Reinforcement Learning with Markov Logic Networks

Weiwei Wang, Yang Gao, Xingguo Chen, and Shen Ge

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, PRC
elegate@gmail.com

Abstract. In this paper, we propose a method to combine reinforcement learning (RL) and Markov logic networks (MLN). RL usually does not consider the inherent relations or logical connections of the features. Markov logic networks combines first-order logic and graphical model and it can represent a wide variety of knowledge compactly and abstractly. We propose a new method, reinforcement learning algorithm with Markov logic networks (RLMLN), to deal with many difficult problems in RL which have much prior knowledge to employ and need some relational representation of states. With RLMLN, prior knowledge can be easily introduced to the learning systems and the learning process will become more efficient. Experiments on blocks world illustrate that RLMLN is a promising method.

1 Introduction

State representation is a critical task in RL. Function approximation is an approach to dealing with high-dimension tasks. However, RL usually does not consider inherent relations or connections of the features. Otherwise we need to introduce additional features to represent such connections. Therefore, we need a high level relational and abstract representation in RL for real world problems where there are enormous state spaces.

Relational reinforcement learning (RRL) is concerned with upgrading the representation of RL methods to the first-order case, that is, reasoning and learning about objects and relations between Objects ([1]). RRL is modeled by relational MDPs (RMDP). For a detailed definition, please see ([1]). Recently, researchers have presented a lot of methods to solve RRL problems, which can be classified into three classes: model-free, partially modeling and model-based. Among these methods, many integrate first-order logic with traditional method and gain the ability to compactly and declaratively represent complex problems. For example, LOMDP ([2]) use clauses or formulas to partition state space and learn state values for those formulas.

Markov logic networks (MLN), proposed by Matthew Richardson and Pedro Domingos ([3]), attaches weights to first-order formulas and takes them as features of the network. It can be viewed as a template for generating ground Markov networks. In this way, MLN achieves the goal of combining graphical

A. Gelbukh and E.F. Morales (Eds.): MICAI 2008, LNAI 5317, pp. 230–242, 2008.

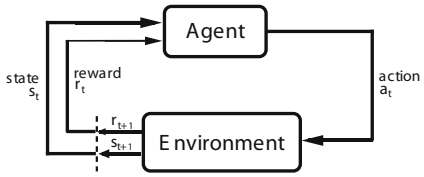

Fig. 1. The agent-environment interaction in reinforcement learning

model and first-order logic, so that it handles the complexity and uncertainty of the real world in a single framework. Recently, MLN has been used to deal with various kinds of problems. For example, Parag Singla and Pedro Domingos used it to do entity resolution ([4]), collective classification and so on. Transfer learning based on MLN ([5]) is also undertaken.

The above work inspires us to propose reinforcement learning algorithm with Markov logic networks (RLMLN) to combine RL and MLN. In RLMLN, MLN does inference for the action queries and selects a best action, while RL uses the successive state, current state and the reward to update the weights of formulas in MLN. With RLMLN, we can compactly represent a state in RL for those problems which have enormous state space and need abstract state representation. Furthermore, we can easily introduce prior knowledge to a learning system. We apply RLMLN to the problem of blocks world ([6]) and the experimental results show that RLMLN is a promising method.

The rest of this paper is organized as follows. In sections 2 and 3, we briefly review reinforcement learning and Markov logic networks. Section 4 presents RLMLN. Section 5 gives the experiments on blocks world. Finally, section 6 concludes.

2 Reinforcement Learning

Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal([7]). A reinforcement learning agent learns knowledge from interaction with the environment, as shown in Fig. 1. We normally describe a reinforcement learning problem using Markov decision processes(MDPs). A MDP is a four tuple $< S, A, T, R >$ consisting of a finite state space S, a finite action space A, a transition function $T : S \times A \times S \rightarrow \Re$, and a reward function $R : S \times A \rightarrow \Re$([7]). From a given state $s \in S$, a given action $a \in A(s)$ produces an expected reward of $R(s, a)$ and transitions to another state $s' \in S$ with probability $T(s, a, s')$. Monte-Carlo, TD(λ), Q-learning are some of the classical methods in RL.

