Non-parametric Fitted Relational Value Iteration: Unifying Relational and Propositional Discrete Domains

Paper ID 3955

Relational Learning, Adaptive Dynamic Programming, Huber Loss, Functional Gradient Boosting

Abstract

We consider the problem of Approximate Dynamic Programming in relational domains. Inspired by the success of fitted value iteration methods in propositional settings, we develop the first relational fitted value iteration method by representing the value function as a linear combination of relational regression trees. We show how the two steps of Bellman operator application and projection steps can be performed using a gradient-boosting technique. Our proposed framework can be seen as a unified framework that can be effectively applied to both relational and propositional discrete domains.

Introduction

Value function approximation in Reinforcement Learning (RL) has long been viewed using the lens of feature discovery (Parr et al. 2007). A set of classical approaches for this problem based on Adaptive Dynamic Programming (ADP) is the fitted value iteration (Boyan and Moore 1995; Ernst, Geurts, and Wehenkel 2005; Riedmiller 2005), a batch mode approximation scheme that employs function approximators in each iteration to represent the value estimates. Another popular class of methods that address this problem is the Bellman error based methods (Menache, Mannor, and Shimkin 2005; Keller, Mannor, and Precup 2006; Parr et al. 2007). The key intuition is that the Bellman error points to the true value function and thus learning a large number of the Bellman error basis functions can yield a good approximation to the true values. There has been significant progress in these two research directions for the classic RL problems with discrete and continuous state spaces.

We here consider relational domains that are typically described using parametrized state-action spaces. The intuition in these domains is that the world consists of interacting objects and is richly relational. A class of RL methods called Relational Markov Decision Process (RMDPs) have been developed for learning optimal behaviour in these worlds. While it is conceivable to instantiate each object and construct a grounded MDP, given the variable number of objects, this can yield substantially large state-action spaces that render solving the grounded MDP intractable in practice. Also, as noted by Tadepalli et al. (2004), function approximators such as Neural Networks do not generalize well when applied to relational domains.

Consequently, methods that directly learn and reason at the first-order level have been developed in the broad area of Relational Reinforcement Learning (Price and Boutilier 2001; Sanner and Boutilier 2009; Wang, Joshi, and Khardon 2008; Tadepalli, Givan, and Driessens 2004; Wu and Givan 2007). While specific methodologies differ, most of these methods operate at a symbolic level and define the mathematical operations on these symbols to learn the values of the parametrized states (which can essentially be viewed as a "group" of states in classical RL). Most of these methods, however, are exact methods. Guesterin et al. (Guestrin et al. 2003) developed approximate learning for RMDPs based on representing value functions for classes of objects and learning these stage-wise value functions that are later combined to obtain the approximations of the true value functions.

Inspired by the success of approximate value function learning for propositional domains, we here propose an approximate value function learning method for relational domains. The key idea in our work is that we represent the value functions as a set of relational regression trees (RRTs) (Blockeel and De Raedt 1998) learned using gradient-boosting (Friedman 2001). The value of the current state can be obtained by simply evaluating the trees based on the instantiations of the objects in the current state. The projection step of the algorithm would then correspond to learning these boosted trees based on the current values for all the states. To this effect, we adapt the recently successful Relational Functional Gradient Boosting (RFGB) algorithm (Natarajan et al. 2014) for learning these projections.

Our work can be seen as learning functions over relational domains that permit the efficient combination of these functions without explicitly considering the entire space of models. This is achieved by *lifting* or *aggregating*, i.e grouping together of objects that are indistinguishable when seen through the lens of the functions involved in the operation. Wu and Givan (2007) approximate the value function by using beam search with relational features that are iteratively learned from sampled trajectories. Our approach uses gradient boosting where we learn sets of conjunctive features as paths from the root to the leaf of each RRT and lifts the arguments of Ernst, Geurts, and Wehenkel (2005) and Tosatto et al. (2017) from the propositional to the relational level.

Indeed, if we knew that the target value function belongs to a specific class and we use this information to model the value function there is no need of trees and boosting. However, in relational domains this information is almost never available: without difficult feature engineering, the shape of the value function is almost always unknown.

More precisely, we make the following key contributions:

1. We develop a unified framework for handling relational, semi-relational and propositional RL problems by adapting a successful relational algorithm.
2. To our knowledge, this is the first work on learning relational basis functions directly in a sequential manner.
3. We outline the connections between our proposed approach to the classical RL methods for value function approximation (Bellman error, aggregate and tile coding methods). We showed that our method can be viewed as encompassing these different approximation techniques.
4. As far as we are aware, we are the first to explore multiple loss functions, such as LAD and Huber loss inside gradient boosting for learning relational value functions.
5. Finally, we demonstrate empirically the effectiveness and the generalization ability of our proposed approach over classical methods in propositional, semi-relational and fully relational domains. This is sensible as without extensive feature engineering, it is difficult—if not impossible—to apply them within relational domains in which there is a varying number of objects and relations.

