Discovering Relational Domain Features for Probabilistic Planning

Jia-Hong Wu and Robert Givan
Electrical and Computer Engineering, Purdue University, W. Lafayette, IN 47907
$\{$*jw, givan*$\}$*@purdue.edu*

Abstract

In sequential decision-making problems formulated as Markov decision processes, state-value function approximation using domain features is a critical technique for scaling up the feasible problem size. We consider the problem of automatically finding useful domain features in problem domains that exhibit relational structure. Specifically we consider learning compact relational features without input from human expertise; we use neither expert decisions nor human domain knowledge beyond the basic domain definition. We propose a method to learn relational features for a linear value-function representation—numerically valued features are selected by their fit to the Bellman residual of the current value function and are automatically learned and added to the representation when needed. Starting with only a trivial feature in the value-function representation, our method finds useful value functions by combining feature learning with approximate value iteration. Empirical work presented here for Tetris and for probabilistic planning competition domains shows that our technique represents the state-of-the-art for both domain-independent feature learning and for stochastic planning in relational domains.

Introduction

Complex dynamic domains often exhibit regular structure involving relational properties of the objects in the domain. Such structure is useful in compactly representing value functions in such domains, but typically requires human effort to identify. A common approach which we consider here is to compactly represent the state-value function using a weighted linear combination of state features. The usage of features provided by human experts is often critical to the success of systems using such value-function approximations (e.g. TD-gammon (Tesauro 1995; Sutton & Barto 1998)[1]). Our goal is to derive useful state features in complex dynamic domains automatically instead of relying on human experts. In this paper, we propose and evaluate a relational learning system that finds automatically features useful for approximate value-function representation in highly structured dynamic domains.

[1]The performance of TD-gammon improved to compete with world-class human players after the introduction of a set of human-constructed features.

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. 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 ranking of the states that correlates (or anti-correlates) usefully but imperfectly with the true state value. Our method aims to also find compact features that correlate usefully with true state value.

True state value is intractable to compute directly. Here, we instead compute the Bellman error relative to the current approximate value function. Intuitively, this detects regions of the statespace 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 labelled with a low value which has an action available that leads only to high value states. Our approach is to use machine learning to fit relational features to such regions of local inconsistency in the current value function, learning a new feature. We can then train an improved value function, adding the new feature to the available feature set.

We use Markov decision processes (MDPs) to model the dynamics of domains. We measure inconsistency in Bellman equation caused by using an approximated value function and use a relational machine-learning approach to create new features describing regions of high inconsistency. Our method is novel relative to our previous work (Wu & Givan 2005) due to the following innovations: The method in (Wu & Givan 2005) created only binary features, leveraging only the sign, not the magnitude, of the inconsistency in Bellman equation for states to determine how features are created. In (Wu & Givan 2005) features are decision trees learned by using the algorithm C4.5, representing binary-valued functions. Instead, here, we use a beam-search algorithm to find real-valued features that are relationally represented. The resulting feature language is much richer, leading to more compact, effective, and easier to understand learned features.

Another previous method, from (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. The results shown in (Wu & Givan 2005) suggest that empirically superior feature selection results from fitting the Bellman error of the current approximation, so again here we choose to work directly on fitting that Bellman error.[2]

Some other previous methods (Gretton & Thiébaux 2004; Sanner & Boutilier 2006) find useful features by first identifying goal regions (or high reward regions), then identifying additional dynamically relevant 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, and "optimizations" (in particular, controlling or eliminating overlap between regions and dropping regions corresponding to unlikely paths) must be used to control the number of features considered. 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.

However, effective regression requires a compact declarative action model, which is not always available[3], and is a deductive logical technique that can be much more expensive than the measurement of Bellman error by forward action simulation. Deductive regression techniques also naturally generate many overlapping features and it is unclear how to determine which overlap to eliminate by region intersection and which to keep. The most effective regression-based first-order MDP planner, described in (Sanner & Boutilier 2006) is only effective when disallowing overlapping features to allow optimizations in the weight computation, yet clearly most human feature sets in fact have overlapping features. Finally, repeated regression of first-order region definitions typically causes the region description to grow very large. These issues have yet to be fully resolved and can lead to memory-intensive and time-intensive resource problems. 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 (Sanner & Boutilier 2006) in our empirical results. We also show empirical results for our planner on Tetris, where a declarative PDDL action representation is not available or natural, making FOALP inapplicable.

In (Džeroski, DeRaedt, & Driessens 2001), a relational reinforcement learning (RRL) system learns logical regression trees to represent Q-functions of target MDPs. To date, the empirical results from this line of work have failed to demonstrate an ability to represent the value function usefully in more complex domains (e.g., in the full standard blocksworld rather than a greatly simplified version). Thus, these difficulties representing complex relational value functions persist in extensions to the original RRL work (Driessens & Džeroski 2002; Driessens, Ramon, & Gärtner 2006), where again there is only limited applicability to classical planning domains.