3 Markov Logic Networks

Complexity and uncertainty is the nature of many real world applications ([8]). Logic mainly handles the former while statistical learning focuses on the latter.

However, a first-order KB has a hard constraint on the possible worlds because if one formula is violated, then the world will has zero probability. Markov logic soften the constraints by combining logic and probability with attached weights for first-order formulas: if a world violates a formula then it becomes less probable, but not impossible([9]).

3.1 Markov Logic

A Markov network is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \chi$([10]). It consists of an undirected graph G and a set of potential functions ϕ_k. There's a node for each variable in the graph and the model has a potential function for each clique in the graph. The joint distribution represented by a Markov network is given below:

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \ . \tag{1}$$

where $x_{\{k\}}$ is the state of the kth clique, Z=$\sum_{x \in \chi} \prod_k \phi_k(x_{\{k\}})$. For convenience, we often represent Markov networks as *log-linear* models:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right) \ . \tag{2}$$

where Z is a normalization factor, and $f_j(x)$ is one of the feature of state x. Markov logic introduces first-order logic to Markov networks. Recall that a first-order knowledge base is a set of formulas represented in first-order logic. Constants(e. g. , Jack, Johnson), variables, functions and predicates(e. g. , On(a, b), Has(x, y)) are the four types of symbols to construct these formulas. A grounded predicate is a predicate with its variables replaced by constants(e. g. , move(A, Floor), Has(John, computer)). A ground formula is a formula with no variables but constants. In Markov logic, a formula is a first-order logic formula attached with a weight. The weight of each formula reflects how strong a constraint the first-order formula has: the higher the weight, the greater the difference in probability between a world that satisfies the formula and one that does not. And a set of these formulas is called a Markov logic network or a MLN([8]).

The definition is given below([3]):

Definition 1. *A Markov logic network L consists of a set of pairs* (F_i, w_i)*, where* F_i *is a first-order logic formula and* w_i *is* F_i*'s weight(a real number).* $C = c_1, c_2, \ldots, c_{|C|}$ *is a set of constants. Markov network* $M_{L,C}$ *is defined as follows:*

1. $M_{L,C}$ *consists of binary nodes for each possible grounding of each predicate in L. The value of each node is* 1 *if the ground predicate is true,* 0 *otherwise.*
2. $M_{L,C}$ *contains one feature for each grounding of each formula* F_i *in L. The value of this feature is* 1 *if the ground formula is true,* 0 *otherwise. The weight of the feature* F_i *is* w_i*.*

Table 1. Example of MLN for blocks world

First-order logic	Clausal form	Weight
$\forall a, b, c, d$ on$(a, b) \wedge$ clear$(a) \wedge$ on(c, d)	$\neg$on$(a, b) \vee \neg$clear$(a) \vee \neg$on(c, d)	1.0
$\wedge$clear$(c) \Rightarrow$ move$(a, Floor)$	$\vee\neg$clear$(c) \vee$ move$(a, Floor)$	
$\forall a, b, c, d$ on$(a, b) \wedge$ clear$(a) \wedge$ on(c, d)	$\neg$on$(a, b) \vee \neg$clear$(a) \vee \neg$on(c, d)	1.0
$\wedge$cl$(c) \Rightarrow$ move$(x, Floor)$	$\vee\neg$clear$(c) \vee$ move(a, c)	

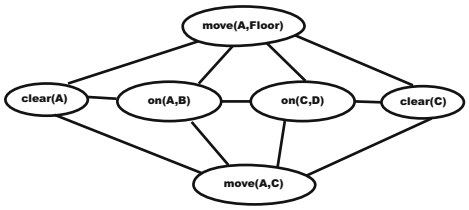

Fig. 2. Ground Markov network obtained by applying the formulas in Table 1 to constants `A`, `B`, `C` and `D`. And we delete nodes such as `move(A, A)`, `on(A, A)`.