The rest of the paper is organized as follows. After introducing the necessary background on RL, value function approximation and relational learning in the next section, we outline the algorithm and analyze its properties in Section 3. Next we present the empirical evaluation of the proposed approach on three domains before concluding by outlining areas of future research.

Background

Before introducing the unified framework, let us briefly review the mathematical background required.

Markov Decision Processes (MDPs): An MDP is described by a set of discrete states S, a set of actions A, a reward function $R(s, a)$ that describes the expected immediate reward in state s when executing an action a, and a state transition function $p^a_{ss'}$ that describes the transition probability from state s to state s' under action a. For infinite horizon problems a discount factor γ is specified to trade-off between current and future reward. A policy π is a mapping from states to actions. The optimal value (expected reward) of a particular state is given by the Bellman optimality equation:

$$V^*(s_i) = max_a\{R(s_i, a) + \gamma \sum\nolimits_j p^a_{s_i,s_j} \cdot V^*(s_j)\} \quad (1)$$

For the general case of any policy π, the max operator is dropped from the above equation. The Bellman operator T^* can be used to define $V^* = T^*V$ for simplifying operations on value functions and can be shown to be a contraction (Parr et al. 2007). Given two vectors V_1 and V_2

$$\max(V_1 - V_2) = d_{\max} \Rightarrow \max(TV_1 - TV_2) \leq kd_{\max}, \quad (2)$$

for some $k \leq \gamma$. The $\max$ difference or the infinity norm between the vectors after application of the operator monotonically decreases. The Bellman error for each state is defined as the difference between the previous value and the value after application of the Bellman operator, $TV - V$.

Approximate Dynamic Programming (ADP): For large state spaces (or continuous states), it is not possible to use a tabular representation for states and they are factored into features $f \in F$. However, in many large problems (structured ones specifically), the size of F can be prohibitively large for performing exact policy evaluation. Hence, the value function can be approximated using a linear combination of state features (Boyan and Moore 1995).

Let there be a set of weights $w \in W$ corresponding to each feature in F, then the approximate value of a state s_i described using features in F with weights W is:

$$\hat{V}(s_i) = W \cdot F \equiv \hat{V}(s_i) = \sum\nolimits_k w_k \cdot f_k. \quad (3)$$

The features, also called *basis functions*, are ideally linearly independent. Least Squares Temporal Difference (LSTD) Learning (Boyan 1999) and Least Squares Policy Evaluation (LSPE) (Yu and Bertsekas 2009) solve for this approximate value as a function of these basis functions.

According to the Bellman equation, as already mentioned, the value of a state following a particular policy is equal to the reward at that state plus the discount expected value of the next state. In fitted value iteration, we can approximate the value (TV) using a function approximator. Typically, a linear combination of features defined by Φ is used to defined the value of the state as $\Phi \cdot w$, where w are weights on the features. In our method however, the approximation is non-parametric. The approximation of the value of the state is a linear combination of relational regression trees as opposed to fixed features a priori. This lends itself to efficient calculation in relational domains by a technique called Relational Functional Gradient Boosting (RFGB) (Natarajan et al. 2012).

The ADP framework is a collection of all such established RL methods where the values or policies are approximated using information from samples. LSTD for example uses the least squares approximation to the temporal difference learning paradigm. Our proposed contribution can be well understood w.r.t fitted value iteration where the value iteration step is carried out through non-parametric approximation through samples. The main component of ADP approaches such as LSTD is the computation of weights for the approximation of the value function. If it is possible to compute non-parametric basis functions that fit the Bellman error directly, calculation of weights for each iteration of LSTD can be avoided and their values can be computed directly. In relational domains, groups of states are typically indistinguishable w.r.t. the values that they assume aka *parameter sharing*. For example, in a Wumpus World, many grid locations could have similar values as they exhibit similar properties. Consequently, it is possible to learn a *lifted representation* such as the one by Wu et al. (2007) that can reason over sets of feature values.

As mentioned earlier, we employ Functional Gradient Boosting (Friedman 2001) in relational domains (Getoor

2007; De Raedt et al. 2016) to capture the Bellman error for every iteration of Approximate Dynamic Programming to obtain *lifted* basis functions in a non-parametric fashion. In domains where we require probabilistic policies, a Gibbs distribution over the values can be used to model probabilistic action choices. While it is conceivable to learn policies directly as performed by Kersting and Driessens (2008) and Natarajan et al. (2011), our method can retrieve all probabilistic policies instead of the optimal one and guarantee that the learned basis functions are orthogonal to the existing basis functions F that is not guaranteed in prior work.

