Online Feature Discovery in Relational Reinforcement Learning

Scott Sanner SSANNER@CS.TORONTO.EDU
Computer Science Department, University of Toronto, Toronto, ON, M5S 3H5, CANADA

Abstract

We introduce a technique for model-free relational reinforcement learning in indefinite horizon undiscounted domains with bounded reward. Previous work has represented the value function as a ground relational naive Bayes net and has leveraged Bayes net parameter and structure learning techniques to refine successive approximations of the value function. Unfortunately, while value function evaluation and parameter inference are very efficient under this framework, even greedy optimal learning of highly restricted naive Bayes net structure can be computationally prohibitive in practice. In this paper, we propose a novel learning algorithm that focuses Bayes net structure learning on the frequently visited portions of state space. Inspired by the Apriori frequent-itemset data mining algorithm, this structure learning algorithm has the dual benefits of efficiency and low-variance parameter estimates. To demonstrate the efficacy of this approach, we present encouraging initial results in the game domains of Tic-Tac-Toe, Backgammon, and Othello.

1. Introduction

Relational reinforcement learning (RRL) has emerged in recent years as a major area of focus in the reinforcement learning community (Tadepalli et al., 2004; van Otterlo & Kersting, 2004). Relational representations are a natural paradigm for modeling learning domains where states consist of configurations of objects and the relations between them. While RRL is an attractive approach for learning from delayed reward in such state representations, its generality does not come without severe representational and computational drawbacks:

- As the domain size and the arity of the relations increase, there is a combinatorial explosion in the number of ground relations that describe a state. This results in an intractably large state space, even if there are only a few relations in the problem specification.
- One must carefully decide on the hypothesis space from which a value function or policy is selected. In most non-trivial domains, a compact representation of the optimal value function is impossible thus requiring that the value function be approximated. However, the approximate representation of the value function can greatly impact its quality: A representation that is too simple may not adequately approximate the optimal value function, resulting in poor policies; And a more complex representation capable of a good approximation of the optimal value function may have intractable data complexity requirements for learning a low-variance estimate of the value function.

In the special case of reinforcement learning in indefinite horizon undiscounted domains with a single terminal reward of success or failure (e.g. planning tasks or games with stationary opponents), the value function has a probabilistic interpretation: it is simply the conditional probability of eventual success (or failure) given an observed state.[1] Armed with this insight, previous work (Sanner, 2005) has sought to mitigate the aforementioned computational and representational drawbacks of relational reinforcement learning by representing the value function as a ground relational naive Bayes net. Under this approach, the value function and restricted versions of its internal structure can be compactly approximated *and* well-known Bayes net parameter and structural inference techniques can be leveraged to do this. Given that this technique proposed to learn both the parameters and structure of the value function, it has been labelled *structure and value relational reinforcement learning* (SVRRL).

SVRRL relies on the common relational domain assumption that many fewer ground literals appear positively than negatively in a given state.[2] Based on this, it can be shown that policy evaluation and parameter inference in the naive

Presented at the ICML Workshop on Open Problems in Statistical Relational Learning, Pittsburgh, PA, 2006.

[1] We can easily handle any domain with bounded reward in this setting by normalizing all rewards to the range $[0, 1]$ and transforming the terminal nodes of the domain to stochastically transition to success (failure) with a probability equal to (one minus) the normalized reward. The optimal policy in this transformed domain will be equivalent to the optimal policy for the original domain.

[2] This is exploited in the closed-world assumption of Prolog that allows it to compactly represent many relational domains.

Bayes net value function can be computed efficiently by recording only the relational atoms occurring positively in a state. Unfortunately, this efficiency does not extend to naive Bayes net structure learning. In this case the computational overhead of the SVRRL approach has prevented any known practical implementation aside from a severely restricted version known as FAA-SVRRL (Sanner, 2005).

However, in some sense the original SVRRL algorithm is too eager to learn value function structure and we can take a more pragmatic approach in practice. That is, SVRRL maintains joint parameter estimates over *all* nodes in its naive Bayes network in order to greedily determine the structure that should be learned. However, many portions of state space are visited too infrequently to yield low-variance estimates of joint probabilities. For these reasons, it only makes sense to focus structure learning on frequently visited portions of state space. Inspired by the Apriori frequent-itemset data mining (DM) algorithm, we present a novel structure learning algorithm DM-SVRRL, which has the dual benefits of efficiency and low-variance parameter estimates. To demonstrate the efficacy of this approach, we present encouraging initial results in the game domains of Tic-Tac-Toe, Backgammon, and Othello.

2. MDPs and Reinforcement Learning

Reinforcement learning (RL) tasks are often cast in a Markov decision process (MDP) framework. Formally, a finite state and action MDP (Puterman, 1994) is a tuple $\langle S, A, T, R\rangle$ where: S is a finite state space; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a transition function, with $T(s, a, \cdot)$ a distribution over S for all $s \in S, a \in A$; and $R : S \times A \to \mathbb{R}$ is a bounded reward function. Our goal is to find a policy π that maximizes the value function, defined using the infinite horizon, discounted reward criterion: $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r^t | S_0 = s]$, where r^t is a reward obtained at time t, $0 \leq \gamma < 1$ is a discount factor, and S_0 is the initial state.