In our experiments we start with a constant value function, and learn new features and weights from automatically generated sampled state trajectories. We evaluate the performance of the policies that select their actions greedily relative to the learned value functions. We demonstrate that our learner generates superior value functions in Tetris comparing to best performance in the RRL work (Driessens, Ramon, & Gärtner 2006) and in (Wu & Givan 2005). Our learner also demonstrates superior ability to learn value functions in the planning domains comparing to the evaluation in the latest RRL work (Driessens, Ramon, & Gärtner 2006), and shows generally superior performance in success ratio comparing to state-of-the-art probabilistic planners FF-Replan (Yoon, Fern, & Givan 2007) and FOALP in the probabilistic planning competition domains.

Background: Markov Decision Process

In this section, we follow the definitions and terminology from (Wu & Givan 2005). For more detail, see (Bertsekas & Tsitsiklis 1996) and (Sutton & Barto 1998). A Markov decision process (MDP) D is a tuple (S, A, R, T) with finite state and action spaces S and A, reward function $R : S \times A \times S \rightarrow \mathbb{R}$, and a transition probability function $T : S \times A \rightarrow \mathcal{P}(S)$ that maps (state, action) pairs to probability distributions over S.

Given discount factor $0 \leq \gamma < 1$ and *policy* $\pi : S \rightarrow A$ for an MDP, 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 other 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

[2]The empirical differences shown in (Wu & Givan 2005) may also have resulted in part from greedy feature construction methods in addition to the differences in the scoring criterion discussed here.

[3]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.

V^π as follows. First, $V^\pi(s) = Q(s, \pi(s), V^\pi)$. Then, $V^*(s) = \max_{a\in A} Q(s, a, V^*)$. 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 $\arg\max_{a\in A} Q(s, a, V)$.

Value iteration iterates the operation $V'(s) = \max_{a\in A} \sum_{s'\in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$, computing the " Bellman update" V' from V, producing a sequence of value functions converging 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 $V'(s) - V(s)$. We will be inducing new features based on their correlation to 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)).

Linear Approximation of Value Functions. We address very large S and/or A by implicitly representing value functions in terms of state space *features* $f : S \to \mathbb{R}$. Here, we assume S and A are *relationally* represented, i.e., that there is a finite set of objects O, state predicates P, and action names N used to define S and A as follows. A *state fact* is an application $p(o_1, \ldots, o_n)$ of an n-argument state predicate p to arguments objects o_i. The state space S is the powerset of the set of all state facts, with each state containing 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. The action space A is the set of all action instances. Given this statespace representation, our features f must select a real value for each set of state facts.

Thus, we represent value functions using a linear combination of features extracted from s, i.e., as $\tilde{V}(s) = \sum_{i=0}^{l} w_i f_i(s)$. 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^*.

Many methods have been proposed to select weights w_i for linear approximations (Sutton 1988; 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) from randomly selected initial states. We can then compute $w_i^{\beta+1}$ from the training states using $w_i^{\beta+1} = w_i^\beta + \frac{1}{n_i}\sum_j \alpha f_i(s_j)(V'(s_j) - V^\beta(s_j))$, 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.

Our feature-discovering AVI method is initialized to have one constant feature with initial weight zero. We then repeatedly perform two steps: adjust the weights using AVI as just described, and then select and add a new feature as described in the next section.

Feature Construction using Relational Function-Approximation

The feature construction approach in (Wu & Givan 2005) selects a Boolean feature attempting to match the sign of the Bellman error, and builds this feature greedily as a decision-tree based on Boolean statespace features. Here, we construct numerically valued relational features from relational statespace structure, attempting to correlate to the actual Bellman error, not just the sign thereof.

We consider any first-order formula with one free variable to be a feature. Such a feature is a function from state to natural numbers which maps each state to the number of objects in that state that satisfy the formula—we normalize to a real number between zero and one by dividing the feature value by the maximum value any such function can map to in a given problem. We select first-order formulas as candidate features using a beam search with a beam width W. The search starts with basic features derived automatically from the domain description or provided by a human, 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 number of 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," which we take to be a function mapping states to their Bellman error. We note 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 V' of the current value function V. This would give us a way to conduct value iteration exactly without enumerating states, except that the Bellman error feature may have no compact representation and if such features are added repeatedly, the resulting value function may in its computation require considering exponentially many states. We view our feature selection method as an attempt to tractably approximate this exact value iteration method.