Gradient-Boosted Relational Adaptive Dynamic Programming (GBAD)

We consider learning to act in relational, noisy domains where the domains consist of multiple interacting objects. To this end, we employ Statistical Relational Models (SRMs). They combine the expressive power of relational models to represent complex relationships and the ability of probability theory to model uncertainty. As mentioned earlier, we use RFGB (Natarajan et al. 2014; 2012; 2011) for approximating the value functions in relational domains. The key idea in RFGB is that like its propositional counterpart, a functional representation is used to model the target distribution (typically using a sigmoid function). The gradients are obtained w.r.t. this function and are approximated using regression functions at each step of gradient computation. In relational domains, these regression functions are *Relational Regression Trees* (RRTs) (Blockeel and De Raedt 1998).

Learning Relational Regression Trees for Gradients: In each iteration, gradients are computed for each training example w.r.t the loss function and an RRT is learned at each gradient step to capture all these gradients. The intuition is that the direction of the approximately learned gradients is similar to the original gradients. A typical loss function used is the squared error, $L = \frac{(y-F)^2}{2}$ where y is the true (regression) value and F is the approximated value. The (functional) gradient is therefore $\delta_i = \hat{y_i} - F_{:m-1}(m_i)$, where $F_{:m-1}$ denotes the sum of all the gradients till the m^{th} iteration of boosting. At each iteration j, every training example i includes a gradient δ_i^j and a new RRT is trained to fit these $\delta_i^j, \forall i$. The final regression value over an example x_i is the sum of regression values after M iterations of boosting, $\hat{y_i} = F_{:M}(x_i)$.

The problem with the squared error loss is that, if the gradients get too large because of outliers in the samples, the boosting will fit the outliers and result in over-fitting. Hence, we also consider the Least Absolute Deviation loss (LAD) function $L = |y - F|$, to fit to the sign of the gradients instead of the actual values and is hence more robust to outliers. The gradient is now $\text{sign}(\hat{y_i} - F_{:m-1}(x_i))$. The process of obtaining the regression values for example x_i is the same as with the squared error loss. While the LAD loss function does solve the problem of fitting outliers, the functional form is not smooth. To ensure smoothness, it would be favorable if when the gradient is within δ, the squared error loss can be used and if it is higher than δ then the LAD loss can be used. A loss function that behaves in this manner is the Huber loss function:

$$L(y,F) = \begin{cases} \frac{(y-F)^2}{2} & : |y-F| \leq \delta \\ \delta(|y-F| - \delta/2) & : |y-F| > \delta \end{cases} \Bigg\}. \quad (4)$$

Thus the functional gradients δ with respect to the Huber loss function for example x_i is $\Delta(x_i) =$

$$\begin{cases} \hat{y_i} - F_{:m-1}(x_i) & : |\hat{y_i} - F_{:m-1}(x_i)| \leq \delta \\ \delta \cdot \text{sign}(y_i - \hat{F_{:m-1}}(x_i)) & : |\hat{y_i} - F_{:m-1}(x_i)| > \delta \end{cases} \Bigg\}. \quad (5)$$

Again, for each example, these gradients are calculated and a RRT is fitted over all the examples for the current iteration.

The GBAD Algorithm: With RRTs for gradients at hand, we now present the algorithm for learning the approximate value functions of relational MDPs using this machinery. Alg. 1 presents the outline of our approach, called Gradient-Boosted Relational Approximate Dynamic Programming (GBAD).

GBAD takes as input the number of training trajectories, the number of of iterations for basis function computation and an access to the simulator (a setting similar to that of Parr et al. (2007). At each iteration a set of trajectories are sampled from the simulator. For every state, action, next state tuple in the trajectory, the value for the current state is updated using the Bellman operator. In this step, we make a crucial difference to the standard fitted VI methods. We add an averaging factor (α) that allows us to keep the running average of the value for states that are visited multiple times. Since we are in a relational setting, it is quite possible that the same state could be visited multiple times and it is necessary to not skew the values due to the multiple visits states. In standard fitted VI, $\alpha = 1$. Next, every state and its corresponding value pair is added to the training set for the RFGB procedure.

The idea here is that RFGB now learns a set of relational regression trees that closely fits the regression values of the examples. In our case, these correspond to finding the combinations of the set of (relational) features that best approximate the value of the current state (example). Note that in each iteration of the RFGB procedure (indexed by m), a single regression tree is learned and added to the model. In the next iteration, the regression value is updated to be the difference between original value (that was the input to the RFGB procedure) and the value returned by the current set of trees (see Equation 6). The key here is that RFGB is essentially performing a series of gradient steps to best fit the value of each state according to the training set. Each tree corresponds to a single gradient in that direction and the final value is the sum of all the values from the different regression trees. Hence, each call to the RFGB procedure returns a single set of regression trees.

