Rule Value Reinforcement Learning for Cognitive Agents

Chris Child and Kostas Stathis
City University, London, UK
{c.child, k.stathis}@city.ac.uk

ABSTRACT

RVRL (Rule Value Reinforcement Learning) is a new algorithm which extends an existing learning framework that models the environment of a situated agent using a probabilistic rule representation. The algorithm attaches values to learned rules by adapting reinforcement learning. Structure captured by the rules is used to form a policy. The resulting rule values represent the utility of taking an action if the rule's conditions are present in the agent's current percept. Advantages of the new framework are demonstrated, through examples in a predator-prey environment.

Categories and Subject Descriptors

I.2.1 [**AI/GEN**]

General Terms

Algorithms

Keywords

Reinforcement learning, perception, action, planning, situated agents, stochastic, environment, probabilistic logic.

1. INTRODUCTION

The overall aim of our research is to build agents that can learn to act autonomously in a stochastic environment through experience gathered from interaction with the environment. Acquired stochastic logic rules are used to provide a compact model of the effects of agent action in the environment, and reinforcement learning techniques are used to plan within that model. RVRL provides a method for planning and action within this context.

If the environment the agent is modelling can be described in terms of a set of state variables, a factored state-model can be used. This describes the environment in terms of the dependencies between state variables and the evolution of these variables with respect to the actions taken by an agent.

The method used in this research is to create planning operators from experience of interactions with the environment. These rules predict how the environment will change when the agent takes an action. The general form of a stochastic planning operator is expressed as a rule of the form:

$p: e \leftarrow a, c$

p is the probability that the effects (*e*) of this operator will become true given the conditions (*a, c*) of the operator hold. *a* is the action taken by the agent, and *c* is a set of state variables representing the context of the agent's perception of the environment for the operator. Both *a* and *c* may be empty. In order to restrict the number of possible operators, *e* is defined to be a *single* state variable for each operator.

AAMAS'06, May 8-12, 2006, Hakodate, Hokkaido, Japan.

The process of building a rule set from experience requires the identification of conditions relevant to the effects of a rule. An effective method of building planning operators from experience is to use statistical significance to identify whether additional conditions are relevant to the outcome. This is the method used to create the rule sets in this work using ASDD [2]. ILP has also been used to learn rules of this form [3]. Precedence between rules is used to resolve conflicts in situations where two or more rule-sets match the conditions for the same output variable (for full details of the "precedence" algorithm, see [2]). This paper is an abridged version of the work presented in [1].

2. RULE VALUE REINFORCEMENT LEARNING

Section 1 outlined the use of rules to model an environment. The next task is to use this rule model to develop an effective policy. One method of achieving this is to use Q-Learning [1]. The update function for Q-learning is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R_{s'} + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2.1)$$

s and a are the states and actions. $Q(s,a)$ indicates the current Q value for the state action pair. This update rule gradually improves estimates on the target function Q. The α parameter is a step-size, indicating how quickly the new estimate should change the old one. γ indicates the discount factor, determining the influence of future rewards on the current state. If we use this function and take sample results (i.e. s' is taken to be the random result after taking action a in state s) the learning is one-step temporal difference (TD) learning.

2.1 The Rule Value Update Function

RVRL uses the same principle as TD learning to update a value associated with each rule, rather than each state. The main advantages of using a state-based aggregation method, such as RVRL, over standard reinforcement learning are that the agent:

a) Does not store a value-map containing every state-action.
b) Can generalize over many states, allowing each value to represent states with similar properties.

If a model of the environment is available, full backup values can be used. Rather than taking a random sample for s_{t+1}, the probability (*P*) of reaching each possible next state (s') given that action (a) was taken in state (s) can be used in the equation, and the best next action taken as the maximum action (a) for each possible next state, as in dynamic programming (DP). The update function for DP [4] is:

$$Q(s,a) \leftarrow Q(s,a) + \sum_{s'} P^{a}_{ss'} \alpha [R_{s'} + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (2.2)$$

The stochastic planning operators act as a model in RVRL. It is therefore possible to use an adaptation of equation (2.2). The rule values for operators cannot be updated directly using this equation because more than one rule will match the next state (`s'`) and would therefore be used to generate consecutive states due to several output variables being present. The rule learning function, therefore, replaces `Q(s',a')` with an average value for all matching rules which have precedence (and would therefore be used in generation of the successor state). The rules with precedence are used to give the most accurate representation of the dynamics of the environment in state s'.

