Online collaborative multi-agent reinforcement learning by transfer of abstract trajectories
van Someren, M.W.; Pool, M.; Korzec, S.

Published in:
BNAIC

Citation for published version (APA):
Online Collaborative Multi-Agent Reinforcement Learning by Transfer of Abstract Trajectories

Maarten van Someren Martin Pool Sanne Korzec
Informatics Institute
Kruislaan 419
1098 VA Amsterdam
University of Amsterdam
email: maarten,skorzec@science.uva.nl

Abstract
In this paper we propose a method for multi-agent reinforcement learning by automatic discovery of abstract trajectories. Local details are abstracted from successful trajectories and the resulting generalized, abstract trajectories are exchanged between agents. Each agent learns a policy for its own environment. By abstracting trajectories and sharing the result the agents benefit from each others learning. This reduces the overall learning time compared to individual learning.

1 Introduction

Intelligent agents typically learn individually. In contrast, humans acquire their skills with a combination of practice and knowledge that they receive from others. Some of our experience with the world is passed on from one person to another in a student-teacher setting. Other knowledge is codified in books or in electronic form. As humans we typically learn skills by combining such knowledge with hands-on experience. One reason for this is that it is not possible to codify and retrieve all knowledge. We codify knowledge that is hard to acquire by experience and yet useful for solving a problem or acquiring a skill. This observation motivated the study of learning settings in which knowledge is both acquired by individual agents and shared between agents. We take a typical skill learning setting as context, Reinforcement Learning.

Standard Reinforcement Learning (RL) methods learn a policy for an environment by observing the effect of the actions and the rewards obtained from the environment. This policy is specific for the environment: the states, the effects of operators and the position and size of rewards. Here we consider learning when some parts or aspects are the same in all environments and others are different. Different agents are learning in these environments. They can benefit from each others learning by exchanging parts of policies that are useful in all environments. In abstract terms this problem is in many respects the same as that of a single learner in a single environment where policies that were learned in one part can be transferred to other parts but in this paper we focus on the multi-agent setting. Our goal is thus a method to speed up learning by transferring parts of policies, in a single environment and also between multiple agents in different environments. This paper describes the setting that is used and preliminary experiments.

Note that we do not consider multi-agent learning in the sense of performing a task collaboratively. Each individual agent has its individual task and environment and do not perform a task collaboratively. They only collaborate by sharing what they have learned: they learn together but do not act together.

In the literature we find several different approaches to the problem of transfer. The first is to apply generalization to (partially learned) policies. Instead of acquiring a policy by basic Q-learning, generalization is used to complete a policy. By learning the relation between (features of) states, (features of) actions and (features of) rewards, predictions can be made about policy entries for which little or no experience is
available. A second approach is to give the learner a predefined, possibly hierarchical, decomposition of the
environment. The learner can then learn policies for parts of the environment and these can be used as com-
pound actions to learn a total policy. The environment is decomposed into parts by specifying an initial state
and a goal state for which the agent can learn a (sub)policy. The decomposition can be hierarchical. This
underlies hierarchical Reinforcement Learning methods such as MAXQ [1]. This decomposition again will
be useful if it captures components of policies that appear multiple times because this enables transfer. Once
a new subpolicy is learned it can be used for learning new tasks, simply by treating it as a (macro-)action.
The main difficulty here is to discover that certain parts of the environment are the same (ie. have the same
policy), before their policies have been learned completely (see for example [7] for a discussion).

We consider a variation of this learning task in which there are several agents that live in different parts of
a large environment. These parts are characterized by different geographical (or perhaps rather: topological)
structures but governed by the same physical laws. This means that the environment cannot be divided into
parts that match completely nor that a policy can be transferred ‘as is’. Instead, abstract descriptions of
states must be constructed that can be used in different environments for learning (sub)policies.

Our approach is based on the notion of Abstract Trajectories. An Abstract Trajectory is a description of
a set of trajectories in terms of changes in a subset of the properties that define a state. An abstract
trajectory corresponds to a part of an environment: the part that matches the property changes. Given an
abstract trajectory and an environment, a policy can be learned for the part of the environment that matches
the abstract trajectory. Thus, a learner can use an Abstract Trajectory by first learning a policy for states
that satisfy the abstract trajectory and then incorporating the result in RL as an option [8], a kind of macro-
operator. RL treats the option as an operator and includes it in its policy. This speeds up learning if the
abstract trajectory is actually a useful macro–operator in the final policy. In the following sections we
discuss how Abstract Trajectories are found (section 2) and how they are used in RL. Then we describe
experiments to study the effect (sections 4 and 5).