In the main GBAD procedure, when evaluating the value of a state at the current time step, the set of trees trees learned in the previous time-step are used to get the values from the previous time. These set of trees at any iteration are the (relational) basis functions for that iteration. Now these trees can technically be discarded as the current values have been computed for the next iteration. It is an interesting future research direction to explore the possibility of retaining the

Algorithm 1 GBAD: Gradient-Boosted Relational Approximate Dynamic Programming

procedure GBAD(# of trajectories N, # of iterations I, Simulator D, Loss Function $loss$)
  $\hat{V}_0 = \mathbf{0}$
  for $i = 1$ to I **do**
    $\mathbf{X} =< s_j, r_j, s_{j+1} >=$ SAMPLETRAJECTORIES(N,D) ▷ Sample trajectories
    $V_{cur} = V_{i-1}$
    for $(s_j, r_j, s_{j+1}) \in \mathbf{X}$ **do**
      $V_{cur}(s_j) = R + (\gamma V_{cur}(s_{j+1})))$ ▷ Apply Bellman operator (R is the immediate reward)
    E = []
    for $(s_j, r_j, s_{j+1}) \in \mathbf{X}$ **do**
      $\mathbf{E} = \mathbf{E} \cup (s_j, V_{cur}(s_j))$ ▷ Create dataset for RFGB
    $\Phi_i = RFGB(\mathbf{E}, loss)$ ▷ Learn a set of relational regression trees
    for $(s_j, r_j, s_{j+1}) \in \mathbf{X}$ **do**
      $V_i(s_j) = \Phi_i(s_j)$
  return Φ_I
procedure RFGB($Data$,$loss$) ▷ Perform RFGB using specified loss
  $F_0 = \{0\}$ ▷ Initialize basis functions to uniform
  for $1 \leq m \leq M$ **do** ▷ Iteration through the gradient steps
    S = GENEXAMPLES($Data$,F_{m-1}) ▷ compute gradient for each example
    Δ_m = FITRELREGRESSTREE(S,$loss$) ▷ learn a tree to fit the error
    $F_m = F_{m-1} + \Delta_m$ ▷ compute value using learned trees
  return F_M ▷ return final set of trees

trees. This could potentially show that the induction of the set of trees is an non-expansive operator and could potentially demonstrate the convergence of GBAD. However, for the purposes of this work, the final set of trees represent the approximation of the relational values.

An example relational regression tree learned in logistics domain is given in Fig. 1, where the inner nodes represent a first order conjunction (test). The left branch indicates that the test is satisfied and the right branch is when it does not hold true. In this example, the root node checks if a truck T is in the destination at the current state, if so, checks for a different truck T2 at the source and if a box B1 is on T2 and so on. The leaf values denote the value of the states that satisfy any particular path. Note that the RFGB procedure learns a set of regression trees for each iteration of GBAD.

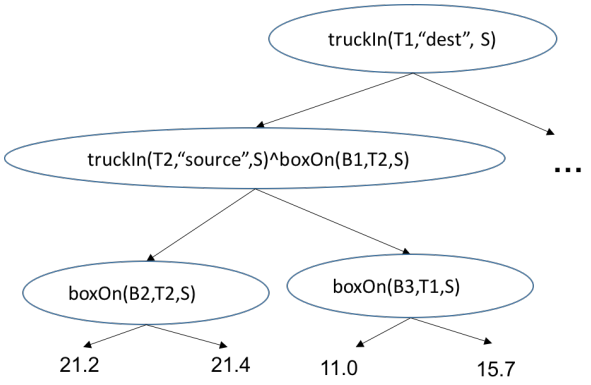

Figure 1: An example of a relational regression tree, where truck in represents the predicate corresponding to a truck being in a city and boxOn corresponding to a box being on a truck.

Relation to Classical ADP Approaches

Approximate Dynamic Programming (ADP) approaches for propositional domains can be classified into *Bellman error* based and *aggregation* based methods. It is quite possible to view our GBAD procedure as a projection method similar to the classical Neural network (Hornik, Stinchcombe, and White 1989) or linear least squares regression approximation (Neter et al. 1996) methods. For this intuition, one needs to make a small change to the algorithm. Instead of returning Φ_I, at every iteration I, if one were to store all the trees from the current iteration, it can be viewed as learning different basis functions. Then it could potentially be shown that these different set of trees indeed form a contraction and hence the method could be viewed as a Bellman error based method with I (number of iterations) partitions to the basis functions. The weights of the different sets of the functions will then be corresponding exponentials of α. We will explore this connection in the future. Currently, we discard the trees from the previous iteration for efficiency and scalability to large relational tasks. The key difference to these methods is that our approach handles relational tasks in a faithful manner without significant engineering of the features.

Aggregate approaches, group (cluster) states and learn values over these clusters. For instance let $S = \{s_1, ..., s_n\}$ denote the set of states and $C = \{C_1, ..., C_R\}$ the clusters. Then the value of a state is calculated as $V(s) = \sum_i \phi_i \cdot I(s \in C_i)$ where ϕ_i is the weight on the cluster. In our method, the grouping is obtained naturally. Each path from root to leaf is a first-order logic clause which is a *lifted* representation of the ground states. ϕ_i can be viewed as the weights learned in the leaves of each branch of a tree.