A stationary policy takes the form $\pi : S \to A$, with $\pi(s)$ denoting the action to be executed in state s. Policy π is optimal if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and all policies π'. The optimal value function V^* is the value of any optimal policy and satisfies the following:

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{t \in S} T(s, a, t) \cdot V^*(t) \right] \quad (1)$$

In this paper, we will assume that the horizon is indefinite (i.e., starting from any state, the system will eventually transition to a terminal state with probability 1), the reward is undiscounted (i.e., $\gamma = 1$), and there is a *single* reward of *success* (1) or *failure* (0) upon transition into the terminal absorbing states. As mentioned previously, domains

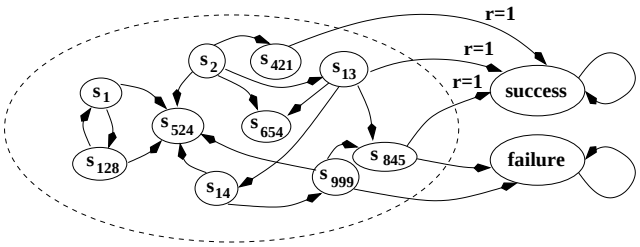

Figure 1. An example Markov chain corresponding to an undiscounted MDP evaluated under a fixed policy π. States are indexed by a unique ID and transitions between states are labelled with a reward r *when it is non-zero*. Note that in this restricted setting, there is only a *single* reward of *success* (1) or *failure* (0) upon transition into the terminal absorbing states. Thus, the value of a given state under a policy can be interpreted as the probability of reaching the success state from the given state in the infinite limit.

with bounded reward can be easily transformed to this setting while preserving policy optimality. Consequently, this setting is adequate for modelling many stochastic planning tasks or game-playing with stationary opponents. Under a fixed policy π, such a restricted MDP reduces to a simple Markov chain as demonstrated in Figure 1. Letting w denote that an eventual *win* or *success* state is reached ($S^{t=\infty} = success$), we derive the following simple probabilistic interpretation for a value function under a policy:

$$\begin{aligned} V_\pi(s) &= E_\pi[\sum_{t=0}^{\infty} r^t | S^{t=0} = s] \\ &= P(S^{t=\infty} = success | S^{t=0} = s) \;=\; P(w|s) \end{aligned}$$

We note that this probabilistic interpretation of a policy is very intuitive – it states that the value of any state under a fixed policy is just the probability of reaching a success state in the infinite limit. One way of estimating this value is simple Monte Carlo reinforcement learning (Sutton & Barto, 1998) where we sample the value function from real-world or simulated experience. In the infinite limit of samples, it is known that the value function estimate will converge to its true value.

So far, we have discussed how to determine the value of a policy, but we have not discussed how to find the *optimal policy*. One approach to doing this in a model-based MDP setting is known as *policy iteration* (Puterman, 1994):

1. Pick an arbitrary decision policy π_0 and set $i = 0$.
2. *Policy evaluation*: Given π_i, determine $V_{\pi_i}(s)$.
3. *Policy improvement*: Find a new policy $\pi_{i+1}(s) = \arg\max_{a \in A} \left\{ R(s, a) + \gamma \sum_{t \in S} T(s, a, t) \cdot V_{\pi_i}(s) \right\}$
4. If $\pi_{i+1} \neq \pi_i$ increment i & go to (2) else return π_{i+1}.

Now, if we were to apply policy iteration to the RL setting, we would notice that while Monte Carlo RL suffices

to provide an estimate of $V_{\pi_i}(s)$, we do not have the luxury of obtaining an infinite number of samples so that it converges to the exact estimate. Nonetheless, we note that using a finite but large number of Monte Carlo samples for policy evaluation in step 2 still allows policy iteration to converge in many practical applications.

Setting aside convergence issues of policy evaluation, we also note that we have a representational problem. In most practical problems the state space is often too large to allow an explicit representation of $P(w|s)$, so we must often approximate this probability in practice. This brings us to the topic of the next section.

3. Structure and Value Relational RL

In many RL applications, the state often admits a compact relational description. Consequently, in the restricted RL setting described above where the value function under a policy can be represented as the conditional probability $P(w|s)$, we might try approximating this value function with a relational naive Bayes net. In doing this, we would ideally like to learn both the *structure* and parameters (i.e. *value*) of this Bayes net. This is precisely the objective of the structure and value relational RL algorithm (SVRRL) (Sanner, 2005).

