Multi-agent Case-Based Reasoning for Cooperative Reinforcement Learners

Thomas Gabel and Martin Riedmiller

Neuroinformatics Group
Department of Mathematics and Computer Science
Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
{thomas.gabel, martin.riedmiller}@uni-osnabrueck.de

Abstract. In both research fields, Case-Based Reasoning and Reinforcement Learning, the system under consideration gains its expertise from experience. Utilizing this fundamental common ground as well as further characteristics and results of these two disciplines, in this paper we develop an approach that facilitates the distributed learning of behaviour policies in cooperative multi-agent domains without communication between the learning agents. We evaluate our algorithms in a case study in reactive production scheduling.

1 Introduction

A reinforcement learning (RL) agent must acquire its behavior policy by repeatedly collecting experience within its environment. Usually, that experience is then processed into a state or state-action value function, from which an appropriate behaviour policy can be induced easily [21]. When applying RL approaches to complex and/or real-world domains, typically some kind of function approximation mechanism to represent the value function has to be used. While in previous work [5], we have explored the use of case-based methods for that specific task, the CBR component will play a similar, yet more prominent role in this paper.

Just like a reinforcement learner, a CBR system's competence is based upon the *experience* it comprises. One main difference is, however, that this experience is not processed mathematically into some kind of value function, but explicitly stored in the system's case base. Furthermore, it is rather unusual to speak of autonomous agents in CBR settings. This difference, however, is of minor importance, since it represents a question of notion and reflects only two different views of describing how the system acquires its experience.

Multi-agent domains in which autonomous agents act entirely independently are faced with the problem that the agents behave without any form of central control in a shared environment, having the goal to learn a behaviour policy that is optimal for the respective environment. This heavily increases the degree of difficulty of learning compared to single-agent scenarios. In earlier work [9], we presented an experience- and Q learning-based reinforcement learning

T.R. Roth-Berghofer et al. (Eds.): ECCBR 2006, LNAI 4106, pp. 32–46, 2006.

algorithm for multi-agent learning of independent learners in general stochastic environments. In a data-efficient version, this algorithm has been proven to feature convergence to the optimal joint policy. In that version, however, it is applicable to problems with finite and small state spaces and has been shown to perform well for a small "climbing game" [4], only.

In this paper, we will extend that algorithm and embed it into a CBR framework. Despite losing guarantees of theoretical convergence to the optimum, we will show that our approach can be applied successfully to larger-scale, real-world tasks. For the purpose of evaluation, we focus on complex application scenarios of reactive production scheduling, in particular on job-shop scheduling tasks [16].

The remainder of this paper is structured as follows. In Section 2 we clarify the problem statement for this paper by focusing on multi-agent reinforcement learning, highlighting its difficulties and reviewing in short the experience-based multi-agent learning algorithm mentioned. Furthermore, relevant related work is highlighted. Section 3 introduces our multi-agent CBR approach and discusses issues of case modelling and distributed case-based value function approximation. In Section 4 we introduce and explain in detail our case-based reinforcement learning algorithm for independent learners. Section 5 depicts the application field of reactive production scheduling and summarises the results of a set of experimental evaluations, applying our approach to job-shop scheduling problems. Finally, Section 6 concludes and points to ongoing and future work.

2 Distributed Reinforcement Learning

Promising a way to program agents without explicitly encoding problem-solving routines, reinforcement learning approaches have been attracting much interest in the machine learning and artificial intelligence communities during the past decades. Traditional reinforcement learning approaches are concerned with single agents that act autonomously in their environment and seek for an optimal behaviour. In many applications, however, interaction and/or coordination with other agents is of crucial importance to achieve some goal.

2.1 From One to m Agents

The standard approach to modelling reinforcement learning problems is to use Markov Decision Processes (MDP). An MDP is a 4-tuple (A, S, r, p) where S and A denote the state and action spaces, respectively, $p : S \times A \times S \to [0, 1]$ is a probabilistic state transition function with $p(s, a, s')$ describing the probability to end up in s' when taking action a in state s. Moreover, $r : S \times A \to \mathbb{R}$ is a reward function that denotes the immediate reward that is obtained when taking a specific action in some state. In search of an optimal behaviour, the learning agent must differentiate between the value of possible successor states or the value of taking a specific action in a certain state. Typically, this kind of ranking is made by computing a state or state-action value function, $V : S \to \mathbb{R}$ or $Q : S \times A \to \mathbb{R}$. For more basics and a thorough review of state-of-the-art reinforcement learning methods we refer to [21].

If there is no explicit model of the environment and of the reward structure available, Q learning is one of the reinforcement learning methods of choice to learn a state-action value function for the problem at hand [26]. It updates directly the estimates for the values of state-action pairs according to

$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha(r(s,a) + max_{b \in A(s')}Q(s',b)) \quad (1)$$

where the successor state s' and the immediate reward $r(s,a)$ are generated by simulation or by interaction with a real process. For the case of finite state and action spaces where the Q function can be represented using a look-up table, there are convergence guarantees that say that Q learning converges to the optimal value function $Q^\star$, assumed that all state-action pairs are visited infinitely often and that α diminishes appropriately. Given convergence to $Q^\star$, the optimal policy $\pi^\star$ can be induced by greedy exploitation of Q according to $\pi^\star(s) = \arg\max_{a \in A(s)} Q^\star(s,a)$.