A MLN can be used as a template to construct Markov networks and the resultant networks are called ground Markov networks. And the networks will vary when given different sets of constants. Table. 1 gives part of the formulas in a MLN for blocks world. An example of ground Markov networks for Table. 1 is shown in Fig. 2. The probability of world x specified by $M_{L,C}$ is given by:

$$P(X = x) = \frac{1}{Z}\exp\left(\sum_i w_i n_i(x)\right) = \frac{1}{Z}\prod_i \phi_i(x_{\{i\}})^{n_i(x)} \quad . \tag{3}$$

where $n_i(x)$ is the number of true groundings of F_i in x, $x_{\{i\}}$ is the state of the atoms appearing in F_i, and $\phi_i(x_{\{i\}}) = e^{w_i}$([3]).

3.2 Inference

Here we discuss the MCMC algorithm used in MLN for inference as our method is based on this inference technique. First we introduce the formula to calculate probability that formula F_1 holds given that F_2 does([3]).

$$\begin{aligned} P(F_1|F_2, L, C) &= P(F_1|F_2, M_{L,C}) \\ &= \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} \\ &= \frac{\sum_{x\in\chi_{F_1}\cap\chi_{F_2}} P(X = x|M_{L,C})}{\sum_{x\in\chi_{F_2}} P(X = x|M_{L,C})} \quad . \end{aligned} \tag{4}$$

where χ_{F_i} is the set of worlds where F_i holds, and $P(X|M_{L,C})$ is given by Eq.3.

$P(F_1|F_2, L, C)$ can be approximated using an MCMC algorithm that rejects all moves to states where F_2 does not hold, and counts the number of samples in which F_1 holds. In MCMC algorithm we have two steps to do for inference. First, we construct the minimal subset M of the ground Markov network required to compute $P(F_1|F_2, L, C)$. Second, we use Gibbs sampling to perform inference on this network. Gibbs sampling need a Markov blanket to sample one ground atom. The Markov blanket of a ground predicate is the set of ground predicates that appear in some grounding of a formula with it([3]). Given Markov blanket $B_l = b_l$, the probability of a ground predicate X_l is

$$P(X_l = x_l|B_l = b_l) = \frac{\exp(\sum_{f_i \in F_l} w_i f_i(X_l = x_l, B_l = b_l)))}{\exp(\sum_{f_i \in F_l} w_i f_i(X_l = 0, B_l = b_l)) + \exp(\sum_{f_i \in F_l} w_i f_i(X_l = 1, B_l = b_l))} \quad (5)$$

where F_l is the set of ground formulas that X_l appears in, and $f_i(X_l = x_l, B_l = b_l)$ is the value(0 or 1) of the feature corresponding to the ith ground formula when $X_l = x_l$ and $B_l = b_l$ ([3]).

For more detailed information about MLN, please refer to ([3]).

4 Reinforcement Learning Algorithm with Markov Logic Networks

Based on MLN, we propose our reinforcement learning algorithm with Markov logic networks. First we give a simple view of our method in Fig. 3. From the figure, we can see that given current state s, MLN helps calculate the probability of each action and randomly choose among the best ones, $a = \max_a P(a|s)$, and then next state s', reward r are received by RL agent from the environment. Using gradient-descent method, RL agent updates the weights vector $\vec{w}$ of MLN and then use the new weights to do inference. The inference algorithm we choose for MLN is MCMC as we mentioned before.

