Gradient-Boosted Adaptive Dynamic Programming: A Unifying Framework for Relational Domains

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We consider the problem of Approximate Dynamic Programming in relational domains. Inspired by the success of the Bellman error based methods in propositional settings, we develop a relational Bellman error method that represents 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 generalizes multiple value function approximation methods and can be effectively applied to both relational and propositional domains.

1 Introduction

Value function approximation in Reinforcement Learning (RL) has long been viewed using the lens of feature discovery [17]. A popular class of methods that address this problem for Adaptive Dynamic Programming (ADP) is the Bellman error based methods [12, 9, 17]. 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 this research for the classic RL problems with discrete and continuous state spaces.

We 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 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 in substantially large state-action spaces that can be intractable. Also, as noted by Tadepalli et al. [20], using function approximators such as Neural Networks that 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 [18, 19, 21, 20, 23]. 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. [7] 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 Bellman error based approximate learning for propositional domains, we propose a basis function learning method for relational domains. It can be observed that the Bellman error methods in RL have two steps - applying the Bellman operator and then projecting them back to the span of the basis functions. The key idea in our work is that we represent the value functions as a set of relational regression trees (RRTs) [2] learned using gradient-boosting [5]. Given this

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

representation, we show that applying the Bellman operator is simply evaluating the trees for 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 [14] 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 [23] 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.

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) As far as we are aware, 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 demonstrate clearly how our method can be viewed as encompassing these different approximation techniques. (4) Finally, we show 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 structured 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.

2 Background

Markov Decision Processes: 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:

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

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 [17]. Given two vectors V_1 and V_2

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

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, $TV - V$. The Bellman operator is also a contraction in the weighted L_2 norm.

$$\|V_1 - V_2\|_\rho = d \implies \|TV_1 - TV_2\|_\rho \leq kd \tag{3}$$

where ρ is the stationary distribution of the transition matrix P.

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 [4]. 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:

$$\hat{V}(s_i) = W \cdot F \equiv \hat{V}(s_i) = \sum_k w_k \cdot f_k. \tag{4}$$

The features, also called *basis functions*, are ideally linearly independent. Least Squares Temporal Difference (LSTD) Learning [3] and Least Squares Policy Evaluation (LSPE) [24] solve for this approximate value as a function of these basis functions. Because the Bellman operator T is a contraction and least squares projection Π is non expansive, the difference between the value learned using approximation and the actual value for a state $s \in S$ is bounded as:

$$\|V(s) - \hat{V}(s)\| = \frac{\|V(s) - \Pi V(s)\|}{\sqrt{1-\gamma}} \tag{5}$$

However, the projection Π has to be with respect to the distribution over the states $s \in S$ for convergence as then ΠT is a contraction. We start with no basis functions and learn them iteratively using successive approximation where each approximation is performed using the non-parametric gradient-boosting. We can handle both propositional and relational domains in a unified manner.

Bellman Error Basis Functions: The Bellman error for a policy π is $T\hat{V}_\pi - \hat{V}_\pi$, since $\hat{V}_\pi$ is the perpendicular projection of $T\hat{V}_\pi$ on to the span of F and hence, the Bellman error is orthogonal to the span of F and can be included as an additional basis function. 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. [23] that can reason over sets of feature values.

We employ Functional Gradient Boosting [5] in relational domains [6] 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 [10] and Natarajan et al. [13], 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.

3 Gradient-Boosted Relational Adaptive Dynamic Programming

We consider learning to act in relational, noisy domains where the domains consist of multiple interacting objects. Statistical Relational Models (SRMs) combine the expressive power of relational models to represent complex relationships and the ability of probability theory to model uncertainty. We adapt a recently successful *Relational Functional Gradient Boosting* (RFGB) algorithm [14, 15, 13] for learning basis 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. 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) [2]. In each iteration, a loss function is used and gradients are computed for each training example and a 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 $\delta_i = \hat{y}_i - F_{:k-1}(x_i)$ where $F_{:k-1}$ denotes the sum of all the gradients till the kth 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 K iterations of boosting i.e $\hat{y}_i = F_{:K}(x_i)$. 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 $sign(\hat{y}_i - F_{:k-1}(x_i))$. The process of obtaining the regression values for example x_i is the same as with earlier case. 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.

$$L(y, F) = \begin{Bmatrix} \frac{(y-F)^2}{2} & : |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) : & |y - F| > \delta \end{Bmatrix}. \tag{6}$$