If there are multiple agents acting concurrently, actions are *vectors* of individual actions. Each agent i contributes its own action component a_i – to be termed *elementary action* subsequently – to the joint action vector $\boldsymbol{a}$. Given the current state and the joint action vector the environment transitions to a successor state s', while at the same time all agents receive the (global) reward signal r. The agents' overall goal is to find an optimal joint behaviour policy, i.e. a mapping from states to actions, that maximises the sum of expected rewards [1].

In [4] the distinction between joint-action learners and independent learners is introduced. While the former know about the actions taken by the other agents, the latter only know about their own contribution a_i to the joint action. As a consequence of their lack of action information, the attempt to estimate an elementary action's value in a specific state would most likely fail: The reward signals for different joint action vectors (differing in the action contributions of the other agents) would mix for any state s and any elementary action a_i. In this paper, we focus on independent learners: As argued before, the key problem of independent reinforcement learners is that they must somehow be enabled to distinguish between different joint actions to which they contributed the same elementary action. We shall explore that issue more thoroughly in the following.

2.2 Core Ideas of a Learning Algorithm for Independent Learners

In [9] we presented an algorithm that realises the idea of implicit agent coordination: Each agent is endowed with the capability to differ between joint action vectors without knowing what the joint actions are, i.e. without knowing which elementary actions are taken by the other agents. As follows, we briefly summarise the core ideas of that algorithm, while in Sections 3 and 4 we further-develop and fully integrate it into a CBR framework.

Implicit Coordination: Each agent i manages for each state $s \in S$ an experience list $E_i(s)$, which on the one hand contains exactly one entry e for every joint action vector and on the other hand for reasons of efficiency is sorted with respect to the estimated value of that piece of experience. The list entry

e is a 3-tuple $e = (Q, a_i, n)$. Here, Q denotes the actual value of e and thus the value of taking the corresponding, not explicitly known action vector in state s (the sorting is done w.r.t. Q), a_i is the elementary action taken by agent i and n stands for the number of times the estimated value Q of this experience has already been updated.

If it can be assured, that at every point of time each agent choosing an action uses the same index x to access its experience list, i.e. selecting $e := E_i(s)[x]$, and moreover, selects the action $e.a_i$ given by that experience, then the reward signal generated by the environment can be related correctly to the (implicitly) referenced joint action vector. Accordingly, the corresponding Q value $e.Q = E_i(s)[x].Q$ may be updated appropriately.

Index Selection Harmonisation: The procedure to select index x within an experience list $E_i(s)$ must be implemented in exactly the same manner in every agent. It has to guarantee that at each instant of time each agent selects the same index, which implies that all agents must be aware of the same time.

Efficient Exploration: Due to the sorting of all experience lists, the best actions are to be found at their beginnings. Therefore, an index selection mechanism that aims at greedily exploiting the knowledge contained in its lists, would always select the first index. However, to trade off exploration and exploitation it is suitable to select the top entries more frequently, but also to choose the last entries with a non-zero probability. The implementation of a corresponding procedure is straightforward.

For further details of the algorithm as well as for a proof of its theoretical convergence properties we refer to [9]. An obvious limitation of this algorithm is its usability for discrete state-action spaces, only. However, the application of generalisation techniques, to deal with large state/action spaces, is of great importance, in particular in multi-agent domains where the size of the joint action spaces can grow exponentially with the number of agents [4]. Therefore, taking the learning algorithm sketched so far as a starting point, in the next section we will present an extended, case-based version of it that may be employed for larger-scale problems.

2.3 Related Work

Different authors have investigated the use of case-based technology in RL-related and multi-agent settings. Powell et. al [17] introduce automatic case elicitation (ACE), a learning technique where the CBR system initially starts without domain knowledge and incrementally gains competence through real-time exploration and interaction within the environment. Their system is successfully applied in the domain of checkers. Macedo [13] focuses in depth on the issue of efficient CBR-based exploration of an autonomous agent in an eventually non-stationary environment an evaluates his approach in the context of robotics. The aspect of agent-cooperation is highlighted, for example, in the work of Ontanon and Plaza on ensemble CBR [14]. The focus there is on case retention of collaborative agents trying to solve some analytical classification task

while having different degrees of competence. In a later work, Plaza spots the issue of distributed case-based reuse, assuming cooperation of multiple agents in problem-solving and tackling configuration tasks [15]. There has also been much work into the direction of multi-case-based reasoning (MCBR [10]), where case bases contain knowledge collected in different contexts and for different tasks or where each case base may specialise on certain regions of the problem space.

