Multi-Agent Relational Reinforcement Learning
Explorations in Multi-State Coordination Tasks

Tom Croonenborghs[1], Karl Tuyls[2], Jan Ramon[1], and Maurice Bruynooghe[1]

[1] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
[2] Institute for Knowledge and Agent Technology, Universiteit Maastricht, The Netherlands

Abstract. In this paper we report on using a relational state space in multi-agent reinforcement learning. There is growing evidence in the Reinforcement Learning research community that a relational representation of the state space has many benefits over a propositional one. Complex tasks as planning or information retrieval on the web can be represented more naturally in relational form. Yet, this relational structure has not been exploited for multi-agent reinforcement learning tasks and has only been studied in a single agent context so far. In this paper we explore the powerful possibilities of using Relational Reinforcement Learning (RRL) in complex multi-agent coordination tasks. More precisely, we consider an abstract multi-state coordination problem, which can be considered as a variation and extension of repeated stateless Dispersion Games. Our approach shows that RRL allows to represent a complex state space in a multi-agent environment more compactly and allows for fast convergence of learning agents. Moreover, with this technique, agents are able to make complex interactive models (in the sense of learning from an expert), to predict what other agents will do and generalize over this model. This enables to solve complex multi-agent planning tasks, in which agents need to be adaptive and learn, with more powerful tools.

1 Introduction

In recent years, Relational Reinforcement Learning (RRL) has emerged in the machine learning community as a new interesting subfield of Reinforcement Learning (RL) [7, 4, 20]. It offers to RL a state space representation that is much richer than that used in classical (or propositional) methods. More precisely, states are represented in a relational form, that more directly represents the underlying world and allows the representation of complex real world tasks as planning or information retrieval on the web in a more natural manner (see section 2 for an example).

Compared to single agent RL, learning in a MAS is a complex and cumbersome task. Typical for a MAS is that the environment is not stationary and the Markov property is not valid. These characteristics make the transition from a one-agent system to a multi-agent system very hard. Furthermore, an agent in

a MAS needs to take in account that other agents are also trying to attain the highest utility for their task. A possible solution would be to provide all possible situations an agent can encounter in a MAS and define the best possible behavior in each of these situations beforehand. However, such a solution suffers from combinatorial explosion and is not the most intelligent solution in terms of efficiency and performance.

Yet different approaches have been introduced to solve this multi-agent learning problem ranging from joint action learners [10] to individual local Q-learners. All of these approaches have as well their own merits as disadvantages in learning in a multi-agent context. In the first approach, i.e., the joint action space approach, the state and action space are respectively defined as the Cartesian product of the agent's individual state and action spaces. More precisely, if S is the set of states and $A_1, ..., A_n$ the action sets of the n different agents, the learning will be performed in the product space $S \times A_1 \times ... \times A_n$, where each agent has a reward function of the form: $S \times A_1 \times ... \times A_n \rightarrow \mathbb{R}$. This implies that the state information is shared amongst the agents and actions are taken and evaluated synchronously. It is obvious that this approach leads to very big state-action spaces, and assumes instant communication between the agents. Clearly this approach is in contrast with the basic principles of many contemporary multi-agent applications such as distributed control, asynchronous actions, incomplete information, cost of communication. In the local or selfish Q-learners setting, the presence of the other agents is totally neglected, and agents are considered to be selfish reinforcement learners. The effects caused by the other agents also acting in that same environment are considered as noise. In between these approaches we can find examples which try to overcome the drawbacks of the joint action approach, examples are [13, 2, 15, 23, 17, 18]. There has also been quite some effort to extend these RL techniques to Partially Observable Markovian decision problems and non-Markovian settings [11].

However, to our knowledge, almost all of these different techniques have been used so far in combination with a state space which is in propositional form, where by no means relations between different features are expressed or exploited. To our belief, multi-agent RL in general could greatly benefit from the ideas of RRL, which has proved to be very successful in the single agent case. In this paper we illustrate the advantages of using RRL in MAS, by doing experiments in an abstract multi-state coordination task. This problem can be considered as a variation and extension of the abstract Dispersion Game (DG), introduced in [8]. More precisely, DGs are a stateless abstract representation of typical load-balancing problems. The goal of the game is to achieve coordination between the different participating agents and fast convergence to a maximally dispersed situation by avoiding to work on cross-purposes. This coordination problem has been acknowledged to be difficult and complex, but can be solved quite good by different RL approaches as for instance local Q-learning. Unfortunately, this problem remains stateless, which is of course a great simplification of real world complex coordination tasks. In this paper we study RRL in an abstract multi-state coordination problem in which tasks consists of multiple

