can find a single feature/weight vector combination that generalizes good performance across problem sizes, i.e., for the value function V defined by such combination, whether Greedy(V) performs similarly well in different problem sizes.

Throughout Section 7, we have demonstrated the direct application of learned feature/weight vectors to target problem sizes, (without retraining of weights)—these results are shown in the target-size lines in the result tables for each domain. In TETRIS, BLOCKSWORLD, CONJUNCTIVE-BOXWORLD, TIREWORLD, and LIFTED-FILEWORLD3, the target-size lines demonstrate direct successful generalization to target sizes even when the current problem sizes is significantly smaller. (In the other domains, there was either no notion of problem size (SYSADMIN), or insufficient planning progress to significantly increase problem size when learning from small problems (EXPLODING BLOCKSWORLD, ZENOTRAVEL, and TOWERS OF HANOI).)

In this subsection, we consider the generalization from (larger) target sizes to selected intermediate sizes in these five domains. Specifically, we take the weight vectors and feature vectors resulting from the end of the trials (i.e. with weight vector retrained at the target sizes), and apply directly to selected intermediate problem sizes without weight retraining. For the trials that terminate before learning reaches the target problem sizes[12], we take the weights and features that result in the best performing policy at the terminating problem sizes. The generalization results are shown in Figure 26; for comparison, that table also shows the performance on the same intermediate-sized problems of the value function that was learned directly at the that size, as well as the performance of random walk on that size.

12. Note that one of the trials in TETRIS terminates before reaching target size due to non-improving performance, and the two trials in LIFTED-FILEWORLD3 terminate as target-size performance already reaches optimality before learning reaches the target size. Still, although a few of the value functions were learned at smaller sizes than the target size, all of the value functions evaluated for generalization were learned at significantly larger sizes than the intermediate evaluation size.

In each domain shown the random walk result is much weaker than the generalization result, showing the presence of generalization of learned value functions across problem sizes. In the four goal-oriented planning domains, applying the value functions learned in the target sizes equals the performance achieved by value functions learned directly in the intermediate sizes (with better performance in CONJUNCTIVE-BOXWORLD). In TETRIS, however, the generalization result does not match the result of learning in the intermediate size. We note that in some domains, solution strategy is invariant with respect to the problem size (e.g. destroying incorrect towers to form correct ones in BLOCKSWORLD). For some domains the best plan/strategy may change dramatically with size. For example, in TETRIS, a larger number of rows in the board allows strategies that temporary stack uncompleted rows, but smaller number of rows favors strategies that complete rows as quickly as possible. Thus one should not necessarily expect generalization between domain sizes in every domain—this conclusion can be expected to hold whether we are considering the generalization of value functions or of policies.

We have included a discussion of policy-based generalization in the related work section (Appendix A.4), focusing on our previous work on approximate policy iteration. However, we note that policies that generalize between problems of different sizes are no more or less well defined than value functions which generalize between such problems. In our previous API work, we defined policies that select actions for states of any domain size; in this work we define value functions that assign numeric values to states of any domain size. None of this work guarantees finding a good or optimal policy or value function; as far as we know, some problems admit good compact value functions, some admit good compact policies, some admit both, and some neither.

8. Discussion and Future Research

We have presented a general framework for automatically learning state-value functions by feature-discovery and gradient-based weight training. In this framework, we greedily select features from a provided hypothesis space (which is a parameter of the method) to best correlate with Bellman error features, and use AVI to find weights to associate with these features.

We have proposed two different candidate hypothesis spaces for features. One of these two spaces is a relational one where features are first-order formulas with one free-variable, and a beam-search process is used to greedily select a hypothesis. The other hypothesis space we have considered is a propositional feature representation where features are decision trees. For this hypothesis space, we use a standard classification algorithm C4.5 (Quinlan, 1993) to build a feature that best correlates with the sign of the statewise Bellman error, instead of using both the sign and magnitude.

The performance of our feature-learning planners is evaluated using both reward-oriented and goal-oriented planning domains. We have demonstrated that our relational planner represents the state-of-the-art for feature-discovering probabilistic planning techniques. Our propositional planner does not perform as well as our relational planner, and cannot generalize between problem instances, suggesting that knowledge representation is indeed critical to the success of feature-discovering planners.

Although we present results for a propositional feature-learning approach and a relation feature-learning approach, the knowledge representation difference is not the only difference between the approaches. Historically, our propositional approach was originally conceived as a reduction to classification learning, and so does not attempt to capture the magnitude of the Bellman error during feature selection, but rather focuses only the sign of the error. In contrast, our relational approach counts objects in order to match the magnitude of the Bellman error.

Because of this difference, we cannot attribute all of the performance differences between the approaches to the knowledge representation choice. Some differences in performance could be due to the choice to match sign only in the propositional feature selection. A possible future experiment to identify the sources of performance variation would use a propositional representation involving regression trees (Dzeroski, Todorovski, & Urbancic, 1995) to capture the magnitude of the error. This representation might possibly perform somewhat better than the decision-tree representation shown here, but of course would still not enable the generalization between sizes that the relational feature learner exhibits.

Bellman-error reduction is of course just one source of guidance that might be followed in feature discovery. During our experiments in the IPPC planning domains, we find that in many domains the successful plan length achieved is much longer than optimal, as we discussed above in Section 7.3.5. A possible remedy other than deploying search as in our previous work (Wu et al., 2008) is to learn features targeting the dynamics inside plateaus, and use these features in decision-making when plateaus are encountered.

Acknowledgments

This material is based upon work supported in part by the National Science Foundation under Grant No. 0905372.