A comprehensive article reviewing and comparing a number of memory-based approaches to value function approximation in reinforcement learning is the one by Santamaria and Sutton [20]. With our case-based Q function representation we are in line with these authors as well as with the work of Bridge [3], since we will consider the actions as a part of the cases' solutions.

Multi-agent learning has been an important topic in the RL literature for years. Foundational issues and algorithms as well as a number of approaches to extend the Q learning algorithm into the area of multi-agent domains can be found, e.g., in [7,11,23]. While most of these works focus on discrete problems with finite state spaces, there have also been attempts to tackle larger multi-agent problem domains in conjunction with function approximation. Focusing not on cooperative MA learning, but on adversarial settings with one agent and one opponent, Uther [25] uses a piecewise linear function approximator similar to decision trees in an abstracted soccer game. For the domain of Robotic Soccer Simulation we have learnt a number of strategic behaviors in which multiple agents are involved using neural networks for value function representation [18]. Bowling and Veloso [2] use the CMAC for function approximation in adversarial multi-robot learning in conjunction with policy gradient RL. Cooperative multi-agent learning of independent learners (as we do) and the aspect of inter-agent communication is investigated in the work of Szer and Charpillet [22] where, however, mutual communication between the agents is allowed.

In the evaluation part of this paper we will focus on the application field of reactive production scheduling. For a deeper understanding of that domain we refer to [16] and to our own previous work in that area [19,6] using RL with neural net-based function approximation. Moreover, there have been also attempts to solve scheduling problems with case-based methods (e.g. [8,24,12]).

3 A CBR Approach to Multi-agent Reinforcement Learning

The CBR paradigm tells that similar problems have similar solutions. We may transfer that statement to a terminology that is more closely related to reinforcement learning tasks and say it is likely that in similar situations similar or identical actions are of similar utility. Based on that assumption, case-based techniques have been employed at times to generalise and approximate value functions for RL problems in large/continuous state-action spaces. The same principle also holds when multiple agents are involved: In similar situations a collective of agents will obtain similar rewards when taking similar joint actions.

3.1 Distributed Case-Based Value Function Approximation

For large and/or continuous state spaces S it is impossible to store the expected value of all state-action pairs explicitly (e.g., in a Q table). So, we suggest to utilize the capability to generalise that can be achieved via CBR: We intend to cover S by a finite number of cases, where the expected values (Q values) of different actions are stored in a special form in the cases' solution parts.

Each agent manages its own case base to cover the state space. When a new state is entered, the best-matching representative (nearest neighbour) from the case base is retrieved and the action which promises to yield the highest reward is selected for execution. For single-agent scenarios the implementation of appropriate algorithms seems intuitive, but when multiple agents are involved in decision-making, a number of substantial problem arise:

- Since we consider independent learners that have no idea of the actions taken by their colleagues, we must find a way to enable each agent to differentiate between different joint action vectors in the light of case-based Q value function approximation.
- Efficient and effective exploration of the joint action space are indispensable to learn good policy behaviours. Accordingly, some synchronisation mechanism is required.
- Lack of adequate (state) information in some agents might imply that learning proceeds differently than in other agents. Consequently, retrieval and reuse of experience could not be made in a harmonised way.

The issues raised above manifest necessary conditions for a distributed case-based value function approximation to work successfully. Regarding the last problem we emphasise that we always assume all agents to have the same information on the current global state. We stress that the state information may very well be incomplete, i.e. the environment may be partially observable only (as in our experiments in Section 5), which adds further difficulty to the learning problem. However, then for each agent the same parts of the global state are hidden. As a consequence of the identical global view each agent has and assuming identical case-base management and retrieval algorithms to be used by all agents, it can easily be guaranteed that all agents have case bases with identical contents and that retrieval produces the same results across all agents.

3.2 Case Representation and Retrieval

Pursuing the traditional way to model the case representation, we consider cases that are made up of a problem and a solution part, in the following. Note, that each agent present in our multi-agent settings has its own case base and has no access to any other agent's case base and that no inter-agent communication is allowed. The overall case structure is sketched in Figure 1.

3.2.1 The Problem Part

The cases problem parts are meant to represent the instances s of the state space S. Our algorithms do not pose any requirements on the modelling of the

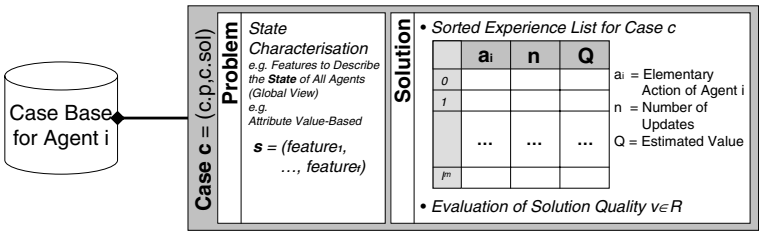

Fig. 1. Case Representation

case problem parts as long as adequate similarity measures can be defined, which reflect similarities between states. So, we assume the existence of some similarity measure $sim : S \times S \rightarrow [0, 1]$ that assesses the degree of similarity between two states. Consequently, given a case base CB, the nearest neighbour $\nu \in CB$ of any query $q \in S$ can be determined easily, and[1] it holds $NN(q) := \nu.p = \arg\max_{c \in CB} sim(c.p, q)$. In Figure 1 the cases' problem parts are exemplarily realised by a number of features (attribute-value based representation).