Thus the functional gradients δ with respect to the Huber loss function for example x_i is:

$$\Delta(x_i) = \begin{Bmatrix} \hat{y}_i - F_{:k-1}(x_i) & : |\hat{y}_i - F_{:k-1}(x_i)| \leq \delta \\ \delta \cdot sign(y_i - \hat{F}_{:k-1}(x_i)) : & |\hat{y}_i - F_{:k-1}(x_i)| > \delta \end{Bmatrix}. \tag{7}$$

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

3.1 Algorithm

Given that we discussed learning RRTs for a given set of regression examples, we now present the algorithm for learning the basis functions of relational MDPs using this machinery. Algorithm 1 presents the outline of our approach. Gradient-Boosted Relational Approximate Dynamic Programming (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. [17]). At each iteration a set of trajectories are sampled from the simulator. For every state, action, next state tuple in the trajectory, the values are updated using the Bellman operator and these are added to the training example set. Then a new set of relational trees are learned from these examples and the process is repeated. When evaluating the value of a state at the current time step, the 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. The final set of trees represent the set of relational basis functions learned for the domain. Next, we analyze the properties of this algorithm by positioning it w.r.t basis function learning and present its convergence.

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
            $V_{cur}(s_j) = V_{cur}(s_j) + \alpha(r_j + (\gamma V_{cur}(s_{j+1}) - V_{cur}(s_j))$ ▷ Apply Bellman operator
        end for
        $\mathbf{E}$ = []
        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
        end for
        $\Phi_i = RFGB(\mathbf{E}, loss)$          ▷ Project value to span of relational basis functions
        for $(s_j, r_j, s_{j+1}) \in \mathbf{X}$ do
            $V_i(s_j) = \Phi_i(s_j)$
        end for
    end for
    return $\Phi_I$          ▷ Return relational basis functions
end procedure