Appendix A. Other Related Work

A.1 Other Feature Selection Approaches

A.1.1 Feature Selection via Constructive Function Approximation

Automatic feature extraction in sequential decision-making has been studied in the work of Utgoff and Precup (1997), via constructive function approximation (Utgoff & Precup, 1998). This work can be viewed as a forerunner of our more general framework, limited to propositional representations, binary-valued features, and new features that are single-literal extensions of old features by conjunction. Also in the work of Rivest and Precup (2003) a variant of Cascade-Correlation (Fahlman & Lebiere, 1990), a constructive neural network algorithm, is combined with TD-learning to learn value functions in reinforcement learning. Cascade-Correlation incrementally adds hidden units to multi-layered neural networks, where each hidden unit is essentially a feature built upon a set of numerically-valued basic features. Our work provides a framework generalizing those prior efforts into a reduction to supervised learning, with explicit reliance on the Bellman error signal, so that any feature hypothesis space and corresponding learner can be deployed. In particular, we demonstrate our framework on both binary propositional features using C4.5 as the learner and rich numeric-valued relational features using a greedy beam-search learner. Our work provides the first evaluation of automatic feature extraction in benchmark planning domains from the several planning competitions.

While the work of Utgoff and Precup (1997) implicitly relies on Bellman error, there is no explicit construction of a Bellman error training set or discussion of selecting features to correlate to Bellman error. For instance, their work focuses first on refining a current feature for which weight updates are converging poorly (high variance in weight updates), whereas our work focuses first on finding a feature that correlates to statewise Bellman error, regardless of whether that feature refines any current feature. In addition, their work selects features online while the weights of the current features are being adjusted, so there is no stationary target value function for which the Bellman error is considered in the selection of the next new feature. In contrast, our work separates weight training and new feature selection completely. (These differences are perhaps in part due to the reinforcement learning setting used in Utgoff & Precup, 1997, as opposed to the planning setting of our work.)

The selection of hidden unit feature in Cascade-Correlation (Fahlman & Lebiere, 1990) is based on the covariance between feature values and errors of the output units. For output units that are estimating a value function, with training data providing the Bellman update of that value function, the output unit error is just Bellman error. Thus, the hidden units learned in the work of Rivest and Precup (2003) are approximations of Bellman-error features just as our learned features are, although this is not made explicit in that work. By making the goal of capturing Bellman error explicit here, we provide a general reduction that facilitates the use of any learning method to capture the resulting feature-learning training sets. In particular, we are able to naturally demonstrate generalization across domain sizes in several large domains, using a relational feature learner. In contrast, the single test domain in the work of Rivest and Precup (2003) has a small fixed size. Nonetheless, that work is an important precursor to our approach.

A.1.2 FEATURE CONSTRUCTION VIA SPECTRAL ANALYSIS

Feature-learning frameworks for value functions based upon spectral analysis of state-space connectivity are presented in the work of Mahadevan and Maggioni (2007) and Petrik (2007). In these frameworks, features are eigenvectors of connectivity matrices constructed from random walk (Mahadevan & Maggioni, 2007) or eigenvectors of probabilistic transition matrices (Petrik, 2007). Such features capture aspects of long-term problem behaviours, as opposed to the short-term behaviours captured by the Bellman-error features. Bellman-error reduction requires iteration to capture long-term behaviors.

Reward functions are not considered at all during feature construction in the work of Mahadevan and Maggioni (2007); but in the work of Petrik (2007), reward functions are incorporated in the learning of Krylov basis features, an variant of our Bellman error features (Parr et al., 2008), to complement the eigenvector features. However, even in Petrik's framework, reward is only incorporated in features used for policy evaluation rather than in the controlled environment we consider.

Essential to our work here is the use of machine learning in factored representations to handle very large statespaces and to generalize between problems of different sizes. Both of these spectral analysis frameworks are limited in this respect (at least at the current state of development). The approach by Petrik (2007) is presented only for explicit statespaces, while a factorization approach for scaling up to large discrete domains is proposed in the work of Mahadevan and Maggioni (2007). In that approach, features are learned for each dimension in the factorization, independent of the other dimensions. We believe the assumption of independence between the dimensions is inappropriate in many domains, including the benchmark planning domains considered in our work. The Mahadevan and Maggioni factorization approach also suffers the same drawbacks as our propositional approach: the solution has to be recomputed for problems of different sizes in the same domain and so lacks the flexibility to generalize between problems of different sizes provided by our relational approach.

A.2 Structural Model-based and Model-free Solution Methods for Markov Decision Processes

A.2.1 Relational Reinforcement Learning

In the work of Džeroski et al. (2001), a relational reinforcement learning (RRL) system learns logical regression trees to represent Q-functions of target MDPs. This work is related to ours since both use relational representations and automatically construct functions that capture state value. In addition to the Q-function trees, a policy tree learner is also introduced in the work of Džeroski et al. (2001) that finds policy trees based on the Q-function trees. We do not learn an explicit policy description and instead use only greedy policies for evaluation.

The logical expressions in RRL regression trees are used as decision points in computing the value function (or policy) rather than as numerically valued features for linear combination, as in our method. Generalization across problem sizes is achieved by learning policy trees; the learned value functions apply only to the training problem sizes. To date, the empirical results from this approach have failed to demonstrate an ability to represent the value function usefully in familiar planning benchmark domains. While good performance is shown for simplified goals such as placing a particular block A onto a particular block B, the technique fails to capture the structure in richer problems such as constructing particular arrangements of Blocksworld towers. RRL has not been entered into any of the international planning competitions. These difficulties representing complex relational value functions persist in extensions to the original RRL work (Driessens & Džeroski, 2004; Driessens et al., 2006), where again only limited applicability is shown to benchmark planning domains such as those used in our work.