3.1. Value Learning

In the SVRRL framework, we assume that the state is described using a fixed set of relations $R = \{R_1, \ldots, R_i\}$, each having some finite arity. Each relation argument is assigned an attribute type from a set $A = \{A_1, \ldots, A_j\}$ where each attribute takes an assignment from a set of legal values. We refer to a relation and the attribute specification for its arguments as a *relation template*.

To make this more concrete, we use the example of Tic-Tac-Toe where we use a single relation template $At(Mark, Row, Col)$ where $Mark = \{X, O\}$ and $Row = Col = \{1, 2, 3\}$. This leads to $2 \times 3 \times 3 = 18$ possible ground relational features, each of which can be treated as a binary proposition taking a truth assignment for every possible state. The entire game state of Tic-Tac-Toe can be determined by a truth assignment to each of these 18 ground relations.

In the following discussion, we use binary propositions F_i to denote generic ground atoms (a.k.a. features) which can take on the value *true* denoted by f_i and the value *false* denoted by $\bar{f}_i$. For representational parsimony, we assume the state is represented by only the positive (true) atoms, which we arbitrarily label $\{f_1, \ldots, f_p\}$. Then, given that that there are a total of n ground atoms in a problem domain, we use the closed-world assumption to infer that the remaining atoms $\{\bar{f}_{p+1}, \ldots, \bar{f}_n\}$ are false. We let $F = \{F_1 \ldots F_n\}$

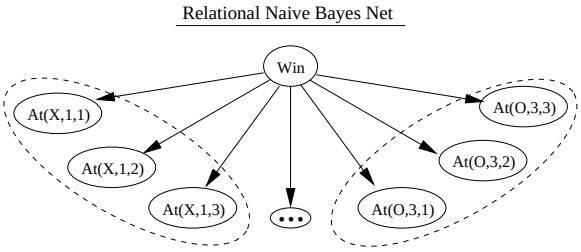

Figure 2. The initial relational naive Bayes net used in the game of Tic-Tac-Toe. In all, the binary-valued Win node has 18 binary-valued children corresponding to each of the ground relational features F_i in the state description. This representation requires one conditional probability table (CPT) for $P(Win)$ and 18 individual CPTs $P(F_i|Win), i \in \{1 \ldots 18\}$. In value learning, the goal is to learn the parameters of this Bayes net to maximize log-likelihood. In structure learning,the goal is to learn which child nodes to join to maximize the log-liklihood. The dashed lines show two different sets of node joins that can be useful in Tic-Tac-Toe – if all individual nodes take the value *true*, they represent a line of X's across the top row and a line of O's across the bottom row.

(all ground atoms) and represent a state instantiation $s \in \{F_1 \times \cdots \times F_n\}$ as $s = \{f_1, \ldots, f_p \bar{f}_{p+1}, \ldots, \bar{f}_n\}$ (a truth assignment to all ground atoms). The omission of negative atoms from the state often yields a compact representation since the number of positive atoms is usually small (and easily identifiable) in comparison to the total number of atoms (i.e., $p \ll n$).

Even very small relational RL domains can have hundreds of ground atoms and it would be impossible to represent the exact distribution, which in its fully enumerated form would require roughly one probability entry for every distinct truth assignment to ground atoms. For 100 ground atoms, this would require approximately 2^{100} distinct probability entries, which is clearly intractable. Thus, we need to focus on a compact, factored representation of $P(w|s)$. SVRRL does this by representing $P(w|s)$ using the simple relational naive Bayes structure given in Figure 2. For our previous example of a relational domain with 100 ground atoms, we need only record 201 probability entries to approximate the value of $P(w|s)$; two probability entries $P(f_i|w)$ and $P(f_i|\bar{w})$ for each ground atom and one entry $P(w)$ for the prior over winning. While this is only an approximation, we show that we can "patch up" this simple representation through structure learning. But, first we focus on how to learn the value (parameters) of this network.

Figure 3 shows the learning task. Given a number of trials, each involving some *finite* number of time steps, the learner is presented with a relational specification of the positive state features $\{f_1, \ldots, f_p\}$ and chooses an action according to a fixed policy. This is repeated in each trial until the terminal state is reached and the terminal reward

