STAIRS 2012
K. Kersting and M. Toussaint (Eds.)

doi:10.3233/978-1-61499-096-3-162

Hierarchical Action Selection for Reinforcement Learning in Infinite Mario

Mandar JOSHI[a,1], Rakesh KHOBRAGADE[a], Saurabh SARDA[a], Umesh DESHPANDE[a] and Shiwali MOHAN[b]
[a] *Visvesvaraya National Institute of Technology, Nagpur*
[b] *University of Michigan, Ann Arbor*

Abstract. In this work, we analyze and improve upon reinforcement learning techniques used to build agents that can learn to play *Infinite Mario,* an action game. We use the object-oriented representation with the hierarchical RL model as a learning framework. We then extend the idea of hierarchical RL by designing a hierarchy in action selection using domain specific knowledge. Using experimental results, we show that this approach facilitates faster and efficient learning for the domain.

Keywords. hierarchical reinforcement learning, action games, Infinite Mario, object-oriented representation, action selection.

Introduction

Action games are an interesting platform for research in artificial intelligence and machine learning. They encourage quick decision making, as penalties are incurred if the agent fails to finish the game quickly. Players have to often choose between competing goals and make decisions that can give favorable long term rewards.

In this work, we concentrate on designing a reinforcement learning agent for a variant of a popular action game, Super Mario Bros, called *Infinite Mario.* This domain, like most action games, poses a challenge in the form of a very large search space (in terms of time and memory).

Hierarchies are often employed to scale up traditional reinforcement learning algorithms to complex environments. This is done by dividing a complex task into simpler subtasks using well-formulated domain knowledge. We use an object-oriented representation along with hierarchical RL as a learning framework for the *Infinite Mario* domain. We then extend the idea, by designing a hierarchy in action selection as well, using domain specific knowledge about the behaviour of game entities. Using experimental results, we claim that this approach provides for a higher domain-specific performance level as it brings about a reduction in both state and action spaces.

[1] Corresponding Author: Mandar Joshi, E-mail: mandarjoshi.90@gmail.com

1. Related Work

For an extensive overview of RL related work, we refer to [1]. Several hierarchical reinforcement learning (HRL) techniques have been developed e.g., MAXQ [2], Options [3], HAMs [4]. A summary of work in HRL is given by [5]. The motivation and scope for research in computer games is documented in [6]. Marthi et al. [7] applied hierarchical RL to scale to complex environments where the agent has to control several effectors simultaneously. They describe a concurrent partial programming language for hierarchical reinforcement learning and use the language to design an agent that learns navigation policies in a limited real-time strategy game. Ponsen et al. [8] employed a deictic state representation that reduces the complexity compared to a propositional representation and allows the adaptive agent to learn a generalized policy, i.e., it is capable of transferring knowledge to unseen game instances.

In the domain of action games, Diuk et al. [9] introduce Object-Oriented representation that is based on interaction of game entities called objects. They show that this representation results in performance gains in Pitfall, an action game. Mohan and Laird [10] use this representation to propose a reinforcement learning framework that gives promising results on a subset of game instances for the *Infinite Mario* domain. Their approach aims at creating learning agents that begin with little or no background knowledge about the game environment. In contrast, we use domain specific knowledge to extend the object oriented representation by introducing the concept of object classes, which play a major role in constraining the state space. We also use domain specific knowledge to design a hierarchy in action selection. We integrate this action selection hierarchy with the learning framework and show that this approach gives better results in terms of the average final reward, as well as the time required to learn the game playing task.

2. Problem Specification

Infinite Mario is a reinforcement learning domain developed for RL-competition, 2009 [11]. It is a complete side scrolling game with destructible blocks, coins, monsters, pits, and raised platforms. The objective of the game is to maximize the total reward by collecting coins and killing monsters on the way to the finish line. One run from the starting point to the finish line (or to the point where Mario gets killed) is called as an episode. If Mario reaches the finish line, the game restarts on the same level.

2.1. Environment

At any given time-step, the agent perceives the visual scene as a two-dimensional [16 x 22] matrix of tiles, where each tile can have 13 different values. The agent has access to a character array generated by the environment corresponding to the given scene. The domain also provides attributes like speed and position of the dynamic elements as double precision arrays.

Each action is composed of three action classes – motion (left, right, stationary), jump (yes, no), and speed (high, low). For each interaction with the entities in the environment, the agent receives a reward.

The RL competition software is capable of generating several instances of the game varying in difficulty level. As the difficulty level increases, interaction with game elements becomes more complex, i.e., the agent has to deal with more entities of different types. This makes decision making more difficult.