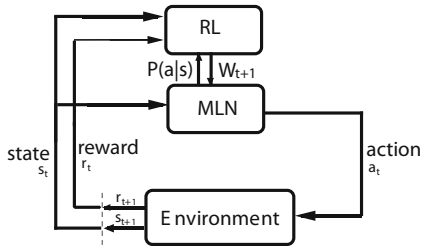

Fig. 3. Framework of reinforcement learning algorithm with Markov logic network

4.1 Update Weights with Gradient-Descent Methods

Gradient-descent method is a popular method used in RL and other fields of AI. The simple form is given below([7]):

$$\begin{aligned}\vec{\theta}_{t+1} &= \vec{\theta}_t + \frac{1}{2}\alpha \bigtriangledown_{\vec{\theta}_t} [V^\pi(s_t) - V_t(s_t)]^2 \\ &= \vec{\theta}_t + \alpha[V^\pi(s_t) - V_t(s_t)] \bigtriangledown_{\vec{\theta}_t} V_t(s_t) \ . \end{aligned} \tag{6}$$

where α is a positive step-size parameter, $V_t(s)$ is a smooth differentiable function of $\vec{\theta}_t$ for all $s \in \mathcal{S}$, $V^\pi(s_t)$ is the exact, correct value for each s_t, and $\bigtriangledown_{\vec{\theta}_t} V_t(s_t)$, for any function f, denotes the vector of partial derivatives. Normally, we can't get the exact value $V^\pi(s_t)$, so we use some approximation of it, denoted by v_t. Now we try to take MLN output as a representation of Q value and get a new formula:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \left[r + \gamma \max_{a'} P(a'|s') - P(a|s) \right] \bigtriangledown_{\vec{\theta}_t} P(a|s) \ . \tag{7}$$

where $P(A = a|S = s) = \frac{1}{Z}\exp(\sum_j w_j f_j(x))$(Equ. 2). For simplicity of calculation of $\bigtriangledown_{\vec{\theta}_t} P(a|s)$, we adopt the ln form of $P(a|s)$. So we get

$$\frac{\partial}{\partial w_i} \ln P_w(A = a|S = s) = n_i(x) - \sum_{x'} P_w(X = x') n_i(x') \ . \tag{8}$$

where $n_i(x)$ is the number of true groundings in the data x for the ith formula. However, Equ. 8 is difficult for calculation. As mentioned before, we use MCMC algorithm in our method, so $P(a|s)$ is calculated using Equ. 5. In this way, we can get $\bigtriangledown_{\vec{\theta}_t} P(a|s)$ as below in a simple form:

$$\begin{aligned}&\frac{\partial}{\partial w_i} \ln P(X_l = x_l|B_l = b_l) \\ &= \sum_{f_i \in F_l} \left\{ f_i(X_l = x_l, B_l = b_l) - \sum_{x' \in 0,1} f_i(X_l = x', B_l = b_l) P(X_l = x'|B_l = b_l) \right\} \end{aligned} \tag{9}$$

where F_l is the set of ground formulas with weight w_i that X_l appears in, and $f_i(X_l = x_l, B_l = b_l)$ is the value(0 or 1) of the feature corresponding to the ith ground formula when $X_l = x_l$ and $B_l = b_l$; $P(X_l = x'|B_l = b_l)$ is the probability that $X_l = x'$ holds given Markov blanket b_l.

In such a way, we can use this gradient-descent method to update the weight of each formula in MLN in each state transition process. Different from MLN, we introduce state and action, which only exist in RL, to MLN. The original weight learning methods in MLN use an evidence database to do gradient-descent learning while in RL we have states, actions and state transitions which can't be represented simply in an evidence database. That's why we can't directly use the weight learning methods of MLN. Now we can see that MLN helps RL to choose actions while RL helps MLN to learn weights. We call this method reinforcement learning algorithm with Markov logic networks.