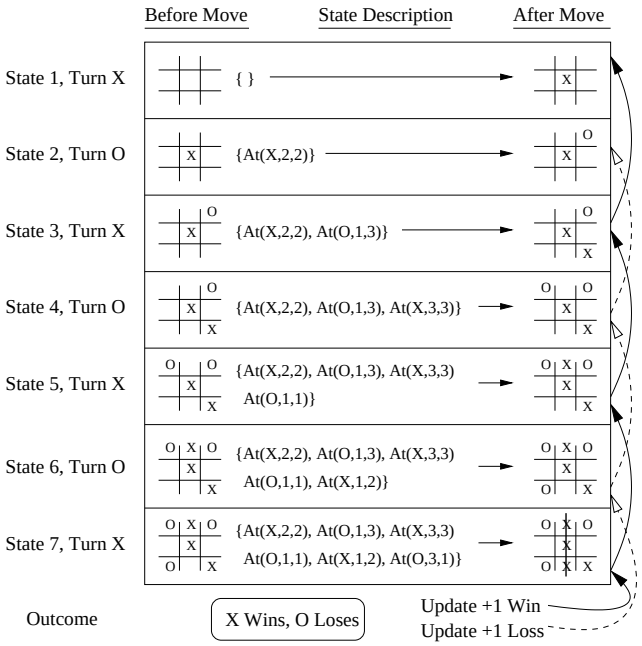

Figure 3. A diagram depicting a single training trial of two SVRRLs agent in the domain of Tic-Tac-Toe. In Tic-Tac-Toe, players alternate turns, placing their mark in a free square until one player wins by completing a line, or both players draw by running out of moves. On every turn, the player extracts the true ground relational features $\{f_1, \ldots, f_p\}$ of the game state. The $At(\cdot, \cdot, \cdot)$ relation has respective attributes indicating the mark placed and its row and column position. At the end of the game, each SVRRL agent records all features resulting from its moves and immediately updates their conditional probability tables $P(F_i|W)$ as well as the global winning prior $P(W)$. Training can proceed for any number of these trials.

is received. At the end of each trial, the overall win/loss count is immediately updated along with that of the features for the *after-states*[3] occurring in the move history for the player.

How does SVRRL learn the parameters of the value function during training and use the value function to determine a policy? For *parameter learning* it is well-known that the max-likelihood parameters of a Bayes net are simply the observed probabilities (denoted by $\hat{P}$) for each CPT, so we can efficiently determine the max-likelihood value function by tracking overall and feature-specific win/loss counts.

Now, in order to *execute the optimal policy* w.r.t. to the current parameters in our naive Bayes net value function, we leverage the fact that the state s which maximizes $\hat{P}(w|s)$ will also maximize the log winning odds $\log(\frac{\hat{P}(w|s)}{\hat{P}(\bar{w}|s)})$. Previous work (Sanner, 2005) showed that we can express the log winning odds of a state described by *only the set of positive feature instances*:

$$\log \frac{\hat{P}(w|s)}{\hat{P}(\bar{w}|s)} = C + \sum_{i=1}^{p} \left(\log \frac{\hat{P}(f_i|w)}{\hat{P}(f_i|\bar{w})} - \log \frac{\hat{P}(\bar{f}_i|w)}{\hat{P}(\bar{f}_i|\bar{w})} \right) \quad (2)$$

Since C is a constant common to all states, we can ignore it during comparisons of log winning odds of states. Thus, even in a relational naive Bayes net with a large number of negative features, it is still possible to efficiently determine the highest-valued after-state in a comparative manner. This suffices to execute the optimal policy w.r.t. a value function.

3.2. Exploiting Relational Structure

Before we continue it is important to note that we can use locally weighted regression techniques (Atkeson et al., 1997) to exploit relational structure in the conditional probability tables (CPTs) of the Bayes net. While the ground Bayes net technique we have proposed so far is not relational in the traditional sense, it is also not purely propositional due to the fact that we can exploit relational structure in the CPT representation to reduce memory usage and increase learning generalization.

Our Tic-Tac-Toe example has been didactic from the standpoint of structure learning but the relational representation is too simple to demonstrate how we can exploit structure within the CPTs. For this, we need to examine more complex domains such as Backgammon and Othello.[4]

In **Backgammon**, we find it useful to use the following five feature attributes: (1) PT: Point on the board $\{1, \ldots, 24\}$ where the feature occurs. (2) BAR: Number of opponent pieces $\{0, \ldots, 15\}$ on the bar that must escape to the board before the opponent can resume normal play. (3) ON: Number of opponent pieces $\{0, \ldots, 15\}$ nearby, i.e. within 6 points of the current point. (4) OA: Total number of opponent pieces $\{0, \ldots, 15\}$ ahead of the point. (5) SZ: Size of a block $\{1, \ldots, 7\}$, i.e. number of contiguous points with at least two of the player's pieces. We then use these attributes to build the following three relational features:

1. $Attack(PT, BAR)$: The current player has attacked an exposed opponent on a point, moving it to the bar.
2. $Expose(PT, BAR, ON, OA)$: The current player has exposed a piece alone on a point.
3. $Block(PT, BAR, ON, OA, SZ)$: The current player has built a block of consecutive points, impeding the progress of opponents ahead of the block.

[3] An after-state (Sutton & Barto, 1998) is simply the state resulting from an agent's action before any other agent, if present, has chosen its respective action.

[4] While space limitations provide us from describing Backgammon and Othello here, we refer the reader to the web sites *http://www.bkgm.com/rules.html* and *http://www.rainfall.com/othello/rules/othellorules.asp* for a quick illustrated overview of the respective rules for these games.