subtasks as opposed to in DGs. We believe that RRL can overcome many of the problems encountered by classical RL techniques when trying to solve this type of multi-agent learning problems. These issues include large state spaces, generalization of knowledge, building models of agents and computational cost of the problem. In a first step we study how agents can learn the relational structure of such a problem. We do not consider at this moment the allocation of tasks to agents. Interference of agents in different tasks will be subject of our future work. Questions we answer now include: Can agents learn the relational structure of the problem and other agents? Can they build models of each other improving learnability? Does this improve convergence properties and can agents generalize over the knowledge they learned about other agents?

The rest of this document is structured as follows. Section 2 introduces Relational Reinforcement Learning, Section 3 gives an overview of relevant existing work and introduces the multi-agent RRL task. The experiments are presented in Section 4 and finally Section 5 concludes.

2 Single Agent Relational Reinforcement Learning

2.1 Reinforcement learning

Reinforcement Learning offers a general framework, including several methods, for constructing intelligent agents that optimize their behavior in stochastic environments with minimal supervision. The problem task of RL [19] using the discounted sum of rewards is most often formulated as follows: Given a set of possible states S, a set of possible actions A, unknown transition probabilities t: $S \times A \times S \rightarrow [0,1]$ and an unknown real-valued reward function $r : S \times A \rightarrow \mathbb{R}$, find a policy which maximizes the expected discounted sum of rewards $V(s_t) = \mathbb{E}\left(\sum_{i=0}^{\infty} \gamma^i r_t\right)$ for all s_t, where $0 \leq \gamma < 1$.

At every time step t, the learning agent is in one of the possible states s_t of S and selects an action $a_t = \pi(s_t) \in A$ according to his policy π. After executing action a_t in s_t, the agent will be in a new state s_{t+1} (this new state is chosen according to the transition probabilities) and receives a reward $r_t = r(s_t, a_t)$.

A drawback of most work on RL, using a propositional representation[3], is the difficulty to represent states that are defined by the objects that are present in this state and the relations between these objects. The real world contains objects. Objects with certain properties, that relate to each other. To apply RL in such complex environments, a structural or relational representation is needed.

To illustrate the need for these structural representations, we will describe the blocks world domain as a Reinforcement Learning problem. The blocks world consists of a number of blocks, which can be on the floor or onto each other. It is assumed that an infinite number of blocks can be put on the floor and that all blocks are neatly stacked onto each other, e.g. a block can only be

[3] In propositional representations a feature vector is used with an attribute for every possible property of the agent's environment

on one other block at the same time. The possible actions consist of moving one clear block (e.g. a block with no other block on top of it) onto another clear block or onto the floor. It is impossible to represent such blocks world states with a propositional representation without an explosion of the number of states. Using First-Order Logic, a blocks world state can be represented as a conjunction of predicates, describing the relations between the blocks, e.g. $\{on(s, b, floor) \wedge on(s, a, b) \wedge clear(s, a) \ldots\}$.

2.2 Relational Reinforcement Learning

Relational Reinforcement Learning combines the RL setting with relational learning or Inductive Logic Programming (ILP). Because of this structural representation, it is possible to abstract from and generalize over specific goals, states and actions and exploit the results of previous learning phases when addressing new and possibly more complex situations.

Furthermore, because relational learning algorithms are used, there is the possibility to use background knowledge. Background knowledge consists of facts or general rules relevant to the examples or problem domain in the context of RL. E.g., in the blocks world, a predicate $above(S, A, B)$ could be defined as the transitive closure of the predicate $on(S, A, B)$. These predicates in the background knowledge can be used in the learning process, i.e., in the representation of a Q-function.

Although, RRL is a relatively new domain, several approaches have been proposed during the last few years, we refer to [20] for an overview.

One of the first methods within RRL, is relational Q-learning [7]. In this work, a Q-learning algorithm is proposed that allows a relational representation for states and actions. The Q-function is represented and learned using an incremental relational regression algorithm. So far, a number of different relational regression learners are developed[4].

Besides relational Q-learning, there has been some work on other methods which is not discussed here. So far, all work on RRL has focused on the single agent case. To our knowledge, there is no existing work on applying relational reinforcement learning in a multi-agent system. Earlier, van Otterlo et al. [24] already mentioned the possible benefits of using RRL in (multi-) agent systems. They state that cognitive and sapient agents especially need a learning component where the reinforcement learning paradigm is the most logical choice for this. Since these agents are (usually) logic-based, RRL is indicated as a very suitable learning method for intelligent agents.

