Relational State-Space Feature Learning and Its Applications in Planning

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

Abstract

We consider how to learn useful relational features in linear approximated value function representations for solving probabilistic planning problems. We first discuss a current feature-discovering planner that we presented at the International Conference on Automated Planning and Scheduling (ICAPS) in 2007. We then propose how the feature learning framework can be further enhanced to improve problem solving ability.

Introduction

The use of domain features is important in approximately representing state-values in decision-theoretic planning and reinforcement learning. How to find useful domain features automatically instead of relying on human experts is an interesting question. In this paper, we first introduce a relational feature learning system that finds useful features automatically for linear-approximated value functions in Markov decision processes (MDPs). This part is mainly an excerpt from our previous work (Wu & Givan 2007), titled "Discovering Relational Domain Features for Probabilistic Planning", to appear in International Conference on Automated Planning and Scheduling (ICAPS) 2007. We then propose future research directions on how the feature learning framework can be improved. We focus on discussing how learning features affect problem solving and how improvements can be made in the learning of features [1].

Relational Feature-Discovering Planner

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)). Our goal is to derive useful state features in complex dynamic domains automatically instead of relying on human experts.

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 al-

[1] A similar paper with different future research directions focused on incorporating search techniques in the framework is planned for the ICAPS-07 Workshop on Artificial Intelligence Planning and Learning, which will be held earlier in the year.

gorithm to find integer-valued features that are relationally represented. The resulting feature language is much richer, leading to more compact, effective, and easier to understand learned features.

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, 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.

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 trivial value function (a constant or an integer related to the problem size), 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.

Methodology: 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 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[2]. 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.

Experimental Results We show the results of the blocksworld domain from the first IPPC here in Figure 1 and briefly discuss the experimental environment used in this domain. The non-reward version of the domain is used in the experiments. 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. The performance of learned features is also evaluated on a large problem size, which is 20 blocks here. We use a cutoff of 1000 steps during learning and a cutoff of 2100 for evaluating on large problems and when comparing against other planners.

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

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 1: ***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.*

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.

We also evaluate our method by using Tetris and planning domains from the first and the second international probabilistic planning competitions (IPPCs) in (Wu & Givan 2007). The comparisons in (Wu & Givan 2007) to other methods show that our technique represents the state-of-the-art for both domain-independent feature learning and for stochastic planning in relational domains.

Future Research Directions

Here we discuss ideas for improving our feature learning framework. Learning new features essentially changes the value function representation, and also affects how greedy policies solve planning problems. We discuss several topics on how the learning and representation of features can be changed to improve the problem solving ability of feature-discovering planners.

Removing Value Function Plateaus in State Space We observe in our experiments that the existence of large plateaus may be a cause of long solution length. Here we define a plateau to be a state region where all states in the region are equivalent in estimated value, typically because all selected features are constant within the plateau. When greedy policies are used to select actions, there may be no obvious choice in a plateau and the agent may be forced to "guess". This may lead to a sub-optimal plan.

There can be various reasons why plateaus exist in the state space. Insufficiently many features, restrictive feature language bias, and inadequate exploration may all contribute to a state region not being properly represented. It is possible to eliminate the plateaus caused by lack of features by adding more features to the value function representation in our system; but since our system learns features by considering the inconsistency in the Bellman equation over the entire state space, there is no guarantee that plateaus can be identified and removed in this way.

Instead of features that represent the dynamic structure in the overall state space, we may learn features that focus on certain plateaus in the state space. Such features require a two-level hierarchical representation, where the first level identifies the state regions containing the plateaus, and the second level represents the dynamic structure not identified previously in those regions. The learning of the first-level structure is essentially a classification task. If we regard the state regions identified in the first-level structure as a new state space, the second-level structure may be learned using a method that learn global features (such as the one we presented earlier in the paper) by forming a sub-problem containing the new state space.

Modelling Solution Flows using Features The two-level feature hierarchy can also be used in representing other dynamic structures in planning and reinforcement learning. Instead of using the first level to identify certain regions of state space, we can use it to identify solution "flows" (i.e. how problem solutions progress in the state space). The second level of a feature then represents how an agent can act during a certain point (stage) in the flow.

A solution flow may be identified by considering successful solution trajectories and separating states in the trajectories into groups according to the characteristics of the states and actions taken from the states. For example, a common tactic that human use to solve blocksworld is to first put all blocks on the table and then assemble the correct towers. A solution flow of two stages can be identified here by observing how the height of the towers changes and how actions on the blocks are taken.

Generalization in Feature Learning In (Wu & Givan 2007), we demonstrate that the features learned in our system can be generalized to large problems. The "learning from small problems" approach we used is effective here as well as in previous researches (Martin & Geffner 2000; Yoon, Fern, & Givan 2002) due to the smaller state space and the ability to obtain positive feedback (i.e., reach the goal) in a smaller number of steps. The use of a relational feature language is the main reason why we can use the same features in problems of different sizes.

One issue when learning features in small problems is that the resulting features may "overfit" the small problems and not be able to describe a useful concept in general sizes. For example, if we learn in 3-blocks blocksworld problems, we may find features that correctly describe how the 3-blocks blocksworld works but are not useful in 4 or more blocks problems. This may happen when attempts are made to improve the plan quality beyond some limit. How to detect and prevent this type of "over-fitting" automatically may be an interesting research topic.

Some domains have multiple size-dimensions where objects in some of the dimensions are more independent of each other than objects in other dimensions. For example, in the probabilistic planning domain boxworld, a problem size consists of 2 dimensions: the number of boxes and the number of cities. If all cities are reachable from any city, we can move the boxes to their destination one at a time, essentially decomposing a problem into subproblems centering on a certain object (box) where each subproblem has a simpler subgoal and such subgoal can be reached without affecting whether other subproblems can be solved. Features can be learned to represent subproblems instead of the more complex original ones. The feature-learning planner FOALP (Sanner & Boutilier 2006) uses this approach by learning features for generic subproblems for a domain and problems of different sizes in the same domain are solved using the solutions to the generic subproblems. However, some domains such as blocksworld have interacting subgoals and learning features for subproblems may not be desired. In blocksworld, features that attempt to maximize the correct *on* relations may be insufficient in a situation where a huge tower has an incorrectly placed block at the bottom but all others are in correct order. How to correctly identify whether and how problems in a domain can be decomposed is therefore critical in using feature learning in problem solving.

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*.

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.

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

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

Wu, J., and Givan, R. 2007. Discovering relational domain features for probabilistic planning. In *ICAPS*. To Appear.

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