In **Othello**, we use a very simple representation that abstracts heavily over the board state. While in future work, we plan to extend this representation to more accurately capture spatial patterns on the board (Buro, 1998), we find that the current relational representation suffices for reasonable performance against intermediate opponents. In our representation, we use the following two attributes: (1) CT: A count $\{0, \ldots, 64\}$ of board positions satisfying the feature. (2) $TURN$: The turn number $\{1, \ldots, 60\}$ at which the feature occurs. Strategy in Othello is highly dependent upon game stage and the use of this last attribute allows us to capture this dependency. We use these attributes to build the following five relational features:

1. $Pieces(CT, TURN)$: A simple count of the number of the current player's pieces on the board. Maximizing pieces is the objective of Othello although it can be harmful if done too early in the game.
2. $Mobility(CT, TURN)$: A count of the number of moves available to the current player.
3. $Adjacent(CT, TURN)$: The number of open positions adjacent to a current player's pieces.
4. $Edges(CT, TURN)$: The number of the current player's pieces at the non-corner edges of the board.
5. $Corner(CT, TURN)$: The number of the current player's pieces at the corners of the board.

Given the feature descriptions for these two games, we can now envision how we might exploit relational structure of these features to reduce memory usage and increase learning generalization. Suppose that we have learned that the Backgammon feature $Block(PT = 14, BAR = 0, ON = 7, OA = 8, SZ = 4)$ is predictive of a win. However, suppose that in the future, we can encounter the feature $Block(PT = 13, BAR = 0, ON = 6, OA = 7, SZ = 5)$, for which we have no previous experience. Given the intuitive similarity of these two features, we should be able to predict a win for the second. Likewise in Othello, if we know that the feature $Pieces(CT = 16, TURN = 24)$ is bad, then we should also be able to generalize that $Pieces(CT = 17, TURN = 26)$ is potentially bad even if we have never encountered it before. However, we can say less about $Pieces(CT = 34, TURN = 56)$ given information about $Pieces(CT = 16, TURN = 24)$ since these two features are much less similar.

To formalize these intuitions, we can treat ground atoms of an n-arity relation as a point in n-D space. To define the distance measure between points, we assign comparison metrics for attributes. In our experiments, we used the following three attribute distance measures:

1. *Linear (Lin)*: The distance between one attribute value and another is the difference of the values normalized by the attribute span (so as to have a range of $[0, 1]$). This is a natural distance measure for the PT, SZ and $TURN$ attributes.
2. *Zero-Diff-Linear (ZDL)*: This distance is identical to *Linear* except that the attribute value 0 has a maximal distance of 1 to all other attribute values. This measure is useful in features that do counting where the 0 count is extremely dissimilar to any count ≥ 1. For example, in Backgammon, having 0 opponents on the bar allows for normal play whereas having ≥ 1 opponent on the bar severely restricts the opponent's play. BAR, ON, OA, and CT use this distance metric.
3. *No similarity (NS)*: Here the max distance of 1 is assigned between all attribute values. This is used for the Tic-Tac-Toe attributes of $Mark$, Row, and Col and explains why we cannot exploit ground relational structure in this domain. The discrete nature of each of these attributes makes it difficult to generalize from one attribute value to another in this case.[5]

Now that we have assigned a distance measure appropriate for each attribute we can easily define the total Euclidean distance $D(F_i, F_j)$ between two ground atoms $F_i = R(v_{i,1}, \ldots, v_{i,k})$ and $F_j = R(v_{j,1}, \ldots, v_{j,k})$ of the same k-arity relation template $R(A_1, \ldots, A_k)$. To make this concrete, we compute the following function where $d(A, v_1, v_2)$ computes the assigned distance measure (i.e., (Lin, ZDL or NS) for attribute A given values v_1 and v_2:

$$D(F_i, F_j) = \sqrt{\sum_{p=1}^{k} \left(d(A_p, v_{i,p}, v_{j,p})\right)^2} \qquad (3)$$

Now, when we encounter a true ground atom target feature t at runtime, we first look in our *ground feature cache* for t's relation to see if we can find a set of features $F = \{f \in \textit{ground feature cache} \mid D(t, f) \leq \epsilon\}$. If we can, we assign each feature $f \in F$ a match *similarity* weight normalized by the sum of all match similarity weights for F:

$$f.match\text{-}weight = \frac{\sqrt{k} - D(t, f)}{\sum_{g \in F} \sqrt{k} - D(t, g)} \qquad (4)$$

Then for each $f \in F$, we compute $P(t|w) \approx \sum_{f \in F} P(f|w) \cdot f.match\text{-}weight$. Likewise when we update feature CPTs at the end of the trial, we update the respective wins or losses of f by $f.match\text{-}weight$ to reflect f's expected contribution to the outcome. On the other hand, if $F = \emptyset$, then we add t to our ground feature cache with a win-biased initial CPT and update it with weight 1 at the end of the trial.[6] In this way, ϵ serves as a parameter that determines how densely we populate our feature/CPT cache. The lower ϵ is, the more features/CPTs we will store and the less we will generalize – albeit for the tradeoff of higher prediction accuracy with sufficient experience.