Since each state only satisfies one path, each tree will yield only one ϕ for a state and R is the number of trees. A key advantage of GBAD over the aggregate features is that in the latter, the transition function of one state from a cluster to another $P(C_i(s)|C_j(s), a)$ is not easy to compute and need to be designed carefully. However, in GBAD, this is easy to compute based on the RRTs learned for standard SRMs (Natarajan et al. 2014).

A related method for solving ADP is that of tile-coding (Whiteson et al. 2007), a linear piece-wise approximator that partitions the state space into (potentially overlapping) regions called *tiles*. These are essentially axis-parallel hyper-cuboids learned from the original state space. It is easy to see that our method can be considered as learning these tiles since each tree can be considered as defining a tile in the original space. While the original method (Albus 1981) also used Bellman error to identify the tiles, it was a heuristic based method. GBAD on the other hand, can be viewed as performing a gradient descent in the Bellman error space (when viewed in a functional form).

Experimental Evaluation

After discussing the relationship of GBAD to propositional ADP methods, we now focus on our empirical evaluation where we investigate the following questions:

Q1: Does faithfully modeling relational problems improve learning performance?

Q2: How does the choice of loss function affect the performance of GBAD?

Q3: Can GBAD be effective even in semi-relational and propositional domains?

Q4: Does using a relational representation facilitate transfer across related domains?

Experimental domains: We validate GBAD on three standard propositional domains (50-state-chain, Tetris and Pong), one semi-relational domain (Blackjack) and three fully relational domains (Wumpus World, Blocks World and Logistics). *Semi-relational* domains are those that are naturally relational but their relational structure can be flattened using a well engineered propositional feature. For example, in the blackjack cards game domain, the sum of the cards involved in the players hand is this propositional feature. It allows us to effectively model the sum irrespective of relational features such as the suit of the card. Using this kind of feature allows for propositional learning to be performed. But given the immense size of the state space, in many real tasks, it is not easily possible to encode important relational features and the relational approximator is still expected to perform better.

We now describe each of our experimental domains.

1. 50-state-chain: The world is a one dimensional grid of 50 grid cells (Lagoudakis and Parr 2003). Lateral movements costing a reward of -1 in both directions are the possible actions. Two of the grid cells are gold locations where the reward is 100 and the goal is to reach one of these locations. We can encode these naturally using predicate logic format.

2. Tetris: The Tetris domain, an atari game, is trained on features that measure the number of blocks of each type of shape (for example, L shaped, square shaped etc). The factored state corresponds to count of each kind of block occurring in play. The game ends when screen can fit no more blocks vertically.

3. Pong: The Pong domain, another atari game, consists of features about player paddle position, ball position, opponent paddle position and finally the score. The factored state is the description of the same split into the two dimensions used to represent the ball position and paddle positions. Each player receives a point when the other player misses the ball. The game ends in a win for the first player to score 5 points (goal state). Goal state has a reward of 100.

4. Blackjack: This domain is a simulation of the blackjack card game. The goal is to get the sum of the player's cards higher than that of the dealer's without exceeding 21. Admissible actions include hitting, in which the player receives a new card from the deck and standing in which the player receives no new cards. We define this using a parameterized representation over the players and dealer's cards. Note that this can be represented using sums of the cards of the dealer and the player making it possible for propositional approaches to handle this task. Winning incurs a reward of 100 and losing earns -50.

5. Wumpus World: The Wumpus World domain is a 4×4 grid with wumpuses at certain locations in the grid. One of the grid cells is a goal state and the objective is to reach this goal state by avoiding the wumpuses. States adjacent to a wumpus location receive a stench percept. This is classically modeled using predicate logic notation and hence serves as a good test bed for the relational task. The reward is -1 for every action, 100 to reach the goal and -50 for the square that contains the wumpus.

6. Blocks World: The Blocks World domain consists of 4 blocks that may be placed on a table or stack on other blocks. We start with the configuration of all blocks on the table and the goal is create a stack where the order is randomly chosen. There is a -1 reward per action taken and a reward of 100 is obtained on attaining the goal state.

7. Logistics The Logistics domain consists of a number of boxes, trucks and cities. The goal is to transfer a certain number of boxes to a city or more of them via trucks. The right sequence of actions governed by stochastic actions that have to be learned in order to achieve the goal configuration.

Domain Representations: In the 50-state-chain domain we represent the ground states using radial basis function. Pong and Tetris are represented using the state features listed in their domain descriptions. We represent Blackjack using factored states corresponding to the sum of the cards in the players hand and the value of the dealer's face card. Wumpus World has a relational state representation containing the agents position as well as any wumpus locations in the grid. We randomize the number and location of the wumpuses. Blocks World is represented using relations: a block being On another block (or the table), a block being $Clear$ (no blocks on top of it). Finally, the Logistics domain uses tIn to represent a truck in a city, bIn to represent a box in a city and bOn to represent a box on a truck.