We construct a training set of states by drawing trajectories from the domain using the current greedy policy Greedy(V), and evaluate the Bellman error feature f' for the training set. Each candidate feature f is scored with its correlation coefficient to the Bellman error feature f' as estimated by this training set. The correlation coefficient between f and f' is defined as $\frac{E\{f(s)f'(s)\} - E\{f(s)\}E\{f'(s)\}}{\sigma_f \sigma_{f'}}$. Instead of using a known distribution to compute this value, we use the states in the training set and compute a sampled version instead. 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 a means for automatically constructing a basic set of features from a relational domain, and a means for constructing more complex features from simpler ones for use in the beam search. For basic features, we first enrich the set of state predicates P by adding 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 we also add a version of each predicate p called goal-p to represent the desired state of the predicate p in the goal, and a means-ends analysis predicate correct-p to represent facts that are present in both the current state and the goal. So, $p+(x, y)$ is true of objects x and y connected by a path in the binary relation p, goal-$p(x,y)$ is always true if and only if $p(x, y)$ is true in the goal region, and correctly-$p(x,y)$ is true if and only if both $p(x, y)$ and goal-$p(x,y)$ are true. The relation max-$p(x)$ is true if object x is a maximal element w.r.t. p, i.e., there exists no other object y such that $p(x, y)$ is true. The relation min-$p(x)$ is true if object x is a minimal element w.r.t. p, i.e., there exists no other object y such that $p(y, x)$ is true.

After this enrichment of P, we take as basic features the one-free-variable existentially quantified applications of (possibly negated) state predicates to variables[4]. 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 and retain the W best scoring features. The new candidate features are created as follows. Any feature in the beam is combined with any other, or with any basic feature. The combination is by moving all existential quantification to the front, conjoining the bodies of the feature formulas, each possibly negated. The two free variables are either equated or one is existentially quantified, and then each pair of quantified variables, chosen one from each contributing feature, may also be equated. Every such combination feature is a candidate.

Experiments

Tetris We run two different experiments on the Tetris domain. We first run feature-discovering AVI on 8×8 Tetris and compare the performance against our previous propositional method, described in (Wu & Givan 2005). In our Tetris domain, a reward is received for each full row of blocks removed, and the next block to fall is chosen uniformly at random. We cannot compare this domain to probabilistic planners requiring PDDL input because we have found no natural PDDL definition for Tetris. We also show the performance of the greedy policy based on the value function learned in the smaller 8×8 Tetris domain when executing in the standard 10×20 size.

We also test a learn-from-small-problems approach by learning initially from 10×5 Tetris, and increasing the number of rows from five by adding two new rows for every 10 learned features up to the size of 10×9, and finally evaluate the performance on the standard 10×20 Tetris using

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

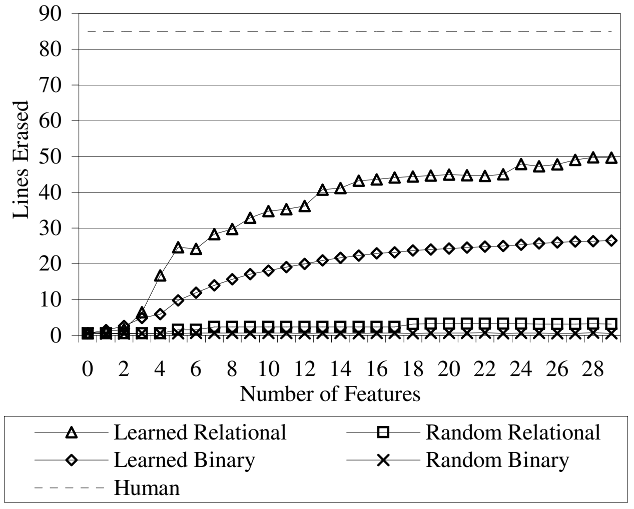

Figure 1: *Plot for the average number of lines erased in 10,000 8×8 **Tetris** games for the learned features. All learned/random features are evaluated by 4 separate trials, averaged together here. The standard deviation at 29 features for the learned integer features is 8.03.*

greedy policies with 29 features in $\tilde{V}$. We compare the performance in both approaches to results in (Driessens, Ramon, & Gärtner 2006), where 10×20 Tetris is used directly.

We represent Tetris using rows and columns as objects. We use three primitive predicates: **fill**(x, y), which means the square on column x, row y is occupied; **below**(x, y), which represents row x is directly below row y; **beside**(x, y), which represents column x is directly to the left of column y. While our representation here uses only primitive domain predicates, the RRL result we compare to uses human-specified Tetris-specific functions in the representation such as "number of holes" (Driessens, Ramon, & Gärtner 2006).