[5] We *could* exploit relational structure in Tic-Tac-Toe in the form of spatial symmetry, but this is not yet implemented.

[6] This optimistic bias guarantees exploration of new features.

3.3. Structure Learning

The above framework permits us to learn a value function based on a simple relational naive Bayes net and to efficiently execute a policy derived from this value function, but often the naive Bayes structure is too impoverished to adequately approximate the optimal value function. To remedy this problem, previous work (Sanner, 2005) presented two types of relational Bayes net structure learning: feature attribute augmentation (FAA) and feature conjunction (FC). While FAA was intended to fine-tune the probability estimates in an individual CPT, FC was intended as the main feature discovery component and is the structure learning that we focus on here.

The concept of FC-learning is simple: at each structure learning step, greedily conjoin the CPTs for the two feature nodes that yield the greatest increase in log likelihood of the naive Bayes net. Joining feature nodes is a useful type of structure learning since it can learn non-independent correlations between features while maintaining the naive Bayes structure that makes policy evaluation efficient. As shown previously (Sanner, 2005), the feature join yielding the highest log-likehood $l^*(\theta|D)$ of the network parameters θ given the data D is the one that maximizes the following expression where M is the number of data samples, C is a constant, and $I(F_a, F_b|W)$ is the mutual conditional entropy of two features given the outcome W:

$$l^*(\theta|D) = C + M \cdot I(F_a, F_b|W) \tag{5}$$

The drawback of this structure learning algorithm is that it requires maintaining the joint probability over all node pairs in the naive Bayes net in order to calculate $I(F_a, F_b|W)$. For a relational naive Bayes net with only 100 ground features, this requires maintaining 10000 additional probability estimates. It can be seen that this quadratic growth quickly becomes intractable as the number of ground atoms increases. Dealing with this intractability is the main contribution of this paper.

4. Data Mining SVRRL Algorithm

Our goal is to learn structure in the relational naive Bayes net that maximizes utility to the learning agent. In attempting this, the first issue we note is that many portions of state space are visited too infrequently to yield low-variance estimates of joint probabilities (or mutual conditional entropies for that matter). Thus, it *makes sense to focus structure learning on those portions of state space that are visited frequently* in order to maximize the payoff of learning the structure while minimizing the variance (and noise) of the joint probability estimates of the learned features.

One popular method used in data mining for identifying frequent itemsets is the *Apriori* algorithm (Agrawal & Srikant, 1994). The basic motivating idea underlying Apriori is that a set of features can only positively co-occur with some minimum frequency if all proper subsets also positively co-occur with at least that minimum frequency. By building up frequent itemsets from the frequent singleton features, one can very efficiently find *all* sets of frequently positively co-occuring features.

In data mining SVRRL (DM-SVRRL), our goal is to discover structure in the form of frequently positively co-ocurring features.[7] The Apriori algorithm gives us an immediate method for doing this: Whenever two features co-occur in a state with some minimum threshold frequency, we build a new joint feature and keep track of its joint frequency (via its win/loss count). This serves two purposes: The joint frequency estimate allows us to keep track of frequency estimates when building even larger joint features and it also gives us the parameters for the CPT representing this joint node in the naive Bayes net. Thus, as a byproduct of applying an Apriori algorithm variant to a data stream of after-state features occurring during training, we can also learn joint probability estimates for *all* frequently co-occurring features!

To make the DM-SVRRL algorithm concrete, we present the training and update procedures for a game context in Algorithms 1 and 2. We note that each DM-SVRRL player maintains its own win/loss counts (overall, for each cached ground feature, and for each learned conjoined feature) in addition to recording all after-states from the game. The $Train$ procedure simply plays the game for a number of trials and updates each player upon receiving an outcome (a draw is considered a win for both). The $Update$ procedure tracks the overall win/loss counts as well as win/loss counts for ground and conjoined features occurring in the recorded after-states. When a conjoined feature reaches the min frequency threshold, it becomes available to the player for value estimation during normal game play. Finally we note that this algorithm always executes the greedy policy w.r.t. it's most recently updated value function. While there are no convergence guarantees for such interleaved policy evaluation and update methods, we find that we obtain convergent policies in practice.

5. Empirical Evaluation

We applied the DM-SVRRL algorithm to the domains of Tic-Tac-Toe (18 ground features), Othello (13,200 ground features), and Backgammon (786,816 ground features). We used the same relational features and attribute distance metrics as outlined for each game in Sections 3.1 and 3.2. In addition, we set ϵ to 0.02 (see Section 3.2).

[7] Ideally we would like to learn joint probabilities for pairs of positive and negative features as well. Unfortunately, this is non-trivial since most ground features occur negatively in a state and this would return us to the same difficulty that made FC-learning in SVRRL intractable.