Experimental Protocol: We evaluate our experimental

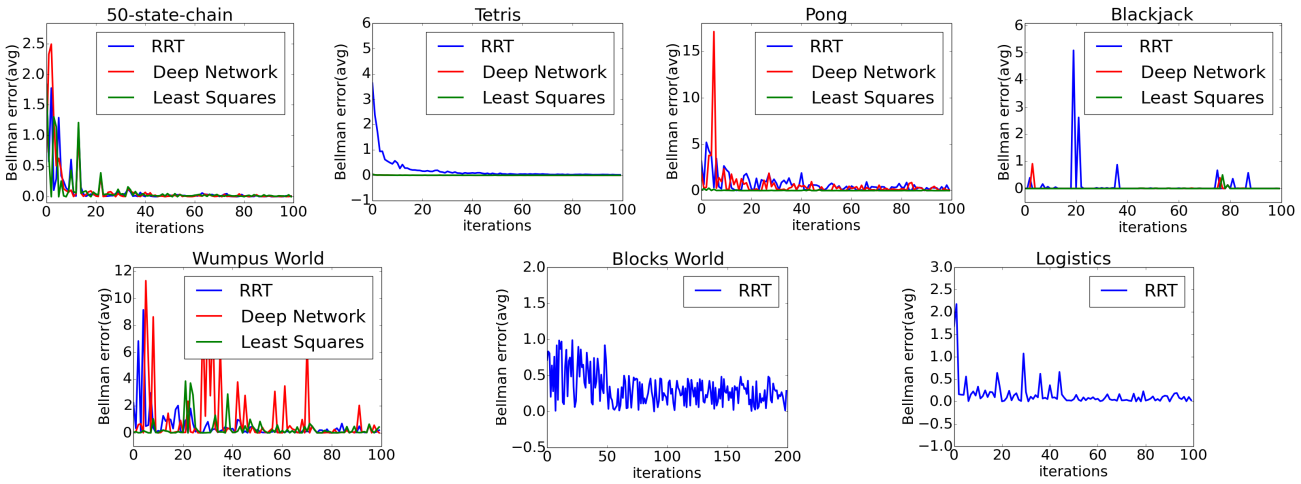

Figure 2: Comparison of GBAD to a relational baseline RRT (blue) and standard baselines using deep networks (red) and linear least squares (green). We plot the differences in the average Bellman error for each baseline and GBAD, i.e., Error(listed method) − Error(GBAD) for standard domains and relational domains: the higher, the better GBAD is. Clearly, GBAD outperforms the standard baselines in nearly all the domains. (Best viewed in color)

questions in three distinct ways. First, we compare GBAD against two other non-relational function approximators that are popular in the literature: linear least squares (Neter et al. 1996) which is often used in standard ADP and deep networks (Hornik, Stinchcombe, and White 1989) which are state-of-the-art in many reinforcement learning tasks. This allows us to demonstrate the power of a faithful relational representation across tasks. For a relational baseline, we learn a single RRT. Next, we compare several different loss functions for GBAD. Finally, we demonstrate the ability to transfer the basis functions in several domains. In Blackjack, we transfer from one run to another. In Wumpus World, we transfer to different wumpus locations and grid sizes (4×4 to 5×5). In Blocks World, we transfer to a different number of blocks (4 to 6). In the Logistics domain, we transfer from 3 boxes, 4 trucks and 2 cities to 5 boxes and 8 trucks.

Throughout the experiments for the deep network baseline, we used 25 hidden layers of size 5 each with a logistic sigmoid activation function and uses Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) optimization for training. The results are averaged over 10 runs.

Evaluation in relational/standard domains: The results in the relational domains (Wumpus World, Blocks World and Logistics) are shown in Fig. 2 where we plot the difference in the average Bellman error over all the states between the corresponding method and our GBAD approach. Hence, a positive number means that GBAD is better (lower error rate). For Wumpus World, we compare against a relational baseline (RRT) as well as the standard baselines (deep network and least squares). While we are able to flatten the representation in the domain, the other relational domains have large state spaces that make flattening the domain impractical. Across all of the relational domains, GBAD outperforms the baselines, especially in early iterations. Furthermore, Wumpus World demonstrates clearly that relational representations are significantly better than standard approximators that can not take advantage of the structure *even when the representation can be flattened*. This suggests that modeling the relational structure in domains improves performance (**Q1**).

Next, we compare GBAD with the standard and semi-relational domains in Fig. 2. It can be observed that GBAD always outperforms the baselines as the difference in Bellman error is always positive. It can also be observed that similar to the relational domains, our approach often outperforms the baselines in early iterations. The key difference is that in the propositional domains, there is less underlying structure for GBAD to utilize and so the baselines converge relatively more quickly. This shows that GBAD is effective even in the case of propositional and semi-relational domains **(Q3)**.