`Q(s,a)` is replaced by the value of the rule which will be updated. All matching rules are updated in turn by the update function because their estimated value will be improved by the update, whether they have precedence or not. The update function for RVRL is defined as:

$$forEach\ (rule \in MatchingRules(s,a)) \left\{ \begin{array}{l} Q(rule) \leftarrow Q(rule) + \\ \sum_{s'} P^{a}_{ss'} \alpha [R_{s'} + \gamma \max_{a'} AvgQ(WinningRules(s',a')) \\ -Q(rule)] \end{array} \right\} \quad (2.3)$$

`AvgQ(WinningRules(s',a'))` returns the average rule value for rules which have precedence in state `s'` if action `a'` is taken. `AvgQ(WinningRules(s',a'))` finds the average value of the winning rules and returns the value. `MatchingRules(s,a)` returns all rules whose conditions match the current state and action. The values of all the returned rules are updated by equation (2.3).

2.2 Iterative Rule Value Evaluation

The process of building successor states described in [2], combined with the rule-value update function (2.3) allows continuous generation of next states from an initial state and updates the rule values until satisfactory values have been generated. This process is described by the following algorithm:

```
Initialise Q(rule) = 0, for all rule ∈ rules;
Repeat {
  Initialise s = random state, a = random action;
  Generate next states, s' and prob(s') for s,a
  totalValue = 0; totalReward = 0;
  For each s' ∈ successor states {
    totalReward += reward(s') * prob(s');
    maxActionValue = −∞;
    maxAction = null;
    For each a' ∈ actions {
       actionValue = AvgQ(WinningRules(s',a'));
       if (actionValue > maxActionValue)
         maxActionValue = actionValue; }
    totalValue += maxActionValue * prob(s'); }
  For each rules ∈ matchingRules(s,a) {
    Q(rule) = Q(rule) +
             α[totalReward + γ*totalValue;
             -Q(rule)]; }
} for n steps
```

The sampling (TD learning) equivalent to this method would take a sample next state `s'` rather than calculating the probability of each next state. The process is otherwise the same.

3. EXPERIMENTATION

We tested RVRL in a predator prey environment, consisting of a 4x4 grid surrounded by a wall and containing a predator agent (P) and a prey agent (A). P has caught A if A is on the same square as P at the end of A's move, represented as A being under P (or `Agent_U`). A selects a random action every move. Both P and A have four actions: move north (`Move_N`), east (`Move_E`), south (`Move_S`) and west (`Move_W`). An action moves an agent one square in the direction, unless there is a wall, in which instance there is no effect. P and A move alternate turns. An agent's percept gives the contents of the four squares around it and the square under it. Each square can be in one of three states: empty (`E`), wall (`W`) or agent (`A`).

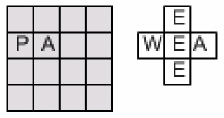

Figure 3-1: P, A in 4x4 grid. P's percept is shown to the right.

The task for RVRL was to construct an effective policy under the above, allowing the predator to catch the prey with optimal frequency. The task is complicated by the fact that the predator (P) is only adjudged to have captured the prey (A) if it moves into, or remains in, the predator's square at the end of its turn. P and A will continue to move after the prey is caught (continuous task).

In experiments it was found that, with a small amount of experience in the environment, P will tend to move next to A but not on-top of it. This is a good tactic as A is likely to move onto P and be caught. The optimal tactic, however, was found to be one in which P moves into A's square every move. This enables the predator (P) to be constantly in sight of the prey (A) and catch it whenever it moves into a wall. An example of a rule which captures this behaviour is: `Agent_N, Move_N`. Our experiments showed that RVRL gives high value to this rule and the `S, E` and `W` equivalents. Rules which attain higher value than the above have conditions such as: `{Agent_U, Move_N, Wall_N, Wall_E}`, in which P is on-top of the A in the NE corner of the map and choose action move north, into a wall (giving P a 50% chance of catching A). `Wall_N` and `Wall_E` are state variables representing that there is a wall north and east. The "effects" of the rules are not shown, because equal value will be learned for all rules with the same conditions. A sample of the final rule weights from rules learned from 60,000 moves experience after RVRL was run on the rule set for 15,000 iterations is given in Table 1.

Rules 1 and 2 have the same conditions but different actions (P is in the SW corner of the grid, with A underneath it). The rule has a positive value if P moves into a wall (1) and a negative value if the P moves away from the wall (2). P would, therefore, move into the wall and thus have the highest chance of catching A (50% chance of A moving into a wall).

Rules 3 and 4 have the same weight. If P takes the move north action (3), it will be on-top of A and will catch it if it moves into the wall to the west (25% chance). If P takes the move north action in (4) it will move into the wall and therefore stay on-top of A. P will then catch A if it moves into the wall to the north (25% chance). These two situations should be of equal utility to the agent, which has been successfully learned by RVRL.

Table 1: Sample rule weights for rules learned from 60,000 moves experience and 15,000 iterations of RVRL.