A.2.2 Policy Learning via Boosting

In the work of Kersting and Driessens (2008), a boosting approach is introduced to incrementally learn features to represent stochastic policies. This is a policy-iteration variant of our feature-learning framework, and clearly differs from our work as policy representations are learned instead of value function representations. Using the regression tree learner TILDE (Blockeel & De Raedt, 1998), the feature learner demonstrated advantages against previous RRL work in the task of accomplishing on(A,B) in a 10-block problem. Applicability to a simple continuous domain (the corridor world) is also demonstrated. As in the line of RRL work, only limited applicability to benchmark planning domains is shown here. One probable source of this limited applicability is the model-free reinforcement-learning setting where the system does not model the problem dynamics explicitly.

A.2.3 Fitted Value Iteration

Gordon (1995) has presented a method of value iteration called *fitted value iteration* that is suitable for very large state spaces but does not require direct feature selection. Instead, the method relies on a provided kernel function measuring similarity between states. Selection of this kernel function can be viewed as a kind of feature selection, as the kernel identifies which state aspects are significant in measuring similarity. To our knowledge, techniques from this class have not been applied to large relational planning problems like those evaluated in this paper. We do note that selection of a single relational kernel for all domains would measure state similarity in a domain-independent manner and thus we believe such a kernel could not adapt to the individual domains the way our work here does. Thus we would expect inferior performance from such an approach; however, this remains to be investigated. Selection of domain-specific kernels for stochastic planning domains, automatically, is also yet to be explored.

A.2.4 EXACT VALUE ITERATION IN FIRST-ORDER MDPS

Previous work has used lifted techniques to exactly solve first-order MDPs by reformulating exact solution techniques from explicit MDPs, such as value iteration. Boutilier et al. (2001) and Holldobler and Skvortsova (2004) have independently used two different first-order languages (situation calculus and fluent calculus, respectively) to define first-order MDPs. In both works, the Bellman update procedure in value iteration is reformulated using the respective calculus, resulting in two first-order dynamic-programming methods: symbolic dynamic programming (SDP), and first-order value iteration (FOVI). Only a simple boxworld example with human-assisted computation is demonstrated in the SDP work, but the method serves as a basis for FOALP (Sanner & Boutilier, 2009), which replaces exact techniques with heuristic approximation in order to scale the techniques to benchmark planning domains. Application of FOVI on planning domains is only demonstrated on the colored blocksworld benchmark, and is limited to under 10 blocks (Holldobler, Karabaev, & Skvortsova, 2006).

In the work of Kersting et al. (2004), constraint logic programming is used to define a relational value iteration method. MDP components, such as states, actions, and rewards, are first abstracted to form a Markov decision program, a lifted version of an MDP. A relational Bellman operation (ReBel) is then used to define updates of Q-values and state values. Empirical study of the ReBel approach has been limited to 10-step backups from single-predicate goals in the blocksworld and logistics domains.

Exact techniques suffer from difficulty in representing the full complexity of the state-value function for arbitrary goals in even mildly complex domains. These previous works serve to illustrate the central motivation for using problem features to compactly approximate the structure of a complex value function, and thus to motivate the automatic extraction of features as studied in this work.

A.3 Comparison to Inductive Logic Programming Algorithms

The problem of selecting a numeric function on relational states to match the Bellman-error training set is a "first-order regression" problem for which there are some available systems described in the Inductive logic programming (ILP) literature (Quinlan, 1996; Karalic & Bratko, 1997).

It is important to note that most ILP work has studied the learning of *classifiers* on relational data (Muggleton, 1991), but here we are concerned with learning *numeric functions* on relational data (such as our states). The latter problem is called "first-order regression" within the ILP literature, and has received less study than relational classification. Here, we choose to design our own proof-of-concept relational learner for our experiments rather than use one of the few previous systems. Separate work is needed to compare the utility of this relational learner with previous regression systems; our purpose here is to demonstrate the utility of Bellman-error training data for finding decision-theoretic value-function features. Our simple learner here suffices to create state-of-the-art domain-independent planning via automatic feature selection.

ILP classification systems often proceed either from general to specific, or from specific to general, in seeking a concept to match the training data. For regression, however, there is no such easy ordering of the numeric functions to be searched. We design instead a method that searches a basic logical expression language from simple expressions to more complex expressions, seeking good matches to the training data. In order to control the branching factor, while still allowing more complex expressions to be considered, we heuristically build long expressions out of only those short expressions that score best. In other words, we use a beam search of the space of expressions.

There are several heuristic aspects to our method. First, we define a heuristic set of basic expressions from which our search begins. Second, we define an heuristic method of combining expressions to build more complex expressions. These two heuristic elements are designed so that any logical formula without disjunction, with one free variable, can be built by repeated combination from the basic expressions. Finally, the assumption that high-scoring expressions will be built only out of high-scoring parts is heuristic (and often not true). This critical heuristic assumption makes it likely that our learner will often miss complex features that match the training data well. There is no known method that guarantees tractably finding such features.

A.4 Approximate Policy Iteration for Relational Domains

Our planners use greedy policies derived from learned value functions. Alternatively, one can directly learn representations for policies. The policy-tree learning in the work of Džeroski et al. (2001), discussed previously in Appendix A.2.1, is one such example. Recent work uses a relational decision-list language to learn policies for small example problems that generalize well to perform in large problems (Khardon, 1999; Martin & Geffner, 2004; Yoon et al., 2002). Due to the inductive nature of this line of work, however, the selected policies occasionally contain severe flaws, and no mechanism is provided for policy improvement. Such policy improvement is quite challenging due to the astronomically large highly structured state spaces and the relational policy language.