Evaluation of different loss functions: When comparing the different loss functions, we present the difference in average Bellman error over all the states between the Huber loss and the LAD and LS loss functions (Fig. 3). In this case, two significant patterns emerge. First is that the smoothing due to LAD loss function can result in underfitting of the model and hence has significantly worse performance in two of the four domains. However, the choice of LS vs Huber loss is not statistically significant across domains (even though there is a gain in performance of Huber loss in early iterations). This is illustrated more clearly in Blocks World, even after 2000 iterations the green line (LS) keeps oscillating between only very slightly worse to matching performance against Huber loss. Hence, **(Q2)** can be answered by noting that a smoother loss function does not necessarily model the problem well.

Evaluation of transferring basis functions: When analyzing the transfer property in relational domains, as they facilitate transfer by generalization naturally, we present the

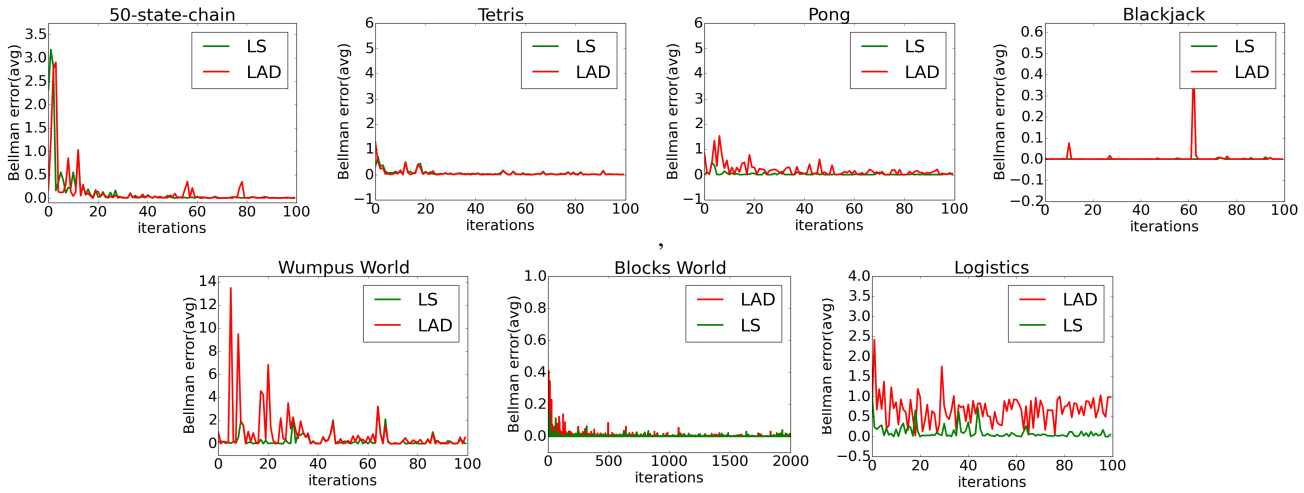

Figure 3: Performance of GBAD using different loss functions: least absolute deviation (LAD, red) and least squares (LS, green). We plot the differences between these loss functions and huber loss: the higher, the better the huber loss is. As one can see, GBAD with LAD loss performs significantly worse for some domains where as the difference between the huber loss and LS loss is not statistically significant. (Best viewed in color)

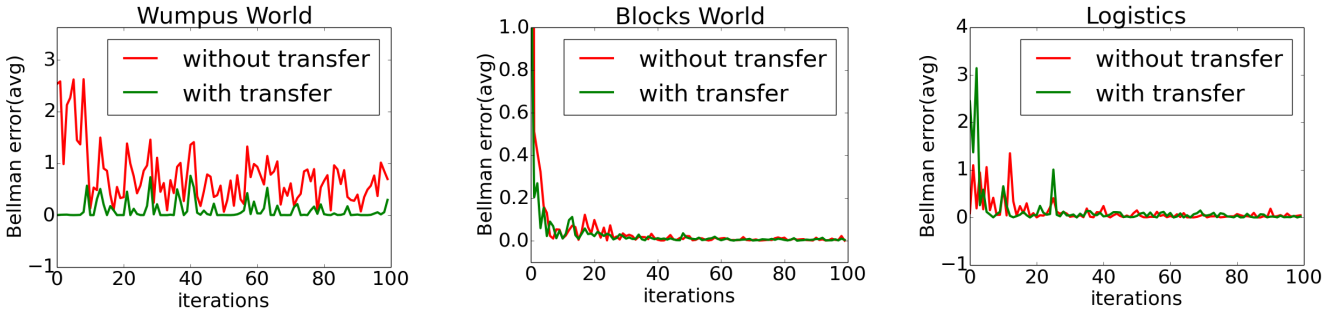

Figure 4: Transfer of relational basis functions for Wumpus World, Blocks World and Logistics. The average Bellman errors (the higher, the better) without transfer (red) and with transfer (green) are shown. As one can see, transfer learning can easily be incorporated into GBAD and can boost performance, especially in fully relational domains. (Best viewed in color)