Each Tetris feature-learning training set is generated by drawing trajectories to yield a 20,000 state training set, using Greedy$(\tilde{V})$ from the initial empty grid state and removing duplicate states. Each AVI training set is generated by drawing 50 trajectories from the empty grid state to the end of the game. We set the discount factor γ to 0.9. AVI learning is allowed a total of 1200 iterations each time. The learning rate α is $\frac{1}{1+k/100}$, where k is the number of AVI iterations already executed. The only initial feature is the constant function 0.25.

We plot 8×8-performance as features are learned in Figure 1, together with the performance of AVI-trained human features (taken from (Bertsekas & Tsitsiklis 1996)) and the result of using the propositional feature learning in (Wu & Givan 2005). We also compare our result to integer-valued features randomly selected from the same feature space we learn in.

We find that the features learned by our technique outperform those from the decision-tree learner in (Wu & Givan 2005), even though we learn here from a much smaller training set. Also the performance for the learned features is much better comparing to the randomly generated features.

From each of the four 8×8 learning trials of our new method, we take the final learned value function $\tilde{V}$ and use it as a value function for greedy policy execution in 10×20 Tetris, exploiting the relational generalization our representation enables. The four $10,000$-games averages that resulted were 130, 280, 343, and 573. The four $10,000$-games averages for the learn-from-smaller-problems approach were 557, 390, 135, and 235. For comparison, the best RRL result reported in (Driessens, Ramon, & Gärtner 2006) was around 55. Both of our approaches produce performance far superior to the policy learned by RRL, even though RRL is using human-engineered Tetris-specific features well beyond those in the primitive domain description.

Our performance in 10×20 Tetris is still far worse than that obtained by using the best-known human-selected features with AVI or temporal difference (TD) learning to train weights or that obtained by any good human Tetris player. Our feature learner has still not fully replaced the value of human engineering in selecting features in this domain, though it does produce the best machine-learned policy known to date that is found without exploiting such human engineering of the feature set.

Planning Domains: Overview and Setup We evaluate our method on a range of domains from the first and the second international probabilistic planning competitions (IPPCs). We have restricted consideration to domains that were provided in the competitions with problem generators, rather than simply with a single instance, as our planner requires a problem generator.

We show results on the goal-oriented ("non-reward") version of **blocksworld** and **boxworld** from the first IPPC, as well as **tireworld** from the second IPPC. We find that our planner learns good policies in these domains, and shows superior or equal performance in success ratio in most of the problem instances compared to state-of-the-art probabilistic planners FF-Replan (Yoon, Fern, & Givan 2007) and FOALP (Sanner & Boutilier 2006). In addition, we show results on **zenotravel** and **exploding blocksworld** from the second IPPC; we compare the result on zenotravel to only FF-Replan, since FOALP has not demonstrated success on this domain.

We have also tried our planner in Towers of Hanoi (TOH) from the first IPPC. Our learner fails to obtain a good policy that is able to generalize on this domain. In TOH, our learner is able to achieve success ratio 0.71 and 0.74 over 600 attempts for 3 disc problems for two trials during learning, but the success ratio drops to zero for 4 disc problems. TOH has a recursive-policy solution structure that our value-function feature language is not able to describe. FOALP has also not demonstrated success on this domain. FF-Replan achieves a 0.37 SR over 30 attempts for the 5 disc problem from the first IPPC.

We take a "learning from small problems" approach and run our method in small problems until it performs well, increasing problem size whenever performance measures (success rate and average plan length for randomly generated problems of the current size) exceed thresholds that we provide as domain parameters. Learning initially in small

Trial #1							
# of features	0	1	2	3	3	4	4
# of blocks	3	3	3	3	4	4	5
Success ratio	1	1	1	1	0.94	1	1
Plan length	83	51	52	20	117	17	35
20 blocks SR	0	0	0	0	0	0.98	0.98
20 blocks length	–	–	–	–	–	753	744
# of features	4	4	4	5	4	5	5
# of blocks	10	15	17	17	18	18	18
Success ratio	1	0.97	0.90	0.90	0.86	0.83	0.86
Plan length	166	385	473	471	490	497	505
20 blocks SR	0.99	0.98	0.98	1.00	0.98	0.99	0.98
20 blocks length	752	765	768	790	805	773	738

Trial #2							
# of features	0	1	2	3	3	4	4
# of blocks	3	3	3	3	4	4	5
Success ratio	1	1	1	1	0.94	1	1
Plan length	90	54	53	18	124	18	34
20 blocks SR	0	0	0	0	0	0.98	0.99
20 blocks length	–	–	–	–	–	742	749
# of features	4	4	4	5	6	7	
# of blocks	10	15	18	18	18	18	
Success ratio	1.00	0.97	0.85	0.84	0.87	0.88	
Plan length	179	388	502	522	525	500	
20 blocks SR	0.98	0.99	0.98	0.98	0.98	0.99	
20 blocks length	764	751	743	786	757	743	