In the work of Fern et al. (2006), an approximate version of policy iteration addressing these issues is presented. Starting from a base policy, approximate policy iteration iteratively generates training data from an improved policy (using policy rollout) and then uses the learning algorithm in the work of Yoon et al. (2002) to capture the improved policy in the compact decision-list language again. Similar to our work, the learner in the work of Fern et al. (2006) aims to take a flawed solution structure and improve its quality using conventional MDP techniques (in that case, finding an improved policy with policy rollout) and machine learning. Unlike our work, in the work of Fern et al. (2006) the improved policies are learned in the form of logical decision lists. Our work can be viewed as complementary to this previous work in exploring the structured representation of value functions where that work explored the structured representation of policies. Both approaches are likely to be relevant and important to any long-term effort to solve structured stochastic decision-making problems.

We note that feature-based representation, as considered here and generally in the MDP literature, is used to represent value functions rather than policies. Compact representation of policies can be done via value functions (with greedy execution) or more directly, for example, using decision lists. The previous API work just discussed uses a direct representation for policies, and never uses any compact representation of value functions. Instead, sampling of value functions is used in the policy evaluation step of policy iteration.

One can imagine a different and novel approach to API in which the compact feature-based representation is used for value functions, with greedy execution as the policy representation. In that approach, feature discovery similar to what we explore here for value iteration could be designed to assist the policy evaluation phase of the policy iteration. We leave further development and evaluation of that idea to future work. We expect the two approaches to API, as well as our current approach to value iteration, to have advantages and disadvantages that vary with the domain in ways that have yet to be well understood. Some domains have natural compact direct policy representations ("run if you see a tarantula"), whereas others are naturally compactly represented via value functions ("prefer restaurants with good review ratings"). Research in this area must eventually develop means to combine these compact representations effectively.

A.5 Automatic Extraction of Domain Knowledge

There is a substantial literature on learning to plan using methods other than direct representation of a value function or a reactive policy, especially in the deterministic planning literature. These techniques are related to ours in that both acquire domain specific knowledge via planning experience in the domain. Much of this literature targets control knowledge for particular search-based planners (Estlin & Mooney, 1997; Kambhampati et al., 1996; Veloso et al., 1995), and is distant from our approach in its focus on the particular planning technology used and on the limitation to deterministic domains. It is unclear how to generalize this work to value-function construction or probabilistic domains.

However, the broader learning-to-plan literature also contains work producing declarative learned domain knowledge that could well be exploited during feature discovery for value function representation. In the work of Fox and Long (1998), a pre-processing module called TIM is able to infer useful domain-specific and problem-specific structures, such as typing of objects and state invariants, from descriptions of domain definition and initial states. While these invariants are targeted in that work to improving the planning efficiency of a Graphplan based planner, we suggest that future work could exploit these invariants in discovering features for value function representation. Similarly, in the work of Gerevini and Schubert (1998), DISCOPLAN infers state constraints from the domain definition and initial state in order to improve the performance of SAT-based planners; again, these constraints could be incorporated in a feature search like our method but have not to date.

Appendix B. Results and Discussions for Five Probabilistic Planning Competition Domains

In Section 7.3, we have presented the results of our relational and propositional feature learners for BLOCKSWORLD and CONJUNCTIVE-BOXWORLD. Here we present the results of our relational feature learner for the following five probabilistic planning competition domains: TIREWORLD, ZENOTRAVEL, EXPLODING BLOCKSWORLD, TOWERS OF HANOI, and LIFTED-FILEWORLD3.

B.1 Tireworld

We use the TIREWORLD domain from the second IPPC. The agent needs to drive a vehicle through a graph from the start node to the goal node. When moving from one node to an adjacent node, the vehicle has a certain chance of suffering a flat tire (while still arriving at the adjacent node).

Trial #1												
# of features	0	1	2	3	3	3	4	4	5	5	5	5
Problem difficulty	4	4	4	4	5	6	6	9	9	10	20	30
Success ratio	0.52	0.81	0.84	0.86	0.86	0.84	0.88	0.85	0.86	0.86	0.91	0.91
Plan length	4	3	4	2	2	2	3	3	4	4	5	5
Accumulated time (Hr.)	0.3	3.1	12	17	18	18	19	21	22	23	29	36
Target size SR	0.17	0.53	0.81	0.83	0.83	0.82	0.90	0.91	0.91	0.91	0.92	0.92
Target size Slen.	5	4	9	5	4	4	6	6	6	6	5	6

Trial #2										
# of features	0	1	2	3	3	3	4	4	4	4
Problem difficulty	4	4	4	4	5	6	6	10	20	30
Success ratio	0.52	0.81	0.85	0.86	0.93	0.81	0.89	0.85	0.86	0.88
Plan length	4	3	3	2	3	2	3	4	4	5
Accumulated time (Hr.)	0.5	3.7	6.9	10	11	11	12	14	18	24
Target size SR	0.19	0.49	0.80	0.82	0.91	0.62	0.92	0.91	0.90	0.88
Target size Slen.	7	3	9	4	5	2	5	5	6	6

Figure 27: TIREWORLD performance (averaged over 600 problems) for relational learner. We add one feature per column until success ratio exceeds 0.85 and average successful plan length is less than $4n$, for n nodes, and then increase problem difficulty for the next column. Plan lengths shown are successful trials only. Problem difficulties are measured in number of nodes, with a target problem size of 30 nodes. Some columns are omitted as discussed in Section 7.1.

The flat tire can be replaced by a spare tire, but only if there is such a spare tire present in the node containing the vehicle, or if the vehicle is carrying a spare tire. The vehicle can pick up a spare tire if it is not already carrying one and there is one present in the node containing the vehicle. The default setting for the second-IPPC problem generator for this domain defines a problem distribution that includes problems for which there is no policy achieving the goal with probability one. Such problems create a tradeoff between goal-achievement probability and expected number of steps to the goal. How strongly our planner favors goal achievement versus short trajectories to the goal is determined by the choice of the discount factor made in Section 6.1.