average Bellman errors with and without transfer. In the without transfer case, for the target problem, the learning occurs from scratch. In the transfer case, the set of RRTs learned from the final step of source task is used to initialize the models (values of states in line 1 of GBAD algorithm). We present the transfer results for the relational domains in Fig. 4. Across all domains, transferring the learned model to initialize during source task helps clearly in learning a better model by achieving a jump start, a higher slope and better asymptote (for Wumpus World). This allows us to answer **(Q4)** optimistically: the relational basis functions have significant potential for enabling transfer across related tasks.

Conclusion

We introduced Relational Approximate Dynamic Programming (ADP) and presented the first algorithm for it, called Gradient-Boosted Relational ADP (GBAD). The intuition underyling GBAD is that one can approximate the value function over a set of objects and relations as a set of RRTs learned in a sequential manner. The Bellman operator application step corresponds to the evaluation of these trees for a given state and the projection step corresponds to the learning of these trees using gradient-boosting. Our experiments clearly show that relational ADP can outperform state-of-the art methods on relational domains. Most importantly, gradient boosting paves the way to deal jointly with propositional and relational features; one only has to adapt the gradient regression examples correspondingly. We also demonstrated the generalization ability of GBAD for transfer learning.

Relational ADP and GBAD suggest several interesting avenues for future work. Extending our work to generalized continuous state-action spaces, multi-agent settings and potentially Partially Observable Markov Decision Processes (POMDPs) are interesting directions. Exploring the closer connections to the Bellman error based methods is an important direction. Currently our work does policy evaluation and extending this to policy improvement similar to Kersting et al. (2008) is another future direction.

References

Albus, J. S. 1981. *Brains, behavior, and robotics*.

Blockeel, H., and De Raedt, L. 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence* 101(1-2):285–297.

Boyan, J. A., and Moore, A. W. 1995. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems* 369–376.

Boyan, J. A. 1999. Least-squares temporal difference learning. In *ICML*, 49–56.

De Raedt, L.; Kersting, K.; Natarajan, S.; and Poole, D. 2016. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* 1189–1232.

Getoor, L. 2007. *Introduction to statistical relational learning*. MIT press.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th international joint conference on Artificial intelligence*, 1003–1010. Morgan Kaufmann Publishers Inc.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2(5):359–366.

Keller, P. W.; Mannor, S.; and Precup, D. 2006. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 449–456. ACM.

Kersting, K., and Driessens, K. 2008. Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the 25th international conference on Machine learning*, 456–463. ACM.

Lagoudakis, M. G., and Parr, R. 2003. Least-squares policy iteration. *Journal of machine learning research* 4(Dec):1107–1149.

Menache, I.; Mannor, S.; and Shimkin, N. 2005. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research* 134(1):215–238.

Natarajan, S.; Joshi, S.; Tadepalli, P.; Kersting, K.; and Shavlik, J. 2011. Imitation learning in relational domains: A functional-gradient boosting approach. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, 1414.

Natarajan, S.; Khot, T.; Kersting, K.; Gutmann, B.; and Shavlik, J. 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning* 86(1):25–56.

Natarajan, S.; Kersting, K.; Khot, T.; and Shavlik, J. 2014. Introduction. In *Boosted Statistical Relational Learners*. Springer. 1–3.

Neter, J.; Kutner, M. H.; Nachtsheim, C. J.; and Wasserman, W. 1996. *Applied linear statistical models*, volume 4. Irwin Chicago.

Parr, R.; Painter-Wakefield, C.; Li, L.; and Littman, M. 2007. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, 737–744. ACM.

Price, B., and Boutilier, C. 2001. Imitation and reinforcement learning in agents with heterogeneous actions. In *Conference of the Canadian Society for Computational Studies of Intelligence*, 111–120. Springer.

Riedmiller, M. 2005. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, 317–328. Springer.

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

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, 1–9.

Tosatto, S.; Pirotta, M.; D'Eramo, C.; and Restelli, M. 2017. Boosted fitted q-iteration. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 3434–3443.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research* 31:431–472.

Whiteson, S.; Taylor, M. E.; Stone, P.; et al. 2007. *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin.

Wu, J.-H., and Givan, R. 2007. Discovering relational domain features for probabilistic planning. In *ICAPS*, 344–351.

Yu, H., and Bertsekas, D. P. 2009. Convergence results for some temporal difference methods based on least squares. *IEEE Transactions on Automatic Control* 54(7):1515–1531.