Learning in Multi-Agent Systems

Marco Wiering, Ben Kröse, and Frans Groen
Department of Computer Science
Faculty of Mathematics, Computer Science, Physics, and Astronomy
University of Amsterdam
Kruislaan 403, NL-1098 SJ, Amsterdam, The Netherlands
{wiering,krose,groen}@wins.uva.nl

January 4, 2000

Abstract

There is an increased interest in multi-agent systems (MASs) for computing robust solutions to complex real world problems. In this paper we analyze different aspects of multi-agent systems, in particular multi-agent architectures, multi-agent problems, and optimization algorithms for MASs. Furthermore, we present a scheme for mapping multi-agent problems to architectures which can be used for solving them and a mapping from multi-agent problem features to optimization algorithms. Finally, we review the solutions of previous work on many different multi-agent problems.

1 Introduction

Multi-agent systems (MASs). The study of multi-agent systems enables us to come up with robust solutions to complex problems. In the past many monolithic approaches have been constructed to solve such tasks. As problems have become more complex during the last decades, more modular systems have been developed for solving them. The study of multi-agent systems (MASs) is becoming an active field of research in artificial intelligence (AI) which studies how particular problems consisting of multiple components can be solved. For this the MAS engineer identifies local components in the problem (sometimes a problem has to be broken down into local components first) and then uses agents to construct individual solutions for each of the components. MASs are systems in which there is no central control: individual agents receive inputs from the environment (or through communication with other agents) and use these inputs to select individual actions (outputs, decisions). The global (complex) behavior of MASs depends on the local behavior of each agent (the way it acts) and on the interactions between the agents. Many complex problems can be naturally described as MASs such as traffic control, network routing, stock supply, pollution detection, elevator dispatching, forest fire fighting, and transportation problems. E.g., in traffic control, traffic lights and cars are modeled as agents and the total throughput of the system is a result of the complex interactions of the agents and the traffic environment (infrastructure).

Why MASs? The question may arise why we want to construct multi-agent systems instead of single centralized agent systems. There are different reasons for using MASs:

- **External.** Some problems require MASs. Consider different organizations which trade with one another. Each organization has its own goals and proprietary information and wants to keep its own information hidden from other organizations so that it will not give authority to any single person to build a representation that integrates them all. Instead the different organizations need their own systems (Stone and Veloso, 1997).

- **Internal.** MASs are more modular than single agent systems which gives particular advantages: (1) It becomes easier to change the behavior of single agents or to add new agents to the system (scalability); (2) MASs provide a convenient way to integrate distributed computing algorithms to speed up computing solutions for the different agents; (3) MASs are more robust, especially if there are redundant agents available. If an agent in a MAS breaks down, the other agents can still continue working so that the system degrades gracefully and the problem can still be solved. This is very different from a central controller breaking down — when this happens there is not another way for selecting control actions for the agents so that the whole system cannot operate anymore.

Learning in MASs. The first possibility to construct a MAS is to design a MAS architecture and to program the interaction protocols and behavior of all agents in the system. The behavior of an agent is usually captured by a decision module (policy) which uses the inputs of the agent to choose an action. As problems get more complex, programming the correct action for each possible input is a demanding and time-consuming task. Even if we are able to program the behavior of an agent in a particular environment, this behavior may fail entirely when the environment changes. Therefore a more convenient way to construct MASs is to make use of machine learning (ML) techniques which are able to adapt an agent's behavior automatically and have the aim to optimize the agent's behavior in the environment. Training an agent is done by having the agent interact with the environment and using feedback (reward) from the environment to adjust the agent's behavior. If the environment changes, so will the agent's behavior. Since programming agents is a hard task, we will focus on the use of ML techniques or other optimization methods to automatically search for an (optimal) behavior of an agent.

Outline. In this article we will first consider different multi-agent systems and will describe their most important features and architectures in section 2. Then in section 3, we will describe a set of algorithms and methods which can be used to optimize a system. In section 4, we will consider a set of multi-agent problems and describe work which has been done to solve them. In section 5, we will draw some conclusions.

2 Multi-agent systems (MASs)

Multi-agent systems can be used to naturally solve a wide variety of problems. An important topic in MASs is the coordination of the behaviors of different agents. Although in single agent systems, an agent pursues its own goal, there may not be such a thing as a single goal for MASs. MASs can also consist of different self-interested rational agents which learn behaviors which fulfil only their own goals. Since the MAS designer wants to optimize the behavior of the global system, we have to evaluate its global performance. When the agents pursue a common goal, e.g. in network routing, the performance of the global system can easily be evaluated. Otherwise the performance of the global system is dependent on the performances of all single agents, and the global goal would consist of all goals of the individual agents (which may not always be completely attainable). Often these agents interact and can help or hinder each other when they pursue their goals. Therefore, to optimize the global system, agents should cooperate which is made possible in different ways. E.g., we can use communication between agents so that they can negotiate about their goals and actions and then execute joint plans, or we can use management agencies or social laws which determine which behaviors should and which ones should not be used.

2.1 Description of agents

Although we do not want to provide a theory of agency here (we refer to (Wooldridge, 1999) for this), we will shortly describe how an agent interacts with its environment. An agent uses real or virtual sensors to obtain information from its environment, and uses actuators for executing actions. Its input consists of information of the environment, but may also include information obtained through communication with other agents. By using a decision policy, the agent maps inputs to actions. The goal of an agent is usually provided in the form of an evaluation or reward function. At particular time steps the agent may be evaluated and the given amount of reward determines how far the agent is successful in reaching its goal. The higher the long term reward intake, the better the agent performs. See figure 1 which depicts a two-agent system.

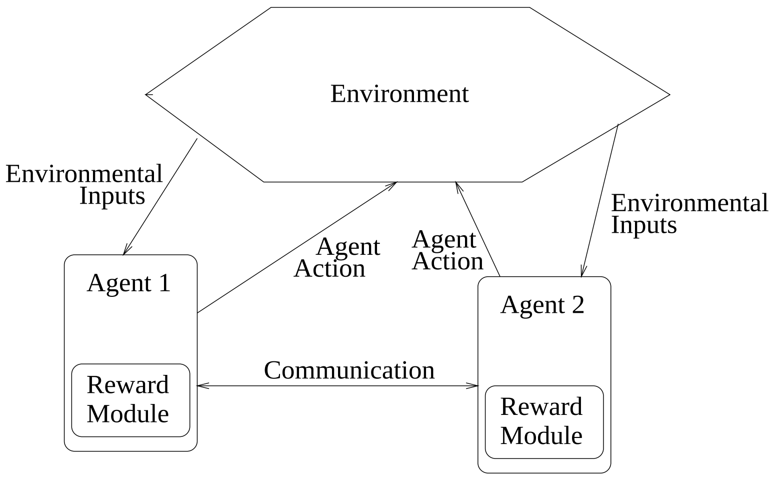


Figure 1: *A MAS in which two agents interact simultaneously with an environment and have the possibility to communicate with each other.*

All agents interact simultaneously with the environment. After all agents have selected their action, the environment is changed (to a new state) and the agents receive a reward (evaluation signal) based on their local reward function which maps the arrival in the new state to a scalar reward value. The goal of each agent is to achieve the highest possible sum of rewards (utility) and in this way each agent is self-interested. To achieve this goal, agents adapt their policy based on their experiences and try to find that policy in the space of all possible policies which leads to the maximal future reward sum given some state. As agents optimize their individual policy, we would like to see that the performance of the whole MAS improves as well. This is not always the case, however. Sometimes goals of different agents are dependent on each other so that if one agent pursues its own goal it hinders another agent reaching its goal. Therefore agent's behaviors should be coordinated so that system performance (or the social welfare: the sum of all agent's performances) is optimized when agent behaviors are optimized.