We start with 4-node problems in our relational learner and increase from n nodes to $n + 1$ nodes whenever the success ratio exceeds 0.85 and the average successful plan length is better than $4n$ steps. The target problem size is 30 nodes. The results are shown in Figures 18 and 27.

In TIREWORLD, our relational learner again is able to find features that generalize well to large problems. Our learner achieves a success ratio of about 0.9 on 30 node problems. It is unknown whether any policy can exceed this success ratio on this problem distribution; however, neither comparison planner, FOALP nor FF-Replan, finds a higher success-rate policy.

We note that some improvements in success rate in this domain will necessarily be associated with increases in plan length because success-rate improvements may be due to path deviations to acquire spare tires.

Trial #1											
# of features	0	1	1	2	3	4	5	6	7	8	9
Problem difficulty	3,1,1	3,1,1	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2
Success ratio	0.79	0.8	0.59	0.52	0.54	0.55	0.54	0.52	0.56	0.53	0.55
Plan length	253	255	413	440	437	450	411	440	426	428	451
Accumulated time (Hr.)	0.75	1.7	3.4	7.1	11	15	19	25	30	36	41
Target size SR	0.06	0.08	0.09	0.09	0.12	0.11	0.10	0.08	0.11	0.08	0.12
Target size Slen.	916	1024	1064	1114	1050	1125	1111	1115	1061	1174	1195

Trial #2											
# of features	0	1	2	2	3	4	5	6	7	8	9
Problem difficulty	3,1,1	3,1,1	3,1,1	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2
Success ratio	0.77	0.79	0.80	0.55	0.55	0.50	0.53	0.12	0.12	0.12	0.10
Plan length	262	254	233	391	425	415	422	0	0	0	0
Accumulated time (Hr.)	1.3	2.3	3.3	5.3	8.9	13	17	22	29	36	43
Target size SR	0.05	0.10	0.10	0.09	0.09	0.08	0.10	0.02	0.02	0.02	0.01
Target size Slen.	814	1008	1007	1067	1088	1014	1078	0	0	0	0

Figure 28: ZENOTRAVEL performance (averaged over 600 problems) for relational learner. The problem difficulty shown in this table lists the numbers of cities, travelers, and aircraft, with a target problem size of 10 cities, 2 travelers, and 2 aircraft. We add one feature per column until success ratio exceeds 0.8, and then increase problem difficulty for the next column. Plan lengths shown are successful trials only.

B.2 Zenotravel

We use the ZENOTRAVEL domain from the second IPPC. The goal of this domain is to fly all travelers from their original location to their destination. Planes have (finite-range, discrete) fuel levels, and need to be re-fuelled when the fuel level reaches zero to cont inue flying. Each available activity (boarding, debarking, flying, zooming, or refueling) is divided into two stages, so that an activity X is modelled as two actions start-X and finish-X. Each finish-X activity has a (high) probability of doing nothing. Once a "start" action is taken, the corresponding "finish" action must be taken (repeatedly) until it succeeds before any conflicting action can be started. This structure allows the failure rates on the "finish" actions to simulate action costs (which were not used explicitly in the problem representation for the competition). A plane can be moved between locations by flying or zooming. Zooming uses more fuel than flying, but has a higher success probability.

We start with a problem difficulty of 3 cities, 1 traveler, and 1 aircraft using our relational feature learner. Whenever the success ratio exceeds 0.8, we increase the number n of travelers and aircraft by 1 if the number of cities is no less than $5n-2$, and increase the number of cities by one otherwise. The target problem size is 10 cities, 2 travelers, and 2 aircraft. ZENOTRAVEL results for the relational learner are shown in Figures 18 and 28.

The relational learner is unable to find features that enable AVI to achieve the threshold success rate (0.8) for 3 cities, 2 travelers, and 2 aircraft, although 9 relational features are learned. The trials were stopped because no improvement in performance was achieved for several iterations of feature addition. Using a broader search ($W = 160$, $q = 3$, and $d = 3$) we are able to find better features and extend the solvable size to several cities with success rate 0.9 (not shown here as all results in this paper use the same search parameters, but reported in Wu & Givan, 2007), but the runtime also increases dramatically, to weeks. We believe the speed and effectiveness of the relational learning needs to be improved to excel in this domain, and a likely major factor is improved knowledge representation for features so that key concepts for ZENOTRAVEL are easily represented.

Trial two in Figure 28 shows a striking event where adding a single new feature to a useful value function results in a value function for which the greedy policy cannot find the goal at all, so that the success ratio degrades dramatically immediately. Note that in this small problem size, about ten percent of the problems are trivial, in that the initial state satisfies the goal. After the addition of the sixth feature in trial two, these are the only problems the policy can solve. This reflects the unreliability of our AVI weight-selection technique more than any aspect of our feature discovery: after all, AVI is free to assign a zero weight to this new feature, but does not. Additional study of the control of AVI and/or replacement of AVI by linear programming methods is indicated by this phenomenon; however, this is a rare event in our extensive experiments.

B.3 Exploding Blocksworld

We also use EXPLODING BLOCKSWORLD from the second IPPC to evaluate our relational planner. This domain differs from the normal Blocksworld largely due to the blocks having certain probability of being "detonated" when they are being put down, destroying objects beneath (but not the detonating block). Blocks that are already detonated once will not be detonated again. The goal state in this domain is described in tower fragments, where the fragments are not generally required to be on the table. Destroyed objects cannot be picked up, and blocks cannot be put down on destroyed objects (but a destroyed object can still be part of the goal if the necessary relationships were established before or just as it was destroyed).