2.2. Challenges

- *Large state space*: One of the main challenges of the domain is a large and continuous state space. As reinforcement learning uses the trial and error paradigm, large state spaces are a major deterrent to the learning process. To make learning tractable, some form of discretization and abstraction is required. The learning framework, that uses the object-oriented representation combined with hierarchical RL, enables the agent to constrain the state space and facilitates effective learning.
- *Action selection*: At any time step, the visual scene is occupied by a large number of objects and corresponding goals. The action selection problem, given these multiple goals, is combinatorial in the number of objects and is largely intractable. Additionally, some of the actions may be irrelevant or useless in context of a particular goal. The hierarchy introduced in action selection helps in constraining the action space.

2.3. Benchmark: Memory Agent

RL Competition, 2009 provides a sample learning agent which is used as a benchmark for the proposed solution. The sample agent does not use any reinforcement learning but learns through memorization. It stores the sequence of actions it takes as it moves through the episode and uses this memory to take actions during the future trials. This simple strategy enables the agent to learn quickly; however, the final reward obtained by the agent is restricted by its limited exploration. Furthermore, the agent cannot transfer its learning in between different levels of the game. An RL framework, on the other hand, is capable of this transfer.

3. Agent Design

Our agent design is based on the object-oriented representation [9] integrated with the hierarchical model of reinforcement learning. We also incorporate a hierarchy in action selection into this framework to facilitate intelligent action selection.

3.1. State Representation

In [9], object oriented representation is used to model another video-game, *Pitfall.* According to this representation, the environment can be described as a collection of objects – entities that the agent can interact with and thus affecting the agent's behavior. For the *Infinite Mario* domain, the set of *objects* includes monsters, blocks, coins, platforms, pits and pipes. An object is characterized by its attributes, e.g. an instance of the object *monster* will have the (x and y components of the) distance of the monster from Mario, speed of the monster and its *type* as attributes. Mohan and Laird also used a similar representation in their solution for the *Infinite Mario* domain in [10].

However, including all objects in the visual scene as a part of the state will result in a large state table. As an extension to the object oriented representation, we divide objects into various classes, and each class defines the range of inclusion of the object instance in the state. This range is pre-programmed using domain-specific knowledge, e.g., a monster's range is larger than that of a coin or a question block, as monsters pose a threat to Mario. The central idea behind using object classes is to exclude those objects, which do not influence the agent's immediate actions, from the state. The *state table* in our approach is incremental, i.e. a state is added to the state table only when it is encountered for the first time in the game.

3.2. Hierarchical RL

The basic idea behind hierarchical reinforcement learning is to divide a complex task into a hierarchy of simpler subtasks. Each subtask can then be modeled as a single Markov Decision Process. The game playing task is accomplished using combinations of four subtasks, namely *moveRight*, *grabCoin*, *tackleMonster* and *searchQuestionBlock*. Each subtask is initiated with respect to a specific object in the state and spans over multiple one step actions.

Learning to carry out a subtask, therefore, involves identifying a set of primitive actions that allow the agent to interact "successfully" with the selected object. Termination criteria are defined separately for each subtask to identify successful interaction. For example, the terminal state for *tackleMonster* subtask is reached when the agent either kills or avoids the monster.

The state representation at the subtask level also uses object classes. The double precision attributes are rounded off to the nearest integer. Thus, the state representation at the lower (action) level of the hierarchy is more detailed as compared to that at the higher (subtask) level.

3.3. Subtask Selection

After the state is created, the agent tries to select one of the objects in the state and initiates an appropriate subtask. For example, if the agent selects a coin, it initiates the *grabCoin* subtask to grab it. The object selection is done by learnt object selection preferences, where a value is associated with each object in the state and this value indicates the utility of selecting that object. The agent uses SARSA to update this value after the termination of every subtask using the cumulative sum of rewards received from the environment.

3.4. Action Selection

Once a subtask is selected, the first level of the action selection hierarchy uses qualitative measures to reject irrelevant and useless actions corresponding to that subtask. These measures are based on the characteristic properties and behavior of the game entities. For example, in certain cases it might be possible to predict a collision between Mario and the game entities. Consider the case where a monster is very close to Mario. Depending on the position and speed of the monster, the agent may rule out certain actions (like remaining stationary) which might lead to a collision. Similar reductions in action space can be done for other subtasks. Once this reduction is done, the agent uses the learning framework to select an optimal action from the remaining set of actions. Thus, action selection at the second level of the action selection hierarchy is quantitative.

Additionally, whenever a new state is encountered, the action optimal for the *nearest similar* state is selected. We find the nearest similar state as follows. A search is made into the state table to find the set of all states which have the same entities as the present state, j. For every member, i of this set the state distance from the present state is calculated using

$$StateDistance_{ij} = \sum_{all\ entities} \sum_{all\ attribues} |att\_value_i - att\_value_j|$$

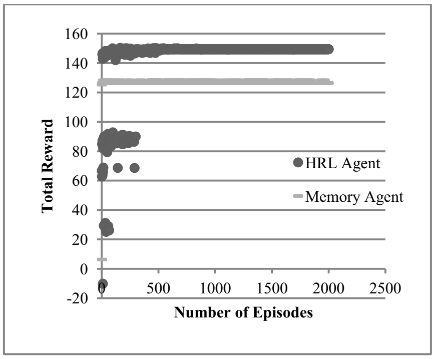
The state for which the state distance is minimum and below a threshold value is the nearest similar state. If no such state exists, a random action is selected.