2 Finding abstract trajectories

The environment for learning consists of a set of states. Each state is described by a set of properties. The
agents know the possible actions for a state. Different agents live in different “sub-environments” that do
not overlap. Abstract Trajectories are found by exploring the environment, here in the context of Q–learning
[9]. An example of an abstract trajectory is:

\[
\begin{align*}
  \text{key}1 & : \text{no}, \text{key}2 : \text{no}, \text{key}3 : \text{no}, \text{door} : \text{closed} \rightarrow \\
  \text{key}1 & : \text{yes} \rightarrow \text{key}2 : \text{yes} \rightarrow \text{door} : \text{open}
\end{align*}
\]

This describes a set of trajectories that is characterized by a change in the property key from value no to
value yes and a change in property door from closed to open. In this example key opens the door.

An abstract trajectory is defined as a sequence of abstract states where an abstract state is a set of attribute
value pairs. An abstract trajectory is used as a kind of learning goal. An agent learns a policy that enables it
to follow the abstract trajectory in its own environment.

What is a useful trajectory and how is it abstracted? Our method is this. During learning, we store all
trajectories. maintain the set of abstract trajectories. An abstract trajectory \( \delta \) subsumes abstract trajectory
\( \epsilon \) if it can be constructed by removing states from \( \delta \). The algorithm to discover the abstract trajectories is
given in Figure 1.

Infrequent Abstract Trajectories are discarded and very frequently occurring (using a threshold) Abstract
Trajectories are sent to other agents. The rationale for this is that property changes that occur frequently in
successful trajectories are likely to be useful for other agents in other environments. It would be possible
to exchange Abstract Trajectories that consist of multiple property changes but it is less clear that these are
useful for other environments. It is useful to first pick up a fire extinguisher and then extinguish the fire but
in a different environment it the order in which an agent should pick up a key and a fire extinguisher may be
different in different environments.

An agent that receives an Abstract Trajectory starts to learn a policy for the part of its environment that
is covered by the Abstract Trajectory. The resulting option has states that satisfy the initial value of the
property, before the change. This becomes the condition for applying the option. Once learning the policy
for the option converges and the policy is stable, it is incorporated into RL. In this way the agent learns when
to use the option.
Abstract trajectories

For each trajectory that ends in a goal state, perform the following actions:

1. Remove properties that change frequently (using a threshold)
2. Remove all subsequences without property change, leaving only the states between which a property changes.
3. Construct an abstract trajectory for each subsequence between the initial state, the final state and each change of a single property.

Our Abstract Trajectories are a form of state abstraction, [3, 5, 6]. In our case, the goal is actually twofold: to find the best abstraction for the agent and also to find the abstraction that is useful for other agents that live in different environments. The current version simply generates an abstract trajectory per property. There are many other possibilities here that need to be explored in future work.

3 Collaboration and Communication

The organization of the agents is quite simple. An agent that has found a useful Abstract Trajectory broadcasts it to all other agents. The agents remember this and will not duplicate it. Communication is synchronous. The system runs in cycles and in each cycle an agent moves to a new state, learns from this, checks if it found a good Abstract Trajectory and if so, broadcasts it.

There are many alternatives. For example, agents could exchange candidate Abstract Trajectories and see if together over all environments they are frequent enough. This gives more communication but this can be compensated by reduced learning time. In this study we choose to explore the simplest organization form.

4 Experiments

The purpose of the experiments is to evaluate the effect of learning and exchanging Abstract Trajectories. First, we perform a preliminary experiment to demonstrate the effect. Good specific trajectories are given to the agent and performance is compared with that of agents without them. Next we consider the effect of learning Abstract Trajectories. Finally we consider the effect of Abstract Trajectories in a multi-agent setting in which agents live in different environments. These environments vary in geography but they are subject to the same physics. Therefore the optimal policies are different and exchanging specific trajectories or policies is not useful but some abstract trajectories are useful in all environments and therefore can contribute to the learning performance of other agents.

The agents perform regular undiscounted Q-learning, starting with $\epsilon = 1.0$ and multiplying by 0.99 (decreasing with 1%) at each episode.

4.1 Mazes

We have conducted our experiments with maze environments. An agent is in search of the exit, which can be blocked by obstacles. The agent needs to use items in the environment to get past the obstacles. For example, to pass a locked door, the agent will first need to pick up an item (e.g. a key) and use that item on the obstacle.

All are 8x11 mazes with a horizontal wall running through the middle. See for an example Figure 2. In the wall is an opening with an obstacle. The nature of the obstacles varies: two obstacles (river and
roadblock) just keep the agent from passing, one (monster) causes the agent to die if he doesn’t have the proper object, and one (fire) does both: the agent may pass if he has the correct item (a fire extinguisher), but he will die if he also carries the associated ‘evil item’ (a jerrycan of petrol). The mazes also contain two irrelevant items. The mazes differ in the positioning of the starting point and exit. The exit is always in one of the four corners and the starting point is exactly on the opposite side of the maze. The size of the state space of the small mazes is 1408 states.