Furthermore, there has been some work in the past on guiding learning agents [5] and combinations with behavioral cloning [14].

[4] A thorough discussion and comparison can be found in [4]

3 Relational Multi-Agent Reinforcement Learning

During the 90's multi-agent systems have become a very popular approach in solving computational problems of distributed nature as for instance load balancing or distributed planning systems. They are a conceptually proved solution method for problems of this nature.

However, designing a cooperative multi-agent system with both a global high utility for the system and high individual utilities for the different agents is still a difficult problem [21, 9].The joint actions of all agents derive some reward from the outside world. To enable local learning, this reward has to be divided among the individual agents where each agent aims to increase its received reward. However, unless special care is taken as to how reward is assigned, there is a risk that agents in the collective work at cross-purposes. For example, agents can reach sub-optimal solutions in the blocks world example by competing for the same block or goal state, i.e., by inefficient task distribution among the agents as they each might only consider their own goals which can result in a Tragedy of the Commons situation, or policy oscillations [15].

In this setting different researches have already obtained some very nice results within the framework of stochastic Dispersion Games [8, 21, 9]. Dispersion games are an abstract representation of typical load balancing and niche selection problems. The games are played repeatedly, during which the agents learn to disperse. Still, these dispersion games are far more simple than for instance blocks world planning problems, as there is only one state to consider. We extend this type of work to large planning problems with multiple states, which we will try to solve by an agent-based system, consisting of learning agents in a relational state space.

Combining multi agent systems and relational reinforcement learning combines two complex domains. We believe that, in order to study the integration of these both settings, one should take care not to make the learning task too complex at once, as a mix up of the many different effects playing a role in both domains could make (especially experimental) results difficult to interpret. Therefore, we will first try to separate a number of effects we want to investigate as much as possible independently. In a second part of this section, we will then propose a number of settings of increasing difficulty in which we plan to conduct experiments.

3.1 Complexity factors

One could describe the main complexity factors of multi agent systems as uninformedness, communication and interference. First, agents are often assumed to be unaware of parts of the world far away where the other agents are operating. This essentially makes the world only partially observable, and hence agents are less informed than in the Markovian situation. Second, agents are unaware of each other's knowledge and intentions. Though these cannot be observed, these can be (partially) revealed by communication. In fact, a lot of work has been published on the study of agent communication. Third, plans and actions of

agents can interfere. To act optimally, an agent should take plans and actions of other agents into account (e.g. by knowing them, or by predicting them, or by making his own plan robust).

Relational learning adds extra complexity with increased state and hypothesis spaces, generalization and informedness. This does not necessarily mean that relational learning has a negative impact on the time complexity. First, RRL has been proposed in answer to the need to describe larger state spaces. Through their generalization ability, relational languages allow the compact representation of state spaces, policies, reward, transition and value functions. However, they do not take away the fundamental problem of difficult tasks that the optimal policy is complex and that the learning hardness increases with the state space size. This is illustrated by the fact that in the single agent relational case only some initial convergence results exist [12, 16], but no global guarantees that the policy will converge to an optimal policy in the case generalization over the state space is performed. Second, while the generalization ability is usually beneficial, it also often has the consequence that due to the generalization the world is only partially observable, i.e., it may be difficult to see the difference between states over which the agent generalizes. Third, while a reason for introducing relational languages was the ability to introduce background knowledge and hence make the agent better informed and put him in a better situation to act intelligently, the background knowledge will only be useful when the algorithm has the ability to exploit it. This often adds an extra level of complexity to the algorithm.

3.2 Settings

We will introduce first some terminology. A **setting** is a set of properties of the problem and the agent abilities under consideration. We say a setting has the `comm_reward` property iff the agents are trying to maximize the same common reward function (if one agent gets a reward, all other agents get the same reward). We say a setting has the `know_aim` property iff the agents know what function the other agents are trying to maximize. A setting has property `full_obs` iff the agents can observe the full world, including actions performed by other agents (but excluding internals of the other agents). A setting has property `know_abil` iff the agents know the ability of the other agents, i.e., how good they are at their task (e.g., whether an other agent will perform random actions, or whether an other agent always performs the optimal action). A setting has the property `comm_schedule` iff the agents have a way of communication for deciding who is performing actions at which time point. This e.g., would allow to let more experienced or specialized (but maybe also more costly) agents perform actions in certain types of situations. In a setting with the property `talk` the agents have the ability to communicate about their knowledge.