2.2 MAS architectures

An important task for a system engineer when constructing a MAS is to design an architecture which determines how agents are organized. E.g., the organization determines whether and how agents can communicate and is essential for coordinating agent behaviors. We would like to set up an architecture which allows for cooperation between groups of agents. We will describe a number of MAS architectures, and examine their complexities and whether they are static or dynamic architectures.

- We define the **complexity** of an architecture as the total size of the search (policy) space.
- An architecture is **static** if it remains invariant during the learning process. An architecture is **dynamic** if some components of it change during the adaptation process.

The multi-agent architectures can be separated according to the problem they deal with. First of all, architectures may deal with the problem of breaking down a very large and complex central controller into parts and assigning tasks and goals to individual agents. Second, architectures may deal with handling the problem of partial observability of the global state and with the separation of problem-solving information over agents. Here, combining information from multiple sources may reduce the uncertainty in the decision making of single agents. Finally, architectures may deal with the coordination of multiple agents. To deal with multiple problems at the same time architectures may be combined.

2.2.1 Centralized control

We can use a *single group policy* which maps a description of the environmental state to a set of agent actions. This can be considered as a single agent system, since the central policy is not broken down into local components. E.g., in a network routing problem we could have a superagent which receives information about the global state of the network and selects for each node to which neighbor it should send its currently processed package. Although such a policy may guarantee finding optimal cooperative policies of all agents, the number of environmental states and the product action space of the superagent is usually much too large to be stored. Furthermore, action selection will in general cost a lot of time. Finally, the system is not robust: if the central controller breaks down, the performance of all agents collapses.

Mataric (1997) uses centralized policies for two robot agents in a box-pushing task. In her system, the agents receive the sensory information of the other agent (by communication) and take turns in selecting the actions for both of them. The complexity of this system can be measured as the number of allowable policies which equals: $A^{N^{S^N}}$, where A is the number of actions, S is the number of input states, and N is the number of agents. The architecture is static. It deals with the problems of partial observability and can be used to coordinate agents.

2.2.2 Policy sharing

In order to make the policy space much smaller, we can use policy sharing. In policy sharing, each agent uses the same policy for selecting an action, although behaviors between these homogeneous agents differ since they receive different inputs. This approach is pursued in (Sałustowicz et al., 1998; Wiering et al., 1998) for learning soccer strategies. Furthermore, the ant colony system (Dorigo, 1992; Dorigo and

Di Caro, 1999) uses a global pheremone trail which is shared by all agents (ants) for decision making. The advantage of policy sharing is that training them can be very fast, but a disadvantage is that it does not allow for task specialization or self-interested agents (agents can only pursue a common goal). The complexity of this system equals: A^S, a huge reduction compared to the centralized system if there are many agents. The architecture is static. Policy sharing is good for breaking down the large centralized controller so that the architecture does not suffer from space limitations anymore. For some applications total policy sharing is unwanted, and we may only want to use shared policies for subgroups of agents.

2.2.3 Local agents

We can also reduce the size of the policy space by breaking a central controller down into local non-directly interacting agents. Thus, we can construct a system where each agent has to learn its own policy and the whole system is evaluated by examining the group behavior. E.g., in a network routing problem, at each time-step each node may use its individual policy to decide to which neighboring node it will send its currently processed package given the destination address of the package. A disadvantage of using totally uncoupled systems is that it may be hard for the system to find optimal cooperative behaviors, since agents usually have a very local view and agent behaviors are not directly coordinated. Since dependencies between tasks of agents are not modeled explicitly, a degradation of performance may result if a local agent system has to solve tasks with many dependencies. For such problems we could supply the agents with communication facilities to coordinate their behaviors. Another possibility is to let agents learn models of other agent's behavior so that agents themselves can reason about and overcome possible conflicts. Finally, an important topic is which reward function to use. E.g. agents can use global or local reward functions or functions so that the reward intake of the agent and its neighbors is optimized (Schneider et al., 1999).

Using a system with local learning agents is used by (Crites and Barto, 1996) for learning to control elevators and by (Littman and Boyan, 1993) for network routing. The complexity of this system is: NA^S, since we have N policies of size A^S.[1] The architecture is static. The local agent architecture also concentrates itself on the problem of breaking down the large centralized controller so that the architecture does not suffer from severe space limitations.

2.2.4 Task schedules

When multiple tasks have to be executed by a number of agents, we can use task schedules to divide the tasks over the agents. The schedules tell the orderings of the operations performed by each of the agents. Each agent can have specialized skills and the goal of the engineer or learning algorithm is to divide a task into sequences of subtasks for individual agents. E.g., in a job-shop scheduling problem, a schedule may order the tasks each machine (agent) has to carry out in order to make the total processing time of all jobs as short as possible. One disadvantage of these precomputed schedules is that they are often not reactive to dynamic changes of the environment, since they are usually precomputed. Therefore if some agent breaks down, a completely new solution has to be computed unless there are ways for assigning its subtask to other agents. Schedules may be useful when tasks can be broken down into parts for which special skills are desired. The complexity of this system is: $NA^S + T(N)$ where the second term $T(N)$ refers to the number

[1]Although there are more possible team policies (A^{NS}), we neglect policy-dependencies and assume agents can improve their policies regardless of other policies. Therefore, when making a single policy change at each time-step we only have to consider one out of NA^S possibilities.

of possible orderings of the tasks. The architecture is dynamic, since for different global world states, different task schedules may have to be generated. A task schedule architecture concentrates itself on the problem of breaking down the central controller and on assigning tasks to agents and thus coordinating them.

2.2.5 Hierarchical systems

Instead of breaking down the problem entirely into agents which do not share global information or interact directly, we can also use a hierarchical design of the system to ensure cooperation in the system. E.g., we could think of a supervisor (management agency) which determines which behaviors are allowed for agents (social conventions) and which agents have to cooperate to solve some task. In general the supervisor will have a more global view and it can ensure better cooperation by coordinating agent behaviors. The supervisor combines agent policies in a top-down way in which supervisors can use the current global world state to choose which agents cooperatively solve some subtask of the task (and thus may share reward functions). We could also have different agent roles (policies) and learn global strategies for combining these roles (Tambe, 1997). The difference with a central controller is that hierarchical systems can combine a small number of agents into a policy instead of combining them all. A disadvantage of such a system is that it may be less robust, since management agents may break down and cause havoc in the total behavior of the system.