Figure 2: ***Blocksworld*** *performance (averaged over 600 problems). We add one feature per column until SR > 0.9 and average successful plan length is less than $40(n-2)$, for n blocks. Plan lengths shown are successful trials only. We omit columns where problem size repeatedly increases with no feature learning, for space reasons and ease of reading. 20 blocks SR uses a longer length cutoff as discussed in the text.*

problems (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) is more effective due to the smaller state space and the ability to obtain positive feedback (i.e., reach the goal) in a smaller number of steps. We find that once a good value function is learned in a small problem, it tends to generalize well enough to large problems to enable additional learning directly in large problems.

In modeling a given planning instance as an MDP, the only relevant feedback to the agent is the non-zero reward when a goal is reached (a reward of 1 is received and a zero-reward absorbing state is entered), or when the agent has no legal action (a reward of -1 is received before a zero-reward absorbing state). The standard PPDDL (Younes *et al.* 2005) domain descriptions for these domains provide the state predicates P needed for our representation.

We have added several mechanisms that empirically appear to help guide AVI to good value functions. Features with weights very close to zero are dropped after AVI. AVI can sometimes severely degrade policy performance and get lost, never restoring it. For this reason, we monitor the performance of the greedy policy as weights are adjusted. If this performance degrades and does not improve we restore the last good weights and lock the magnitude and/or the sign of one or more weights to prevent the weight change that caused the degradation.

We draw feature-learning training sets by generating

Trial #1									
# of features	0	1	2	2	2	2	2	2	2
# of boxes, 5 cities	1	1	1	2	3	5	10	15	17
Success ratio	0.97	1	1	1	1	1	1	1	1
Plan length	241	82	24	36	43	57	77	92	97
(15 BX,5 CI) SR	0.98	1	1	1	1	1	1	1	1
(15 BX,5 CI) len.	1078	359	91	91	90	92	90	90	92
(10 BX,10 CI) SR	0.23	0.88	0.96	0.97	0.96	0.97	0.97	0.97	0.98
(10 BX,10 CI) len.	1636	1031	242	244	244	239	237	233	228

Trial #2									
# of features	0	1	2	2	2	2	2	2	2
# of boxes, 5 cities	1	1	1	2	3	5	10	15	17
Success ratio	0.97	1	1	1	1	1	1	1	1
Plan length	229	82	25	35	43	55	76	91	97
(15 BX,5 CI) SR	0.98	1	1	1	1	1	1	1	1
(15 BX,5 CI) len.	1086	362	91	92	91	92	91	90	91
(10 BX,10 CI) SR	0.22	0.91	0.97	0.97	0.98	0.97	0.97	0.96	0.97
(10 BX,10 CI) len.	1657	1020	239	237	234	233	236	239	235

Figure 3: ***Boxworld*** *performance (averaged over 600 problems). We add one feature per column until SR > 0.9 and average successful plan length is less than $30n$, for n boxes. Plan lengths shown are successful trials only. We omit columns where problem size repeatedly increases with no feature learning, for space reasons and ease of reading.*

Trial #1											
# of features	0	1	2	2	3	3	3	3	3	3	3
# of nodes	4	4	4	5	5	6	10	20	30	40	50
Success ratio	.52	.78	.87	.80	.9	.89	.87	.87	.9	.90	.93
Plan length	4	3	3	4	3	3	3	5	6	7	7
30 nodes SR	.14	.53	.80	.80	.91	.88	.91	.89	.91	.91	.90
30 nodes len.	7	4	9	9	6	6	6	6	6	6	6

Trial #2											
# of features	0	1	2	2	3	3	3	3	3	3	3
# of nodes	4	4	4	6	6	7	10	20	30	40	50
Success ratio	.53	.80	.86	.84	.90	.90	.88	.87	.90	.90	.91
Plan length	4	3	3	3	3	3	3	5	5	7	7
30 nodes SR	.14	.51	.78	.80	.88	.90	.91	.88	.91	.91	.89
30 nodes len.	7	4	9	9	6	6	6	5	6	6	5

Figure 4: ***Tireworld*** *performance (averaged over 600 problems). We add one feature per column until SR > 0.85 and average successful plan length is less than $4n$, for n nodes. Plan lengths shown are successful trials only. We omit columns where problem size repeatedly increases with no feature learning, for space reasons and ease of reading.*

