Proceedings of the KI 2015 Workshop on Formal and Cognitive Reasoning

Learning Rules for Cooperative Solving of Spatio-Temporal Problems

Daan Apeldoorn

Information Engineering Group,
Technische Universität Dortmund, Germany
`daan.apeldoorn@tu-dortmund.de`

Abstract. This paper addresses the issue of creating agents that are able to learn rules for cooperative problem solving behavior in different multi-agent scenarios. The proposed agents start with given rule fragments that are combined to rules determining their action selection. The agents learn how to apply these rules adequately by collecting rewards retrieved from the simulation environment of the scenarios. To evaluate the approach, the resulting agents are applied in two different example scenarios.

Keywords: multi-agent simulation, cooperative problem solving, rule learning, knowledge extraction

1 Introduction

Learning to solve problems cooperatively is an interesting and challenging task for agents in multi-agent scenarios with possible applications in logistics, scheduling or robotics (e. g., [5]). In this paper, it is tried to approach this issue by introducing agents that learn to apply rules which are combined from given rule fragments. The agents are then evaluated in the context of different spatio-temporal problem solving scenarios that are defined using the multi-agent framework AbstractSwarm [1, 2] and the learned rules are extracted from the agents' epistemic state.

Section 2 introduces the agent model. Section 3 evaluates the agent model in the context of two different example scenarios and the results are presented. A conclusion and an outlook on future work are given in Section 4.

2 Agent Model

The agent model is considered a template which is instantiated for every agent that is part of a multi-agent scenario. Thus, only homogeneous multi-agent scenarios are considered here, where every agent follows the same implementation. Agents have a *collective* epistemic state, but every agent infers its own actions also considering its current percepts.

In AbstractSwarm the two basic concepts *agents* and *stations* exist. The former refer to the active components of a scenario (being able to act) and the latter represent the passive components of a scenario (serving as *locations* of the agents). These concepts are used in the following as a basis for the agents' percepts and actions.

2.1 Percepts, Actions and Communication

Following the agent interface of the AbstractSwarm simulation framework, the percepts of an agent a consist of:

- a's current location (i. e., the station where a is currently located)
- the current location of all other agents (i. e., the stations where the other agents are currently located)
- the next action of every agent $a_i \neq a$ that was computed in the same time step *before* a (i. e., the station every agent a_i selected as its next target)[1]

An agent's action is defined as the decision about its next target location chosen from a given set of potential target locations.

Before the action of an agent a is executed (i. e., after the agent decides about its next target location, but before the agent starts moving towards this new target location), the agent communicates its decision to all other agents $a_j \neq a$ that are computed *subsequently* to a. This information then is part of the percepts of the agents a_j (see the third point at the beginning of Section 2.1).

2.2 Rules

The agents are provided with a set of *rule fragments* that can be combined to *rules*. The resulting rules are used by the agents to select their target locations. A rule fragment is either the *subject* or the *strategy* of a rule. A rule subject represents a property of a location (e. g., the current free space of the location) and a rule strategy is a selection criterion (e. g., whether a location should be selected according to the maximum or the minimum value of a given subject). A complete rule is of the form *subject_strategy*, stating that locations will be selected based on the given subject and according to the given strategy.

As an Example, consider the rule *FreeSpace_Max*: Following this rule, an agent would choose, among all potential target location, the one with the maximum free space as next target.

Since locations can have a broad variety of different properties that could potentially serve as selection criteria, a subset of rule subjects must be selected here. The focus is set to subjects that seem to be relevant for the example scenarios considered in Section 4:[2]

[1] Note that in the AbstractSwarm framework, in every discrete time step, all agents of a scenario are computed subsequently (in random order) and agents can consider the information communicated by other agents that were computed before.

[2] Note that the selection of possible rules is also restricted by the limited number of properties that are available in the AbstractSwarm framework.

- *FreeSpace*: The current available space of a location (e. g., the remaining number of car agents that are still able to enter a parking deck until it will be completely covered)
- *MaxSpace*: The total available space of a location (e. g., the total number of parking boxes of a parking deck)
- *VisitTime*: The duration of visiting a location, excluding the time to get to the location (e. g., the time that will be needed to do a transaction on a bank counter)
- *RemainingTime*: The time left until a fully covered location becomes available again (e. g., the remaining time a bank counter will be occupied while a bank customer agent is finishing its transaction)

The following rule strategies are selected, which are covering the most common cases:

- *Min*: Selection according to the minimal value of the rule subject
- *Max*: Selection according to the maximal value of the rule subject
- *Avg*: Selection according to the average value of the rule subject
- *None*: No selection according to the value of the rule subject (random selection regarding the rule subject)

In case the application of a rule results in the selection of more than one target location (i. e., the values of the rule subject are equal), multiple rules can be chained, meaning that the rules are applied subsequently.

