Department of Electrical and Computer Engineering

Robot Architecture for Interactively Learning Rules in Non-Stationary Domains

Prashanth Devarakonda
5-11-2015

I declare that this report is my own work, and any external material has been properly referenced.

This report was not written in collaboration with any other person.

Prashanth Devarakonda

Robot Architecture for Interactively Learning Rules in Non-Stationary Domains

Prashanth Devarakonda

Abstract— **With the demand for robots in complex environments increasing, it is beneficial to have an autonomous robots that are able to adapt to their environment. This report discusses reinforcement learning and the different approaches to a reinforcement learning problem, the proposed architecture for robots in complex environments and related works.**

I. INTRODUCTION

Robots deployed to assist and interact with humans and objects in complex domain often face fundamental learning and representation challenges. For instance, it is difficult to equip a robot with accurate domain knowledge when some rules governing the domain dynamics may be unknown or change at the time. For an artificial agent to be fully autonomous and robust, it needs to be able to learn and adapt to the environment. The objective of the project is to have a robotic arm be able to optimally stack blocks. The blocks are of different colour and shape and the robot needs to learn to distinguish the blocks according to these attributes. These attributes are imposed by rules governed by the environment as well as governed by humans. For example the robot must be able learn that a cube cannot be stacked on top of a pyramid, or that a blue cube shouldnt be stacked on top of a red cube. This report discusses the theory of relational reinforcement learning, the proposed architecture and related works that have extensively studied the combination of relational languages with reinforcement learning.

II. PROBLEM FORMULATION

A. Reinforcement Learning

Reinforcement learning is method of machine learning that utilizes the agent's experience of the environment and what actions to take to maximize some notation of reward. It can be formulated as a Markov Decision Process (MDP), whose entries correspond a set of states, a set of actions, unknown state transition function, unknown real-value reward function. Unlike a traditional MDP it relaxes some of these assumptions including determinism and complete knowledge of the domain [1].The agent can be in one state s_t of S and stochastically selects an action at which causes it to transition to s_{t+1}. The agent also receives a reward r_t which is dependent on the state and action. The transition function is also dependent on the state and action chosen. The objective of reinforcement learning is to find π* such that V^π is maximum, where
$V^\pi = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. [5]
Unlike supervised learning algorithms, reinforcement learning algorithms do not require large sets of data as the agent can learn from its experience from the environment. There are multiple methods formulated to solve the reinforcement learning problem.

1) Dynamic Programming: Dynamic Programming is collection of algorithms that find the optimal policy, given a perfect model of the environment as an MDP. Many dynamic programming algorithms use policy iteration. Policy iteration consists of two parts: policy evaluation and policy improvement. Policy evaluation considers how to compute V^π given an arbitrary policy π*. Consider the state-value function
$V^\pi(s) = \sum_a \pi(s,a) \sum_{s\prime} P^a_{ss\prime}[R_{ss\prime} + \gamma V^\pi(s\prime)]$[5], where $\pi(s,a)$ is the probability of action a being chosen in state s. The algorithm computes V^π iteratively where the old approximation of $V^\pi(s')$ is used to compute the new value of $V^\pi(s)$. As number of iterations increases $V^\pi(s)$ would converge, as long as $0 < \gamma < 1$. Once the $V^\pi(s)$ converges to a specific action, then it is evaluated using policy improvement method. Policy improvement method checks if the action chosen in state s is the best by comparing all values of $V^\pi(s)$ for the set of actions in a given state s. The major problem with dynamic programming is that they require operations on the entire state set. If the state set is large then it can be computationally expensive.

2) Monte Carlo Methods: Monte Carlo (MC) methods differ from dynamic programming methods where they utilize on line or simulated interaction with the environment. A model is still required by MC methods, however the model only generates sample transitions and not full probability distribution of all possible transitions[5]. MC methods work well for episodic tasks, where a policy is divided into episodes. The methods are incremental in an episode-by-episode sense. Once the end state of an episode is reached, the $V^{\pi}(s)$ is the average $V^{\pi}(s)$ of s that appear in all episodes. A disadvantage of MC methods is that $V^{\pi}(s)$ is updated at the end of an episode instead of at the end of a state transition.