Another way of using hierarchical systems is to use coalition structures (CSs) between agent policies (Sandholm, 1996). Coalition structures determine which agents work together to solve a particular task and how they do that. The formation of coalition structures could be the result of a negotiation between agent so that they cooperate to handle some task. CSs can also be used to match policies of agents so that each time that some CS is used, two specific adaptable policies are combined which have (already learned) to cooperate. Note that in this system explicit communication between agents would also be possible. The complexity of the system is about: $NMA^S + K(MN)^C$, where M is the number of policies allowed for each agent, K is the number of coalitions and C is the number of agents which can be combined inside a single coalition. Note that if M and C are 1, the system would essentially be the same as using local agents. The architecture is dynamic, since at different moments different agents may have to work together. The hierarchical system provides us with a solution to the problem of how to coordinate different agents.

2.2.6 Global world models

Agents may also share a global world model which they use for their decision making and which they can alter themselves. In this way the global world model provides a way to communicate (abstract) information between agents. This helps to overcome limited perception of some agents. E.g., if some agent has detected an event of interest, this is communicated to the global world model, and other agents may respond to this information. Global world models are used in e.g. robot soccer (Stone et al., 1999). Here, sensory information over time is gathered to make the global world model as accurate as possible. This methods is also pursued in (Ye. and Tsotsos, 1997) to search with a team of robots for an object in a 3D environment. It resembles policy sharing somewhat, although with global world models only state information is shared — decision policies can be learned locally. The method requires the abilities to match agent's local views to the global world model and to adapt the world model accordingly. The complexity of this system depends a lot on how agents select actions. If we consider an agent which uses the state of the global world model to select actions, and all agents abstract their local views to a state in the global world model, the complexity is $NI^S + NA^{NI}$. Here, I refers to the number of abstract states for each agent. The term NI^S refers to the number of possible abstractions from states to abstract states. It is clear that for very large or continuous state spaces, this should be done using function approximators. The term NA^{NI} refers to the number of policies for all agents when NI states are available in the (abstracted) global world model. The architecture is static, although the global world model may be dynamic, e.g. it can involve different numbers of possible states over time. Global world models deal with the problem of partial observability of the global state.

2.2.7 Communicating agents

When agents have to cooperate, it may be very useful that agents can directly communicate with each other to give information about their plans so that future behaviors of the agents are better coordinated. This can be done by using language protocols such as KQML (Finin et al., 1992) or finite state automata (FSA) which cause each agent's description of the world to be augmented with abstract information received by the communication with other agents. Thus, communication provides a way to expand the limited input of an agent so that it "knows" about the existence of other agents. Agents may even communicate recurrently so that they can match their independent actions so that an action of an agent can be conditioned on other agent's actions (Gmytrasiewicz and Durfee, 1992). This repetitive communication should stop, however, and therefore it should be limited to a specific number of steps. Other ways of communication are to use blackboards containing messages of agents transmitted to the whole system (Carver et al., 1991). In this way, the agents communicate immediately to all other agents which has the drawback that communication bandwidth is enlarged. Problems with increased communication is that input spaces are enlarged and distributed implementation may become less efficient if communication bandwidth is large. Although communication provides good ways for letting agents cooperate, it is still a hard task for an engineer to specify which protocols agents should use and therefore algorithms which are able to learn communication protocols would be quite interesting. Some research in this direction is described in (Steels, 1997; Balch and Arkin, 1994; Murciano and Millan, 1989). Finally agents may negotiate on joint plans and in this way maximize their individual utilities (Zlotkin and Rosenschein, 1996). Mataric (1997) discusses communication in a multiple-robot setting to deal with hidden state and credit assignment. The task is a box-pushing task with two six-legged robots. The robots are equipped with a radio communication mechanism and have to coordinate their efforts to be able to push the box to a light source. Each robot communicates its state-information to the other agents and learns a function mapping the combined perceptual state to the best action for itself and the other agent. The robots take turns in controlling the box and in this way, the behaviors of the robots were matched effectively. By using the combined perceptual state and taking turns, the credit assignment problem was solved, whereas by communicating the sensory data the hidden state problem was made less worse. Since she essentially used a central controller, the approach could not be used for systems consisting of many agents. When we allow full communication, the complexity of the system is $NA^{I^N S} + NI^{(N-1)^S}$, where I is the number of different messages an agent may get from another agent. The first term reflects the number of policies when the state space is enlarged (to $I^N S$) due to communication. The second term refers to the number of communication policies (for each agent state, one out of I messages can be sent to each of the $N-1$ other agents). We can note that for very large or continuous state spaces, function approximators should be used to store the communication function and full communication should be broken down into local communicating groups. The architecture is in principle static, although different interaction protocols could be designed to make the system dynamic. Communication can be used to deal with the problem of coordinating different agents or for dealing with partial observability of the global state and separated problem solving information. E.g. a blackboard full with messages could be seen as some form of global world model.

2.3 Characteristics of multi-agent problems

There are many problems which are naturally described by multi-agent systems. In this section we will list a set of problems and create a classification scheme based on features inherent in these problems. Important questions which are of interest when labeling multi-agent problems are:

- **Is the problem environment static or dynamic?** The traditional job-shop schedule problem is an example of a static problem since all jobs, machines and ways of processing each job are usually assumed to be fixed. In contrast a network routing problem can be highly dynamic since traffic load can increase or decrease significantly during the operation of the system resulting in a non-stationary environment.

- **Do agents communicate as part of the task?** In some problems, such as robot soccer (RoboCup soccer), agents are allowed to communicate. For some other problems, communication could be possible, but is not necessarily posed in the problem's description. If communication is used, the engineer is requested to set up communication protocols which can make the problem appear harder. Communication can also suffer from malicious communication acts of opponents. E.g. consider an intelligent opponent making use of the same communication method with the intentions to distort agent communication. Especially for surveillance and security robots, communication methods should be reliable which could be done by using identification protocols of senders.

- **Is the state representation for an agent continuous or discrete?** The agent receives inputs through its sensors. Sometimes this information can be discrete such as in network routing problems, for which we make the state space of each agent discrete by just considering as local input the destination place of each package (eventually augmented by information which indicates whether neighboring nodes are busy or not). For some vehicle routing problems we are given a set of destination addresses to which packages should be delivered. Given a set of vehicles and distances between destination addresses, we could just compute an ordered list of places where each agent has to go to. In this way the state space representation is discrete, since we only consider discrete orderings of destination addresses. This will not work well for highly dynamic problems, however, in which the current environmental state changes a lot and changes in the architecture have to be done online. E.g., for dynamic stock supply problems each agent may require sensors (e.g. camera's) to obtain local input information which describes the state of the environment such as possible obstacles, positions of packages which are waiting to be delivered, and positions of the other agents. This provides us usually with a high-dimensional continuous feature (input) space. In this case, we may want to discretize the space first so that we are able to use efficient planning methods working with discretized spaces. Discretizing a continuous space is still an interesting topic of research (see e.g. Thrun (1998) for a robot application constructing a discrete map of a building and Yamauchi et al. (1998) for a multi-agent map-building application).