As an example, following the rule chain *FreeSpace_Max*$\rightarrow$*VisitTime_Min*, agents will first select the locations with a maximum amount of free space (compared to all other potential target locations). If more than one location are currently having the same amount of free space, the one with the minimum visiting time will be selected from these.

If all rules are chained and there are still more than one location in the result set, one of the locations in the result set is chosen randomly.

The order of the rule chains is learned based on the agents' experiences and is inferred from the agents' current epistemic state. The epistemic state together with the learning and the inference process will be explained in the following.

2.3 Epistemic State

The epistemic state of an agent comprises its knowledge about which rules (or rather rule chains) are most preferable. Since the usefulness of rules strongly depends on the problem to be solved (and in many cases there is no a priori knowledge about which rules could be useful), the agents learn their epistemic state by acting in the problem scenario and by earning rewards for their actions. The epistemic state is represented as a Bayesian Logic Network [3] which is trained using Statistical Relational Learning. By this, the learned rules can later be extracted easily from the epistemic state.

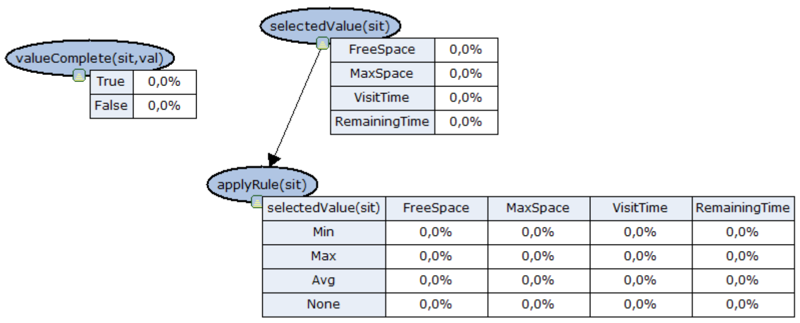
In the following three subsections, the Bayesian Logic Network representing the epistemic state will be introduced and both the learning and the inference process will be described.

Bayesian Logic Network. A Bayesian Logic Network (BLN), introduced by Jain et al. in [3], is a Bayesian Network extended by (many-sorted) first-order Logic. Functions and predicates (which are considered functions with the co-domain {True, False}) are associated with the nodes of an underlying Bayesian Network, such that every node represents a (conditional) probability distribution over the co-domain of the associated predicate. Logical formulas can be added, which then will be considered by the inference mechanism (in addition to the probability distributions of the network). A BLN can be trained such that the (conditional) probability distributions are learned from a given set of data. The BLN is implemented and visualized in the following using ProbCog [4], a software suite for Statistical Relational Learning.

Besides some logical formulas, the BLN for modeling the epistemic state of the agents consists of only three nodes:

- The node *selectedValue(sit)* represents the function with the corresponding probability distribution for the rule subjects.
- The node *applyRule(sit)* represents the function with the corresponding conditional probability distribution for the rule strategies, given a rule subject. The associated function of the node is later queried for inference.
- The isolated node *valueComplete(sit, val)* represents a predicate which is only relevant for the logical formulas implemented in the BLN. (These formulas are used later to handle special cases where some of the potential target locations of an agent are lacking a property and therefore are incomparable regarding this subject, see explanation of the logical formulas below).

The functions associated with the nodes *selectedValue(sit)* and *applyRule(sit)* depend on the current *situation* of an agent (which is used to specify evident knowledge for the inference queries and for the training data later). The predicate *valueComplete(sit, val)* depends on the current situation of the agent and on the subject of a rule. Figure 1 shows the described BLN in the initial state.