We start with 3-block problems using our relational learner and increase from n blocks to $n + 1$ blocks whenever the success ratio exceeds 0.7. The target problem sizes are 5 and 10 blocks. EXPLODING BLOCKSWORLD results for the relational learner are shown in Figures 19 and 29. The results in EXPLODING BLOCKSWORLD are not good enough for the planner to increase the difficulty beyond 4-block problems, and while the results show limited generalization to 5-block problems, there is very little generalization to 10-block problems.

Our performance in this domain is quite weak. We believe this is due to the presence of many dead-end states that are reachable with high probability. These are the states where either the table or one of the blocks needed in the goal has been destroyed, before the object in question achieved the required properties. Our planner can find meaningful and relevant features: the planner discovers that it is undesirable to destroy the table, for instance. However, the resulting partial understanding of the domain cannot be augmented by random walk (as it is in some other domains such as BLOCKSWORLD and CONJUNCTIVE-BOXWORLD) to enable steady improvement in value, leading to the goal; random walk in this domain invariably lands the agent in a dead end. Very short successful plan length, low probability of reaching the goal, and (not shown here) very high unsuccessful plan length (caused by wandering in a dead end region) suggest the need for new techniques

Trial #1												
# of features	0	1	2	3	4	5	6	7	7	8	9	10
Problem difficulty	3	3	3	3	3	3	3	3	4	4	4	4
Success ratio	0.56	0.58	0.56	0.63	0.56	0.68	0.62	0.71	0.4	0.45	0.43	0.44
Plan length	1	2	1	2	1	1	2	2	4	5	4	5
Accumulated time (Hr.)	0.6	1.4	2.2	3.1	4.2	5.9	8.7	11	12	20	28	38
Target size #1 SR	0.12	0.12	0.14	0.22	0.20	0.31	0.16	0.34	0.33	0.31	0.31	0.29
Target size #1 Slen.	3	3	3	5	4	6	9	6	6	5	5	5
Target size #2 SR	0	0	0	0.00	0.00	0.03	0	0.02	0.03	0.02	0.02	0.01
Target size #2 Slen.	–	–	–	10	4	24	–	19	26	23	22	15

Trial #2											
# of features	0	1	2	3	4	5	5	6	7	8	9
Problem difficulty	3	3	3	3	3	3	4	4	4	4	4
Success ratio	0.56	0.56	0.55	0.63	0.55	0.75	0.45	0.45	0.43	0.42	0.36
Plan length	1	2	1	2	1	2	4	5	5	4	4
Accumulated time (Hr.)	0.6	1.3	2.1	2.9	3.7	4.6	5.3	14	22	31	39
Target size #1 SR	0.14	0.15	0.12	0.18	0.17	0.33	0.31	0.32	0.31	0.28	0.30
Target size #1 Slen.	4	3	4	6	4	6	6	6	6	5	5
Target size #2 SR	0	0	0	0.01	0.00	0.02	0.01	0.01	0.02	0.01	0.01
Target size #2 Slen.	–	–	–	19	18	26	27	15	21	15	18

Figure 29: EXPLODING BLOCKSWORLD performance (averaged over 600 problems) for relational learner. Problem difficulties are measured in number of blocks. We add one feature per column until success ratio exceeds 0.7, and then increase problem difficulty for the next column. Plan lengths shown are successful trials only. Target problem size #1 has 5 blocks, and target problem size #2 has 10 blocks.

aimed at handling dead-end regions to handle this domain. These results demonstrate that our technique relies on random walk (or some other form of search) so that the learned features need not completely describe the desired policy.

B.4 Towers of Hanoi

We use the domain TOWERS OF HANOI from the first IPPC. In this probabilistic version of the well-known problem, the agent can move one or two discs simultaneously, but there is a small probability of going to a dead-end state on each move, and this probability depends on whether the largest disc has been moved and which type of disc move (one or two at a time) is being used. We note that there is only one planning problem in each problem size here.

It is important to note that 100% success rate is generally unachievable in this domain due to the unavoidable dead-end states.

Trial #1														
# of features	0	1	1	2	3	3	4	5	6	7	8	8	20	38
Problem difficulty	2	2	3	3	3	4	4	4	4	4	4	5	5	5
Success ratio	0.70	0.75	0.11	0.44	0.73	0	0	0	0	0	0.51	0	0	0
Plan length	4	2	43	26	4	–	–	–	–	–	4	–	–	–
Accumulated time (Hr.)	0.0	0.0	0.1	0.2	0.3	0.4	0.5	1.1	1.2	2.1	2.2	2.3	18	53
Target size #1 SR	0.07	0.15	0.01	0.08	0.03	0	0	0	0	0	0.52	0.53	0	0.43
Target size #1 Slen.	13	9	90	95	37	–	–	–	–	–	4	4	–	4
Target size #2 SR	0.00	0	0	0	0.00	0	0	0	0	0	0	0	0	0
Target size #2 Slen.	11	–	–	–	107	–	–	–	–	–	–	–	–	–

Trial #2														
# of features	0	0	1	2	3	3	4	5	6	7	8	8	20	38
Problem difficulty	2	3	3	3	3	4	4	4	4	4	4	5	5	5
Success ratio	0.71	0.23	0.14	0.42	0.75	0	0	0	0	0	0.53	0	0	0
Plan length	4	12	37	25	4	–	–	–	–	–	4	–	–	–
Accumulated time (Hr.)	0.0	0.0	0.2	0.3	0.3	0.4	0.5	1.1	1.9	2.3	2.6	2.7	6	16
Target size #1 SR	0.1	0.09	0.0	0.09	0.03	0	0	0	0	0	0.49	0	0	0
Target size #1 Slen.	14	11	105	95	41	–	–	–	–	–	4	–	–	–
Target size #2 SR	0.00	0.1	0	0	0.00	0	0	0	0	0	0	0	0	0
Target size #2 Slen.	16	29	–	–	107	–	–	–	–	–	–	–	–	–

Figure 30: TOWERS OF HANOI performance (averaged over 600 problems) for relational learner. We add one feature per column until success ratio exceeds 0.7^{n-1} for n discs, and then increase problem difficulty for the next column. Plan lengths shown are successful trials only. Problem difficulties are measured in number of discs, with a target problem size #1 of 4 discs and size #2 of 5 discs. Some columns are omitted as discussed in Section 7.1.