3.5. Results

Figures 1 and 2 show the average performance of the proposed agent on difficulty level 0 and difficulty level 1 respectively, on seed 121. All results have been averaged over 10 trials of 2000 episodes each. Seed 121 of the game has been specifically chosen to enable a comparison with the work presented in [10]. On both levels the agent is able to converge to a policy, and gives a higher final reward as compared to the Memory Agent. This improvement in performance is due to the fact that the proposed HRL agent tries to explore the environment to maximize the total reward while the memory agent mainly concentrates on survival.

While comparing with other RL agent designs, two performance measures can be used – the average final reward, and the number of trials it takes the agent to converge to a policy. On difficulty level 0, the average final reward obtained by the proposed agent is close to 149.48 and the agent converges to a policy in about 300 episodes. Thus, the proposed agent performs better than the agents in [10] in terms of both the performance measures, on difficulty level 0. On difficulty level 1, the proposed agent converges to a policy in about 1000 episodes and earns an average final reward of 127.64 while agents proposed in [10] failed to learn a good policy.

The improved performance can be attributed to the following reasons. The divison



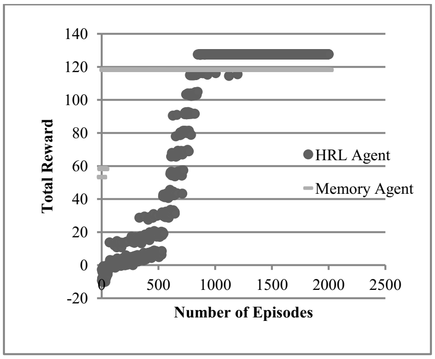


Figure 1. Performance of HRL Agent at seed 121, level 0

Figure 2. Performance of HRL Agent at seed 121, level 1 of objects into classes helps in constraining the state space. The hierarchical RL model introduces abstraction in state representation at higher levels while retaining the benefits of a detailed state representation at lower levels. By exploiting the interplay between high-level and low-level representations, the agent learns more efficiently and more accurately. The hierarchy introduced in action selection facilitates intelligent action selection. The reduction in action space allows the agent to learn quickly as has fewer relevant actions to choose from.

4. Conclusion and Future Work

The *Infinite Mario* domain is characterized by continuous, enormous state-action spaces and complex relationships between participating entities which encourage good decision making. To learn near optimal policies, the ability of agents to constrain state and action spaces is often crucial.

Through this work, we combine an object-oriented representation with hierarchical reinforcement learning to bring about a reduction in state and action spaces in the Mario domain. We also show that introducing a hierarchy in action selection in the above framework improves performance.

A major limitation of this work is that the proposed agent is unable to transfer previously acquired knowledge to identify optimal actions in related (but not necessarily *similar*) states. We plan to study function approximation schemes that can represent the states and actions as a combination of features. We are interested in approaches that can represent domain knowledge as a feature of a state and not merely as a tool for constraining action spaces.

References

[1] R. Sutton and A. Barto, Introduction to reinforcement learning, 1st Edition ed., MIT Press, 1998.

[2] T. G. Dietterich, "The MAXQ method for hierarchical reinforcement learning," in *the Fifteenth International Conference on Machine Learning*, 1998.

[3] R. S. Sutton, D. Precup and S. Singh, "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning," *Artificial Intelligence,* vol. 112, pp. 181-211, 1999.

[4] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems [NIPS-97]*, 1997.

[5] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," in *Discrete Events Dynamic Systems: Theory and Applications*, 2003.

[6] J. Laird and M. van Lent, "Human-Level AI's Killer Application: Interactive Computer Games," in *the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 2000.

[7] B. Marthi, S. Russell, D. Latham and C. Guestrin, "Concurrent hierarchical reinforcement learning.," in *The 20th National Conference on Artificial Intelligence*, 2005.

[8] M. Ponsen, P. Spronck and K. Tuyls, "Hierarchical reinforcement learning with diectic representation in a computer game," in *the 18th Belgium-Netherlands Conference on Artificial Intelligence*, 2006.

[9] C. Diuk, A. Cohen and M. Littman, "An object oriented representation for efficient reinforcement learning," in *the 25th International Conference on Machine Learning*, 2008.

[10] S. Mohan and J. Laird, "An object-oreiented approach to reinforcement learning in an action game," in *the 7th Artificial Intelligence for Interactive Digital Entertainment Conference*, 2011.

[11] B. Tanner and A. White, "RL-Glue: Language-indepenent software for reinforcement learning experiments," *The Journal of Machine Learning Research,* vol. 10, pp. 2133-2136, September 2009.