True	0,0%
False	0,0%

FreeSpace	0,0%
MaxSpace	0,0%
VisitTime	0,0%
RemainingTime	0,0%

selectedValue(sit)	FreeSpace	MaxSpace	VisitTime	RemainingTime
Min	0,0%	0,0%	0,0%	0,0%
Max	0,0%	0,0%	0,0%	0,0%
Avg	0,0%	0,0%	0,0%	0,0%
None	0,0%	0,0%	0,0%	0,0%

Fig. 1. Initial BLN for the agents' epistemic state.

Note that the agents are using a *collective* epistemic state (see Section 2), such that all agents access the same BLN (both for learning and inference).

However, every agent a infers its own actions, which are additionally depending on its current external state σ_t^a at time t in the environment.

The logical rules implemented in the BLN handle the cases where some locations are lacking some of the properties represented by the rule subjects, which can lead to anomalies for certain subjects (e. g., a closed bank counter does not have a limited duration for visiting, thus applying the rule *VisitTime_Max* would always lead to the selection of the closed bank counter). These kinds of problems are covered by the following formulas:

$$selectedValue(sit, VisitTime) \wedge \neg valueComplete(sit, VisitTime) \\ \Rightarrow \neg applyRule(sit, Max)$$

$$selectedValue(sit, RemainingTime) \wedge \neg valueComplete(sit, RemainingTime) \\ \Rightarrow \neg applyRule(sit, Max)$$

The first formula states that the rule *VisitTime_Max* is never applied in case not all of the potential target locations have a defined duration for visiting. The second formula states this analogously for the time left until a fully covered location is available again.

Learning. While a simulation episode is running, the agents select their locations by trying out different rule subjects and strategies. After the execution of an action is completed by an agent $a \in A$ (i. e., after the agent visited a selected location), the agent gets a *local* reward r from the simulation environment, which is calculated as follows:

$$r := \frac{1}{t_{rwd} - t_{act} + 1} \sum_{t=t_{act}}^{t_{rwd}} \left(\frac{1}{|A|} \sum_{a \in A} w(\sigma_t^a) \right) \tag{1}$$

where t_{act} is the point in time when a selected a target location, t_{rwd} is the point in time when a finished visiting this location, σ_t^a describes the state of a at time t, and function w is defined as:

$$w(\sigma_t^a) := \begin{cases} 1, \text{ if } a \text{ is visiting a location at time } t \\ 0, \text{ otherwise} \end{cases} . \tag{2}$$

Thus, the reward for the action of agent a is calculated from the number of agents in the scenario, that are visiting locations as a consequence of the action performed by a, averaged over time t_{act} to time t_{rwd}: Agents gain higher rewards, the more their actions allow other agents to simultaneously perform their respective actions and lower rewards the more they are restricting other agents.

An agent can easily calculate a *global* reward from the local rewards of its actions by summing up all local rewards until the end of a simulation episode.

Before a new simulation episode starts, the agents store their experiences from the previous episode (i. e., which rules were applied in which situations) by logging training datasets that consist of value assignments to the functions. The following example shows three exemplary training data records:

$$\begin{aligned} selectedValue(sit_1) &= FreeSpace \\ applyRule(sit_1) &= Max \\ \\ selectedValue(sit_2) &= FreeSpace \\ applyRule(sit_2) &= Max \\ \\ selectedValue(sit_3) &= VisitTime \\ applyRule(sit_3) &= Min \end{aligned}$$

A situation identifier sit_i contains the name of the agent that was applying a rule and an index value (including the time when the rule was applied). In the given example, it was preferable in two cases to select a target location according to the rule *FreeSpace_Max* and in only one case it was preferable to select a target location according to the rule *VisitTime_Min*.

The amount of records that are stored depends on the global reward earned by the agent during an episode: The higher the reward, the more training data records are stored. After storing the training data records, the (conditional) probability distributions of the BLN are updated by counting relative frequencies from the training data.

Inference. For inferring results from the trained BLN, the function associated with the node *applyRule(sit)* is queried for every rule subject (i. e., for every value of the co-domain of the function associated with the node *selectedValue(sit)*). The results are the conditional probabilities over the different rule strategies, given the subjects.

Based on the conditional probabilities retrieved from the BLN, if an agent determines its next target location, it performs the following steps:

1. The rule (i. e., the subject-strategy-combination) with the overall highest probability value is applied to select a target location.
2. If this results in more than one location (i. e., the resulting locations are indistinguishable regarding the rule subject), the rule with the next lower probability value is applied to the results from the previous rule. This is continued until there is only one location left in the result set or until all rule subjects were used. Thus, the conditional probabilities $P(Str_1|Sub_1) > ... > P(Str_n|Sub_n)$ would lead to the rule chain $Sub_1\_Str_1 \rightarrow\ ...\ \rightarrow Sub_n\_Str_n$. (If in this case there are still more than one location in the result set, one of the remaining locations is selected randomly.)