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
        $\Delta_m$ = FitRelRegressTree($S$,$loss$)          ▷ learn relational basis functions to fit error
        $F_m = F_{m-1} + \Delta_m$          ▷ compute value using learned basis
    end for
    return $F_M$          ▷ return final set of basis functions
end procedure
```

3.2 Convergence using Functional Gradient Boosting

The classical proof of convergence of Approximate Dynamic Programming approaches operating on projection that aim to minimize Bellman error $TV - V$ rely on two observations. First, $TV(s)$ is a contraction and second, the projection operator Π is non expansive. Thus, it can be shown that the difference between the true and approximated values can be bounded as shown in Equation 3. Given that our approach both uses the contraction and projection steps and that the contraction step remains the same, we simply need to prove that the projection step (RFGB) is non-expansive.

While learning these trees, one constraint that is implicitly placed on these trees is that each state $s \in S$ satisfies only one path in a single regression tree. The paths in each tree form independent partitions of the state space and hence can be viewed as independent basis functions. Given that there are multiple trees, it may appear that the partitions from the different trees may not be independent. However, these different RRTs can be combined (added) to a single tree similar to the procedure of First-order Decision Diagrams (FODDs) [21]. Thus the final combined tree will satisfy the single-path semantics and can ensure the presence of independent basis functions.

Let k_ϕ be the regression value learned at the leaf of a branch of the tree representing a first-order predicate ϕ. Each ϕ is a conjunction of first-order features $f \in F$ and is in the span of F. Each of the K RRTs learned during boosting represents a hypothesis $\hat{h}$ such that

$$\hat{h}(s) = \sum_{\phi} k_\phi \cdot I(\phi(s) = True) \tag{8}$$

where I is an indicator function that is true if state s satisfies the formula $\phi(s)$. If we denote the set of k_ϕ across all RRTs $\hat{h}$ as K_Φ and the set of predicates (first-order features) $\phi(x_i)$ as $\Phi(s)$ then,

$$\hat{V}(s) = K_\Phi \cdot \Phi(s) \tag{9}$$

Equation 9 is similar in form to Equation 4 and this shows that the gradient boosted regression trees are a linear combination of basis functions. Thus, non-parametric projection using RFGB is similar to the parametric projection over the span of basis functions and is therefore non-expansive by extension. The projection operator Π w.r.t. a distribution ρ denoted as Π_ρ for a set of basis functions Φ is

$$\Pi_\rho = \Phi(\Phi^{\mathbf{T}}\Delta\Phi)^{-1}\Phi^{\mathbf{T}}\Delta \tag{10}$$

where Δ is a diagonal matrix of the distribution of states ρ. The number of states satisfying each branch of a tree i.e. predicate ϕ can be used to construct Δ as that represents a distribution ρ over the states. Thus, the projection achieved using RFGB analogous to Equation 10 can be written as

$$K_\Phi = \Phi(\Phi^{\mathbf{T}}\Delta\Phi)^{-1}\Phi^{\mathbf{T}}\Delta \tag{11}$$

3.3 Relation to Classical ADP Approaches

ADP approaches for propositional domains can be classified into *Bellman error* based and *aggregation* based methods. In the previous section, we established the connection to the Bellman error based methods as our RFGB can be considered as a projection method similar to the classical Neural network [8] or linear least squares regression approximation [16] methods. 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 on the other hand, 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

$$V(s) = \sum_{i} \phi_i \cdot I(s \in C_i) \tag{12}$$

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 [14].

A related method for solving ADP is that of tile-coding [22], 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 [1] 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).

4 Experimental Results

After discussing the convergence properties of GBAD and its relationship to propositional ADP methods, we now focus on empirical evaluation where we answer the following questions explicitly:

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?

Domains: We validate GBAD on two fully relational domains (Wumpus world and Blocks World), one semi-relational domain (Blackjack) and a standard domain (50-state chain).

1. 50-state chain: The world is a one dimensional grid of 50 grid cells [11]. 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. 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.
3. 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.
