Applied Soft Computing 25 (2014) 118–128

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Automatic abstraction controller in reinforcement learning agent via automata

Seyed Sajad Mousavi*, Behzad Ghazanfari, Nasser Mozayani, Mohammad Reza Jahed-Motlagh

School of Computer Engineering, Iran university of Science and Technology, Tehran, Iran

ARTICLE INFO

Article history:
Received 1 November 2013
Received in revised form 17 July 2014
Accepted 24 August 2014
Available online 27 September 2014

Keywords:
Reinforcement learning
Hierarchical reinforcement learning
Cluster
Multi-agent learning

ABSTRACT

Reinforcement learning (RL) for solving large and complex problems faces the curse of dimensions problem. To overcome this problem, frameworks based on the temporal abstraction have been presented; each having their advantages and disadvantages. This paper proposes a new method like the strategies introduced in the hierarchical abstract machines (HAMs) to create a high-level controller layer of reinforcement learning which uses options. The proposed framework considers a non-deterministic automata as a controller to make a more effective use of temporally extended actions and state space clustering. This method can be viewed as a bridge between option and HAM frameworks, which tries to suggest a new framework to decrease the disadvantage of both by creating connection structures between them and at the same time takes advantages of them. Experimental results on different test environments show significant efficiency of the proposed method.

1. Introduction

RL is a branch of machine learning in which the agent must learn through interaction with the environment. RL has two main unique features: (1) Learning is based on trial and error, and (2) Signals of rewards may be delayed. One of the major challenges the scaling up of reinforcement learning faces with is the well-known "curse of dimensionality" problem or combinatorial explosion (In dealing with large environments, the state space grows exponentially).

To cope with the curse of dimensionality, the state space reduction techniques can be used. In this ways, the original model is mapped to another model with fewer states. In other words, the new structural model is modified to obtain an efficient solution so that the transformed model is an approximate of the main model. This approach of reinforcement learning has a better ability to deal with the curse of dimensionality in solving problems and introduces methods such as Monte Carlo, temporal difference backups and functions approximation. Another approaches use techniques of aggregation/disaggregation [1]. The method is used in cases where the system model can be viewed as an interacting set of tightly coupled subsystems. The solution method is generally an iterative one in which sub-models are solved, then the obtained results are used to improve the sub-models repetitively until the optimal convergence is achieved.

Recent attempts to address the challenge of the curse of dimensionality in RL tend to methods based on abstraction which are indeed based on the aggregation and disaggregation techniques. Using these techniques leads to hierarchical control architectures and their associated learning algorithms known as hierarchical reinforcement learning (HRL). In most cases, hierarchical solutions provide near optimal solutions in their performance, less cost at runtime and also at the learning time and required solving space in comparison with mere RL techniques [2].

HRL is wide and active branch of RL that applies the hierarchical structure in the state, action and strategy space. Some frameworks based on HRL use temporal abstraction techniques. Making a decision at each step is not necessary, but instead the temporally extended actions (the macro actions) or sub-tasks can be selected to reach the goal [3]. Indeed, HRL is a general framework for scaling up RL to problems with large state spaces by using the task structure (action) to restrict the space of policies. The key principle considered by HRL is that the development of learning algorithms does not need to learn a policy from scratch, but instead it reuses available policies for simpler subtasks (macro) and uses the divide-and-conquer strategy. HRL models to apply the temporal abstraction must be able to make use of actions with variable lengths. A statistical model known as semi-Markov decision processes (SMDP) is used to perform behaviors with actions of variable lengths. These works on SMDP approach have led to

* Corresponding author. Tel.: +98 9304054874; fax: +98 21 73225322.
E-mail address: smousavi71@gmail.com (S.S. Mousavi).

http://dx.doi.org/10.1016/j.asoc.2014.08.071
1568-4946/

the development of powerful HRL models, including Hierarchical Abstraction Machines (HAMs) [4,5], options [3], and MAXQ [6].

The goal of the option approach is to learn global policies, being given completely partial polices to do subtasks; but HAMs emphasize on limiting policies that can be performed instead of developing the actions. In option framework, the set of agents' actions are increased by produced actions instead of being simplified or even reduced. In the framework of reinforcement learning using option, choosing temporally extended actions is too expensive due to their large number of time steps. Now the issue is to create a smart choice of temporally extended actions to increase the learning speed.

The main innovation introduced in this paper is the use nondeterministic automata like HAM as a high-level controller to select temporally extended actions in low level. The proposed approach tries to present a framework which uses the capabilities of option and HAMs, and also reduces the disadvantages of both methods by building communication structures between them.

Option and HAM frameworks have certain operational features. Obviously, a structural design based on the two approaches in such a way that they cover each other's weaknesses leads to a flexible structure with high performance. These approaches are strong methods but, as mentioned before, have some disadvantages the purpose of this paper is to address.

The rest of the paper is divided into the following sections: Section 2 gives a review of the background of the work. Section 3 discusses multi-agent learning. In Section 4, we describe the algorithm and its steps. Section 5 defines a framework of the proposed method and that how it is used in RL. Section 6 offers some computational experiments to evaluate the performance of the proposed algorithms on single-agent and multi-agent environments separately. Finally, we conclude our work in Section 7.