We start with the 2-disc problem in our relational learner and increase the problem difficulty from n discs to $n+1$ discs whenever the success ratio exceeds 0.7^{n-1}. The target problem sizes are 4 and 5 discs. TOWERS OF HANOI results for the relational learner are shown in Figures 19 and 30.

The learner is clearly able to adapt to three- and four-disc problems, achieving around 50% success rate on the four disc problem in both trials. The optimal solution for the four disc problem has success rate 75%. This policy uses single disc moves until the large disc is moved and then uses double disc moves. Policies that use only single disc moves or only double disc moves can achieve success rates of 64% and 58%, respectively, on the four disc problem. The learned solution occasionally moves a disc in a way that does not get closer to the goal, reducing its success.

Unfortunately, the trials show that an increasing number of new features are needed to adapt to each larger problem size, and in our trials even 38 total features are not enough to adapt to the five-disc problem. Thus, we do not know if this approach can extend even to five discs. Moreover, the results indicate poor generalization between problem sizes.

We believe it is difficult for our learner (and for humans) to represent a good value function across problem sizes. Humans deal with this domain by formulating a good recursive policy, not by establishing any direct idea of the value of a state. Finding such a recursive policy automatically is an interesting open research question outside the scope of this paper.

B.5 Lifted-Fileworld3

As described in Section 6.1, we use the domain LIFTED-FILEWORLD3, which is a straightforwardly lifted form of FILEWORLD from the first IPPC, restricted to three folders. To reach the goal of filing all files, an action needs to be taken for each file to randomly determine which folder that file should go into. There are actions for taking out a folder, putting a file in that folder, and returning the folder to the cabinet. The goal is reached when all files are correctly filed in the targeted folders.

We note that both FILEWORLD and LIFTED-FILEWORLD3 are very benign domains. There are no reachable dead ends and very few non-optimal actions, each of which is directly reversible. Random walk solves this domain with success rate one even for thirty files. The technical challenge posed then is to minimize unnecessary steps so as to minimize plan length. The optimal policy solves the n-file problem with between $2n + 1$ and $2n + 5$ steps, depending on the random file types generated.

Rather than preset a plan-length threshold for increasing difficulty (as a function of n), here we adopt a policy of increasing difficulty whenever the method fails to improve plan length by adding features. Specifically, if the success ratio exceeds 0.9 and one feature is added without improving plan length, we remove that feature and increase problem difficulty instead.[13]

We start with 1 file problems in our relational learner and increase from n files to $n + 1$ files whenever the performance does not improve upon feature addition. The target problem size is 30 files. LIFTED-FILEWORLD3 results for the relational learner are shown in Figures 20 and 31.

The results show that our planner acquires an optimal policy for the 30-file target size problem after learning four features, in each of the two trials. The results in this domain again reveal the weakness of our AVI weight-selection method. Although four features are enough to define an optimal policy, as problem difficulty increases, AVI often fails to find the weight assignment producing such a policy. When this happens, further feature addition can be triggered, as in trial 1. In this domain, the results show that such extra features do not prevent AVI from finding good weights on subsequent iterations, as the optimal policy is recovered again with the larger feature set. Nonetheless, here is another indication that improved performance may be available via work on alternative weight-selection approaches, orthogonal to the topic of feature selection.

References

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116*, 123–191.

Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Bonet, B., & Givan, R. (2006). Non-deterministic planning track of the 2006 international planning competition. Website. http://www.ldc.usb.ve/ bonet/ipc5/.

13. It is possible to specify a plan-length threshold function for triggering increase in difficulty in this domain, as we have done in other domains. We find that this domain is quite sensitive to the choice of that function, and in the end it must be chosen to trigger difficulty increase only when further feature addition is fruitless at the current difficulty. So, we have directly implemented that automatic method for triggering difficulty increase.

Trial #1																						
# of features	0	1	2	3	3	4	4	4	4	4	4	4	4	4	5	5	5	6	7	7	7	7
Problem difficulty	1	1	1	1	2	2	3	4	8	10	11	12	13	14	14	15	16	16	16	18	19	20
Success ratio	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Plan length	14	8	4	3	7	6	9	11	21	25	30	29	31	49	37	35	55	37	37	41	43	45
Accumulated time (Hr.)	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	2.4	3.8	4.8	5.9	7.3	8.9	10	13	15	17	19	37	49	62
Target size SR	1	1	1	0	0	0	0	1.00	1.00	1	1	1	1	1	1	1	1	1	1	1	1	1
Target size Slen.	251	134	87	–	–	–	–	87	82	91	88	93	65	90	91	65	91	65	65	65	111	65

Trial #2																						
# of features	0	1	2	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Problem difficulty	1	1	1	1	2	2	3	4	5	8	9	10	14	15	16	17	18	19	20	23	24	25
Success ratio	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Plan length	14	8	4	3	7	6	9	12	14	21	23	25	33	35	62	65	41	43	49	91	53	55
Accumulated time (Hr.)	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.2	0.6	2.5	3.1	3.9	9.0	11	13	19	27	30	34	50	66	74
Target size SR	1	1	1	0	0	0	0	0.96	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Target size Slen.	251	135	88	–	–	–	–	85	88	82	82	91	96	87	91	93	97	65	65	107	82	65

Figure 31: LIFTED-FILEWORLD3 performance (averaged over 600 problems) for relational learner. We add one feature per column until success ratio exceeds 0.9 and adding one extra feature does not improve plan length, and then increase problem difficulty for the next column (after removing the extra feature). Plan lengths shown are successful trials only. Problem difficulties are measured in number of files, with a target problem size of 30 files. Some columns are omitted as discussed in Section 7.1.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 690–700.

