FEATURE-DISCOVERING APPROXIMATE VALUE ITERATION METHODS

A Technical Report

Submitted to the ECE Department

of

Purdue University

by

Jia-Hong Wu and Robert Givan

School of Electrical and Computer Engineering

465 Northwestern Ave.

Purdue University

West Lafayette, IN 47907

November, 2004

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We study the problem of automatically selecting problem features to use in approximately representing value in Markov decision processes. We propose and evaluate a simple approach reducing the problem of selecting a new feature to standard classification learning—we learn a classifier that predicts the sign of the Bellman error over a training set of states. By iteratively adding new classifiers as features with this method, training between iterations with TD-learning, we find a Tetris feature set that outperforms randomly constructed features significantly, and obtains a score of about one-third of the highest score obtained by using a carefully hand-constructed feature set. We also show that features learned with this method outperform those learned with the previous method of Patrascu et al. [1] on the same SysAdmin domain used for evaluation there.

1. Introduction

Decision-theoretic planning and reinforcement learning methods facing astronomically large state spaces typically rely on approximately represented value functions (see, e.g., [2, 3]). Many such approximate representations rely on an appropriate set of problem features; for example, by taking a weighted combination of the feature values as the value function [4]. Human engineering of the problem features used has repeatedly proven critical to the success of the resulting system, e.g. in [5].

Here, we study the problem of automatically selecting problem features for use in approximately representing value in Markov decision processes. We focus our initial work on this problem on binary features, i.e., mappings from state to Boolean values. We view each such feature as a set of states, those states where the feature is true.

We propose and evaluate a simple, greedy approach to finding new binary features for a linear-combination value estimate. Our heuristic approach assumes an initial "base" value estimate described by a linear approximation where the weights have already been tuned to minimize Bellman error. We attempt to reduce the Bellman error magnitude of this value estimate further by learning a new feature that is true in statespace regions of positive statewise Bellman error and correspondingly false in regions of negative statewise Bellman error, or vice versa. The learning problem generated is a standard supervised classification problem, and for this work we address this problem using the decision-tree learner C4.5 [6].

One view of this approach is that we are conducting approximate value iteration with an added mechanism for extending the available feature set. Given an initial feature set, imaging a sufficient period of approximate value iteration (or any similar weight adjustment method) to achieve convergence of the approximation to a value function $\tilde{V}$. We can think of the approximate value iteration process as "stuck", in that it can represent $\tilde{V}$ but not the

Bellman update of $\tilde{V}$. (Of course, this assumes that the updates being performed would find $\tilde{V}$ if they could represent it, which is only heuristically true.) We are then trying to induce features to enable representation of the Bellman update of $\tilde{V}$, so that the approximate value iteration process can continue to reduce Bellman error, with the larger feature space.

If the learner succeeds in capturing features that describe the statespace regions of positive and negative Bellman error, we can guarantee that adding these features makes available weight assignments closer to the Bellman update of the base value estimate. Our practical method retrains the weights including the new feature(s), using TD-learning, and then repeats the process of selecting a new feature.

We have found surprisingly little previous work in this area. Patrascu et al. [1] give a linear programming technique for selecting new features to reduce the L_1 error of the approximation. Our technique works instead directly to reduce the Bellman error magnitude, but relies on classification learning. We show below that our technique empirically outperforms this previous work on the planning domain used in their evaluation. Specifically, we require fewer new features to achieve the same Bellman error, and can achieve a lower overall Bellman error given enough features.

We also evaluate our technique in the computer-game domain of Tetris. Starting from a constant value function based on only the uniformly true feature, our technique can add features automatically to produce performance that is significantly better than a randomly constructed feature set, and is at about one-third of the performance of a carefully hand-constructed feature set.

In what follows, we first provide technical background on Markov decision processes and value-function approximation, then describe our technique for inducing new features to reduce approximation error, and finally present empirical results on two domains showing improvement over the state of the art, before concluding.

2. Technical Background

2.1 Markov Decision Processes

We define here our terminology for Markov decision processes. For a more thorough discussion of Markov decision processes, see [2] and [3]. A Markov decision process (MDP) D is a tuple (S, A, R, T) where state space S is a finite set of states, action space A is a finite set of actions, $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, and $T : S \times A \rightarrow \mathcal{P}(S)$ is the transition probability function that maps (state, action) pairs to probability distributions over S. $R(s_1, a, s_2)$ represents how much immediate reward is obtained by taking action a from state s_1 and ending up in state s_2. $T(s_1, a, s_2)$ represents the probability of ending up in state s_2 if the action a is taken from state s_1.