2. Background

As mentioned previously, the classical methods for solving Markov decision processes (MDPs) with a large state space sizes are faced with the problem of curse of dimensionality. A common way to overcome this challenge is to use technologies of decomposition and aggregation, and another is to use factored state space. Decomposition and aggregation techniques are generally special cases of the classic divide-and-conquer framework to split a large problem into smaller components and solve the parts in order to construct the global solution. Finally, all local solutions are compounded in order to reach an overall solution. Methods that are based on this technique are classified based on the deterministic and non-deterministic nature of the problems they must solve.

Following methods can be used for deterministic problems. Decomposition principles, for solving a very large linear program, divide it into many correlated linear programs of smaller sizes. Dantzig–Wolfe decomposition [7] is one of the most well-known methods of this kind. In [8] how to use the Dantzig–Wolfe decomposition [7] to solve MDP as linear programming is shown. Also, decomposition techniques of Ross and Varadarajan [9], and Abbad and Boustique [10] for solving certain MDPs of large size have introduced. These works propose some algorithms to compute optimal strategies for several categories of MDPs (average, discounted and weighted) and also can be applied to many practical planning problems. One of the most important steps of the decomposition algorithm of Ross and Varadarajan [9] is to solve the aggregated MDP which is itself an MDP. Ross and Varadarajan [9] used a classical algorithm for solving the aggregated MDP and did not give any new method for solving the aggregated MDP. The proposed algorithm by Abbad and Boustique [10] computes an average optimal strategy which is based on the graph adapted with the original MDP, introducing some hierarchical structure for this graph. Main part of this algorithm is to infer levels of Graph G and solve the limited MDP corresponding with each level. The local solutions of sub MDPs provide an optimal strategy for the final MDP.

But wherever the problem encounters uncertainty, methods such as Dean and Lin decomposition [11], that are one of the first methods presented in random contexts, are introduced. They have shown that divide-and-conquer algorithms can be used for solving some MDPs which are loosely coupled. Using divide-and-conquer strategy in RL has led to HRL. Temporal abstraction is used in HRL to divide the state space into separate regions. Then for each region, a policy is calculated that is called a macro action. Macro actions dramatically reduce the number of decisions in solving large MDPs [4,9]. So in the HRL, there is no need to learn policies at the beginning, but instead, the existing policies are used for subtasks [12].

One of the most important attributes of using temporally extended actions as the smallest development of HRL is to establish SMDPs. Actions in SMDPs take varying amounts of time to complete. They try to model temporally extended actions. At SMDP, in contrast with MDP where state changes only by doing an action, the state of the system may change continuously during performing a temporally extended action and for this reason, they are commonly used in hierarchical reinforcement learning concept. In the following sections, hierarchical reinforcement learning will be explained in more detail.

Another way to solve the curse of dimensionality utilizes the state space variables. The state space of the MDP can be considered as a cross-product of sets of state variables $E_i (E = E_0 \times E_1 \times \ldots \times E_n)$, which is called factored MDP (FMDP). In [13,14] the authors exploit such a factored state space to reduce the amount of computation and the memory required to compute the optimal solution; although it is a challenge to obtain their state variables.

2.1. Markov and semi-Markov decision processes

To model single-agent environments in RL problems is used of MDPs. In the MDP framework, at each time step t, $t = 1, 2, \ldots$ a learning agent interacts with an environment and observes its environmental state $s_t \in S$, where S is the set of possible states. The agent chooses an action from the set of available actions at state s_t, $A(s_t)$, based on a policy, a description of the behavior of an agent. As a result of each action a_t, the agent receives a scalar reward $r_t \in R$, where R is a the reward distribution, and observes state $s_{t+1} \in S$ one step time later. The agent's objective is to learn an optimal policy for selecting actions which maximize the expected discounted reward over time, $E\{\sum_{t=0}^{\infty} \gamma^t r_t\}$ where $0 \langle \gamma \langle 1$ is called discount factor. The popular RL algorithms based on MDP are Q-learning, $TD(\lambda)$ and *SARSA* [2]. It is difficult to apply purely RL algorithms to real world problems with continuous space. Therefore, the new model is proposed that gives the possibility to the agent to do its action in several time steps, which is called SMDP [3].

A SMDP is a five-tuple, $\langle S, A, P, R, J \rangle$ where S is a finite set of states, A is a finite set of actions, $P : S * A * S * N \rightarrow [0, 1]$ is a multi-step state transition function, and $P(s', N|s, a)$ denotes the probability of making a transition from state s to state s' by taking action in N time steps; and the transition is performed only in decision stages. SMDP model displays a snapshot of the system at key decision points. $R : S * A \rightarrow \mathbb{R}$ is a reward function for SMDP model and $R(s, a)$ is the total expected reward that will be received between the two decision stages. The update rule of SMDP Q-learning is as follows:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[r(s, a) + \gamma^N Q(s', a')] \quad (1)$$

2.2. Hierarchical reinforcement learning

HRL is used to decompose the original task into several subtasks so that learning complexity is reduced. Studies in HRL have led to