4.2 RLMLN with Eligibility Trace

Combining with eligibility trace, we can get the backward view([7]) of our RLMLN algorithm.

$$Q^{\pi}(s,a) = \ln((r + \gamma \max_{a'} P(A = a'|S = s')))$$
$$Q(s,a) = \ln P(A = a|S = s)$$
$$\delta = Q^{\pi}(s,a) - Q(s,a)$$
$$\vec{w}_{t+1} = \vec{w}_t + \alpha\delta \frac{\partial}{\partial w_t} Q(s,a)\vec{e}_t$$
$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \frac{\partial}{\partial w_t} Q(s,a)$$
$$\frac{\partial}{\partial w_i} Q(s,a) = \frac{\partial}{\partial w_i} \ln P(A = a|S = s)$$
$$\frac{\partial}{\partial w_i} \ln P(X_l = x_l|B_l = b_l)$$
$$= \sum_{f_i \in F_l} \left\{ f_i(X_l = x_l, B_l = b_l) - \sum_{x' \in 0,1} f_i(X_l = x', B_l = b_l) P(X_l = x'|B_l = b_l) \right\}$$

where $Q^{\pi}(s,a)$ is the exact, correct value for state-action pair (s,a), and here we use an estimate $\ln((r + \gamma \max_{a'} P(A = a'|S = s')))$ to substitute it. $\vec{w}_t$ is the weights of formulas in MLN; $Q(s,a)$ is the Q-value of state-action pair (s,a) given by MLN inference.

A description of this algorithm is given in Fig. 4. In this algorithm, we get $P(a|s)$ from the inference of MLN(Equ. 5) and then apply RL to update the weight of each formula(Equ. 8). Our implementation is based on the open source project–Alchemy.

5 Experiment on Blocks World

5.1 Blocks World Problem

A blocks world contains a fixed set of blocks. Each block has two positions:on the top of another block or on the floor. The action in this world is `move(a,b)`, where `a` must be a block and `b` is either a block or the floor. Marina Irodova and Robert H. Sloan([11]) applied RL and function approximation to this task and got a better result than RRL. However, at present it is still a difficult task in AI. In our experiment, we also consider three goals as other researchers do.

Stack States with all blocks in a single stack
Unstack States with every block on the floor
On(a, b) States with block `a` on top of block `b`

For ease of introducing prior knowledge, we introduce three predicates to blocks world, they are `ontop(x,y)`, `above(x,y)`, `maxheight(x)`. Predicate

Initialize $\vec{w}$ of MLN formulas with prior knowledge or just a constant arbitrarily chosen, $\vec{e} = \vec{0}$
Repeat(for each episode):
  $s \leftarrow$ initial state of episode
  For all $a \in \mathcal{A}(s)$:
    $Q_a \leftarrow P(A = a|S = s)$ //(MLN)
  $a \leftarrow \arg\max_a Q_a$
  With probability $\epsilon : a \leftarrow$ a random action $\in \mathcal{A}(s)$
  Repeat(for each step of episode):
    Take action a, observe reward, r, and next state s'
    $Q(s, a) = \ln Q_a$
    For all $a \in \mathcal{A}(s')$
      $Q_a \leftarrow P(A = a|S = s')$ //(MLN)
    $a' \leftarrow \arg\max_a Q_a$
    With probability $\epsilon : a' \leftarrow$ a random action $\in \mathcal{A}(s')$
    $Q^{\pi}(s, a) = \ln(r + \gamma Q_{a'})$
    $\delta \leftarrow Q^{\pi}(s, a) - Q(s, a)$
    $\frac{\partial}{\partial w_i} Q(s, a) = \frac{\partial}{\partial w_i} \ln P(A = a|S = s)$ //(MLN, Equ. 9)
    $\vec{w}_{t+1} = \vec{w}_t + \alpha\delta \frac{\partial}{\partial w_t} Q(s, a)\vec{e}_t$
    $\vec{e}_{t+1} = \gamma\lambda\vec{e}_t + \frac{\partial}{\partial w_t} Q(s, a)$
    $a \leftarrow a'$
  until s' is terminal

Fig. 4. Reinforcement learning algorithm with Markov logic networks

ontop(x, y) means block x is on the top of block y; above(x, y) means block x is above block y; and maxheight(x) means block x has the maximum height than other blocks. We define height of block x like this: if on(x, Floor), then height(x) = 0, and if on(x, y), height(x) = height(y) + 1. Predicates ontop(x, y), above(x, y) can be defined using on(x, y), clear(x, y) recursively as Markov nets allow cycles.