$20,000$ states (5000 in zenotravel) via trajectories of up to 1000 steps drawn using Greedy($\tilde{V}$) on random problems of the current size and removing duplicate states. Training sets for AVI are drawn using 30 trajectories of up to 1000 steps (fewer if a goal is reached), and AVI is allowed to run at most 1000 iterations, stopping if converged. The learning rate α is $\frac{4}{1+k/100}$, where k is the number of AVI iterations already executed. Random exploration is introduced from the 6th to 100th iterations of AVI—action selection by Greedy($\tilde{V}$) has a chance of being replaced by a uniformly randomly selected one, with the probability being set at 10% and decreases when problem size increases without introducing new features. We use a discount factor γ of $1/(1 + (0.1/(n)))$, where n is the total number of objects (including constants) in a problem. The single initial feature is 0.5. As feature-learning continues, the problem size increases, and we show success ratio and plan length on random problems of that size. We also show these measurements for random problems of a selected large problem size to show how generalization to that size is developing.

The results for two complete trials are shown in Figs. 2 thru 6. To interpret these tables, focus first on the two rows labeled "# of features" and "# of blocks" (or other problem size indicator for other domains). These rows show the progress of the trial. Each column in the figures represents the result in the indicated problem size using the indicated number of learned features. When success ratio (SR) and plan length meet the preset quality thresholds, the problem size will increase. Otherwise, a new feature is added and the number of features will increase. As the trial progresses, SR generally increases with number of features but decreases with problem size. In some tables, each trial wraps around to a second set of rows with the same labels. Overall progress on a fixed large problem size can be tracked in the row with a large problem size label[5]. A low plan-length cutoff is used for the SR evaluations during learning in order to speed learning. A longer plan-length cutoff of 2100 steps is used for the comparison to other planners below in Fig. 8 and in the large problem measurements shown during learning.

Blocksworld results are shown in Fig. 2. Our learner consistently finds value functions with perfect or near-perfect success ratio up to 16 blocks. This performance compares very favorably to the recent RRL (Driessens, Ramon, & Gärtner 2006) results in the blocks world, where goals are severely restricted in a deterministic environment, for instance to 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 believe this is due to plateaus in the learned value function, statespace regions where all the selected features do not vary. Future work is planned to identify and address such plateaus explicitly in the algorithm.

The evaluation on 20 block problems shows that a good set of features/weights that is found in small problem size can be generalized to form a good greedy policy in far larger problem size without the need of further training, an asset of using relational learning structure here. In boxworld and tireworld experiments we have similar observations, showing that such ability to generalize is not limited to a single domain.

Boxworld results are shown in Fig. 3. Only 2 features are required to have good performance in problems we have tested. The features learned in boxworld appear to be straightforward and can be easily interpreted in English. The first feature counts how many boxes are correctly at their target city. The second feature counts how many boxes are on

[5]For domains with multi-dimensional problem sizes, it remains an open research problem on how to change problem size in different dimensions automatically during learning. Here in boxworld and zenotravel, we hand-design how the problem size is changed.

Trial #1							
# of features	0	1	2	3	4	4	5
# of CI,PR,AT	3,1,1	3,1,1	3,1,1	3,1,1	3,1,1	3,2,2	3,2,2
Success ratio	0.78	0.80	0.87	0.86	0.98	0.68	0.59
Plan length	252	231	261	248	187	328	241
(10,2,2) SR	0.06	0.10	0.16	0.26	0.61	0.59	0.40
(10,2,2) len.	765	1204	1123	1304	1175	1113	867
# of features	6	6	7	8	9	10	
# of CI,PR,AT	3,2,2	4,2,2	4,2,2	4,2,2	4,2,2	4,2,2	
Success ratio	0.92	0.60	0.45	0.49	0.48	0.61	
Plan length	246	247	218	226	209	266	
(10,2,2) SR	0.87	0.60	0.31	0.33	0.44	0.59	
(10,2,2) len.	762	644	852	750	638	651	

Trial #2								
# of features	0	1	2	3	4	4	5	6
# of CI,PR,AT	3,1,1	3,1,1	3,1,1	3,1,1	3,1,1	3,2,2	3,2,2	3,2,2
Success ratio	0.81	0.79	0.87	0.85	0.98	0.68	0.86	0.97
Plan length	248	288	249	253	174	312	362	258
(10,2,2) SR	0.03	0.10	0.13	0.25	0.65	0.58	0.76	0.96
(10,2,2) len.	760	1169	1129	1237	1193	1164	1085	812
# of features	6	6	6	7	8	8	8	
# of CI,PR,AT	4,2,2	5,2,2	6,2,2	6,2,2	6,2,2	7,2,2	8,2,2	
Success ratio	0.94	0.92	0.88	0.87	0.90	0.92	0.87	
Plan length	324	395	432	432	417	439	473	
(10,2,2) SR	0.98	0.98	0.98	0.97	0.98	0.96	0.97	
(10,2,2) len.	758	752	765	772	736	659	672	

Figure 5: ***Zenotravel*** *performance (averaged over 600 problems). The problem sizes shown in this table correspond to varying number of cities, people, and aircraft. We add one feature per column until SR* > 0.9. *Plan lengths shown are successful trials only. (10 cities, 2 people, 2 aircraft) SR uses a longer length cutoff as discussed in the text.*

trucks.