3.2.2 The Solution Part

For the cases' solution parts we need to define some specific data structures that are aligned with the distributed learning algorithm sketched in Section 2.2. A solution $c.sol$ of a case c consists of two parts, $c.sol = (E, v)$. Whereas $v \in \mathbb{R}$ represents an evaluation value of the solution quality, E is a list into which the experience is compressed the agent has made within its environment. Specifically, $E = (E[1], \ldots, E[l^m])$ contains exactly l^m entries (with m the number of agents and l the number of elementary actions selectable by each agent[2]). Hence, an experience list reserves one entry to implicitly reference each possible joint action vector. Each list entry $e = E[x]$ of E can be accessed by its index x (for ease of notation we will also allow the shortcut $e = c.sol[x]$ instead of $c.sol.E[x]$) and is a 3-tuple, $e = (a_i, n, Q)$ as indicated in Section 2.2. The key point of this representation is that, no matter which case is considered and no matter which list entry of the case's solution is regarded, the agent only knows its own elementary action a_i. Despite that, it is enabled to implicitly differentiate between all l^m joint action vectors by means of the mechanism that shall be explained subsequently.

4 Multi-agent CBR for Independent Learners

In this section we present our case-based multi-agent RL algorithm in a threefold way. First, in Figure 2 we provide a coarse overview over its components involved. Second, Algorithm 1 gives a possible realisation of its main functionality in

[1] We use the notation $c.p$ and $c.sol$ to access the problem and solution part of case c, respectively.

[2] We assume the number of elementary actions to be finite or, in the case of continuous actions, that an appropriate action discretisation has been introduced.

pseudo-code. And finally, the text describes all elements and their interplay in a much more detailed way.

Since our focus is on multi-agent learning, in the following we must clearly distinguish between the agents' learning and application phases. During the former, the agents interact with their environment, collect experience and extend and refine the contents of their case bases. During the latter (the reuse or application phase), the results of learning are exploited, which means that for each state the best action possible is considered by all agents and collectively executed.

Many practical problems are of episodic nature, characterised by some set G of goal states (an episode ends, when an $s \in G$ has been reached). In order to not complicate Algorithm 1 it has been given in a non-episodic realisation, though it may easily be adapted to handle episodic tasks.

4.1 Solution Index Addressing and Exploration

Each time an agent is provided with a new observation, i.e. a new state s, it takes this state as query and searches its case base for the nearest neighbour ν. Most of our considerations to be made in the rest of this and the following sections will refer to the appending solution $\nu.sol$.

For the moment we ignore the question if the agent ought to add a new case for state s to the case base (cf. Section 4.3). The method `selectIndex(i,t)` (step 3 in Figure 2 and 2b-iii in Algorithm 1), as already indicated in Section 2.2, selects an index x to access the solution's experience list $\nu.sol$ and must be implemented in such a way that it returns the same list index in each agent. For this to happen each agent needs the same implementation and moreover, if random accessing is required, for instance, identical random number generator seedings in each agent. Let $x_i :=$ `selectIndex(i,t)` with t as current time, then the agent will choose $\nu.sol[x_i].a$ as its next elementary action[3].

Using a clever implementation of `selectIndex`, this way an efficient exploration mechanism can be realised. For example, ε-greedy exploration can be implemented by returning index 1 with probability $1 - \varepsilon$ (greedy action choice) and a random index from $[1, \ldots, l^m]$ with probability ε.

After all agents have decided for their elementary action the composite action vector is executed and the successor state s' and reward r are observed by all agents (steps 2c and 2d in Algorithm 1 and steps 5 and 6 in Figure 2).

4.2 Distributed Q Learning Updates

Standard Q learning, as briefly introduced in Section 2.1, converges for MDPs with finite state and action spaces to the expected true rewards, when the learning rate $\alpha_i = \alpha_{n(s,a)}$ in the update rule (Equation 1) is sensitive to the number of updates $n(s, a)$ that have already been made to the state-action pair (s, a) and it holds:

[3] Note, that $x_i = x_j$ for all $i, j \in \{1, \ldots, l^m\}$, but in general it holds $a_i \neq a_j$ for many $i, j \in \{1, \ldots, l^m\}$.