- **Do the agents share a common goal?** For many complex problems the agents naturally cooperate in order to optimize the solution. Then we speak of a global evaluation or reward function which is shared (positively correlated) by all agents. Such agents are called *cooperative* agents. When dealing with multi-agent systems, we could also have the case where different agents have different tasks to perform and use their own reward functions. In such cases agents' reward functions may be negatively correlated (zero-sum games) and the goal becomes to beat an opponent. Such agents then have competing goals and are called *competitive* agents. Other problems may not have purely negatively or positively correlated reward functions and are in the realm of game theoretical problems such as the prisoner's dilemma, bargaining problems or others. For such problems agents may help or hinder each other when they try to optimize their own behavior, and several problems may arise such as non-cooperating policies and the tragedy of the commons (TOC) (Hardin, 1968; Turner, 1993) where agents mutually over-exploit the resources of the system which may finally harm all of them. For these latter problems we need to find a mutual set of policies which cooperate as good as possible. This can be done in different ways such as using communication or the use of management agencies (Turner, 1993). We call such agents *compromising, self-interested* agents. Thus the goal may be a common one, a competing one, or a compromise one. Especially for the latter goal agents have to be coordinated.

- **Is the problem completely or partially observable for the system?** For particular problems such as a job-shop schedule we use a model which contains all relevant data. Such problems are completely observable. Other problems may be partially observable, such as stock supply problems where each agent does in general not have access to a description of the complete working floor since that would require expensive sensory equipment and make the local input space much too large.

- **Static agents/ Dynamic agents.** In some problems, agents, such as traffic lights or nodes in network routing problems, have a fixed determined place in the environment. Static agents can therefore never collide. In other problems, agents are moving physical objects which can explore the environment and usually need sensors to observe their local environment for navigation purposes.

- **Are agents independently changing the state of the environment?** In many problems, we can use the proposed action of each agent individually to compute the new state of the environment. E.g., in a network routing problem the traffic flowing out of a particular node is independent of other actions selected by other agents. However, in some problems this is not the case. E.g., examine two robots which have to carry some heavy object together. If one agent makes some action which is not correlated to another agent's action, the object which they are carrying may drop on the ground. Another example of non-independence is when agents collide since they want to occupy the same physical position in the environment.

Figure 2 shows how different problems can be characterized according to the above distinctions.

	Static / Dynamic Environment	Communication Between Agents	Continuous / Discrete State Space	Common / Competing Goal(s)	Static Agents	Independent Agents
Stock Supply	Dynamic	Yes/No	Continuous	Common	No	No
Network Routing	Dynamic	No	Discrete	Common	Yes	Yes
Explorative Detection	Dynamic	Yes/No	Continuous	Common	No	Yes/No
Transportation Problems	Static	No	Discrete	Common	No	Yes
Robotic Soccer	Dynamic	Yes	Continuous	Common/ Competing	No	No
Predator/ Prey	Dynamic	Yes/No	Discrete	Common/ Competing	No	No
Forest Fire Fighting	Dynamic	Yes/No	Continuous	Common	No	No
Underwater Exploration	Static / Dynamic	Yes	Continuous	Common	No	No
Data Discovery on the Web	Dynamic	Yes/No	Discrete	Common	No	Yes
Traffic Control	Dynamic	Yes/No	Discrete	Common	Yes/No	Yes
Collective Map Building	Static	No	Discrete / Continuous	Common	No	No
Electronic Trading	Dynamic	Yes	Discrete	Compromise	No	No
Job Shop Scheduling	Static	No	Discrete	Common	Yes	Yes

Figure 2: *A classification scheme for multi-agent problems.*

2.4 Matching problems to architectures

Particular problems can be handled easier when particular architectures are used. In figure 3 we show how problems can be matched to architectures. Some of these matches have also been described in previous work.

The centralized controller is almost never advised. Only for scheduling problems it could be used, e.g. combined with local search algorithms or negotiation based algorithms for trading allocated resources. E.g. the M-contract used by Anderson and Sandholm (1998) can make arbitrary complex swap moves and therefore acts like a central controller creating a single schedule.

Policy sharing has been shown helpful for simulated soccer (Sałustowicz et al., 1998; Wiering et al., 1998) and for predator prey problems (Tan, 1993). Here homogeneous agent may profit by combining their experiences so that each time step more experiences are generated for updating the policy, resulting in faster learning and better final policies. In RoboCup, partial policy sharing is also used in the form of agent roles such as defenders (Tambe, 1997) which can be shared by multiple agents. Policy sharing cannot be used for problems requiring different skills of agents. E.g. in network routing, different nodes (agents) map the same input (destination of a message) to different actions (next nodes).

Task schedules are useful for difficult problems where time requirements and/or

	Centralized Control	Policy Sharing	Task Schedules	Local Agents	Hierarchical Systems	Global World Models	Communication
Stock Supply			X		X	X	X
Network Routing				X			X
Explorative Detection					X	X	X
Transportation Problems			X		X		
Robotic Soccer		X			X	X	X
Predator/ Prey		X				X	X
Forest Fire Fighting			X		X	X	X
Underwater Exploration						X	X
Data Discovery on the Web				X			X
Traffic Control				X			X
Collective Map Building					X	X	X
Electronic Trading				X			X
Job Shop Scheduling	X		X	X			

Figure 3: *Matching problems to architectures. Here an 'X' indicates that a particular architecture is suitable for solving a particular problem.*

specialized skills play a role. E.g. in transportation problems, a crane can only start working if a transport wagon has delivered containers. On its turn, once a pile of containers is standing, some crane has to start placing them on the yard. In forest fire fighting, task schedules can be used to plan future sequences of subtasks.

Local agents have been shown fruitful for particular problems such as network routing. Furthermore they can be used to deal with self-interested web- or information-agents. These local agents can learn a model of their private world and use this to make their decisions. For electronic trading, the local agents can be combined with communication to negotiate with other agents. In Vidal and Durfee (1998), a modeling approach is described for trading agents, where agents can learn different models of the other agents, varying in the amount of detail and knowledge about other agent's behaviors.

Hierarchical systems are useful when agent behaviors have to be coordinated. These systems can easily combine policies so that cooperation can be enhanced. E.g. it can be used for explorative detection so that agents will not generate the same paths for doing observations. Instead their policies are dependent on each other by putting them in a hierarchical system. For robot soccer, team strategies can be represented by hierarchical systems in which each agent adopts a particular policy if some team strategy is selected. E.g. in Tambe et al. (1999) this is done by having different agent roles (policies).

Particular problems where navigation of agents play an important role can efficiently make use of global world models. These global world models are shared by all agents which are able to read global state information from the models and update the model according to their individual experiences. E.g. in collective map building (Yamauchi et al., 1998), the task requires building a global world model. Also in robot soccer, a global world model may play an important role to deal with delayed sensory information, uncertainty in observations and goal-directed navigation purposes. The global world model does not say anything about how agent policies are used. If global world models are used this means that the agents use state information through the global world models to select an action instead of directly using the sensory observation. Therefore global world models could also be combined with communication or other architectures.

Finally, communication can be useful for all problems which require coordination or trading. Since communication can play an important role for many different problems, more research should be spend on developing general communication protocols or on evolving communication languages from interaction between agents. Steels (1997) describes an approach to evolve a language between two robot agents. It would also be interesting to compare hierarchical systems to communication for coordinating agents in different problems.

