A Critique of the dissertation "Concurrent Hierarchical Reinforcement Learning"

Max Robinson MAX.ROBINSON@JHU.EDU
Johns Hopkins University,
Baltimore, MD 21218 USA

1. Introduction

(Marthi, 2006)

2. Related Work

3. Summary

Marthi touches on a myriad of topics as he discusses the research conducted in his thesis. Starting with background on "Flat" reinforcement learning, Marthi explains the foundations of the Semi-Markov Decision Process (SMDP), and partial programs upon which the research is built.

The first major contribution of the dissertation, the Concurrent ALisp language, is then explained. A description of the implementation follows. How learning algorithms are applied in Concurrent ALips then follows. These learning algorithms use the early foundations to describe how the learning using Concurrent ALisp is executed.

The experimental results then backup the principles of learning using Concurrent ALisp. The primary themes of partial programs, reward decomposition, and scalability are all tested.

3.1 Background

3.1.1 MDPs and SMDPs

The research done by Marthi, along with much for the reinforcement learning field, builds upon Markov decision processes or MDPs (?), (?). MDPs are often used to model sequential decision making processes. An MDP can be defined as a tuple $M = (S, A, P, R, s_0)$. Each value of the tuple is defined as follows.

- S - state space
- A - action space
- P - transition distribution
- R - reward function. A function that maps a state, action, and next state $R(s, a, s')$ to a member in $\mathbb{R} \cup -\infty$
- s_0 - initial start state

For an MDP to be an accurate representation of the problem, two general properties have to be met or assumed. First, the current state must be derivable just from the last perception of the environment but the agent. Second, the Markov property is assumed. The Markov property states that the probability of entering a given state next only relies on the current state and the action taken from that state. No prior history before that state is taken into account.

Markov decision processes can be solved or estimated with a multitude of different algorithms and approaches. The solution to an MDP is known as a *policy*, denoted by π. A policy describes what actions an agent should take when in a given state. Two types of policies to focus on are stationary and non-stationary policies.

A stationary policy is one in which it depends only on the last state, $\pi(s)$. A non-stationary policy is one in which the action decision relies on additional information than just the current state. Marthi focuses on non-stationary policies as he notes that most hierarchical reinforcement learning breaks agent behavior into tasks. As a result, the goal of an agent might not be recoverable from just the environment state.

From MDPs, a modified version called a semi-Markov decision process (SMDP) can be described. An SMDP is an MDP that also includes a duration distribution for each state action pair. The reasoning behind adding a duration is that actions can take some amount of time to complete. From a hierarchical standpoint with tasks, one might imagine that a task takes a certain amount of time. The SMDP is build to incorporate that duration into the model.

3.1.2 Partial Programs

A goal for Marthi's research is to allow programmers to easily incorporate background knowledge into learning algorithms. To do this Marthi introduces ALisp (Andre & Russell, 2002) and partial programming. Partial programs aim to help programmers incorporate background knowledge while writing a program to learn based on the partial program.

A partial program can thought of as constraints on which policies can be searched for while learning. A policy that finds an optimal policy given these constraints can be considered hierarchically optimal. ALisp is a language in which partial programs can be written. In ALisp, the foundations for Concurrent ALisp, the construct of a choice statements is added to allow for non-determinism in the program execution. This nondeterminism is used for selecting which set of statements to run next and what actions to take in the environment. The program then corresponds to a set of policies that can run the program and choose what to do at the choice statements. The choices made at each statement use a *completion* of the program.

Partial programs when combined with an underlying MDP create an SMDP, called the induced SMDP. The states in the SMDP are derived from the state of the MDP and the *machine state*. The machine state can be thought of as program specific information about the partial program during execution. Actions in the SMDP are the choices made at the choice statements. The actions taken in the program then might result in some action being taken in the underlying environment. The reward for the SMDP is the reward gained during all actions between two choice statements.

By creating an SMDP from the partial program, algorithms for SMDPs can be applied to the induced SMDP. Marthi concludes that finding the hierarchically optimal policy corresponding to the partial program is equivalent to finding the optimal stationary policy for the induced SMDP. This means that by looking at the SMDP, a hierarchically optimal execution of the partial policy can be found.

Partial programs also allow for decomposition of the Q-function for learning algorithms. The Q-function can be thought of as the "action-value function". The Q-function is the expected reward if we start at $\mathbf{s}$, a history of states, and do action a, then follow π. For partial programs, the Q-function can be decomposed into three components, Q_r, Q_c, Q_e. These are the expected reward received while doing the current choice, after the current choice until the subroutine ends, and after the current subroutine respectively.

The motivation behind decomposing the Q-function is to simplify the learning process. Each part of the decomposition may only rely on a particular subset of variables that are part of the state. In these cases, state abstraction can be leveraged, and thus reduce the sample complexity of learning, since each value is learned independently.

3.2 Concurrent ALisp

3.3 Implementation of ALisp System

3.4 Learning Algorithms

3.5 Experimental Results

4. Contributions

5. Relevant Algorithms

6. Applications of Concurrent Hierarchical RL

7. Conclusion

References

Andre, D., & Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, pp. 119–125, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Marthi, B. M. (2006). *Concurrent Hierarchical Reinforcement Learning*. Ph.D. thesis, Berkeley, CA, USA. AAI3253978.