A *policy* π for an MDP is a mapping $\pi : S \rightarrow A$. Given policy π, the value function $V^{\pi}(s)$ gives the expected discounted reward obtained starting from state s and selecting action $\pi(s)$ at each state encountered. Rewards after the first time step are discounted by a factor γ where $0 \leq \gamma < 1$. A Bellman equation relates V^{π} at any state s and successor states s':

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

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. Another Bellman equation governs V^*:

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

From any value function V, we can compute a policy Greedy(V) that selects, at any state s, the greedy look-ahead action $\arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$. The policy Greedy($V^*$) is an optimal policy. *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 V' from V, producing a sequence of value functions converging to V^*, regardless of the initial V used.

We define the statewise Bellman error $E(V, s)$ for a value function V at a state s to be $\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] - V(s)$. We will be inducing new features 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 $\max_{s \in S} |E(V, s)|$.

Linear Approximation of Value Functions. We assume that the states of the MDP have structure. In particular, we assume a state is a vector of basic properties with Boolean, integer, or real values, and that the state space is the set of all such vectors. We call these basic properties *state attributes.* This factored form for states is essential to enable compact representation of approximate value functions.

A common solution to the problem of representing value functions (e.g., value iteration) in very large, structured state spaces is to approximate the value $V(s)$ with a linear combination of features extracted from s, i.e., as $\tilde{V}(s) = \sum_{i=0}^{p} w_i f_i(s)$, where w_i is a real-valued *weight* for the ith feature $f_i(s)$. Our goal is to find features f_i (each mapping states to boolean 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 [7, 8]. Here, we use a heuristic temporal-difference (TD) learning method described in [2] that has performed well in our experiments. Other training methods can be substituted and this choice is orthogonal to our main purpose.

The TD method we deploy constructs a fixed-length sequence of value functions V^1, V^2, ..., V^T, and returns the last one. Each value function V^β is defined by weight values w_0^β, w_1^β, ..., w_p^β as $V^\beta(s) = \sum_{i=0}^{p} w_i^\beta f_i(s)$. Value function $V^{\beta+1}$ is constructed from V^β by drawing a training set of trajectories[1] under the policy Greedy(V^β) and updating the weights according to this training set as follows.

Let s_1, s_2,..., s_n be a trajectory. Define the weight update for the ith weight and state occurrence $s_j \neq s_n$ in this trajectory to be $\alpha f_i(s_j) \sum_{m=j}^{n-1} \lambda^{m-j} d_m$, where d_m is defined as $\gamma \tilde{V}(s_{m+1}) + R(s_m, \pi(s_m), s_{m+1}) - \tilde{V}(s_m)$. Here, α is the learning rate parameter and λ is

[1] The source of this set is a parameter of the algorithm, and it could for example be drawn by sampling initial states from some state distribution and then simulating π to some horizon from each initial state.

the parameter for TD learning as defined in [7]. Then, for each i, the weight updates for all states in all trajectories are averaged and then added to w_i^{β} to get $w_i^{\beta+1}$.

2.2 Decision Tree Classification

A detailed discussion of classification using decision trees can be found in [9]. A decision tree is a binary tree with internal nodes labelled by state attributes (and, in our case, learned features), and leaves labelled 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 partial assignment to the state attributes—each state consistent with that partial assignment is viewed as labelled l by the tree. We learn decision trees from training sets of labelled states using the well known C4.5 algorithm [6]. 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.

3. Feature Construction for MDPs

We propose a simple method for constructing new features given a current set of features and an MDP for which we desire an approximation of V^*. We first use TD-learning, as described above, to select heuristically best weights to approximate V^* with $\tilde{V}$ based on the current feature set. We then use the sign of the statewise Bellman error at each state as an indication of whether the state is undervalued or overvalued by the current approximation. If we can identify a collection of undervalued states (ideally, all such states) as a new feature, then assigning an appropriate positive weight to that feature should reduce the Bellman error magnitude. The same effect should be achieved by identifying overvalued states with a new feature and assigning a negative weight. We note that the domains of interest are generally too large for statespace enumeration, so we will need classification learning to generalize the notions of overvalued and undervalued across the statespace from training sets of sample states. Also, to avoid blurring the concepts of overvalued and undervalued with each other, we discard states with statewise Bellman error near zero from either training set.