3 Optimization algorithms

In this section we will summarize different algorithms which are useful for optimizing MASs. Optimizing a MAS requires that all agents optimize their own behavior (microlearning). This is usually done by evaluating a decision policy (solution), changing the policy in some way, and evaluating the new solution again (keeping the best policy found so far). Optimization methods can be population-based or not. E.g., reinforcement learning (RL) uses a single stochastic policy (the active policy) for each agent to select an action at some time-step and tries to improve this active policy based on the generated experience. Genetic algorithms on the other hand keep a population of policies, evaluate them all and then make a new population based on the previously evaluated population.

Optimizing a MAS requires *microlearning* and *macrolearning*. Microlearning is used to optimize the policies of agents given their reward function, the inputs they receive from the environment and the actions they can select. Macrolearning is used to optimize the global system behavior and is used to optimize aspects of the system which are outside the limited scope of single agents. E.g. we could think of optimizing a MAS architecture, automatically selecting reward functions, or selecting agents which can communicate with each other as possibilities for macrolearning. Thus, microlearning is done on the agent-level and macrolearning is done on the group- or system-level.

Microlearning. In a MAS microlearning refers to the optimization of each single agent's behavior. If all agents use microlearning, we would like to see the behavior of the global system to improve as well. However, this is not always the case. Agents act in their own interests, they only optimize their own reward (evaluation, objective) function and by improving their own sum of rewards, the system as a whole may degrade in performance. As mentioned before, a good example of this is the tragedy of the commons (TOC), where self interested rational agents cause a collapse of the system as a whole.

Macrolearning. To improve the global system behavior, macrolearning can be used. Macrolearning may optimize coalition (cooperation) structures by changing the architecture. It may also learn social conventions so that some behaviors are considered forbidden and agent's are not allowed to settle to them. Furthermore, macrolearning may change single agent reward functions so that when an agent optimizes its own rewards, this is for the global good. In most current systems, macrolearning is not used since the architecture, local reward functions etc. are considered to be fixed. Still (Wolpert et al., 1999b) make use of it in their collective intelligence (COIN) framework, and it may have an important function for the optimization of the whole system. E.g., one important task for macrolearning is coalition formation. Given a set of tasks which have to be carried out by a number of agents, agents can form coalitions to solve particular groups of tasks together. Since there are an exponential number of ways that tasks can be mapped to agents, this requires an optimization module. Finally, in some problems the system has to discover how many agents should be used to solve a particular task (e.g. in forest fire fighting). This also requires some macrolearning to map environmental states to the number of agents (and which agents in case of heterogeneous agents).

In the following we will consider different optimization algorithms and mention how they can be used as microlearning and macrolearning algorithms.

3.1 Reinforcement learning

Reinforcement learning (RL) algorithms (Sutton, 1988; Watkins, 1989; Bertsekas and Tsitsiklis, 1996; Kaelbling et al., 1996) share the goal to optimize the behavior of an agent by learning from the feedback information acquired during the interaction with the environment. Traditionally RL has been used to optimize the behavior of single agent systems. By interacting with the environment, policies are strengthened (reinforced) when they lead to higher long term rewards of the agent. By exploring different policies, the agent may effectively learn which policy leads to the largest reward sums. Successful single agent RL applications are Samuel's checker playing program (1959) and Tesauro's TD-Gammon (1992), a program which trained a neural network to play backgammon at human expert level from self-play.

The problem environment. The problem environment is usually modeled as a Markov decision problem (MDP) containing a state space, a set of possible actions for the agent, a transition rule which determines the new state of the environment given the current state and the selected action of the agent, and a reward function which determines how much reward the agent receives when making a particular transition to a new environmental state (Bellman, 1961). MDPs may be extended to multi-agent MDPs (MMDPs) (Boutilier, 1998) by augmenting the transition function and reward function so that they deal with a set of actions instead of single actions.

Reinforcement learning as microlearning. RL can be applied to optimize the behavior of an agent in a MAS. The agent receives inputs from the environment and possibly from other agents and chooses an action. Then the environment changes and the agent receives feedback from its reward function which tells how much reward it obtains for the last environmental transition. The agent uses this feedback to learn a value function which estimates the long term rewards the agent will receive when it is in a particular state (represented by its environmental inputs and its internal state). Algorithms for learning this value function are Q-learning (Watkins, 1989; Watkins and Dayan, 1992), TD(λ)-learning (Sutton, 1988), and model-based RL (Moore and Atkeson, 1993; Prescott, 1994; Barto et al., 1995; Wiering, 1999).

Storing the value function. When the state space is small, the agent can store the value function in a lookup table, which lists all possible states and their values. However, for high dimensional or continuous state spaces, function approximators should be used such as neural networks (Rumelhart et al., 1986; Tesauro, 1992), neural gas (Fritzke, 1994; Sałustowicz et al., 1997), CMACs (Albus, 1975;

Sutton, 1996; Wiering et al., 1998) or decision trees (Chapman and Kaelbling, 1991; McCallum, 1995). Another big advantage of using function approximators is that they allow for generalization so that not all states have to be visited and the policy space is reduced.

Macrolearning with RL. RL can also be used to alter the global system which provides it a way to coordinate agents. E.g. RL can be used to find coalition structures determining which agents work together to solve some tasks and which policies they have to use for this cooperation. Furthermore, it can be used to learn communication mechanisms between agents such as communication protocols (a language). Another way of using RL as macrolearning algorithm is to let it learn evaluation functions for combinatorial optimization problems. Often, the performances of algorithms searching for solutions can be improved by slight modifications of the evaluation function, since this can make the landscape less flat, thereby biasing the algorithms to search in particular useful directions. Boyan and Moore (1997) used RL to learn predictions with linear networks to improve such combinatorial optimization search.

Applications of RL to MASs. A number of RL systems which have been developed to solve particular problems are mentioned here shortly.

- **Using Q-learning for microlearning.** Crites and Barto (1996) used Q-learning to train a MAS to control elevator dispatchers. A simulated building contains four elevators and each elevator is controlled by its own local policy. Using Q-learning they were able to train neural networks to learn a good value function mapping environmental inputs to elevator actions. Their system outperformed a number of conventional elevator dispatchers. The architecture used local agents and there was no macrolearning involved.

  Littman and Boyan used Q-learning to learn to route packages on a network. Each network node (agent) used a policy to decide to which neighbor a processed package with a particular destination address should be sent. The system used local agents and there was no macrolearning involved.

- **The Ant Colony System.** The ant colony system (ACS) (Dorigo, 1992; Dorigo et al., 1996; Dorigo and Gambardella, 1997; Dorigo and Di Caro, 1999) uses ants (agents) to create solutions to problems which are used to update the global shared policy. E.g., for the Traveling Salesman problem (TSP), each ant makes a tour while updating the pheremone trail so that the best found tours get reinforced most. The ACS has already been shown to find good solutions to the TSP (Dorigo et al., 1996), the quadratic assignment problem (QAP) (Gambardella et al., 1997), and has also been used to find routing policies on the Internet (Di Caro and Dorigo, 1998a; Di Caro and Dorigo, 1998b) outperforming many other algorithms. Note that this algorithm can be used for microlearning or macrolearning. It uses a parallel search of the solution space and clever exploration methods to be able to enhance exploration possibilities for new solutions.