Tireworld results are shown in Fig. 4. In tireworld, our learner is able to find features that generalize well to large problems. Only 3 learned features achieve a success ratio of around 0.9 in 30 node problems. Our learner is unable to achieve a SR on 30 node problems higher than about 0.9; this limitation also applies to both comparison planners, FOALP and FF-Replan. We note that the plan length results can be misleading in this domain because the goal of minimizing plan length is directly opposed to maximizing SR.

Zenotravel results are shown in Fig. 5. Our learner is able to find features that improve the performance of the planner, but only in one trial the results are well enough to expand beyond 4 cities. The learned policies in the second trial generalize well to larger problems; on the other hand, the learned policies in the first trial generalize well to larger problems at one point, but are not able to maintain such performance. Improving performance in this domain is an interesting research challenge.

Exploding blocksworld results are shown in Fig. 6. Again, here, minimizing plan length is opposed to maximizing SR. Our planner is critically affected by a domain-design choice in which the goal state is described as tower fragments, where fragments are not generally required to be on the table. Our feature language is limited in its ability to describe the properties of a good state relative to this goal. We are considering several general-purpose extensions to our

Trial #1												
# of features	0	1	2	3	4	5	5	6	7	8	9	10
# of blocks	3	3	3	3	3	3	4	4	4	4	4	4
Success ratio	.57	.56	.56	.62	.59	.73	.42	.41	.47	.41	.45	.44
Plan length	2	2	1	2	1	2	4	4	4	4	4	5
5 blocks SR	.13	.13	.12	.22	.16	.33	.33	.32	.34	.33	.32	.28
5 blocks len.	4	4	4	5	3	6	6	6	6	6	6	5

Trial #2												
# of features	0	1	2	3	4	5	5	6	7	8	9	10
# of blocks	3	3	3	3	3	3	4	4	4	4	4	4
Success ratio	.56	.55	.52	.63	.57	.74	.43	.44	.41	.38	.42	.39
Plan length	2	1	2	2	1	2	5	5	4	4	4	4
5 blocks SR	.11	.12	.16	.24	.19	.33	.35	.30	.32	.37	.33	.28
5 blocks len.	4	5	4	5	4	5	5	5	6	5	6	5

Figure 6: ***Exploding blocksworld*** *performance (averaged over 600 problems). We add one feature per column until SR* > 0.72. *Plan lengths shown are successful trials only.*

	BW	(15,5)Box	(10,10)Box	Tire	Zeno	EX-BW
SR	0	0.99	0.47	0.31	0.01	0.18
Length	–	771	1455	6	0	4

Figure 7: *Random features performance averaged over three trials.*

KR inspired by the shortcoming as future research. As a result, our planner's performance in this difficult domain is weaker than in other domains, but still exhibits useful adaptation to the domain and generalization to larger problems.

In order to show that our performance is not simply due to the number of features, but to the feature-selection criterion, we generate three greedy policies in each domain using ten random features, alternating AVI training and feature generation as before, and evaluate the average large-problem performance in each domain using the large problem sizes shown for each domain above. The results are shown in Figure 7. In no domain does random feature generation on average approach the performance our technique shows.

Comparison to FF-Replan and FOALP We compare the performance of our learned policies to FF-Replan and FOALP in blocksworld, boxworld, tireworld, zenotravel, and exploding blocks, in multiple problem sizes. We use the problem generators provided by the planning competitions to generate 30 problems for each tested problem size, and evaluate the performance of each planner 30 times for each problem. We report the results in Fig. 8, where we show the success ratio of each planner in each problem size (averaged over a total of 900 attempts). Our policies, learned from two independent trials, are indicated as RFAVI #1 and RFAVI #2. Each planner has 30 minutes time limit for each attempt, but this time limit is only significant for FF-Replan in blocksworld.