These properties are defined as general ones, but the same settings can of course be studied when some of these properties are only partially valid, e.g. there is only partial observability, based on some constraints like the kind of agents, their location or agents only know a part of the function other agents are trying to maximize etc. This will make the learning problem obviously more

difficult, but the ideas about these properties and the settings in which they are used remain the same.

If it is the case that the `full_obs` property holds, agents can use these observations to learn extra information. Therefore we define two properties based on the capabilities of the agents. We say the `learn_beh` property holds iff the agents are capable of learning the behavior of the agents they can observe. When agents use their observations about other agents to learn their abilities, the `learn_abil` holds.

We will now describe a number of settings. Table 1 lists these settings together with their properties. In the column "other's ability" we list the ability of the other agents in the world: 'teacher' means an agent having a good (but perhaps not optimal) policy; 'perfect' means an agent with an optimal policy. In the text we will refer to specific entries in the table with numbers between round brackets. The `comm` column has a 'c' if the `comm_schedule` property holds and a 't' if `talk` holds for that particular setting.

Setting	other's ability	comm rew	know abil	know aim	observe full obs	observe learn beh	observe learn abil	comm or talk
0. Std. RRL (1 agent)	no			x	x			
1. Local learners	any	x						
2. Guidance	teacher	x		x	x			
3. Guided policy learning	perfect	x	x	x	x			
4. Active guidance	teacher	x		x	x			c
5. Actively guided policy learning	perfect	x	x	x	x			c
6. Describing the solution	perfect	x	x	x	x			c + t
7. Collaborative RRL	any	x		x	x			t
8. Find the teacher	any	x		x	x			
9. Learning behavior	any	x			x	x		
10. Learning abilities	any	x			x	x	x	
11. Learning who to imitate	any	x	x		x	x		
12. Knowing and learning abil	any	x	x		x	x	x	
13. Informed active guidance	any	x	x		x	x		c
14. Partially observable world	any	x		x	x			
15. Different interests	any			x				

Table 1. A number of settings with their properties

The first two settings, are the standard settings, (0) for the single agent case and (1) for the situation in which n single (R)RL-agents are put together in the same environment.

Probably one of the simplest settings is the case where `comm_reward`, `know_aim` and `full_obs` hold. Still, even this simple situation is not fully studied in the relational case. One empirical work is on "guidance" [5]. One can see guidance as a setting with two agents where one is a teacher, and the other is a learner (2). Both the teacher and the learner perform actions, and hence it is more likely

that reward is obtained compared to the classical reinforcement learning case. This can be important in difficult planning domains where it would be unlikely to obtain a reward by only exploration. The main advantage is that the teacher directs the exploration towards difficult to reach parts of the state space.

If we also have `know_abil` and the teacher is known to make only optimal actions (3), the learner can directly use the actions performed by the teacher as examples of optimal actions. He could then use a direct learning algorithm that learns actions from states. Another interesting situation (4,5) is the case with `comm_schedule` where the learner may ask the teacher to perform an action (at a certain cost). This is described in [5], while several open questions remain. This can also be seen as a form of active learning.

In the presence of a perfect teacher, the `talk` property together with `know_abil` (6) makes the problem somewhat trivial as the learner can just ask the teacher for the optimal policy (at least if the learner is able to represent that policy). However, the situation gets much more interesting when we have only `comm_reward`, `know_aim`, `full_obs`, `talk` and maybe `know_abil` but no perfect teacher (7). We then have a situation where agents can discuss their learning experiences and even though there is full knowledge, the problem is far from trivial. Indeed, in the relational setting, agents can use very expressive languages to talk. Furthermore, extending the idea of Informed Reinforcement Learning [3], the agents can exchange their learned information. Possible interesting information to share could be subgoals, information about actions like pre- or postconditions but also learned options or macro-actions. No work has been published on which languages are suitable and which questions are most efficient. One could use a language which is a super language of the language to describe states and actions, and can also describe past episodes and statistics. E.g. one could imagine that one agent asks "did you ever see a state with four stacks containing each a red block above a light blue block?" And another agent might answer: "No, but I did see something very similar: I visited 13 states with four stacks containing each a red block above a dark blue one. Is that of interest to you?". Apart from the usual communication issues, one could investigate issues such as the following. What questions are useful in a communication? How to get the desired information at the lowest cost? What generalizations are needed for that? When is it cheaper to explore and find it out yourself?