- **Collective Intelligence (COIN).** COIN (Wolpert et al., 1999b) is devoted to the question of how to set up a RL system which can optimize the collective behavior when the system is put in use. Therefore it tries to find local reward functions for agents which when optimized lead to an optimization of the group behavior. In this way COIN is interested in reverse engineering: given a problem instance how do we set up the local reward functions? COIN's idea is to construct a wonderful life utility function. An agent does not get the global reward (the reward the whole team gets), since this has the problem of agent credit assignment (Versino and Gambardella, 1997) which says that it is hard to evaluate the contribution of each single agent. The local reward function is also not used, since it may result in competitive agents. Instead an agent receives the global reward minus the global reward which would have been obtained if the agent did not do anything at all (or a default action). In this way its own contribution to the global reward is measured. COIN has already been successfully tested on El Farol's Bar problem (Wolpert et al., 1999b) and an Internet routing problem (Wolpert et al., 1999a). COIN also addresses macrolearning aspects in (Wolpert et al., 1999b).

3.2 Evolutionary computation

Evolutionary computation (EC) is inspired on natural evolutionary models in order to find solutions to control, function optimization, and combinatorial optimization problems. Examples of EC algorithms are genetic algorithms (Holland, 1975; Goldberg, 1989), evolutionary strategies (Rechenberg, 1971; Rechenberg, 1989), genetic programming (Koza, 1992), SANE (Moriarty and Miikkulainen, 1996), and PIPE (Sałustowicz and Schmidhuber, 1997). These population-based optimization algorithms make use of a population of agents which allows for a parallel search of the solution space. Each time-step, the current population (generation) is evaluated after which the best individuals get the largest probability for propagating their genes (parts of their policy or solution) to the next population. This propagation is usually done by crossover (two solutions are combined to form a new solution) or mutation (some part of the solution is randomly changed). By selecting the best individuals and throwing out the worst, the population will consist more and more of good individuals of which finally the best is kept as the solution. When applied to navigation, EC is usually combined with a set of behaviors for an agent such as wall-following, random walk, homing etc. This makes the task of learning to control an agent much easier.

EC for microlearning. These algorithms can be used as microlearning methods for optimizing agents in MASs by presenting each agent a population of policies which are adapted and from which the agent finally has to pick the best one. Each time step each agent of the MAS selects one of the members (policies) in their own population, and the whole system is evaluated. Then the fitness of each member in the system is updated according to the behavior of the complete system. A problem of this is that the evaluation of a population member (policy) inside an agent depends heavily on the policies selected by other agents, so that the agent credit assignment problem (ACAP) (Versino and Gambardella, 1997) is difficult to solve. Ways to circumvent this is to use a member inside multiple, different agent-teams to obtain multiple team-evaluations. Then these team-evaluations can be averaged to obtain an evaluation of the individual. This is effectively done by the SANE system described in (Moriarty and Miikkulainen, 1996) which learns a neural network to solve RL problems such as learning to play Othello (Moriarty and Miikkulainen, 1995) or to learn cooperative lane selection policies for cars driving on the highway (Moriarty and Langley, 1998). The latter multi-agent application showed promising results for optimizing traffic flow using machine learning techniques. The SANE system combines different neurons (individuals) inside a set of neural nets and evaluates the neural nets. The neurons which are part of the best performing neural networks are kept and used to create a population of new individuals.

Using macrolearning. Another solution to the agent credit assignment problem is to use macrolearning instead of microlearning. Instead of learning an individual controller for each agent, we can also have a team of controllers as a single individual in a population and thus we store and evolve a population of teams. In this way, complete teams are evaluated which results in a much more accurate evaluation signal. Note, however, that a system in which the whole global system is optimized at the same time resembles a single agent architecture. Hence, it has the problem that each individual has to contain the policies of each agent, and therefore the search space can become prohibitively large (we search for a population of policies inside a population of populations, which is very complex). Just like RL, EC can also be used to evolve coalition structures, reward functions, etc.

3.3 Local iterative search

There also exist a number of single agent algorithms for solving function optimization or combinatorial optimization problems. Examples of these methods are Tabu search (Taillard, 1990; Glover and Laguna, 1997), multiple restarts with local hill-climbing (Colorni et al., 1993), and simulated annealing (Aarts and Korst, 1988).

Tabu search (TS) works as follows (see (Glover and Laguna, 1997) for a more thorough review). A random policy (solution) is generated after which TS tries out all possible policy-changes (by assigning a different action to one of the states). After this, the best policy-change is executed and a new solution is generated. In this way the method is a steepest-descent method. In order not to cycle between solutions, a list of forbidden policy-changes (the Tabu list) is kept. Each time a policy-change is executed, it is placed inside the Tabu list and cannot be executed again for a number of time steps. The Tabu list has a specific length and thus new forbidden policy-changes replace other policy changes when the list is full. When the system observes it is in a (local) minimum, the solution is stored and the system escapes the local minimum by executing a series of random policy changes. Tabu search has been shown to be very effective in solving difficult problems such as the quadratic assignment problem (QAP) (Colorni et al., 1993), vehicle routing problems (Badeau et al., 1997), and job-shop scheduling problems (Taillard, 1994), all problems which can be modeled as multi-agent problems.

Colorni et al.'s Algodesk (Colorni et al., 1993) compares the performances of 8 different optimization methods such as simulated annealing, Tabu search, genetic algorithms, multiple restarts on a set of quadratic assignment problems. They found that using Tabu search and simulated annealing leads to the best results.

Local search as microlearning. Using local search may not be an efficient way for microlearning. When we want to optimize a controller, the policy maps many inputs to many actions. If we make one alteration to such policies and evaluate the policy, it may not reach the goal, since usually such policies are deterministic and become easily trapped in cycles. One way to circumvent that problem is to use stochastic policies which are tested or to use complete behaviors as actions. Still, there are as many as $|A|^{|S|}$ policies (with $|A|$ as the number of actions, and $|S|$ the number of states) so that searching through this space can take a long time with local search methods which only make single policy changes at a time and do not adjust other action values according to the interaction with the environment. Littman (1994) used branch and bound methods to optimize an agent in a partially observable environment and obtained good results, although for such environments the number of inputs can be quite small. For solving Markov decision problems it may be more reliable to use RL. Local search can be used in MASs by changing one individual policy at a time, which leads to quite accurate evaluation signals since we evaluate our single change (maybe using Monte Carlo experiments for stochastic environments), but this requires a long time. We can also change multiple agents synchronously and keep the resulting MAS if its behavior is better than the previous one (which may not work well if the environment is stochastic). For some multi-agent problems containing few states, local search may be useful, especially for scheduling problems in factories which are usually static, fully observable, and for which solutions can be quickly evaluated. For difficult control problems with many input states, it may be very slow, however.

Local search as macrolearning. Local search methods can also be used to find good coalition structures mapping tasks to agents and ensuring that particular agents cooperatively solve their tasks. It can generate a CS, test it by using some algorithm as microlearning method, change the CS by changing the assignment of tasks to agents, and evaluate it again. This also resembles how coalition structures can be constructed in game theory (Sandholm, 1996).

3.4 Game theory

In cases where self-interested agents optimize their own reward functions, interaction problems between agents arise. E.g., when agents do not have entirely positive or negative correlated reward functions, agents may help or hinder each other pursuing their own goals and the need to cooperate between agents may arise. In multi-agent systems, the agents can be provided with an interaction protocol, but each agent will still choose its own strategy. The main question is what social outcomes follow given a protocol which guarantees that each agent's desired local strategy is best for that agent - and thus the agent will use it (Sandholm, 1999).