More formally, we draw a training set of states Σ from which we will select training subsets Σ_+ and Σ_- for learning new features. The training set Σ can either be drawn uniformly at random from the state space, or drawn by collecting all states in sample trajectories starting at uniformly random start states under a policy of interest (typically Greedy$(\tilde{V})$). If using trajectories, each trajectory must be terminated at some horizon. The horizon and the size of Σ are parameters of our algorithm.

For each state s in Σ, we compute the statewise Bellman error $E(\tilde{V}, s)$. We then discard from Σ those states s with statewise Bellman error near zero, i.e., those states for which $|E(\tilde{V}, s)| < \delta$ for a non-negative real-valued parameter δ, and then divide the remaining states into sets Σ_+ and Σ_- according to the sign of $E(\tilde{V}, s)$. So, Σ_+ is the set $\{s|E(\tilde{V}, s) \geq \delta\}$ and Σ_- is the set $\{s|E(\tilde{V}, s) \leq -\delta\}$.

We note that computing statewise Bellman error exactly 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 expectation can be effectively approximately evaluated by sampling next states according to the distribution represented by T.

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 is built from the primitive attributes defining the state space; in our case, we use decision trees over these attributes. We can also interchange the roles of Σ_+ and Σ_-, using the latter as positive examples. In our experiments, we do this interchanging for every other feature constructed.

The concept resulting from supervised learning is then treated as a new feature for our linear approximation architecture, with an initial weight of zero. The process can then be repeated, of course, resulting in larger and larger feature sets, and, hopefully, smaller and smaller Bellman error magnitude.

To conclude our description of our algorithm, we discuss setting the parameter δ dynamically, once in each iteration of feature construction. Rather than directly specify δ, we specify δ in terms of the standard deviation σ of the statewise Bellman error over the same distribution used in selecting states for the training set Σ. The value of σ is easily estimated by sampling the training distribution and computing the Bellman error. We then set δ, at each iteration, to be a fixed multiple η of σ. This approach removes δ as a parameter of the algorithm, replacing it with the parameter η. This dynamic selection of δ allows adaptation to the decrease in Bellman error magnitude over the run of the algorithm.

4. Experiments

In this section, we present some experimental results for our feature construction algorithm. We us two domains in the experiments. The first domain is an 8×8 game of Tetris (Tetris). The second domain is a computer network optimization problem called SysAdmin, which we use primarily in order to compare to the closest previous related work; that work [1] used SysAdmin as a testing domain. Both the state attributes and the learned features in the experiments are binary features.

Tetris. For the Tetris domain, we start with 71 state attributes; 64 attributes which represent if the 64 squares are occupied or not, and 7 attributes which represent which of the 7 pieces is currently being dropped. We select training sets for feature construction by drawing trajectories from an initial state with an empty board and collecting 600,000 states on these trajectories as Σ. The training sets for TD learning are selected by drawing 100 trajectories from an initial state with an empty board and allowing each trajectory to extend to the end of the game. We draw the trajectories using the Greedy$(\tilde{V})$ policy. The discount factor γ is 0.9 for this experiment, and the parameter η is set to 0.3. In addition, λ is fixed at 0 and α at 0.01. TD learning is assumed to converge after 1200 training sets are drawn; at that point, a new feature is learned.

The results are shown in Figure 4.1. The score is determined by the average number of lines erased during a sequence of games. The performance of the learned features are evaluated by the 2,000-game average score for Greedy$(\tilde{V})$ using the weights learned just after TD-learning. Figure 4.1 displays the average of such evaluations over 4 separate trials of feature learning. In addition, we also show in Figure 4.1 the result of using sets of randomly generated features; such features are generated following the same procedure described in the previous section, but label examples in the training set Σ randomly instead of deciding the labeling by statewise Bellman error. Value functions constructed from randomly generated features perform poorly, and do not show improvement as the number of features used increases. Thus, our use of statewise Bellman error to label the training examples plays an important role in the performance of our feature construction algorithm.