By this, stronger rules, that where reinforced through the learning process, are preferred over weaker rules and weaker rules are used with lower priority in a rule chain (in case not all stations could be distinguished by the stronger rules).

To explore the different combinations of rule fragments (even if the agents already learned that some rules are preferable), an exploration probability determines in how many cases an agent decides to use another rule than the one that was inferred from its current epistemic state. The exploration probability depends on the number of already tried rules and the total amount of change in the conditional probability distribution. By this, the exploration probability is slightly discounted over time with increasing experience of the agent.

3 Evaluation

3.1 Test Scenarios

This section introduces the two example scenarios from [1], which are used here as test scenarios to evaluate the agent model. In both scenarios, agents have to solve a problem cooperatively. The scenarios are modeled using the AbstractSwarm framework.

Scenario 1: School Timetabling. In this scenario, a small fictive school is considered, where timetables have to be created for teachers, pupils and rooms. The school comprises:

– 2 math teachers, 2 English teachers and 1 music teacher
– 5 classes (every class has to get 2 math lessons, 2 English lessons, 1 music lesson)
– 4 rooms (3 normal class rooms, 1 special music room)

Teacher agents, class agents and course agents must organize themselves to efficiently create an optimized timetable for the school (i. e., which agent has to be at which time in which room). In this scenario, all locations have the same size (e. g., only one class can be located in a room at a point in time) and the duration of visiting is identical for all locations (i. e., all courses have the same length).

Scenario 2: Production Simulation. In this scenario, a small factory is considered, where workers are producing different products. As part of the quality assurance process, the products have to be analyzed using different machines. The factory comprises:

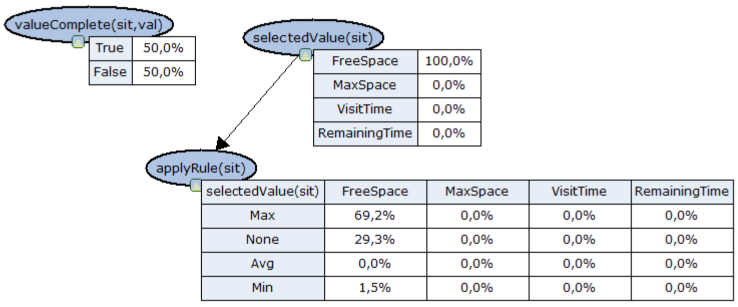
– 8 workers
– 2 kinds of products (5 of each kind, one kind having the need to be analyzed at higher priority)
– 3 machines (2 of which being able to analyze on their own, 1 needing a worker to monitor the analysis

Worker agents and product agents must organize themselves to efficiently create an optimized production plan with few waiting times on the machines. Machines are considered locations with different amounts of space (i. e., one of the machines is able to analyze more than one product at a time) and with different durations of the production and the analysis processes (i. e., the production times depend on the kinds of products and the analysis time needed depends on the specific kind of analysis of a machine.)

3.2 Results

First, the BLNs learned by the agents are inspected in this section and the learned rules (or rule chains) are extracted from the BLNs. After that, the overall performance of the agents for finding adequate solutions for the scenarios will be analyzed.

Learned Rules. In both cases, the agents started without any a priori knowledge about the (conditional) probabilities of the rule fragments (as shown in Figure 1) and 100 runs were performed. The resulting BLNs are shown in Figure 2 and Figure 3.