Algorithm 1: $Train(trials, min\text{-}cache\text{-}freq)$

```
input  : trials, min-cache-freq: number of trials to train and min freq
         for using conjoined features
begin
    // Both players maintain win/loss counts as well counts for all cached
    // ground atom features, conjoined features they have learned so far,
    // and a record of all after-states from each game
    for (i := 1 ... trials) do
        whose-turn := random choice of BLACK or WHITE;
        Initialize game state;
        while (who-has-won() = NEITHER) do
            Generate set of after-states for all possible moves;
            foreach state ∈ after-states do
                TargetState := all positive ground features for state;
                F := ∅;
                foreach t ∈ TargetState do
                    F := F ∪ {f ∈ ground feature cache | D(t, f) ≤ ε}
                    // Note 1: D(t,f) is calculated according to Eq. 3.
                    // Note 2: Each f is assigned a match-weight from Eq. 4.
                Replace sets of ground atoms in F with maximal subsuming
                conjoined features from conjoined-features;
            state := best state ∈ after-states according to Eq. 2;
            Make move corresponding to state and record for player;
            whose-turn := get-other-player(whose-turn);
        Update(WHITE,who-has-won(), min-cache-freq);
        Update(BLACK,who-has-won(), min-cache-freq);
end
```

Algorithm 2: $Update(player, who\text{-}won, min\text{-}cache\text{-}freq)$

```
input  : player, who-won, min-cache-freq: the player (WHITE or
         BLACK), who won the game, and the min freq for using conjoined
         features
begin
    // All updates are local to player
    for (i := 1 ... # moves in game) do
        if who-won = player then
            total-wins := total-wins + 1;
        else
            total-losses := total-losses + 1;
        L_1 := all positive features for after-state from move #i;
        foreach f ∈ L_1 do
            if who-won = player then
                f.wins := f.wins + f.match-weight;
            else
                f.losses := f.losses + f.match-weight;
        // Use Apriori style alg. to find/update frequent joint feature sets
        k := 2;
        while (L_{k-1} ≠ ∅) do
            C_k := all size k supersets of L_{k-1} where all size k − 1 subsets
            meet min-cache-freq;
            foreach cf ∈ C_k do
                if who-won = player then
                    cf.wins := cf.wins + 1;
                else
                    cf.losses := cf.losses + 1;
            L_k := subset of C_k that meet min-cache-freq;
            conjoined-features := conjoined-features ∪ L_k;
            k := k + 1;
end
```

We trained for 5000 games in our experiments. In Tic-Tac-Toe, we trained against an optimal player, in Backgammon we trained against the linear neural network player *PubEval* provided by Gerry Tesauro (*www.netadelica.com/bg/bot/b_pubeval.html*), and in Othello, we trained against an opponent using exhaustive 4-ply lookahead with a specialized evaluation function (intermediate level play). We evaluated three algorithms, SVRRL without structure learning (VRRL), DM-SVRRL (F1) with a $min\text{-}cache\text{-}freq$ setting of 1 (i.e., it learns every joint combination of features seen during a game), and DM-SVRRL (F50) with a $min\text{-}cache\text{-}freq$ setting of 50. DM-SVRRL (F1) was capped at learning the first 2000 joint features since its unrestricted feature learning would exceed memory limits otherwise. Having done this, it only seemed fair to limit DM-SVRRL (F50) to its first 2000 features to avoid the possibility that DM-SVRRL (F50) performed better on account of having more features – in this way, we hope to obtain a reasonably good indication of the quality of the features learned by each algorithm. Table 1 provides final performance results playing the trained players against their training opponents for 1000 games.

Training times on a 3.2 Ghz Pentium ranged from 2 min. for Tic-Tac-Toe to 20 min. for Backgammon. Memory requirements ranged from 200 Kb for Tic-Tac-Toe to 3 Mb for Backgammon. For Tic-Tac-Toe, DM-SVRRL (F1) learns to play optimally since it can quickly learn all possible joint features (there are less than 1000). If we train long enough for all joint features to meet the minimum threshold of DM-SVRRL (F50), it learns the optimal policy as well; unfortunately its learning performance is much worse in the first 5000 games. On the other hand, the inability to learn conjoined features severely hurts VRRL. In Othello, there are more possible joint features than can be stored in memory so it is useful to learn only the high frequency features as evidenced by the fact that DM-SVRRL (F50) nearly doubles the performance of DM-SVRRL (F1). DM-SVRRL (F50) also far outperforms its non-structure learning variant VRRL. VRRL outperforms DM-SVRRL (F1) because the latter's low frequency joint feature probabilities add considerable statistical noise to the value function. In Backgammon, DM-SVRRL (F50) has a slight edge over the other two learners. Since Backgammon is a highly stochastic game, even a small increase in performance is evidence of a strategic advantage.

Overall, these results support two observations: (1) online feature discovery can help the system predict value better than the non-structure learning version, and (2) focusing feature discovery on frequently co-occurring features can outperform near-random structure learning when the relational state space is very large and the number of learned features is bounded by memory constraints.