The large maze (Figure 2) is 10x10. The path to the exit is blocked by a number of walls containing all but one of the obstacles. Also in the maze are all four of the solving items, making one of the items an irrelevant item. In addition, the large maze may or may not contain two more irrelevant items. The large maze has 12,800 states, or 51,200 states when irrelevant items are included.

4.2 Performance measure

We will use speed of convergence to optimality as our measure. More precisely, we say that convergence is reached at the first episode of the first 20 episodes of which the average reward is no more than 5 points away from the optimal reward.

5 Results

5.1 Experiment 1: The effect of Abstract Trajectories

The purpose of this experiment is to test if the Abstract Trajectories work at all. We use a simple 8x11 maze with one obstacle, one key item, and two irrelevant items. First an agent learns abstract trajectory and then acquires the policies for these trajectories. We manually add either the specific trajectories (with policies) or the Abstract Trajectories to a new agent and have it learn the environment. We expect these to learn faster than without abstract trajectories and the agents with specific trajectories should be faster than those with only abstract trajectories.

The comparison in speed of convergence is between:

Options Given: Options are given to the learner explicitly. This is a kind of reference.

Abstract Trajectories Given: In this version, the agent is given the correct Abstract Trajectories and learns its own policy for it.
**Self-used Abstract Trajectory learning:** The agent finds Abstract Trajectories and uses these in its own learning.

**Q:** Plain Q–Learning by a single agent without anything given, nor does the agent learn or use Abstract Trajectories.

We ran each setting 10 times with a maximum of 600 episodes. The results can be found in Table 1.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>142</td>
</tr>
<tr>
<td>Abstract Trajectories</td>
<td>177</td>
</tr>
<tr>
<td>Self-used Abstract Trajectory learning</td>
<td>208</td>
</tr>
<tr>
<td>Q</td>
<td>244</td>
</tr>
</tbody>
</table>

Table 1: Speed of Convergence

As can be seen from the table, the agent that is simply given the complete options performs best, converging to the best solution in 142 episodes on average. The agent that is given the Abstract Trajectories is next best. It has to learn the policies for the Abstract Trajectories which costs extra cycles. The agent that discovers and then uses its own Abstract Trajectories Needs a substantial number of extra cycles. Discovering the Abstract Trajectories is thus fairly expensive. A simple Q–Learner needs 244 cycles on average. This means that we may expect that the exchanging Abstract Trajectories can have a positive effect. In a setting where new agents enter the overall environment, an agent would do some 30% better if the agents in the environment would give it the right Abstract Trajectories.

### 5.2 Experiment 2: Learning Abstract Trajectories

In this experiment, we study the multi–agent setting where learning Abstract Trajectories takes place in parallel in all agents. We compare an agent that just does Q–Learning (Q) with an agent that is given useful Abstract Trajectories and only learns policies for them (Abstract Trajectories), with a agent that generates its own Abstract Trajectories and learns from this (Self-used Abstract Trajectory), and finally with an agent that learns along with four other agents. One agent, the target agent, is learning in a large maze (10 by 10) and four other source agents are in the four smaller mazes (8 by 11). In the smaller mazes Abstract Trajectories will be found earlier than in the larger maze. Distributing these to the other agents, including the one in the larger maze should speed up learning. All agents start learning at the same time. The four agents find the abstract trajectories by exploring their own environment and they transfer them to the agent in the large maze, which starts learning policies for them, in addition to his normal Q–learning, including learning policies for its own trajectories, *Multi–agent Transfer Learning*.

In addition, we vary the number of irrelevant items in the maze, and thus the number of irrelevant properties of states. We uses mazes with no, 1 and three irrelevant items. Figure 2) shows the maze without extra irrelevant items.

<table>
<thead>
<tr>
<th>Agent</th>
<th>Average 1 irrelevant item</th>
<th>Average 3 irrel. items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi–agent Transfer Learning</td>
<td>170</td>
<td>251</td>
</tr>
<tr>
<td>Self-used Abstract Trajectory</td>
<td>187</td>
<td>320</td>
</tr>
<tr>
<td>Q</td>
<td>247</td>
<td>367</td>
</tr>
</tbody>
</table>

Table 2: Convergence for the multi–agent transfer experiment.

Results for this experiment are in Table 2. The results confirm our hypotheses. A simple Q–Learner needs most cycles (some more than in the previous experiment because the maze is more complex). Finding and using its own Abstract Trajectories improves learning. Exchanging Abstract Trajectories between agents gives an extra speedup, despite the fact that the environments of all agents are different.
This is especially when the state space gets larger and the main agent needs more episodes to learn his policy and his subgoals, the difference between transfer of Abstract Trajectories and generation by the agent himself becomes larger.