Chandra, A., & Merlin, P. (1977). Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pp. 77–90.

Davis, R., & Lenat, D. (1982). *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, New York.

Driessens, K., & Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, *57*, 271–304.

Driessens, K., Ramon, J., & Gärtner, T. (2006). Graph kernels and gaussian processes for relational reinforcement learning. *Machine Learning*, *64*, 91–119.

Džeroski, S., DeRaedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Dzeroski, S., Todorovski, L., & Urbancic, T. (1995). Handling real numbers in ILP: A step towards better behavioural clones. In *Proceedings of the Eighth European Conference on Machine Learning*, pp. 283–286.

Estlin, T. A., & Mooney, R. J. (1997). Learning to improve both efficiency and quality of planning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 1227–1232.

Fahlman, S., & Lebiere, C. (1990). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pp. 524 – 532.

Farias, V. F., & Van Roy, B. (2004). Tetris: A study of randomized constraint sampling. In *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer-Verlag.

Fawcett, T. (1996). Knowledge-based feature discovery for evaluation functions. *Computational Intelligence*, *12*(1), 42–64.

Fern, A., Yoon, S., & Givan, R. (2004). Learning domain-specific control knowledge from random walks. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 191–199.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, *25*, 75–118.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, *9*, 367–421.

Gerevini, A., & Schubert, L. (1998). Inferring state constraints for domain-independent planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 905–912.

Gordon, G. (1995). Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 261–268.

Gretton, C., & Thiébaux, S. (2004). Exploiting first-order regression in inductive policy selection. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pp. 217–225.

Guestrin, C., Koller, D., & Parr, R. (2001). Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 673–680.

Holldobler, S., Karabaev, E., & Skvortsova, O. (2006). FluCaP: A heuristic search planner for first-order MDPs. *Journal of Artificial Intelligence Research*, *27*, 419–439.

Holldobler, S., & Skvortsova, O. (2004). A logic-based approach to dynamic programming. In *Proceedings of the Workshop on "Learning and Planning in Markov Processes–Advances and Challenges" at the Nineteenth National Conference on Artificial Intelligence*, pp. 31–36.

Kakade, S. (2001). A natural policy gradient. In *Advances in Neural Information Processing Systems 14*, pp. 1531–1538.

Kambhampati, S., Katukam, S., & Qu, Y. (1996). Failure driven dynamic search control for partial order planners: An explanation based approach. *Artificial Intelligence*, *88*(1-2), 253–315.

Karalic, A., & Bratko, I. (1997). First order regression. *Machine Learning*, *26*, 147–176.

Keller, P., Mannor, S., & Precup, D. (2006). Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pp. 449–456.

Kersting, K., Van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pp. 465–472.

Kersting, K., & Driessens, K. (2008). Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the Twenty-Fifth International Conference on Machine learning*, pp. 456–463.

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

Lagoudakis, M. G., Parr, R., & Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. In *SETN 02: Proceedings of the Second Hellenic Conference on AI*, pp. 249–260.

Mahadevan, S., & Maggioni, M. (2007). Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, *8*, 2169–2231.

Martin, M., & Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, *20*, 9–19.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, *8*(4), 295–318.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., & Littman, M. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pp. 752–759.

Parr, R., Painter-Wakefield, C., Li, L., & Littman, M. (2007). Analyzing feature generation for value-function approximation. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pp. 737–744.

Patrascu, R., Poupart, P., Schuurmans, D., Boutilier, C., & Guestrin, C. (2002). Greedy linear value-approximation for factored Markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 285–291.

Petrik, M. (2007). An analysis of Laplacian methods for value function approximation in MDPs. In *Proceedings of the twentith International Joint Conference on Artificial Intelligence*, pp. 2574–2579.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Quinlan, J. R. (1996). Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, *5*, 139–161.

Rivest, F., & Precup, D. (2003). Combining TD-learning with cascade-correlation networks. In *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 632–639.

Sanner, S., & Boutilier, C. (2006). Practical linear value-approximation techniques for first-order MDPs. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pp. 409–417.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, *173*(5-6), 748–788.

Singh, S., Jaakkola, T., Littman, M., & Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*(3), 287–308.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Szita, I., & Lorincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural Computation*, *18*, 2936–2941.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68.

Tsitsiklis, J., & Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Utgoff, P. E., & Precup, D. (1997). Relative value function approximation. Tech. rep., University of Massachusetts, Department of Computer Science.

Utgoff, P. E., & Precup, D. (1998). Constuctive function approximation. In Motoda, & Liu (Eds.), *Feature Extraction, Construction, and Selection: A Data-Mining Perspective*, pp. 219–235. Kluwer.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, *7*(1), 81–120.

Widrow, B., & Hoff, Jr, M. E. (1960). Adaptive switching circuits. *IRE WESCON Convention Record*, 96–104.

Williams, R. J., & Baird, L. C. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Tech. rep., Northeastern University.

Wu, J., & Givan, R. (2007). Discovering relational domain features for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pp. 344–351.

Wu, J., Kalyanam, R., & Givan, R. (2008). Stochastic enforced hill-climbing. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pp. 396–403.

Wu, J., & Givan, R. (2005). Feature-discovering approximate value iteration methods. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*, pp. 321–331.

Yoon, S., Fern, A., & Givan, R. (2002). Inductive policy selection for first-order MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 568–576.

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pp. 352–358.

Younes, H., Littman, M., Weissman, D., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, *24*, 851–887.