Game theory is interesting for studying the dynamics resulting from having agents interact with the same environment, for designing interaction protocols, and provides us with optimization algorithms in the form of auctions, markets, and voting schemes. The simplest environments are single step decision problems where the reward which each agent obtains is given by a payoff matrix. The payoff matrix returns the rewards each agent (player) receives given its action and the actions of all other players. In the simplest case, the environment is given only by the actions of other players.

Prisoner's Dilemma. A famous example is the prisoner's dilemma (PD) where agents can decide upon a single action: to cooperate (C) or defect (D) with the other agent. The payoff matrix is shown in figure 4. Since defecting is a *dominant strategy* (whatever the other agent does, the best choice is to defect), a rational agent would choose to defect, resulting in a non-cooperating game. The largest payoff can be obtained if both agents choose to cooperate (which is a *a Pareto optimal solution* — a solution in which players cannot change their action to receive higher reward without other players receiving less reward), and hence comes the dilemma. For such problems, agents can be put repeatedly for the same choice, and cooperation may emerge from this interaction. By studying repeated games (fictitious play), given the previous history of choices of both players, the question is how should the agent behave to maximize its future cumulative payoff? Although the defect-defect situation is a *Nash-equilibrium* (given the other players' strategies, no agent will benefit from changing strategies), in iterative games (games which are repeatedly played) the players can explore different possibilities and learn to cooperate and in this way optimize their cumulative payoffs.

	C	D
C	(3,3)	(0,5)
D	(5,0)	(1,1)

Figure 4: *The payoff matrix for the prisoner's dilemma. The matrix should be read as follows: if agent 1 makes decision C and agent 2 decides D, then agent 1 receives reward 0 and agent 2 a reward of 5.*

Game theory and microlearning. Game theory can use two different agents: (1) the rational agent which contains sufficient problem solving knowledge and planning methods to solve a problem, and (2) the reinforcement learner which learns from its interaction with other agents. The reinforcement learner can optimize its behavior in iterated plays (repeated games) with other players in which the agent can learn from the history of moves. E.g., Sandholm (1995) studied how Q-learning can be used to learn the cooperating policy for the Iterated PD (IPD). For this he used lookup tables and recurrent neural networks which were enabled to base their decision policy on the history of previous moves. He found that both systems can learn to cooperate (C-C) with Tit-for-Tat (Axelrod, 1984), an algorithm which starts by cooperating and then uses the previous choice of the opponent as its current move. If the two RL systems play against each other different interactions arise, however, such as alternating C-C and C-D moves. Fogel (1993) uses evolutionary computation to evolve behaviors of agents in the IPD. Brafman and Tennenholtz (1996) study how a RL teacher can learn to teach a student to cooperate in the IPD. They find that a non-adaptive Tit-for-Tat teacher works well and that a RL teacher can be used, but does not always lead to reliable outcomes. Furthermore they study how a cooperative organization can be stable against parasites which may for example work on the same work-floor, but refuse to bring common tools back to their original position. To make the organization stable, penalty agents are constructed which defect against parasites so that a rational parasite will profit more by cooperating as well.

Coordination. For particular games, coordination is needed. E.g., assume the game is a risk-dominant game given by the payoff matrix shown in figure 5(A).

	L	R
L	(2,2)	(-100,0)
R	(0,-100)	(1,1)

	L	R
L	(0,0)	(-5,-5)
R	(-5,-5)	(0,0)

Figure 5: *(A) The payoff matrix for the risk-dominant game. (B) The payoff matrix for the coordination game.*

Here there are two Nash-equilibria: L-L and R-R. Clearly, the Pareto optimal solution is to play L-L, but since choosing L can result in a large loss, R is a *risk-dominant strategy*. Hence, coordination is needed to converge to L-L. An example of coordination is to use communication, e.g., both agents may communicate before making their moves that they will choose L. In complex problems, communication may also be useful to coordinate the activities of different agents. E.g. in soccer, we may want to learn a pass-play strategy in which one agent plays the ball to another agent, runs forward, and receives the ball back. This can be done by communicating the planned actions from one agent to another one. Another game is shown in figure 5(B) in which there are two Nash-equilibria. The problem is which one to chose. Note that this games resembles driving behavior: cars can drive on the left or on the right side of the street. A way to solve this problem is to use social conventions (such as people should drive on the right side) which prune away choices (behaviors) of the agents.

Tragedy of the commons. Another game theoretical problem is the TOC. Here, agents share the same resource and follow their own reward function. By increasing their exploitation of the resource, agents receive higher rewards, and this makes it a rational choice. However, if all agents keep on increasingly exploiting the bounded resource, problems may arise. Take as examples overfishing which has caused the enormous fish population to shrink considerably or environmental pollution. In MASs the TOC can happen in many different ways. Take as example autonomous underwater vehicles (AUVs) which can use radars to sense their environments. If AUVs use their sensors more, they can get a better picture of their environment. The problem is that the use of radars in water causes interference between radar signals, resulting in more uncertainty in the returned information (Turner and Turner, 1998). Methods to deal with such problems use communication, mutual-coercion mutually agreed upon, privatization of resources, social laws, or management agencies to keep the resources from starving out (Turner, 1993).

Negotiation for task decomposition. An important subfield of game theory which deviates from the matrix-games is negotiation between self-interested agents (Sandholm, 1996; Sandholm, 1999; Zlotkin and Rosenschein, 1996). Negotiation may be used as a mechanism for assigning tasks to agents, for allocating resources, and for electronic trading. For this, algorithms from economical theory such as bargaining and auction theory may be used. Bargaining refers to seller/buyer situations in which one agent sells an item to another agent and gets paid for this. Auction theory refers to selling items to a set of buyers so that the buyer which offers most gets the item. A problem which can be solved by bargaining and negotiations is a transportation problem modeled as a vehicle routing problem. There are a number of depots which store goods and a number of customers which have ordered a number of goods. Transportation vehicles are used to deliver the goods to the consumers after which they return to the depot. The goal of the system is to minimize the total distance traveled by all vehicles with the constraint that the vehicles have sufficient capacity to carry all goods to the consumers. In (Sandholm, 1996) negotiation mechanism are described to solve this problem. Here, agent negotiate in order to form coalition structures determining which agents should deliver to which consumers. During this negotiation, agents may pay other agents for solving some task of theirs. In this way, the sum of costs for all agents can be minimized, since agents only handle tasks for other agents if they profit, and agents never pay more to other agents than what would have been their own cost for executing the task. Andersson and Sandholm (1998) discuss using different contracts which can be used for transferring one task among agents (O-contracts), transferring more than 1 task from one agent to another one (C-contracts), having 2 agents swap tasks (S-contracts), or having multiple tasks changed among multiple agents (M-contracts). The trade stops once no more contracts are made and the agents can start solving their own tasks. They found out that if the ratio agents to tasks is great O-contracts work best and if this ratio is small C-contracts work best.