![Figure 3: Results of the experiments. Left: Subgoal transfer with 1 irrelevant item. Right: Subgoal transfer with 3 irrelevant items.](image)

The number of irrelevant items is likely to affect performance. The number of items determines the number of properties of the states and thereby the size of the state space. Because irrelevant items are less likely to appear in Abstract Trajectories that are transferred, the effect of discovering and transferring Abstract Trajectories is likely to be stronger in environments with more irrelevant items. This indeed what we see in Table 2. In particular the difference between the multi-agent setting and the single learner increases.

6 Related Work

Recent years have seen a strong increase in interest for methods that combine Reinforcement Learning and MDP estimation with other forms of Machine Learning. We position our work relative to earlier work that we find most similar.

Our method for constructing Abstract Trajectories is closely related to the HEXQ algorithm by Hengst [2] for discovering hierarchical representations for complex policies that use state abstraction and sequence aliasing. We use a hierarchy of only two levels and multi-agent instead of single agent learning. We do not integrate the values of states and let each learner find these by itself.

Barto and others [11, 6] describe a method for identifying and exploring commonalities between learned policies that is based on identifying types or classes of different objects that all appear in the same policies. A mapping is constructed that maps states and actions from one policy to another. It may involve mappings between objects that are part of states. Our abstract trajectories can be viewed as such a mapping. Finding such a mapping is a form of generalization because the mappings are used to construct a general class of objects or states. Our method is a weak form of this type of learning that searches for a rather simple mapping and leaves policy reconstruction to other agents.

Wilson et al. [10] study a transfer setting using a hierarchical Bayesian approach. MDPs are learned in the form of mixture models with an unspecified number of components. After learning models for some number of tasks or environments, the distribution of parameters of the models over these environments is estimated and used as prior for estimating the distributions of new environments. This approach exploits commonalities in probability distributions over classes as function of labelled states. In our setting this model is less appropriate because we assume a kind of dichotomy in the confidence of these distributions. Some parts of these distributions hold in all tasks and other hold in no other tasks.

Stone and his colleagues [7] study similar issues in the context of transfer learning. They focus on finding a mapping between tasks that enables a policy to be transferred between these tasks. For example, Jong and
Stone [3] search for generalisations over states that enable a simpler description of a policy preserving the agents behavior.

The main distinctive features of our method are: transfer between agents of goals, finding common states to guide abstraction, simple property-based abstraction and not transferring policies. Many of these methods are quite similar and our method can easily be extended to the existing schemes and vice versa. To decide which approach is best depends on the setting and in particular what different learning tasks have in common and what not.

7 Conclusion, Discussion and Further Work

The main contribution of this work is the demonstration of a basic and direct method for learning and transferring abstract trajectories between learning agents. Our method extends existing methods for Q-learning with options by discovering abstract trajectories that have no associated policy. The policy for an abstract trajectory must be learned. Abstract trajectories are generated automatically and used for further learning. Abstract trajectories do need additional learning of a policy but when they capture important aspects of the environment (in the form of Abstract Trajectories) then they focus learning on parts of the environment for which it is useful to create an option with a policy. This improves learning.

The current system is a prototype intended as a basic architecture and a proof of concept. It raises a number of new questions. Most of these have to do with the online character of the learning process. Our method uses a threshold for convergence to trigger exchanging abstract trajectories. There is no guarantee that the abstract trajectory at that point is optimal even for the sending agent, because the trajectory from which it was derived may not be optimal [4]. When a better path and abstract trajectory is found, the revision should be distributed to the other agents but they need to be able to revise the half-learned policy for the original abstract trajectory.

Constructing abstract trajectories can be viewed as a form of causal modelling and corresponding methods can be applied here. The relation between learning policies and learning causal models has been raised before in the context of a single learner. Our experiment suggest that abstracting from specific “subenvironments” to the world in general makes causal modelling attractive relative to learning policies. This suggests that the currently simple generalisation method can be replaced by a more sophisticated causal modelling algorithm. This setting will then be used to study coordinated learning by multiple agents. This involves questions of how the results of learning by different agents can be integrated. Another set of questions concerns the multi-agent coordination and communication: how can the agents optimally exploit each others partially learned abstract options? Who should send what to whom at what moment?

The current system uses a toy domain. Practical applications can be envisaged in the context where multiple Reinforcement Learners are used in a complex environment.

The experiments show that constructing shareable knowledge is feasible and effective in a standard Reinforcement Learning setting.

Acknowledgments

The authors want to thank Attila Houtkoper and Gideon Maillette de Buy Wenniger who developed the first version of the Multiquest system in 2004 and Michael Metternich who revised the original version in 2005. We also thank the reviewers for many useful comments that helped to put this work in perspective.

References