{on(A, B), on(B, Floor), on(E, C), on(E, C), on(C, D), on(D, Floor)} is the initial state in our experiment. The predicate to be inferred is move. In our experiment, the goal state for *on* task is {on(B, D)}. The goal state of *unstack* and *stack* task is trivial. MLN formulas for the three tasks are given below:
On task:

```
//predicates
on(bl, bl)
move(bl, bl)
clear(bl)
maxheight(bl)
ontop(bl, bl)
above(bl, bl)
//formulas
1.0 clear(B) ^ clear(D) => move(B, D)
```

```
1.0 ontop(x, B) => move(x, Floor)
1.0 ontop(x, D) => move(x, Floor)
1.0 on(x, y) => above(x, y)
1.0 on(x, y) ^ above(y, z) => above(x, z)
1.0 clear(y) ^ above(y, x) => ontop(y, x)
```

Stack task:

```
1.0 maxheight(x) ^ clear(y) => move(y, x)
```

Unstack task:

```
1.0 !on(a1,Floor) ^ clear(a1) => move(a1,Floor)
```

In the stack and unstack MLN formulas, we omit the predicate declarations so as to avoid duplication. At the rest of this paper we also omit the declarations in MLN formulas.

5.2 Experiment Results

In this section, we perform an experiment to validate our RLMLN algorithm. The problem domain is a 5-block blocks world task as shown in Fig.5. Fig. 6 gives the experiment result. Parameter setting in our experiment is $\alpha = 0.1, \gamma = 0.99$ and $\epsilon = 0$. Taking account of the accuracy of inference and equivalent initial weight of MLN formulas , we choose actions randomly among the actions whose probability is near the maximum probability, that's why we set $\epsilon = 0$ in this task. Another reason is that we have provided very exact prior knowledge for every task and a positive ϵ will cause unnecessary fluctuation. For different task(*stack, unstack, on*), reward is 1 when the goal is reached, 0 otherwise. No comparisons are provided here because the learned results are optimal for the three tasks and as a result comparisons are not necessary on this problem. Further study of RLMLN on other domains, like Tetris, is undertaken and comparisons on these domains will be taken in the near future.

5.3 Transfer to Larger Blocks World Problems

For blocks world task, we find that our formulas above can easily transfer to larger blocks world problems. We design an experiment of a *on* task with 10 blocks. The task starts with a random initial state and the goal state is $\{\texttt{on}(\texttt{B7}, \texttt{B3})\}$. We use the above learned MLN formulas with only `B`, `D` changed to `B7`, `B3`. We run the experiment twice. The result is given in Fig. 7 and the initial states are(randomly generated in one run):
S1:

$$\{\texttt{on}(\texttt{B9}, \texttt{Floor}), \texttt{on}(\texttt{B8}, \texttt{B7}), \texttt{on}(\texttt{B7}, \texttt{B6}), \texttt{on}(\texttt{B6}, \texttt{Floor}), \texttt{on}(\texttt{B5}, \texttt{B4}), \texttt{on}(\texttt{B4}, \texttt{B3}),$$
$$\texttt{on}(\texttt{B3}, \texttt{Floor}), \texttt{on}(\texttt{B2}, \texttt{B1}), \texttt{on}(\texttt{B1}, \texttt{B0}), \texttt{on}(\texttt{B0}, \texttt{Floor})\}$$