4. 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.

In the 50-state chain domain we represent the ground states using radial basis function, while in Blackjack we represent them 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. Finally, 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).

Methodology: We evaluate our experimental 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 [16] which is often used in standard ADP and deep networks [8] 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 transfer basis functions

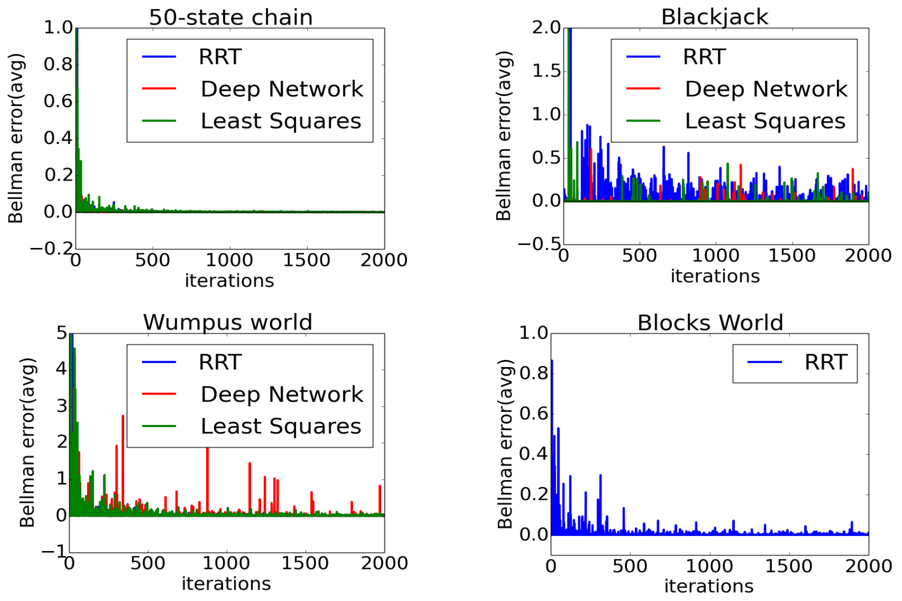

Figure 1: Comparison of GBAD to a relational baseline RRT (blue) and standard baselines using deep networks (red) and linear least squares (green). We plot the difference in the average Bellman error for each baseline vs GBAD for standard domains (**TOP**) and relational domains (**BOT**).

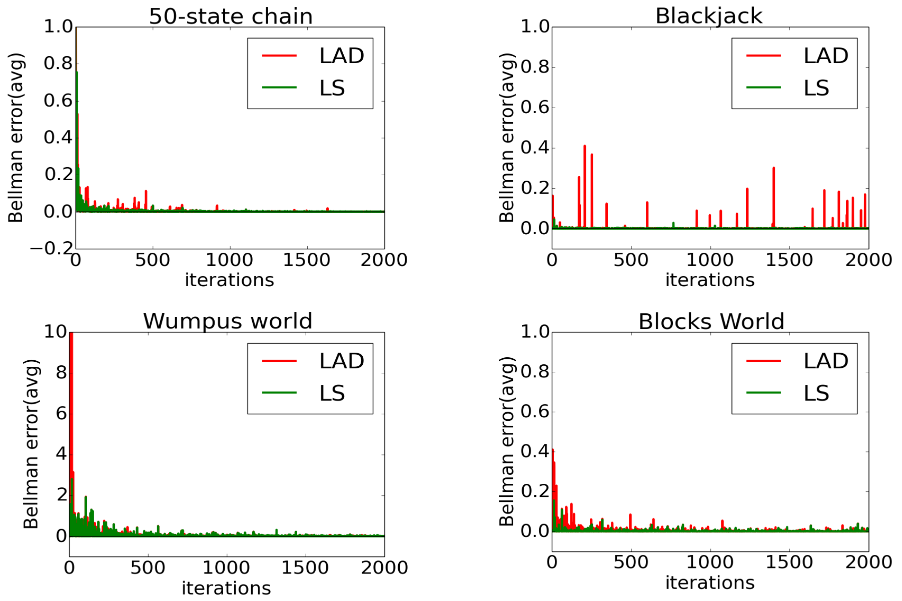

Figure 2: Performance of GBAD using different loss functions: least absolute deviation (red) and least squares (green). We plot the difference between these loss functions and huber loss.

in Wumpus world and Blocks World. The Wumpus worlds have different wumpus locations and different grid sizes (4×4 to 5×5) while the Blocks World has a different number of blocks (4 to 6). The results are average of 10 runs for each method in each domain.

Comparison with standard baselines: We compare GBAD with the standard baselines by measuring the difference in the average Bellman error over all the states between GBAD and the corresponding method in Figure 1.**TOP**. 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 the difference is generally high during early iterations of the algorithms for 50-chain and Blackjack

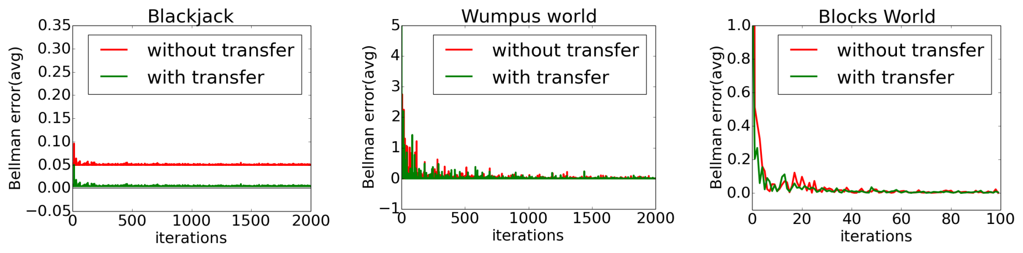

Figure 3: We demonstrate transfer of relational basis functions for Blackjack, Wumpus world and Blocks World. We plot the average Bellman error without transfer (red) and with transfer (green).

domains. GBAD is also better than learning a single tree in both these domains as seen by the blue curve. This helps us to answer **Q3** affirmatively in that GBAD is effective even in the case of propositional and semi-relational domains specifically in early iterations.

The relational domains are shown in Figure 1.**BOT**. For Wumpus world, GBAD significantly outperforms both propositional baselines by converging to the true value in fewer iterations. This demonstrates the effectiveness of relational approximation when the underlying domain representation contains rich structure of which deep networks and linear least squares approximations are not able to take advantage (**Q1**). In Blocks World, it is well known that it is difficult if not impossible to construct engineered features particularly when the target configuration is generated randomly. Hence, we only compare against the relational baseline (RRT). Across Wumpus world and Blocks World, GBAD outperforms learning a single tree by converging more quickly in early iterations.

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 (Figure 2). 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). Hence, **Q2** can be answered by noting that a smoother loss function does not necessarily model the problem well.

When analyzing the transfer property in relational domains, as they facilitate transfer by generalization naturally, we present the 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). The results are presented in Figure 3. A clear pattern emerges in that in propositional domain, the transfer does not help much. However, in both the semi-relational and relational 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 Blackjack). This allows us to answer **Q4** optimistically that the relational basis functions have significant potential for enabling transfer across related tasks.

5 Conclusion

We presented an algorithm for ADP in relational domains. The intuition is that we 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. Moreover, 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 in a transfer learning task. Extending our work to generalized continuous state-action spaces, multi-agent settings and potentially Partially Observable Markov Decision Processes (POMDPs) are interesting directions for future research. Currently our work does policy evaluation and extending this to policy improvement similar to Kersting et al. [10] is another future direction.

References

[1] J. S. Albus. *Brains, behavior, and robotics*. 1981.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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