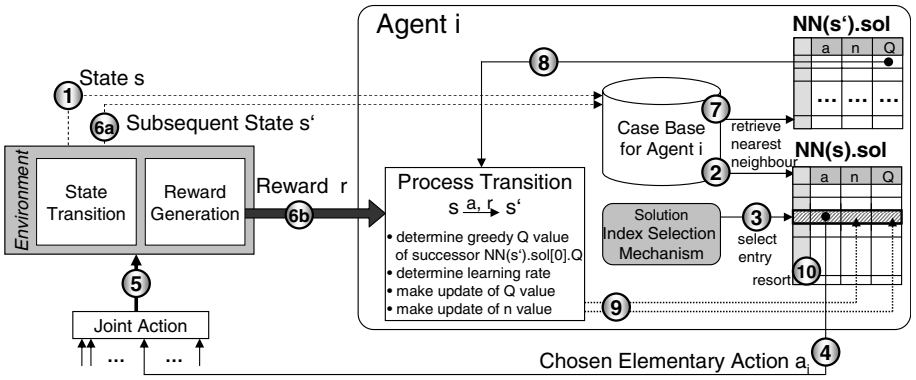

Fig. 2. Procedural View on CBR-Based Multi-Agent Reinforcement Learning

a) the sequence $(\alpha_i)_{i=1}^{\infty}$ fulfills $\alpha_i \in [0,1]$, $\sum_{i=1}^{\infty} \alpha_i = \infty$, $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$, and
b) each state-action pair is visited an infinite number of times.

Although our implementation of the procedure `getLearningRate` (cf. Algorithm 1, step 2e-ii) fulfills the first requirement, convergence to the theoretical optimum could even under fulfillment of b) not be expected, since we do use a case-based function approximator (with a finite number of instances in memory) to cover the state space. Nevertheless, good policies can be learnt even in presence of this kind of generalisation (cf. our empirical result in Section 5).

For an independent learner with a case-based and experience list-based representation of its Q function, the update rule, originally provided in Section 2.1, can now be rewritten with respect to the data structures and case representation we have introduced. Let $T = (s, a_i, s', r)$ be a transition perceived by agent i consisting of state s, (elementary) action a_i of agent i (where a_i corresponds to the previously selected index x_i, i.e. $\nu.sol[x_i] = a_i$), the successor state and the reward, let ν and ν' denote the nearest neighbours of s and s' with respect to CB_i, respectively, and let α_i denote the learning rate calculated by `getLearningRate` (e.g. $\alpha_i := \frac{1}{1+\nu.sol[x_i].n}$), then the agent performs the following updates:

$$\begin{aligned} \nu.sol[x_i].Q &:= (1-\alpha_i) \cdot \nu.sol[x_i].Q + \alpha_i(r + \gamma \cdot \nu'.sol[1].Q) \\ \nu.sol[x_i].n &:= \nu.sol[x_i].n + 1 \end{aligned} \tag{2}$$

After having performed this kind of update, the experience list in $\nu.sol$ is resorted with respect to increasing Q values (see steps 9 and 10 in Figure 2). It is easy to prove by induction that at each instant of time the contents of all agents' case bases and hence, their Q functions, are identical. Due to limited space we omit the proof here.

4.3 Case Base Management

Of course, when the case base is empty the agent has to insert a new, blank case for the state s provided. Otherwise, predicate `addCaseCriterion(s)` must

1. let t be the global time, m be the number of agents, l the number of elementary actions, $CB_i = \varnothing$ an empty case base for each agent, γ the discount factor, and set $s = s' \in S$ to the initial state of the system
2. **repeat**
   (a) **set** $s := s'$
   (b) **for all** agents $i \in \{1, \ldots, m\}$ **do**
      i. **if** $CB_i = \varnothing$ **or** `addCaseCriterion(s)` is true
      then $CB := CB \cup c$
      with $c.p = s$ and $c.sol =$ `emptySolution(i)`
      ii. **retrieve** nearest neighbour $\nu_i := \arg\max_{c \in CB_i} sim(s, c.p)$ of state s
      iii. **set** index $x_i :=$ `selectIndex(i,t)`
      iv. **select** elementary action $a_i := \nu_i.sol[x_i].a_i$
   (c) **apply** joint action $\boldsymbol{a} = (a_1, \ldots, a_m)$
   (d) **observe** successor state $s' \in S$ and reward $r \in \mathbb{R}$
   (e) **for all** agents $i \in \{1, \ldots, m\}$ **do**
      i. **retrieve** nearest neighbour $\nu_i' := \arg\max_{c \in CB_i} sim(s, c.p)$ of state s'
      ii. **set** learn rate $\alpha_i :=$ `getLearningRate(`$\nu_i.sol[x_i].n$`)`
      iii. **set** $\nu_i.sol[x_i].Q := (1 - \alpha_i)\nu_i.sol[x_i].Q + \alpha_i(r + \gamma\nu_i'.sol[1].Q)$
      iv. **increment** $\nu_i.sol[x_i].n$ by one
      v. **resort** the experience list in $\nu_i.sol$
   until `stopCriterion()` becomes true

Algorithm 1. Case-Based Multi-Agent Reinforcement Learning in a Non-Episodic Realisation

make the decision whether to add a new case for s. Here, it must be deliberated whether the experience already contained in CB can be considered reusable for the current situation. In particular, the similarity between s and the problem part of its nearest neighbour in the case base may be a meaningful indicator, provided that appropriate, knowledge-intensive similarity measures have been defined for the task at hand. In our current implementation, `addCaseCriterion(s)` returns true, if the case base size limit has not been exceeded and the similarity between s and its nearest neighbour in CB is less than some threshold ς. Note, that the addition of a new case incurs some necessary follow-up operations:

- Let C_{new} be a case added at time t_{new}. Assume that the transition (s, a, s_{new}, r) has to be processed at $t = t_{new} + \delta$ (with some small δ) where $c := NN(s)$ has been added at time $t_s < t_{new}$. Then, the update according to Equation 2 should take into account that the solution of the nearest neighbour case of s_{new} is most likely rather "uninformed". Therefore, we omit making updates when $NN(s_{new}).sol[1].n = 0$, i.e. when no update for the greedy action in the case responsible for s_{new} has been made, yet. This clearly reduces the speed of learning as long as new cases are added repeatedly.
- After having added a new case c_{new}, the solution parts of all cases in $C := \{c \in C | c_{new} = NN(c)\}$ have to be reinitialised: Let $S_{c_{new}} := \{s \in S | c_{new} = \arg\max_{c \in CB} sim(c.p, s)\} \subset S$ be the subset of the state space, for queries

from which c_{new} is the nearest neighbour in CB. Then, before c_{new} was added to CB, the nearest neighbours of all $s \in S_{c_{new}}$ were elements of C. Hence, the solution parts of all $c \in C$ are no more valid and must be reset.

When resetting as well as initialising a case's solution $c.sol$, i.e. when creating empty solutions (procedure `emptySolution` in Algorithm 1), of course all Q value entries and entries telling the number of updates are set to zero: $c.sol[x].Q = c.sol[x].n = 0$ for all $x \in \{1, \ldots, l^m\}$ in all agents. The field for the elementary action $c.sol[x].a$, however, must be set with some care: The implementation of `emptySolution(i)` must guarantee that – when combining the elementary actions of all agents i over all list entries – there is exactly one entry for each possible joint action, which can be easily achieved by a careful design of the corresponding programming. Given this kind of initialisation and the solution index selection mechanism described, the preconditions for the proof shown in Section 4.2 are satisfied.

Case Base Quality Evaluation: After each learning update step the changed solution $c.sol$ is evaluated with respect to its usability for new problems and the corresponding evaluation is stored in $c.sol.v$. A possible indicator of its quality may be the sum $\sum_{j=1}^{l^m} c.sol[j].n$ of Q learning updates that have already been made for this solution. Currently, however, we employ a simpler, boolean solution evaluation: We consider a solution of a case valid and usable if and only if each of the entries in the belonging experience list has been updated at least once. In other words, each joint action vector possible must have been tried once for each $c \in CB$, until the corresponding case solution is unlocked for use.

5 Experimental Evaluation

To evaluate our case-based and experience list-based approach to multi-agent reinforcement learning we chose the application domain of reactive production scheduling. The learners' task is to autonomously find a cooperative dispatching policy to assign jobs to a limited number of resources, where each job consists of a number of operations that must be performed on specific resources in a pre-defined order. Most classical approaches to solve scheduling problems perform search in the space of possible schedules (e.g. tabu search [16], but also GA-based solutions [12]). By contrast, we take an alternative, reactive approach to job-shop scheduling: We model the environment (the plant) as an MDP and have a learning agent at each resource that decides which job to process next based on its current view on the entire plant (see [19] for more details). A fundamental advantage of this reactive approach is that the agents will also be able to react quickly and appropriately in new, unforeseen situations (e.g. in case of a machine breakdown), whereas most classical scheduling algorithms will have to discard their calculated schedule and start recomputing it. Since, job-shop scheduling problems are well-known to be NP-hard, this can become a time-critical problem.

5.1 Experiment Setup

In our modelling of the scheduling environment a state s must describe the situation of all resources at that point of time – so, s characterises the sets of waiting jobs at all resources. In our current implementation we use an attribute-value based state representation where the state description is made up of $4m$ features, i.e. the sets of waiting jobs at each resource are characterised by four properties. This way, of course, not all of the properties of the currently waiting jobs can be captured, why the environment is perceived as partially observable by each agent. To calculate the similarity between two states we have made use of the local-global principle, defined appropriate local similarity measures for the features and amalgamated them to the global similarity $sim(s_1, s_2)$ using suitable feature weights.

The overall goal of learning is to minimise production costs. Costs arise each time a job is *tardy*, that means when it has not been finished until its due date. So, the overall learning goal is to minimise summed tardiness over all jobs in the system, $\sum_t \sum_j jobTardy(t, j)$. Accordingly, each time one or more tardy jobs are in the system, a negative reward is incurred; if there are no jobs that have violated their due date, the immediate reward is zero.