True	50,0%
False	50,0%

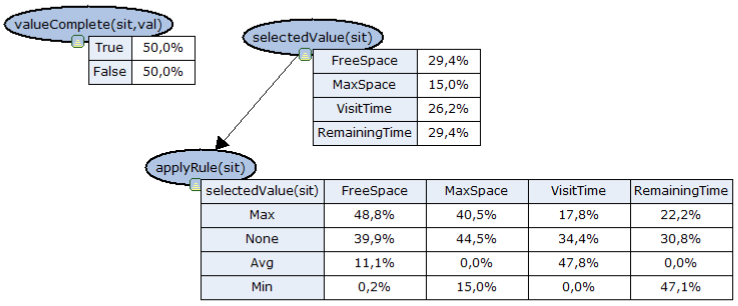
FreeSpace	100,0%
MaxSpace	0,0%
VisitTime	0,0%
RemainingTime	0,0%

selectedValue(sit)	FreeSpace	MaxSpace	VisitTime	RemainingTime
Max	69,2%	0,0%	0,0%	0,0%
None	29,3%	0,0%	0,0%	0,0%
Avg	0,0%	0,0%	0,0%	0,0%
Min	1,5%	0,0%	0,0%	0,0%

Fig. 2. Learned BLN for School Timetabling (Scenario 1) after 100 runs.

From the BLN for Scenario 1 (Figure 2) it can be seen that the agents learned the rule *FreeSpace_Max* with a high success probability. Since in Scenario 1 all locations are of the same size and all courses have the same duration, no further distinctions can be made regarding other rule subjects. Thus, this is the only rule that could be learned.

In case of Scenario 2 (Figure 3), it can be extracted from the BLN, that the rule chain *FreeSpace_Max*$\rightarrow$*VisitTime_Avg*$\rightarrow$*RemainingTime_Min* was learned (since $P(Max|FreeSpace) > P(Avg|VisitTime) > P(Min|RemainingTime)$). Thus, like in case of Scenario 1, it also seems to be useful here to select a target location according to its current available space. But the result is less clear than in Scenario 1: As second and third criteria, agents select their locations according to the average duration of a location (i. e., the average duration of production



True	50,0%
False	50,0%

FreeSpace	29,4%
MaxSpace	15,0%
VisitTime	26,2%
RemainingTime	29,4%

selectedValue(sit)	FreeSpace	MaxSpace	VisitTime	RemainingTime
Max	48,8%	40,5%	17,8%	22,2%
None	39,9%	44,5%	34,4%	30,8%
Avg	11,1%	0,0%	47,8%	0,0%
Min	0,2%	15,0%	0,0%	47,1%

Fig. 3. Learned BLN for Production Simulation (Scenario 2) after 100 runs.

and analysis tasks) and according to the minimal time left until a fully covered production or analysis location becomes available again.

Performance. To analyze the performance of the learning agents, the *total waiting time* of all agents after solving a scenario is considered (i. e., the sum of the idle times of every agent, after all tasks defined in the scenario description were completed). Therefore, 20 repetitions of 100 runs are performed for each scenario and the results are averaged over the 20 repetitions. Every repetition is divided into two phases:

1. The first 50 runs are a *learning phase* where the exploration probability is discounted slightly depending on the experience of the agents (as described in Section 2.3).
2. The second 50 runs are an *exploitation phase*, where the exploration probability is set to zero and the agents act only based on the rules learned in the first phase.

After every repetition, the probability distributions of the BLN are reset to the initial state shown in Figure 1.

Figure 4 and Figure 5 show the performance results for the school timetabling and the production scenarios: The curves represent the minimal waiting time of all agents after r runs (averaged over the 20 repetitions). The gray bars show the waiting time of one selected representative repetition. Note that the agents do not always find a solution for a scenario: The missing gray bars (Figure 5) indicate that the agents could not find a valid solution in this simulation run fulfilling all constraints of the scenario description.

In both Figure 4 and Figure 5 it is shown that the agents are able to quickly find rather good solutions in the two scenarios. Some good solutions are already found randomly at early stages of the learning phase, but it can be seen that the overall performance is getting better through exploitation of the learned rules.