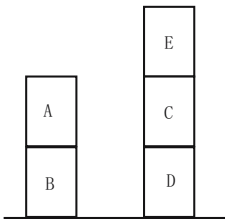

Fig. 5. Stack task

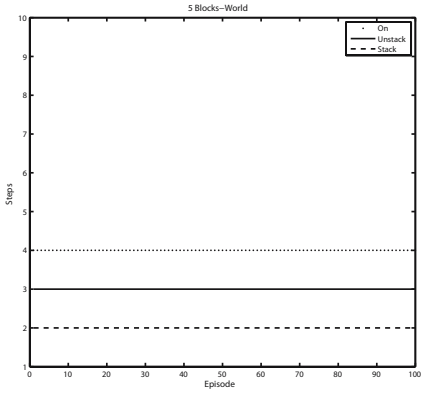

Fig. 6. Experiment with 5 blocks

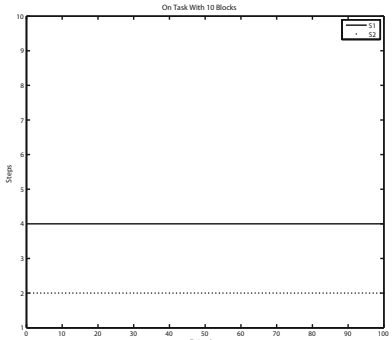

Fig. 7. Transfer to 10 Blocks On Task

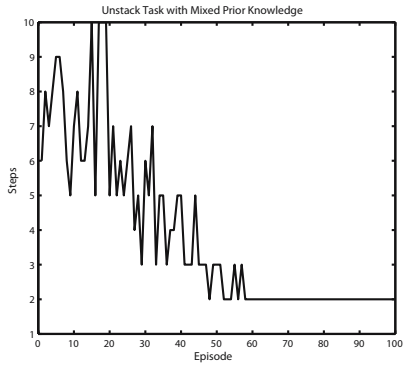

Fig. 8. Experiment with Mixed Knowledge

S2:

$\{\texttt{on}(\texttt{B0},\texttt{Floor}), \texttt{on}(\texttt{B1},\texttt{B0}), \texttt{on}(\texttt{B2},\texttt{B1}), \texttt{on}(\texttt{B3},\texttt{B2}), \texttt{on}(\texttt{B4},\texttt{Floor}), \texttt{on}(\texttt{B5},\texttt{B4}),$
$\texttt{on}(\texttt{B6},\texttt{B5}), \texttt{on}(\texttt{B7},\texttt{B6}), \texttt{on}(\texttt{B8},\texttt{B7}), \texttt{on}(\texttt{B9},\texttt{Floor})\}$

5.4 Mix the Prior Knowledge

In the above experiment, we give an exact prior knowledge about blocks world, so the learning process seems to become useless. The reason is that blocks world problem is easy for humans and we can easily figure out the exact prior knowledge. However, for complex problems like Tetris, it's difficult to give an exact prior knowledge and then we will see the power of RLMLN to distinguish good

formulas and bad formulas(by modifying the weights). In this subsection, we put all the formulas for the three different tasks and do an experiment on unstack task with 4 blocks to validate the power of RLMLN in adjusting weights or selecting formulas(or prior knowledge). In order to confuse RLMLN at the beginning, we set the formulas that are against the unstack goal with higher weights. The initial MLN is:

```
2.0 clear(B2) ^ clear(B0) => move(B2,B0)
1.0 ontop(x,B2) => move(x,Floor)
1.0 ontop(x,B0) => move(x,Floor)
1.0 on(x,y) => above(x,y)
1.0 on(x,y) ^ above(y,z) => above(x,z)
1.0 clear(y) ^ above(y,x) => ontop(y,x)
1.0 clear(a1) ^ !on(a1,Floor) => move(a1,Floor)
2.0 maxheight(x) ^ clear(y) => move(y,x)
```

The learned formulas are given below:

```
1.99943  move(B2,B0) v !clear(B2) v !clear(B0)
1.47049  !ontop(a1,B2) v move(a1,Floor)
1.47049  !ontop(a1,B0) v move(a1,Floor)
1        !on(a1,a2) v above(a1,a2)
1        !on(a1,a2) v above(a1,a3) v !above(a2,a3)
1        ontop(a1,a2) v !above(a1,a2) v !clear(a1)
1.47049  on(a1,Floor) v move(a1,Floor) v !clear(a1)
1.96995  move(a1,a2) v !maxheight(a2) v !clear(a1)
```

In this experiment, $\epsilon = 0.1$ and the other parameters are the same as those used above. We can see the weights of the formulas fit the task are incremented in the learning process while the others are decreased or remain. We run the experiments 10 times and the initial state is generated from a random number generator with the same seed $\texttt{seed} = \texttt{time}(\texttt{NULL})$ in the 10 runs. The result is shown in Fig. 8.