The actions our agents are allowed to take are decisions to act according to one out of l rather simple, established dispatching priority rules (DPR). A DPR chooses a job j out of a set of waiting jobs J subject to some specific criterion. There is a variety, of more or less complex DPRs: For example, the EDD rule chooses the job $j \in J$ which has the earliest due date. The MS rule picks the job with minimal processing slack and the SPT rule, for instance, chooses a job whose next operation has the shortest processing time. In the scope of our evaluation we will focus on the mentioned three DPRs, i.e. the set of available elementary actions for each agent is $A = \{a^{EDD}, a^{MS}, a^{SPT}\}$. Furthermore, in all our experiments we consider two cooperative scheduling resources/agents that work according to the algorithms discussed in the previous sections.

5.2 Results

Each experiment is divided into a training and a testing phase: A random set S_a of training scheduling scenarios and an independent set S_b of testing scenarios are generated (all of them differing in the properties and numbers of jobs to be processed, $|S_a| = 10$, $|S_b| = 50$). During training, the scenarios in S_a are processed repeatedly[4] where the agent picks random actions with $p = 0.5$ (explores) and that way gathers experience. During testing, the scenarios in S_b are processed once, where now all agents behave greedily w.r.t. their current Q functions, stored distributedly in their case bases. The performance is measured in terms of the average summed tardiness on the scheduling scenarios from $S_{a/b}$.

Comparison to DPR-based Agents: We compared the final scheduling capabilities of our learning agents to nine different agent-constellations in which both agents

[4] We call the processing of one scenario an *episode*.

System (Res1+Res2)	Tardiness Training	Tardiness Testing	Rank
EDD+EDD	62.2	70.40	2
EDD+MS	72.4	75.42	4
EDD+SPT	121.7	138.28	8
MS+EDD	70.1	74.52	3
MS+MS	85.2	79.30	5
MS+SPT	121.2	139.18	7
SPT+EDD	149.6	132.28	6
SPT+MS	158.4	133.82	9
SPT+SPT	205.7	225.96	10
CBRELB	**59.7**	**69.88**	**1**

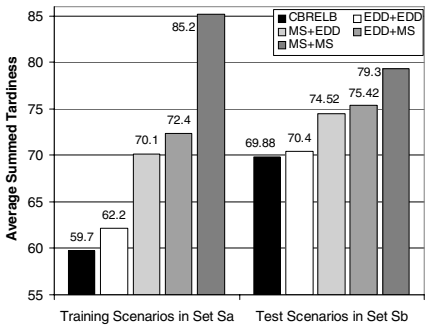

Fig. 3. Performance of the CBR-based Scheduler vs. the Top 4 Heuristic Ones

worked according to some fixed DPR. To be exact, we evaluated each combination of each agent working with one of the rules from $\{EDD, MS, SPT\}$ (see Figure 3). Obviously, the setting when the first as well as the second agent worked with the EDD rule, was the best-performing one (on set S_a as well as on S_b). When both agents were learning, however, the resulting scheduling quality could be increased (data row *CBRELB* in Figure 3). The resulting scheduling system (using a case base of 500 instances) outperformed the best of all DPR-based systems by 4.2% on the training scenarios (average tardiness of 59.7 instead of 62.2). Even on the independent test set, i.e. on 50 entirely different scheduling scenarios the best combination of heuristically acting agents is beaten (average tardiness of 69.88 instead of 70.4). So, one may conclude that the learning agents have discovered and learnt regions of the problem space in which certain joint actions are of extremely high usefulness. We allowed the *CBRELB* agents to learn for 20000 episodes to reach those results.

Case Solution Utilisation: Working within an 8-dimensional problem space, it may happen that for some (query) state q the similarity σ to its nearest neighbour in CB is rather low. Therefore, we allowed each agent to use a fallback action in case that no well-matching case in CB could be found. To be exact, during evaluation an agent used the EDD rule as fallback action in situations when $\sigma < 0.8$ or when the nearest neighbour's solution part had an evaluation value v that marked this solution as not usable (cf. Section 4.3). Of course, the more comprehensive the case base and the longer the learning process has been going on, the less often these situations occur. It is clear that the amount of stored experience must not be too sparse. When experimenting with case bases of sizes 100 and 200 only, the *CBRELB*-setting still outperformed $EDD + EDD$ on the training instances (tardiness of 60.1 and 60.6, respectively), but on the test set an average tardiness of 75.68 and 70.36, respectively, could be achieved only.

6 Conclusions

We have developed and evaluated a CBR-approach that allows for the distributed learning of behaviour policies of independent reinforcement learners. To

tackle the complexity and high-dimensionality inherent in multi-agent settings we employed case-based reasoning as the core technology to facilitate generalisation. Moreover, we combined CBR with a mechanism to achieve implicit coordination of the agents involved which is a necessary prerequisite to make a correct processing of rewards obtained from the environment possible. Our results of a series of experiments in the application domain of reactive job-shop scheduling are very promising, since our approach was able to outperform all schedules generated by a larger number of scheduling systems that worked with fixed dispatching priority rules.