In **blocksworld**, our planner has higher success ratio than both FF-Replan and FOALP. Although the successful plan length tends to be longer comparing to these two planners, our planner is able to find the goal more often. The longer plan length may be due to plateaus as discussed above. FF-Replan fails to scale above 20 blocks. Our improved scaling is likely due to our first-order value function representation as compared to FF-Replan's propositional representa-

	15 blocks BW	20 blocks BW	25 blocks BW	30 blocks BW	20 nodes Tire	30 nodes Tire	40 nodes Tire
RFAVI #1	1 (474)	1 (578)	0.86 (1096)	0.77 (1242)	0.87 (4)	0.83 (7)	0.97 (6)
RFAVI #2	1.00 (477)	1.00 (585)	0.87 (1114)	0.78 (1250)	0.85 (5)	0.86 (7)	0.97 (6)
FF-Replan	0.93 (52)	0.91 (71)	0.7 (96)	0.23 (117)	0.79 (2)	0.71 (3)	0.84 (3)
FOALP	1 (56)	0.73 (73)	0.2 (96)	0.07 (119)	0.92 (4)	0.90 (5)	0.91 (5)

	(10BX,5CI)Box	(10BX,10CI)Box	(10BX,15CI)Box	(15BX,5CI)Box	(10CI,2PR,2AT)Zeno	5 blocks EX-BW
RFAVI #1	1 (73)	0.97 (225)	0.93 (452)	1 (91)	0.87 (803)	0.23 (8)
RFAVI #2	1 (75)	0.97 (221)	0.93 (455)	1 (90)	0.95 (855)	0.20 (10)
FF-Replan	1 (71)	0.98 (245)	0.94 (487)	1 (88)	1 (101)	0.90 (7)
FOALP	1 (35)	0.70 (257)	0.28 (395)	0.99 (56)	N/A	N/A

Figure 8: *Comparison of our planner (RFAVI) against FF-Replan and FOALP. Success ratio for a total of 900 attempts 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.*

tion. FOALP has good performance for 15-blocks problems, but the success ratio drops fast when the number of blocks increases. 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 works well in small problems but does not scale well to large problems.

In **boxworld**, our planner performs similarly to FF-Replan overall, and FOALP again shows a weaker performance. Boxworld is similar to the deterministic logistics, where FF is provably polynomial and very fast and effective. Interestingly our planner is able to find the goal as often by using only two learned features.

In **tireworld**, both first-order approaches perform well, outpacing the propositional FF-Replan.

In **zenotravel**, FF-Replan has a perfect SR as its determinizing technique leads it (somewhat coincidentally) to the optimal policy. FOALP, in contrast, cannot yet plan in this domain. Our results approach but do not equal those of FF-Replan, and are state-of-the-art for first-order approaches. The results shown are for ten cities, two people, and two aircraft. Our planner does not scale much beyond that size, and FF-Replan does.

Our planner does not perform as well as FF-Replan on the **exploding blocksworld**, for reasons discussed above. FOALP has not yet been able to learn a policy directly in the exploding blocks domain, so we do not compare our result to FOALP in that domain.

Overall, in blocksworld, boxworld, and tireworld, our planner has better performance than FOALP and FF-Replan in two out of three domains each, and performs similarly or closely for all problem sizes in the third domain. Our learner shows state-of-the-art performance in these domains, but various research problems remain as discussed in this paper.

Because FF-Replan uses a very different approach, determinizing the problem rather than working with probabilities, it can be expected to perform incomparably to our approach at this stage, as it does. We dominate FF-Replan across several domains, but the reverse holds in zenotravel, exploding blocksworld, and TOH at this time. It is an important topic for future research to try to combine the benefits obtained by these very different planners across all domains.

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).

References

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

Driessens, K., and Džeroski, S. 2002. Integrating experimentation and guidance in relational reinforcement learning. In *ICML*.

Driessens, K.; Ramon, J.; and Gärtner, T. 2006. Graph kernels and gaussian processes for relational reinforcement learning. *MLJ* 64:91–119.

Džeroski, S.; DeRaedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *MLJ* 43:7–52.

Gretton, C., and Thiébaux, S. 2004. Exploiting first-order regression in inductive policy selection. In *UAI*.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *KRR*.

Patrascu, R.; Poupart, P.; Schuurmans, D.; Boutilier, C.; and Guestrin, C. 2002. Greedy linear value-approximation for factored markov decision processes. In *AAAI*.

Sanner, S., and Boutilier, C. 2006. Practical linear value-approximation techniques for first-order mdps. In *UAI*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. MIT Press.

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

Tesauro, G. 1995. Temporal difference learning and td-gammon. *Comm. ACM* 38(3):58–68.

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

Williams, R. J., and Baird, L. C. 1993. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Northeastern University.

Wu, J., and Givan, R. 2005. Feature-discovering approximate value iteration methods. In *SARA*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *UAI*.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*.

Younes, H.; Littman, M.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *JAIR* 24:851–887.