From the experiment result, we can see that our new method works very well on blocks world problem when we introduce some prior knowledge. If the formulas are abstract and exact enough, the learned MLN formulas can be easily transfered to larger tasks. On other tasks other than blocks world, the learned MLN may need some modification or relearning from the learned formulas. Of course we can use function approximation on this problem, however, finding good features is still a difficult and arduous job. With RLMLN, finding a solution for such tasks becomes easier and faster as the formulas give us a high level and comprehensible abstraction of this problem while function approximation in RL often introduces obscure features and needs some programming tricks. Currently, we are taking a study on applying RLMLN to Tetris and in the near future we will be able to give a better example of RLMLN's applications.

6 Conclusion and Future Work

In this paper, we propose reinforcement learning algorithm with Markov logic networks (RLMLN). This method helps us represent states in RL compactly and introduce prior knowledge to a learning system. Furthermore, figuring out a set of formulas of RLMLN is easier than finding features for function approximation in some problem domains, such as the blocks world. Experimental results on blocks world show that RLMLN is a promising method. If the problem can be formulated in RRL framework, like blocks world here, the learned RLMLN can be easily transfered or directly used to larger tasks. However, not all tasks can easily be formulated as relational problems, which inludes mountain car and acrobot, etc([12]).

We are now planning to extensively examine the performance of RLMLN on more problem domains, like Tetris. However, hand coded formulas may be not enough for such a task, so structure learning of RLMLN will be also a collar work for us in the near future.

A recursive random field(RRF)([13]) is a representation where MLNs can have nested MLNs as features. Recursive random fields(RRFs) have more power than MLNs as they can represent disjunctions and conjunctions ,and universal and existential quantifiers symmetrically. As a result, we are now also considering to upgrade MLN in RLMLN to RRF.

Acknowledgements

The work is supported by the Natural Science Foundation of China (No.60775046 and No.60721002). We would like to thank Yinghuan Shi and Tianyin Xu for helpful comments on the preliminary draft.

References

1. van Otterlo, M.: A survey of reinforcement learning in relational domains. Technical Report TR-CTIT-05-31, University of Twente, CTIT Technical Report Series, ISSN 1381-3625 (July 2005)
2. Kersting, K., De Raedt, L.: Logical markov decision programs. In: Proceedings of the IJCAI 2003 Workshop on Learning Statistical Models of Relational Data (2003)
3. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning (2006)
4. Singla, P., Domingos, P.: Entity resolution with markov logic. In: IEEE International Conference on Data Mining (2006)
5. Mihalkova, L., Mooney, R.J.: Transfer learning with markov logic networks. In: Proceedings of the ICML Workshop on Strutural Knowledge Transfer for Machine Learning (2006)
6. Džeroski, S., de Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning 43, 7–52 (2001)
7. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)

8. Domingos, P., Richardson, M.: Markov logic: a unifying framework for statistical relational learning. In: ICML 2004 Workshop on Statistical Relational Learning (2004)
9. Domingos, P., Richardson, M.: Unifying logical and statistical ai. In: Proceedings of AAAI (2006)
10. Pearl, J.: Probabilistic reasoning in intelligent systems: Networks of plausible inference. Morgan Kaufmann, San Francisco (1988)
11. Irodova, M., Sloan, R.H.: Reinforcement learning and function approximation. In: The Florida AI Research Society Conference (2005)
12. Taylor, M.E., Kuhlmann, G., P.S.: Autonomous transfer for reinforcement learning. In: Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008) (2008)
13. Lowd, D., Domingos, P.: Recursive random fields. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007) (2007)