Another unsolved task (8) occurs in the situation where an agent sees several other agents and knows that some of them are quite good agents. In such a situation it may be interesting to try and find the best agent to learn from. But as the reward is collective, it may not be trivial to detect who is performing the best actions. Or maybe, some agents are experts in certain areas of the state space, and it might be interesting to learn concepts describing the other agent's expertise (the areas where they perform well).

So far we have described settings that are determined by the environment, but agents can also learn information that is not provided by the environment. One such interesting setting is when `learn_beh` holds (9). Here the agents will use their observations to learn the behavior of the agents they observe. This can

for instance be obtained by learning a model for every observable agent that predicts the probability that this agent will perform some action in a certain state. Examples to learn such a model can easily be generated if one can observe (some of) the actions made by other agents. Positive examples are the actual actions that are executed by that agent (at least according to their observation), actions executed by other agents can be used as negative examples or they can also be randomly generated. Interesting to note is that since negative examples can be easily generated, one can supply the learner with more examples per observation to accelerate the learning.

This setting becomes particularly interesting when also the `know_abil` property holds (11), because agents can then execute the actions they believe the expert would make in that situation. Note that this learning task boils down to behavioral cloning [1].

If `know_abil` does not hold, one can still have `learn_abil` where the agents can use their observations to try to learn these abilities (10). Learning these abilities can be based on the rewards the agents receive for their actions, but since this information is often not available, another possibility is for instance to learn the average time an agent is working on some task to estimate his abilities for that task. It is of course also possible to have both `know_abil` and `learn_abil` especially when the given information about the agents abilities is only partial (12).

When we also have the `comm_schedule` or even `talk` property (13), an extension of the active guidance settings occurs. This could be seen as informed active guidance since agents can use their learned knowledge about the other agents to restrict communication. The list of learnable knowledge from observations can of course be extended with models that learn what the results of an agents actions are, that learns what other agents are trying to achieve, i.e., the `know_aim` property, etc.

In what precedes, we have listed a number of (partly) unsolved tasks which may be assumed to be 'easy' for the learner. Of course, one can make the problems much more difficult by adding a number of supplementary complexity factors, such as partial observability (14) or situations where not all agents try to reach the same goal (15).

4 Experiments

In this section, we will present results from experiments in some of the settings of Table 1, more specifically on experiments where the `full_obs` property is used to learn extra information about the other agents and hence accelerate convergence. Previously, we reported on preliminary results where the RRL system is used in the blocks world with multiple agents to analyze the problem of interference and incomplete information, see [22] for more details. Here, we will concentrate on experiments showing the gain of learning the relational structure between the different agents in the system.

4.1 Experimental setup

The test environment we use can be seen as a first step toward multi-state dispersion- and cooperation games, but since at the moment we would like to avoid the problem of interference between the agents, all agents are working on their own task without the possibility to interfere with each other. Not only is this still a challenging problem, an isolated study of the relations between the agents and methods that gain maximally from this, are necessary to tackle more complex problems involving communication and especially interference between agents as for instance in multi-state coordination tasks.

The agents need to fulfill some identical task, this task can be solved by sequentially solving a number of subtasks. Since all tasks are identical, all agents need to solve the same subtasks in the same order.

More specifically, the basic setting is a system with four agents, where each agent needs to solve a certain task by sequentially solving four subtasks. A subtask can only be solved by executing the optimal action and only one of a possible 40 actions is optimal.

When all subtasks are successfully solved, the agents are rewarded with a reward of 1.0, for all other actions they receive neither reward nor punishment. Episodes are ended when all agents have solved all of their subtasks or when a maximum of 800 actions is executed (by all agents together). When the quality of the learned policies are tested, episodes are ended after 40 actions to indicate how many agents learned at least a near-optimal policy.

It is also possible that some agents have prior knowledge about some subtask, i.e. they know what the optimal action is for that subtask. In the basic setting, each agent knows a priori how to solve one subtask optimally and hence he can be called an expert for that subtask. So, the goal of the agents is to learn how to optimally solve the full task.

The settings and language bias We will compare results obtained with the basic setting where all agents are local learners (1) and the settings where observations are used to learn extra knowledge about the other agents and the environment (9-12). One can note that for this environment setting (1) converts to the single-agent case, but averaged over the different agents.