3) Temporal Difference Learning: Temporal Difference (TD) learning is a combination of both dynamic programming methods and Monte Carlo methods [5]. Similar to Monte Carlo methods, TD methods can learn from raw experience. It does this by sampling the environment with some policy. It does not need a model of the environment's dynamics. TD is similar to dynamic programming in the sense that it uses previous experience to better approximate the current state estimate[5]. The simplest TD method can be formulated as
$V(s_t) = V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$[5].
Some popular TD algorithms are Q-learning and SARSA, where the state estimates are given as state-action pairs, Q(s,a). The algorithm of Q-learning is described in procedural form in Algorithm 1.

Algorithm 1 Q-learning

```
Initialize Q(s,a) arbitrarily
Repeat(for each episode):
Initialize s
Repeat(for each step of episode):
Choose a from state s using policy derived from Q
Take action a, observe r, $s\prime$
$Q(s,a) = Q(s,a) + \alpha[r_t + \gamma Q(s\prime, a\prime) - Q(s,a)]$
$s \leftarrow s\prime$
until s is terminal
```

TD methods can be superior to Monte Carlo and dynamic programming methods. They are less computationally intensive compared to dynamic programming methods. They can also be applied on-line while Monte Carlo methods only improve their policy at the end of an episode.

B. Relational Reinforcement Learning

As powerful as reinforcement learning maybe if there are any changes in the environment then the agent has to completely retrain the Q-function[3]. In relation to the block world, if the robot was previously asked to stack blue blocks and suddenly asked to stack red blocks, then it would need to relearn how to stack blocks as the environment has changed. Even though the same logic of stacking applies to both coloured blocks. The rule of stacking flat objects on flat objects should be generalized enough to apply to all flat objects. A knowledge base that holds rules of the domain can be useful when learning as previously learned rules can be applied rather than being relearned. Relational Reinforcement Learning (RRL) utilizes this concept by representing the states, actions and policies relationally. This is done by having a logical decision tree structure to show the hierarchical structure of the domain rules. For example if the agent's task was to stack all blocks that are stackable with regards to colour, then the first property that it must check is if the block is stackable or not. If not then it is unnecessary to check the colour of that block. The logical decision tree and the knowledge base can be written in a declarative language such as Answer Set Prolog (ASP). ASP can describe recursive declarations, defaults, causal relations and other constructs that are difficult to express in other logical programming languages.

III. Proposed Architecture

This section is going to talk about the approach taken for incrementally learning the rules governing the environment. The proposed architecture can be divided into four distinct parts as seen in Figure 1. The overall goal of the robot is to stack all objects with respect to colour. It should also do it in such a way as to increase stability of the structure. The robot must learn that flat objects cannot be stacked on top of pyramid shaped objects. However pyramid shaped objects can be stacked on top of flat objects. The domain rules are formulated as answer set prolog code which is part of the ASP KB (knowledge base). The ASP controller uses domain rules to give a generalized instruction to the State Space. The State Space is the module that will obtain the optimal policy with Q-learning. It contains the MDP that represents the environment. The state space then passes the optimal policy to the Robot Controller which uses the actions within the state to appropriately move the robot. The robot's movements are observed by the mounted camera which provides feedback to the state space which is used to update state reward function. An array of Q-values for states is then passed from the State Space to the Rule Generator. Which uses the Q-values to construct new rules that are then stored in the Knowledge Base.

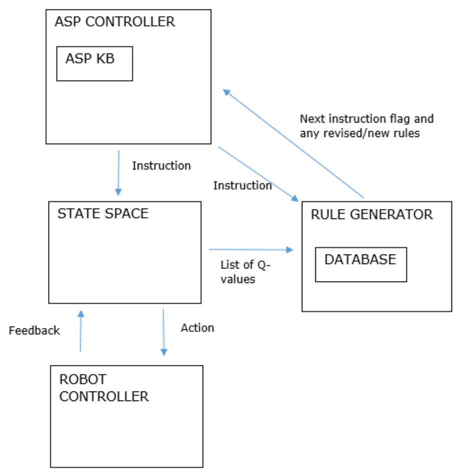

Fig. 1. Proposed Architecture

A. PhantomX Pincher

The PhantoX Pincher, shown in Figure 2, is a robotic arm with 5-degree freedom. The pincher robot works well in picking up small objects. This will be the robot in our dynamic environment. The movement of the robot will be controller by ArbotiX-M Robocontroller. The camera, Figure 3, is a RobotGeek Webcam with 250mm Gooseneck which will be used to provide feedback to the State Space.