No.	*Conditions*	*Value*
1	`Move_W, Wall_W, Wall_S, Agent_U`	0.43
2	`Move_E, Wall_W, Wall_S, Agent_U`	-0.03
3	`Move_N, Wall_W, Agent_N`	0.11
4	`Move_N, Wall_N, Agent_U`	0.11
5	`Move_E, Agent_E`	-0.07
6	`Move_W, Agent_E`	-0.21
7	`Move_S`	-0.28
8		-0.28

Rules 5 and 6 show the weights for moving east onto the prey to the east and moving west away from a prey to the east. Moving onto the prey has higher weight as expected. Rules 7 and 8 have the same value. Rule 7 is the value of moving south with no other information. Rule 8 has no conditions and is thus the value of taking a random move in the environment. These rules are both effectively random and thus have equal weight.

The performance of RVRL was compared with:

a) Dyna-Q: Q-Learning reinforcement learner with frequency based environment model.

b) SR-Q: A stochastic rule based model of the environment was used to build a state, action value map. This is the equivalent of running Q-learning, using the rule based model for experience to build the Q(s, a) map. This method is described in [2].

RVRL builds a Q(rule) map, assigning value to each rule. Table 2 gives a comparison of the three methods.

In each test case the methods were given the same experience with which to build the model. The predator and prey were run for a set number of steps, taking random moves at each step. Using the model, each method ran Q-learning (in the first two cases), or RVRL for 15,000 iterations to build a value map. Once the map had been created, each method ran for 40,000 steps in the predator prey environment, selecting the action with the highest utility at each step. The number of times P caught A was then recorded. Average moves taken for P to capture A is given in Table 2.

The two Q-learning based methods selected the best action at each step picking the highest valued action from all matching Q(s,a) values for the current state (`s`). RVRL picked the highest valued action from all matching `AvgQ(WinningRules(s,a))`.

Table 2: Moves per capture: Dyna-Q, Stochastic Rule model Q (SR-Q) and Rule Value Reinforcement Learning (RVRL).

Method	100	500	1000	10000	15000	30000	60000
Dyna-Q	17.5	16.4	12.0	8.8	7.4	6.2	4.6
SR-Q	13.1	13.4	11.5	9.2	8.8	7.1	4.7
RVRL	13.2	12.7	11.3	9.3	8.1	7.0	4.7

Moves per capture for P taking random moves is 16.01. A trail was also run on a "perfect" model (Dyna-Q built from 400,000 moves). In this instance P averaged 4.32 moves per capture.

The results in Table 2 for 100, 500 and 1000 moves training data show that RVRL is more effective than Dyna-Q when little experience has been gathered. In this case the Dyna-Q agent is forced to take a random move in many of the states encountered in the test, because it has no experience which matches the situation. With this limited model, Dyna-Q "expected" the prey to move in the same way as it did in the training data, resulting in picking a poor action. The RVRL agent, however, was able to make generalisations in two ways: first to generalise the *model* using the stochastic logic rules, which allows the system to predict future states from the current state, even when this state has not been seen before; second, RVRL learned values are applicable across multiple states, allowing learned values to be applied in unseen states. This allows the small amount of experience gathered to be generalised and used, which is demonstrated by the improved performance under these conditions. SR-Q is only able to make use of the first of these generalisations, and therefore performed slightly better than Dyna-Q, but not as well as RVRL.

As the state action map gains a larger amount of experience (10,000, 15,000 and 30,000 steps), its model becomes closer to a perfect model in this test environment, while the generalisations made by the rule learner become less effective. This is due largely to shortcomings in the ASDD modelling method with this level of training data [2] which is reflected in the similar performance of the SR-Q results, rather than shortcomings in the RVRL algorithm. When the learned rules become a near perfect representation of the environment (at 60,000 steps training data), the results show that RVRL is capable of learning near perfect valued rules, and thus the utility of taking an action in the current state, again demonstrating that the rule values are capable of capturing a policy at least as effectively as a state action model under these conditions.

4. CONCLUSIONS

This paper has presented the Rule Value Reinforcement Learning (RVRL) method. RVRL is a state-based aggregation technique, in that states which behave in a similar way with respect to a given action sequence and goal are given the same value. Results in our experimentation are extremely encouraging in that the algorithm is able to learn rule-values which accurately capture the utility of actions in the predator-prey environment without the need for a state-action map.

5. REFERENCES

[1] Child, C. and Stathis, K. "Learning to Act with RVRL agents". City University Technical Report. (2006).

[2] Child, C. and Stathis, K. "The Apriori Stochastic Dependency Detection (ASDD) Algorithm for Learning Stochastic Logic Rules", *In Proceedings of the 4th International Workshop on Computation Logic in Multi-agent Systems (CLIMA-04),* J. Dix, J. Leiter (Eds), Florida, Jan, (2004).

[3] Pasula, H. M, Zettlemoyer, L.S. and Kaelbling, L.P. "Learning Probabilistic Relational Planning Rules." *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling,* ICAPS, 73-82, (2004).

[4] Sutton, R.S., and Barto, A.G. "Reinforcement Learning: An Introduction". A Bradford Book, MIT Press, (1998).

[5] Watkins, C. J. C. H. "Learning from Delayed Rewards." PhD thesis, Cambridge University, (1989).