All the agents in the different settings use the same tree building regression algorithm (RRL-TG [6]). To guide this tree building a declarative bias can be specified. This declarative bias can contain a language bias to specify the predicates that can be used as tests in the tree to partition the state and action space. For all agents this language bias contains predicates that identify the subtask and inequalities to partition the action space.

If agents have the `learn_beh` property they can build a model, using the same algorithm and declarative bias, predicting the probability that an agent will perform some action for some subtask. The examples generated to learn these models consists of the actual actions taken as possible examples and for every positive examples, 10 random actions (different from the actual taken action) are generated as negative examples. In fact, different models will be learned for

every subtask. Their language bias for building the Q-function is extended with predicates that test the probability that a specific agent will perform a specific action for some specific subtask, according to the learned model. Note, that it would be possible to test for the most probable action to be executed by an agent, but with the current implementation of the RRL-TG algorithm this requires to iterate over all possible actions every time this query would be posed. Due to their complexity such predicates are not used in the experiments here.

Agents having the `learn_abil` property will learn models that estimate the percentage of the time spent by some agent on some subtask during an episode. To use this information, tests that partition this probability space can be used in the Q-function.

4.2 Experimental results

First, we will present the results obtained using the above described settings and test environment. In the following subsection, we will discuss the scaling possibilities of these settings and we conclude this section with some notes on generalization. The figures show the reward received by all agents averaged over ten test episodes and five test runs.

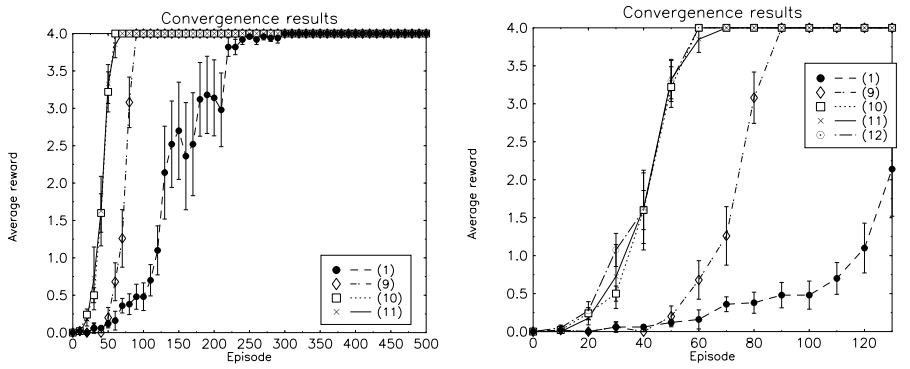


Fig. 1. Standard experiments

Basic settings As illustrated by Figure 1 the learning rate increases significantly when the agents have the `learn_beh` property. It is also clear that having `know_abil` or `learn_abil` outperforms the setting with just `learn_beh`. These experiments also show that for this test environment `learn_abil` performs equally well as having `know_abil`, which means that the abilities of the other agents are easily learned for this environment setup. It is needless to say that in general there can be a big difference between these two properties. Consequently, it is no surprise that having both `know_abil` and `learn_abil` does not further increase the learning rate.

When `learn_beh` holds, the learned Q-functions make use of the subtask identification predicates and the expected actions by the experts for that subtask. Since the learning agent is only interested in distinguishing the optimal actions from the non-optimal actions, it is easier to learn this directly from positive and negative examples. Having this knowledge, it is easy for the learning agent to detect that the Q-values will be higher when executing this action (since he will progress to the next subtask). Hence, when also `know_abil` or `learn_abil` holds, it becomes easier for the learning agent to know which agent to address for every subtask.

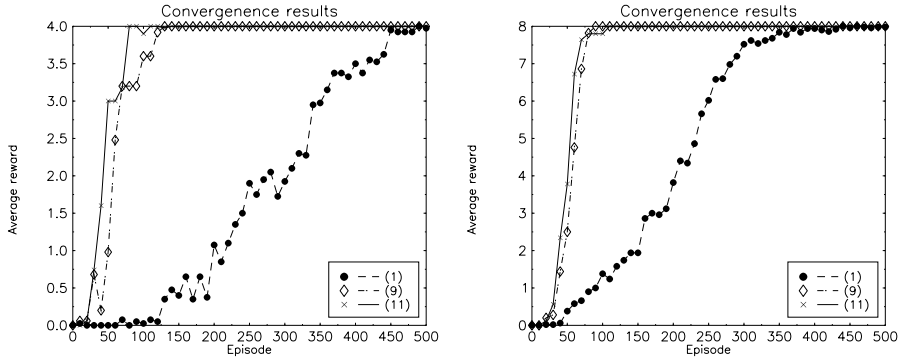
Scaling In this section, we will show the results when the environment parameters are increased: the number of possible actions, the agents in the system and the number of subtasks. Since the settings (10), (11) and (12) performed more or less the same, we only show one in the following graphs. We have omitted the variances in order not to overload the images too much.