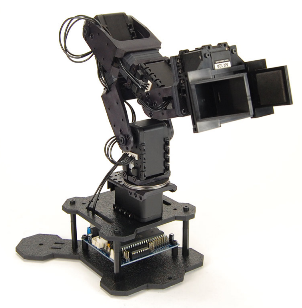

Fig. 2. Pincher Robot [2]

B. Hypothesis

The following hypotheses are based on numerous studies done before. Based on Dzeroski's results it is predicted that relational reinforcement learning increases the accuracy of the approximated optimal policy. These results can be observed by measuring the moving average of the optimal policy and plotting it with the number of iterations done to receive the optimal policy during Q-learning.

Fig. 3. Mounted Camera[2]

IV. Related Work

The main contribution of this work is to show how related reinforcement learning is beneficial to the adaptivity of automated robotics in non-stationary domains. The proposed architecture of related reinforcement learning was inspired from Relational Reinforcement Learning [3] . They proposed the idea of having relational learner in the form of logical decision trees combined with TD methods such as Q-learning and P-learning. P-learning is similar to Q-learning, except that when building the logical decision tree only the optimal policy is considered. In their study, two sets of experiments were done: one was where the number of blocks were kept constant and the other was that the number of blocks were varying during the learning process. In each experiment three distinct goals were attempted by the agent: stacking, unstacking and on(a,b). By using P-RRL, Dzeroski, Raedt and Driessens found that the optimal policies are learned rapidly but as the number of blocks increases the performance of the agent decreases. The academic journal notes that the state space has a large increase with an increase in number of blocks. This results in poor performance in learning as the complexity of the problem also increases.

Similar approach was taken in a masters thesis[4] by Sarah Rainge. The main focus of that paper was to combine the strengths of reinforcement learning with a declarative programming language so that the agent can incrementally learn previously unknown rules of the domain [4]. The architecture described in the thesis consists of a Q-learning algorithm that learns about the domain dynamics and a knowledge base. The Q-learning algorithm uses the previously learned domain rules to bias the action chosen in a state. The experiment conducted to test the architecture was having a simulated robotic agent stack blocks of different shapes. The average reward was collected and plotted against the number of episodes. It was found that the optimal policy was found relatively quickly however there was a difference in performance between the two scenes. One scene involved the robot to stack blocks of different shapes. This increases the complexity of the environment which caused a large standard deviation in values for the average reward obtained by the robot. Compared to the smaller standard deviation with blocks of the same shape. One criticism of this study is that there is no generalization of rules. The set of rules learned about yellow coloured blocks may not apply according to the agent. So it may need to relearn the rules by restarting the algorithm.

V. CONCLUSIONS

This report discusses the architecture that may be used so that robots maybe truly autonomous. It discusses the strengths of relational learning with reinforcement learning to discover unknown rules in non-stationary domains. The domain we discuss is the block world domain, where the objective is to stack blocks based on shape and colour. The approach involves using Q-learning to obtain an optimal policy to achieve a task specified by the Knowledge Base. Actions are chosen stochastically but influenced by previous experience and the information provided by the Knowledge Base. The relationships between states, actions and policies are encoded in a logical decision tree. This architecture will be implemented on robot in the physical realm instead of being simulated as it is important to see the power of such architecture in the physical environment. Similar approaches have been done in previous studies about relational reinforcement learning, however this architecture extends these studies by studying the effect it has in a physical environment.

ACKNOWLEDGEMENTS

The author would like to thank Mohan Sridharan, the project supervisor, and project partner Rashmica Gupta.

REFERENCES

[1] Kurt Driessens Prasad Tadepalli, Robert Givan. Relational reinforcement learning an overview. *Proceedings of the ICML workshop on Re lational Reinforcement Learning*, 2004.

[2] Trossum Robotics. Phantomx pincher robot arm kit. http://www.trossenrobotics.com/.

[3] Luc De Raedt Saso Dzeroski and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, pages 7–52, 2001.

[4] Mohan Sridharan and Sarah Rainge. Integrating reinforcement learning and declarative programming to learn causal laws in dynamic domains. Master's thesis, University of Auckland, 2014.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.