We also tested TD learning on human-constructed features in this domain. The features we used in this case were provided by Bertsekas in [2]. These features are useful features as considered by a human, and according to [2] they were selected after some testing. We tested the performance of the weights learned after each TD iteration by running 2,000 games and taking the average score. The maximum 2,000-game average performance was 91, which was achieved after nearly 30,000 iterations of TD training. This performance was substantially better than the best performance our learned feature set exhibited, which was 29 (using 45 learned binary features).

We note that the human-selected features are all integer-valued, apparently giving the human set a clear advantage (especially per feature). Clearly one approach for futher improvement in feature learning is to design a feature-learning approach that can produce integer-valued features.

SysAdmin. For the SysAdmin domain, two different kinds of topologies are tested: 3-legs and cycle. There are 10 nodes in each topology. We follow the settings used in [1] for testing this domain. The target of learning in this domain is to keep as many machines operational as possible, since the number of operating machines directly affects the reward for each step. Since there are only 10 nodes, the on/off status of each node is used as a basic feature, which means there are a total of 1024 states. We simply use all states as the training set for feature construction. To enable direct comparison to the previous work in [1], we use Bellman error magnitude to measure the performance of the feature construction algorithm here.

For the experiments that use the whole state space as a training set, the plot of average Bellman error for 10 separate trials over the number of features learned is shown in Figure 4.2. We used γ equal to 0.95, η equal to 1, α equal to 0.1 and λ equal to 0. In this experiment there are 50 trajectories drawn in each TD training set, each drawn from

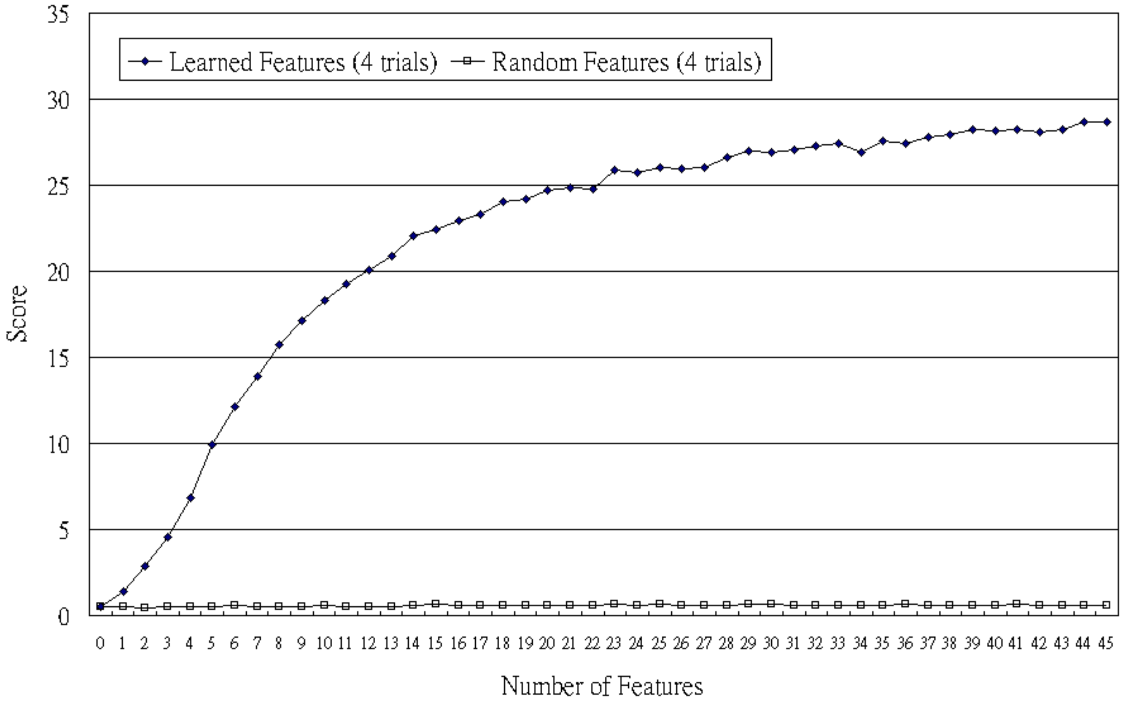

Fig. 4.1.: Score (average number of lines erased in 2,000 games) plot for the learned features and randomly generated features in 8×8 Tetris domain. For reference, the maximum score for the human-selected feature set from [2] was 91.