Figure 2(a) shows the results when we increase the number of actions to 80 possible actions (per subtask). This is a more difficult learning problem for setting (1), although the variance in these 5 runs was rather high, there is no significant change on the learning rate of settings (9) and (11). This can easily be explained by the structure of the learned Q-function as described above.

To test the settings with 8 agents (Figure 2(b)), the length of test episodes was increased to 80 in order to keep the number of available steps per agent constant. Since each agent still has prior expert knowledge about one subtask, the situation arises where two agents are expert in one subtask. This means that the learning task for every agent does not really become more difficult since the percentage of experts per subtask remains the same.

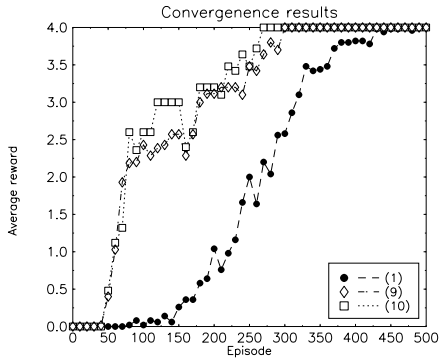
When we increase to number of subtasks to 8 (Figure 2(c)), the learning problem becomes more difficult, even if every agent is expert for two different subtasks, so that the results are not influenced by the lack of experts for some subtask.

Generalization In the previous experiments, the agents learned a different model for every agent. In practice, there will often be a distinction between different kind of agents and the agents can be described by characteristics. The last experiment will show that it is also possible to learn models that generalize over several agents. The setup used for this experiment consists of 6 agents that need to solve 6 subtasks. The main difference with the previous experiments is that the agents and subtasks are described by their characteristics. The agents and subtasks all have a color (green, blue or yellow) and a size (small or large) and every agent is expert for the subtask that has the same characteristics, but the environment is set up in such a way that only the color of the subtask determines the optimal action. So, the agents will no longer learn models for every other agent but they will learn a model for every property an agent can have. The results of this experiment can be found in Figure 2(d) where it is also shown that the agents can benefit from this extra generalization.

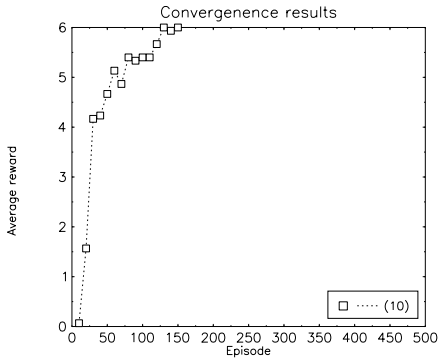


(a) 80 actions

(b) 8 agents



(c) 8 subtasks



(d) generalization

Fig. 2. Scaling and generalization experiments

5 Conclusions

In this paper we introduced the novel idea of cross-fertilization between relational reinforcement learning and multi-agent systems to solve complex multi-state dynamic planning tasks. Current state-of-the-art has mainly focused on stateless (anti) coordination games as for instance Dispersion Games. By using RRL we try to extend this work to more difficult multi-agent learning problems. More precisely, we proposed to use a relational representation of the state space in multi-agent reinforcement learning as this has many proved benefits over the propositional one, as for instance handling large state spaces, a rich relational language, modelling of other agents without a computational explosion, and generalization over new derived knowledge.

We defined different settings in which we believe this research should be carefully conducted, according to six different properties. The different settings are summarized according to their level of complexity in Table 1. Our experiments clearly show that the learning rates are quite good and promising when using a relational representation in this kind of problems and that they can be increased by using the observations over other agents to learn a relational structure between the agents. Moreover, it is clearly shown that in this relational setting it becomes possible and beneficial to generalize over new derived knowledge of other agents.

In our future work we plan to continue along this track and study the addition of communication and interference between the agents since this is an important part of every multi-agent system. A thorough theoretical study of the described properties and settings will also be part of our future research.