6. Related Work

While all of the model-free relational RL approaches are too numerous to mention, many algorithms are variants of

ALGORITHM	WIN/DRAW %	DOMAIN
VRRL	28.3 %	TIC-TAC-TOE
DM-SVRRL (F1)	100.0 %	
DM-SVRRL (F50)	45.8 %	
VRRL	61.3 %	OTHELLO
DM-SVRRL (F1)	49.4 %	
DM-SVRRL (F50)	99.1 %	
VRRL	46.5 %	BACKGAMMON
DM-SVRRL (F1)	45.4 %	
DM-SVRRL (F50)	51.5 %	

Table 1. Win/Draw % ($\pm$ 1.6 %) of VRRL and DM-SVRRL variants in Tic-Tac-Toe, Othello, and Backgammon vs. training opponents. Draws are not possible in Backgammon so the values reported are simply the Win %.

an approach that approximates the value or Q-function with logical regression trees (Dzeroski et al., 1998). While this learning approach is top-down in that it recursively partitions the state space into finer value distinctions, more recent work (Walker et al., 2004; Croonenborghs et al., 2004) has taken a bottom-up approach to finding useful features for predicting value and combining them to make a predictive estimation of state value. While DM-SVRRL takes this latter approach, it is the only known relational RL system to focus on efficient online feature discovery using a relational naive Bayes net representation of the value function.

Our feature discovery ideas are similar in spirit to previous work in conjunctive feature construction in the context of game-playing (Buro, 1998). This work also focused on constructing frequent feature sets, but it explored this in an offline supervised learning setting where frequent conjunctive patterns were mined from a large collection of expert and perfectly evaluated training positions. Using this set of training data, the algorithm used iterative gradient-descent algorithms to fit the feature weights of its linear evaluation function. In contrast, while our feature construction and evaluation functions are quite similar, our algorithms are intended for an online reinforcement learning setting in the absence of expert or perfectly evaluated training data. Consequently, one can characterize our approach as asynchronous policy iteration that continuously bootstraps the value function from previous policy evaluations. To do this efficiently we have used a probabilistic intepretation of the value function that permits efficient closed-form parameter updates and we have exploited relational structure to reduce memory usage and increase learning generalization.

7. Concluding Remarks

We have presented a tractable data mining variant of the SVRRL algorithm that has allowed us to efficiently combine feature discovery with relational reinforcement learning. The empirical results are encouraging and the approach has a remarkably low resource consumption for the large feature domains of Backgammon and Othello.

There are a number of directions for future work, which we divide into two categories: learning more complex structure and learning in more complex domains. For the former, we would like to investigate learning higher level relational structure such as quantifiers, variable matching, rules (e.g., (Landwehr et al., 2005)), and probabilistic relational model (PRM) structure (Friedman et al., 1999). For the latter, we would like to investigate partially observable MDP (POMDP) (Kaelbling et al., 1998) and predictive state representation (PSR) (Littman et al., 2001) frameworks where we can introduce temporal attributes in our features required to learn policies in these domains. Finally, we note that since DM-SVRRL learns to compress state descriptions by representing states with successively larger joint features, it could be used as a means of learning a hierarchical abstraction of the domain to which model-based techniques could be applied. Altogether these possible extensions of the DM-SVRRL algorithm make it an exciting area for future work.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *In Proceedings of VLDB-94* (pp. 487–499).

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review, 11*, 11–73.

Buro, M. (1998). From simple features to sophisticated evaluation functions. *In Proc. of Computers and Games (CG'98)*.

Croonenborghs, T., Ramon, J., & Bruynooghe, M. (2004). Towards informed reinforcement learning. *ICML-04 Workshop on Relational RL*.

Dzeroski, S., de Raedt, L., , & Blockeel, H. (1998). Relational reinforcement learning. *ICML-1998* (pp. 11–22).

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. *IJCAI* (pp. 1300–1309).

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artif. Intell., 101*, 99–134.

Landwehr, N., Kersting, K., & de Raedt, L. (2005). nFOIL: Integrating naïve bayes and FOIL. *AAAI* (pp. 795–800).

Littman, M. L., Sutton, R. S., & Singh, S. P. (2001). Predictive representations of state. *NIPS* (pp. 1555–1561).

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Sanner, S. (2005). Simultaneous learning of structure and value in relational reinforcement learning. *ICML'05 Workshop on Rich Representations for Reinforcement Learning*.

Sutton, R. S., & Barto, A. (1998). *Reinforcement learning: An introduction*. MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. *ICML-04 Workshop on Relational RL*.

van Otterlo, M., & Kersting, K. (2004). Challenges for relational reinforcement learning. *ICML-04 Workshop on Relational RL*.

Walker, T., Shavlik, J., & Maclin, R. (2004). Relational reinforcement learning via sampling the space of first-order conjunctive features. *ICML-04 Workshop on Relational RL*.