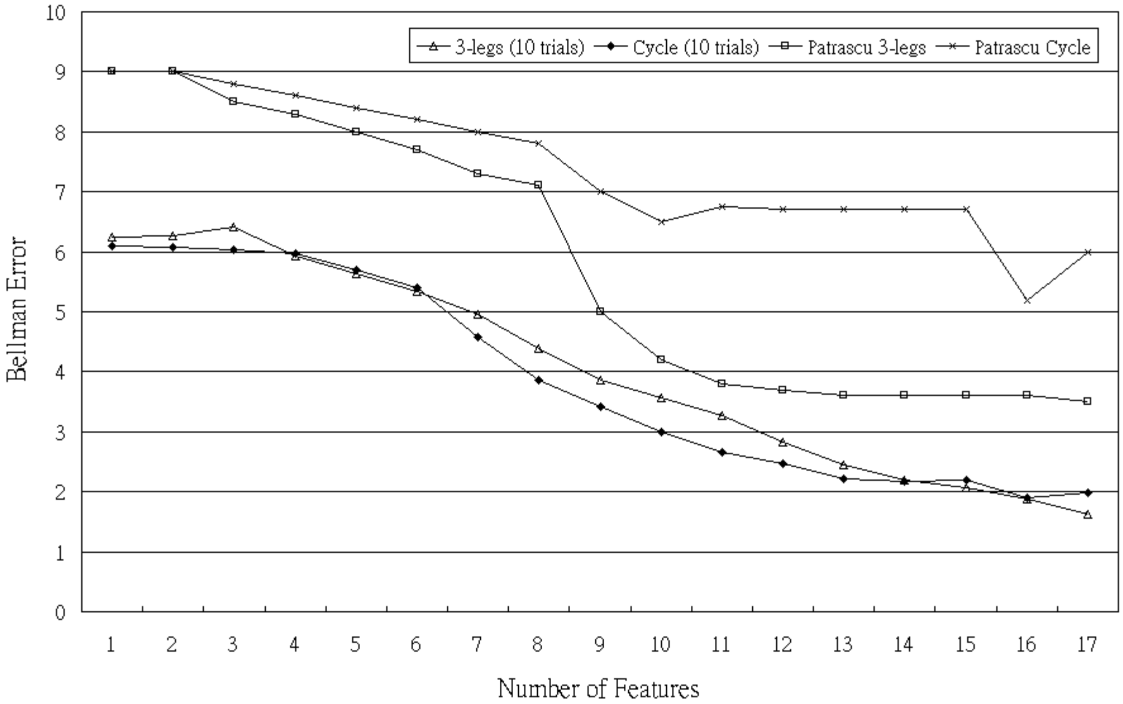

Fig. 4.2.: Bellman error for SysAdmin domain (10 nodes).

a random initial state, and using trajectory length 2. TD learning was considered to have converged after 8000 iterations.

Also included in Figure 4.2 are the results from [1]. We select the best result they show (from various algorithmic approaches) from the 3-legs and cycle domains shown in their paper (their “d-o-s” setting for the cycle domain and their “d-x-n setting” for the 3-legs domain).

Compared to the results in [1], also shown here, our feature construction algorithm achieves a lower Bellman error magnitude in these domains for the same number of features, throughout, and a lower converged Bellman error magnitude when new features stop improving that measure. This is another encouraging result for this proposed feature construction algorithm.

5. Conclusions and Future Work

From the experiments, the results show that our feature construction algorithm can generate features that show significantly better performance in 8×8 Tetris than randomly constructed features, and can produce features that outperform the features produced by the algorithms in [1] for the SysAdmin domain. However, our algorithm cannot learn a feature set for 8×8 Tetris that competes well with the human-constructed feature set provided in [2].

Our technique depends critically on the generalization ability of the classification learner to cope with large state spaces. The features generated by the feature-construction algorithm currently are represented as decision trees. Although the experiments showed that these features are useful in some problems, they are still hard to interpret. One goal for designing a good feature-construction algorithm is to be able to produce features that are understandable by humans. One way we are considering to improve our algorithm in this direction is to use a relational classification algorithm combined with an interesting knowledge representation instead of using C4.5 with decision trees.

6. Acknowledgement

The authors would like to thank Alan Fern for useful discussions and input.

LIST OF REFERENCES

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

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

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

[4] R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation – a new computational technique in dynamic programming. *Math. Comp.*, 17(8):155–161, 1963.

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

[6] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

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

[9] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.