Our research has raised a number of interesting issues to be investigated in on-going and future work. In a next step, we want to evaluate our approach in even larger application scenarios, involving more cooperative learning agents and a larger number of elementary actions. Another interesting issue concerns efficient and effective routines for case base management, case addition and case relocalisation, which need to be developed and further-developed, respectively. Finally, we also seek to design an offline variant of the Q learning update employed which makes more efficient use of gathered experience and, hence, is likely to bring about faster and presumably better learning results.

Acknowledgements. This research has been supported by the German Research Foundation (DFG) under grant number Ri 923/2-1.

References

1. D. Bertsekas and J. Tsitsiklis. *Neuro Dynamic Programming.* Athena Scientific, Belmont, USA, 1996.
2. M. Bowling and M. Veloso. Simultaneous Adversarial Multi-Robot Learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 699–704, Acapulco, Mexico, 2003. Morgan Kaufmann.
3. D. Bridge. The Virtue of Reward: Performance, Reinforcement and Discovery in Case-Based Reasoning. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, page 1, Chicago, USA, 2005. Springer.
4. C. Claus and C. Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Menlo Park, USA, 1998. AAAI Press.
5. T. Gabel and M. Riedmiller. CBR for State Value Function Approximation in Reinforcement Learning. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, pages 206–221, Chicago, USA, 2005. Springer.
6. T. Gabel and M. Riedmiller. Reducing Policy Degradation in Neuro-Dynamic Programming. In *Proceedings of ESANN2006*, Bruges, Belgium, 2006. To appear.
7. J. Hu and M. Wellman. Nash Q-Learning for General-Sum Stochastic Games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.
8. J. Kim, D. Seong, S. Jung, and J. Park. Integrated CBR Framework for Quality Designing and Scheduling in Steel Industry. In *Proceedings of the 7th European Conference on CBR (ECCBR 2004)*, pages 645–658, Madrid, Spain, 2005. Springer.
9. M. Lauer and M. Riedmiller. Reinforcement Learning for Stochastic Cooperative Multi-Agent Systems. In *Proceedings of AAMAS 2004*, pages 1514–1515, New York, NY, July 2004. ACM Press.

10. D. Leake and R. Sooriamurthi. Managing Multiple Case Bases: Dimensions and Issues. In *FLAIRS Conference*, pages 106–110, Pensacola Beach, 2002. AAAI Press.
11. M. Littman. Friend-or-Foe Q-learning in General-Sum Games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 322–328, Williamstown, USA, 2001. Morgan Kaufman.
12. S. Louis and J. McDonnell. Learning with Case-Injected Genetic Algorithms. *IEEE Trans. Evolutionary Computation*, 8(4):316–328, 2004.
13. L. Macedo and A. Cardoso. Using CBR in the Exploration of Unknown Environments with an Autonomous Agent. In *Proceedings of the 7th European Conference on CBR (ECCBR 2004)*, pages 272–286, Madrid, Spain, 2005. Springer.
14. S. Ontanon and E. Plaza. Collaborative Case Retention Strategies for CBR Agents. In *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR 2003)*, pages 392–406, Trondheim, Norway, 2003. Springer.
15. S. Ontanon and E. Plaza. Cooperative Reuse for Compositional Cases in Multiagent Systems. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, pages 382–396, Chicago, USA, 2005. Springer.
16. M. Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Prentice Hall, 2002.
17. J. Powell, B. Hauff, and J. Hastings. Evaluating the Effectiveness of Exploration and Accumulated Experience in Automatic Case Elicitation. In *Proceedings of ICCBR 2005*, pages 397–407, Chicago, USA, 2005. Springer.
18. M. Riedmiller and A. Merke. Using Machine Learning Techniques in Complex Multi-Agent Domains. In I. Stamatescu, W. Menzel, M. Richter, and U. Ratsch, editors, *Adaptivity and Learning*. Springer, 2003.
19. S. Riedmiller and M. Riedmiller. A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling. In *Proceedings of ICJAI'99*, pages 764–771, Stockholm, Sweden, 1999.
20. J. Santamaria, R. Sutton, and A. Ram. Experiments with RL in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2):163–217, 1998.
21. R. S. Sutton and A. G. Barto. *Reinforcement Learning. An Introduction*. MIT Press/A Bradford Book, Cambridge, USA, 1998.
22. D. Szer and F. Charpillet. Coordination through Mutual Notification in Cooperative Multiagent Reinforcement Learning. In *Proceedings of AAMAS 2004*, pages 1254–1255, New York, USA, 2004. IEEE Computer Society.
23. G. Tesauro. Extending Q-Learning to General Adaptive Multi-Agent Systems. In *Proceedings of NIPS 2003*, Vancouver and Whistler, Canada, 2003. MIT Press.
24. P. Tinkler, J. Fox, C. Green, D. Rome, K. Casey, and C. Furmanski. Analogical and Case-Based Reasoning for Predicting Satellite Task Schedulability. In *Proceedings of ICCBR 2005*, pages 566–578, Chicago, USA, 2005. Springer.
25. W. Uther and M. Veloso. Adversarial Reinforcement Learning. Technical Report CMU-CS-03-107, School of Computer Science, Carnegie Mellon University, 2003.
26. C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.