Acknowledgements

Tom Croonenborghs is supported by the Flemish Institute for the Promotion of Science and Technological Research in Industry (IWT). Karl Tuyls is sponsored by the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. Jan Ramon is a post-doctoral fellow of the Fund for Scientific Research (FWO) of Flanders.

References

1. M. Bain and C. Sammut. *Machine Intelligence Agents*, chapter A Framework for Behavioral Cloning, pages 103–129. Oxford University Press, 1995.
2. C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multi-agent systems. In *Proceedings of the 15th International Conference on Artificial Intelligence, p.746-752*, 1998.
3. T. Croonenborghs, J. Ramon, and M. Bruynooghe. Towards informed reinforcement learning. In P. Tadepalli, R. Givan, and K. Driessens, editors, *Proceedings of the ICML2004 Workshop on Relational Reinforcement Learning*, pages 21–26, Banff, Canada, July 2004.
4. K. Driessens. *Relational Reinforcement Learning*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 2004. `http://www.cs.kuleuven.be/publicaties/doctoraten/cw/CW2004_05.abs.html`.
5. K. Driessens and S. Dzeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, Dec. 2004.
6. K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In L. De Raedt and P. Flach, editors, *Proceedings of the 12th European Conference on Machine Learning*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 97–108. Springer-Verlag, 2001.
7. S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
8. T. Grenager, R. Powers, and Y. Shoham. Dispersion games: General definitions and some specific learning results. In *Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, Pages: 398 - 403*, 2002.

9. P. Hoen and K. Tuyls. Engineering multi-agent reinforcement learning using evolutionary dynamics. In *Proceedings of the 15th European Conference on Machine Learning*, 2004.
10. J. Hu and M. P. Wellman. Experimental results on Q-learning for general-sum stochastic games. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 407–414. Morgan Kaufmann Publishers Inc., 2000.
11. L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 1996.
12. K. Kersting and L. De Raedt. Logical Markov Decision Programs and the Convergence of Logical TD(λ). In *Proceedings of the 14th International Conference on inductive logic programming*. Springer-Verlag, 2004. To appear.
13. M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning, p 157 - 163*, 1994.
14. E. F. Morales and C. Sammut. Learning to fly by combining reinforcement learning with behavioural cloning. In *ICML '04: Proceedings of the Twenty-First International Conference on Machine Learning*, page 76, New York, NY, USA, 2004. ACM Press.
15. A. Nowé, J. Parent, and K. Verbeeck. Social agents playing a periodical policy. In *Proceedings of the 12th European Conference on Machine Learning, p 382 - 393, Freiburg*, 2001.
16. J. Ramon. On the convergence of reinforcement learning using a decision tree learner. In *Proceedings of ICML-2005 Workshop on Rich Representation for Reinforcement Learning, Bonn, Germany*, 2005. URL = http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41743.
17. S. Sen, S. Airiau, and R. Mukherjee. Towards a Pareto-optimal solution in general-sum games. In *in the Proceedings of the Second Intenational Joint Conference on Autonomous Agents and Multiagent Systems, (pages 153-160), Melbourne, Australia, July 2003*, 2003.
18. P. Stone. Layered learning in multi-agent systems. *Cambridge, MA: MIT Press*, 2000.
19. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.
20. P. Tadepalli, R. Givan, and K. Driessens. Relational reinfocement learning: An overview. In *Proceedings of the ICML'04 Workshop on Relational Reinfocement Learning*, 2004.
21. K. Tumer and D. Wolpert. COllective INtelligence and Braess' Paradox. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence, pages 104-109.*, 2000.
22. K. Tuyls, T. Croonenborghs, J. Ramon, R. Goetschalckx, and M. Bruynooghe. Multi-agent relational reinforcement learning. In K. Tuyls, K. Verbeeck, P. J. 't Hoen, and S. Sen, editors, *Proceedings of the First International Workshop on Learning and Adaptation in Multi Agent Systems*, pages 123–132, Utrecht, The Netherlands, July 25-26 2005.
23. K. Tuyls, K. Verbeeck, and T. Lenaerts. A selection-mutation model for Q-learning in Multi-Agent Systems. In *The second International Joint Conference on Autonomous Agents and Multi-Agent Systems. ACM Press, Australia*, 2003.
24. M. van Otterlo. A characterization of sapient agents. In *International Conference Integration of Knowledge Intensive Multi-Agent Systems (KIMAS-03)*, Boston, Massachusetts, 2003.