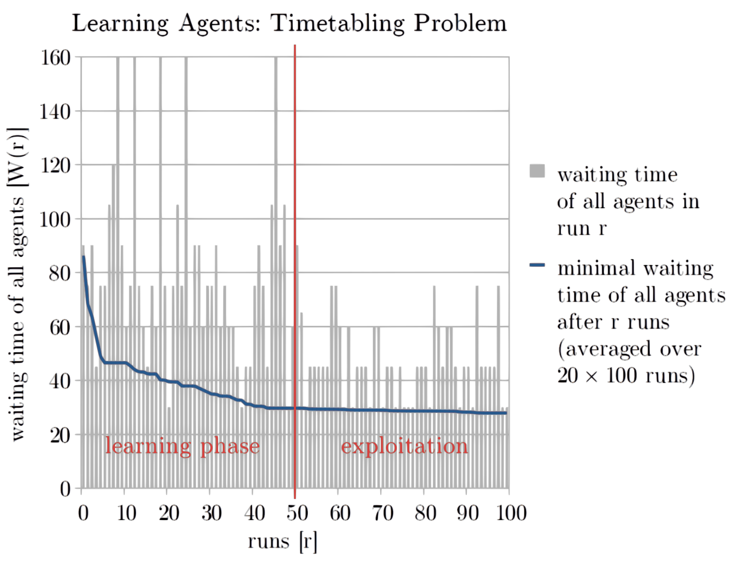


Fig. 4. Performance of learning agents in the school timetabling scenario (Scenario 1).

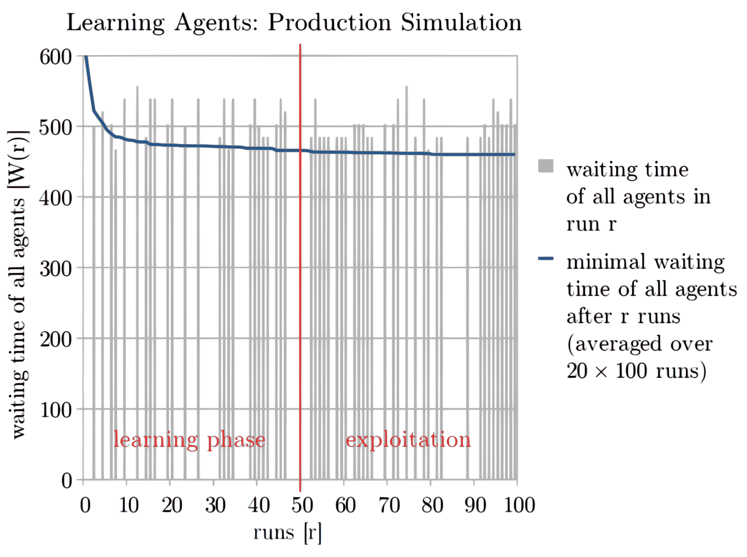


Fig. 5. Performance of learning agents in the production scenario (Scenario 2).

4 Conclusion and Future Work

In this paper, an agent model based on a BLN was presented where multiple agent instances are able to collectively learn rules (and rule chains) for cooperative solving of spatio-temporal problems.

The resulting agents were evaluated in the context of two example scenarios and the results showed that the agents benefit from applying the learned rules (and rule chains).

Unlike other agent-based learning techniques (like Reinforcement Learning), the learned knowledge (i. e., the rules and rule chains) can be easily inspected and extracted from the agents (as shown in Section 3.2) and the agents can be adapted or extended by adding further rule subjects or strategies. Besides that, learning behavioral rules rather than state-action-pairs reduces the state-action-space significantly, which is especially useful in high-dimensional environments (as it is inherently the case in multi-agent-systems, since the state-action-space grows exponentially with the number of agents [6]).

As future work, the rule learning approach could be tested in further, more open environments, as it is the case e. g., for robots cooperating in real world environments (for a related real world scenario see e. g. [5]).

References

1. Apeldoorn, D.: A spatio-temporal multiagent simulation framework for reusing agents in different kinds of scenarios. *To be published in the proceedings of the MATES 2015 conference.*
2. Apeldoorn, D.: Abstractswarm – a generic graphical modeling language for multi-agent systems. In: Klusch, M., Thimm, M., Paprzycki, M. (eds.) Multiagent System Technologies. LNCS, vol. 8076, pp. 180–192. Springer, Berlin Heidelberg (2013)
3. Jain, D., Waldherr, S., Beetz, M.: Bayesian logic networks (extended version). Technical report ias-2009-03, Technische Universität München (2009)
4. Probcog toolbox – a toolbox for statisitical relational learning and reasoning. `http://ias.in.tum.de/software/probcog`, last accessed on 2015-08-15
5. Scheuren, S., Stiene, S., Hertzberg, J., Hartanto, R., Reinecke, M.: The problem of spatio-temporally constrained motion planning for cooperative vehicles. In: Proceedings of the 26th Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2011) (2011)
6. Tan, M.: Independent vs. cooperative agents. In: Proceedings of the Tenth International Conference on Machine Learning. pp. 330–337. Morgan Kaufmann, San Francisco (1993)