The way such a system looks for CSs resembles Tabu search. Tabu search tries out all swap-moves (contracts) and keeps the best one. The difference is that Sandholm (1999) allows for self-interested agents and deals with this by using money transfer while making contracts. Furthermore, contracts come to agents all the time (but not all at the same time), and therefore timing becomes essential for agents: they may want to postpone a contract to see whether other, better contracts are coming, but cannot wait too long. In this way the system could be much more complicated than just using simple swap-moves in a cooperative system.

Negotiation for multi-agent plans. Zlotkin and Rosenschein (1996) study negotiation as a means to determine multi-agent plans. The question they pose is how interaction rules can be designed (mechanism design) so that agents agree on mutually beneficial behavior. Each agent has a goal (set of goal states) which it wants to reach and assigns a specific reward (worth) to achieving its goal. Furthermore each agent can execute a plan to achieve its goal with a specific cost. Agents may also carry out joint plans, which may lead to the goals of all agents being satisfied. Rational agents will only carry out a plan if the reward of reaching the final state is larger than the cost of the plan. In general they want to maximize their utility (their reward of the goal minus the cost of the plan). If we consider two agent systems, agents can decide to pursue their goals by an individual plan, they can cooperate in a joint plan or they can do nothing (since the costs of each plan is larger than its reward). In some cases, joint plans (even partial joint plans which do not lead to any goal), are preferable over individual plans. In such cases, the joint plan is carried out and according to the rewards of goals, agents can divide the work. Even if goals are not both satisfiable (they contradict each other), agent may gain by carrying out a joint plan. E.g., consider a block-world problem in which the goal states of two agents are contradicting, but both goal states have in common that a proportion of the same work has to be done to satisfy them. In this case, the agents can start by cooperatively carrying out the plan which has lower total cost than if an agent would do the work alone, and then they toss a coin to decide which goal will be satisfied. In this way both agents profit on average. To cope with such different situations, agents may choose between pure plans (without a probabilistic component), mixed joint plans (with a probabilistic component to divide work according to goal rewards), semi-cooperative deals (agents carry out a subtask and profit mutually), and multi plan deals which implement post-flip cooperation in which a coin is flipped and the outcome decides whether agents cooperatively execute a joint plan which leads to the goal for agent 1 or the goal of agent 2.

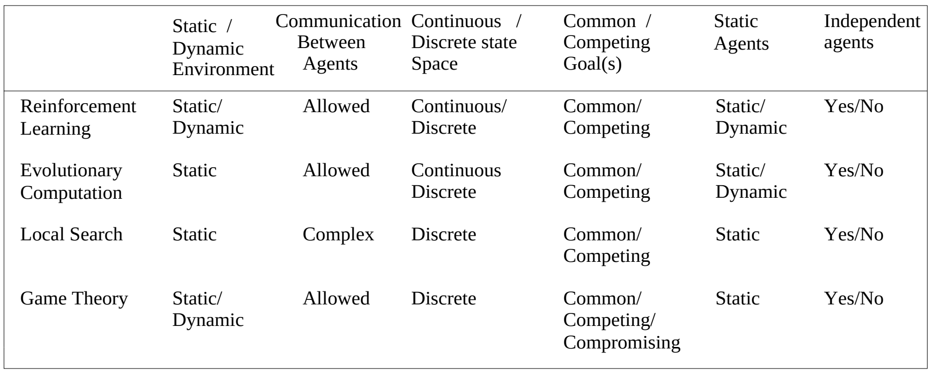
Vickrey auction. When we want to maximize social welfare (the sum of the utilities of all agents) when dealing with multiple agents, it is important that truth-telling of the agents is a dominant strategy. E.g., in some interaction schemes an agent may prefer to lie in order to get paid a higher amount for doing some task or to do less work for solving some task. If agents start lying, the best contracts could be disregarded, leading to a worse social welfare. Therefore, mechanisms have been designed which promote truth-telling as a dominant strategy. One of the mechanisms in auction-theory is the Vickrey (second price, sealed-bid) auction (see Sandholm (1999) for a list of other auctions), in which agents make a bid for some object, and the agent which had offered the largest payment, gets the object for the price of the second bid. Here, truth-telling of the real monetary value of the object is a dominant strategy in single-shot private value auctions with risk neutral bidders, a truthful auctioneer, and no possibility of colluding. The main argument is that if an agent would say that she values the object less, the price would be the same for getting the object, whereas the agent may also loose the bidding, thus not gaining the profit of the difference between its worth of the object and the second largest bid. If the agent would say a higher amount, the agent may win the competition and pay a larger amount than its worth of the object. Thus, the best is to tell the truth. Problems with the Vickrey auction are that buyers may make coalitions (all can cooperate in mentioning a low value so that the winner would have to pay less), or the contractor (the market) may lie: since the second bid is unknown, the contractor could say that its price equaled the winning bid. These subjects are important when dealing with self-interested agents and electronic marketing (Sandholm, 1996). Another mechanism for truth-telling is the Vickrey-Clarke-Gloves Tax (see Appendix A).

3.5 Mapping problem types to algorithms

RL and EC are algorithms which can be used effectively for microlearning, e.g. for learning to control an agent. They could also be used for macrolearning as we have seen. On the other hand, game theory and local iterative algorithms are more useful for macrolearning. They can be used to find coalition structures, task allocations, and schedules.

There are particular differences between algorithms such as Tabu search (TS) and adaptive algorithms such as RL. An important difference is that TS computes and evaluates a complete solution (e.g., a job-shop schedule) and tries to find a better solution by trying a set of changes to the solution, evaluating the resulting solutions and keeping the best one. Usually, TS works very well if evaluating a solution can be very fast and precise, what is the case for standard problems like the traveling salesman problem, but which is not the case for very stochastic problems such as network routing problems. In contrast, RL algorithms incrementally improve their solutions by using feedback information acquired during the interaction with the environment. Using this feedback, they make small changes to a (stochastic) policy. Since these methods rely on stochastic decision policies and evaluate the policy in many different situations for which it should work well, they are disturbed less by the stochasticity inside the problem environment.

We would also like to know when to apply a particular optimization algorithm. For this we made a mapping from problem features to optimization algorithms.



	Static / Dynamic Environment	Communication Between Agents	Continuous / Discrete state Space	Common / Competing Goal(s)	Static Agents	Independent agents
Reinforcement Learning	Static/ Dynamic	Allowed	Continuous/ Discrete	Common/ Competing	Static/ Dynamic	Yes/No
Evolutionary Computation	Static	Allowed	Continuous Discrete	Common/ Competing	Static/ Dynamic	Yes/No
Local Search	Static	Complex	Discrete	Common/ Competing	Static	Yes/No
Game Theory	Static/ Dynamic	Allowed	Discrete	Common/ Competing/ Compromising	Static	Yes/No

Figure 6: *Matching problem features to optimization algorithms. Table entries denote for which problem features the algorithm is suitable.*

In figure 6, we show that RL can handle all kinds of environments. Evolutionary computation cannot handle highly dynamic environments well, since when we are evaluating population members inside a population, we require the problems the agents are solving to be the same and in non-stationary environments this is not the case. For slowly changing environments, both methods will suffer from the same problems and can use the same methods to deal with it. E.g. when the environmental change in cyclic, both methods could track changes of the environment and accordingly switch to new policies. Local search suffers from the same problem as EC, since highly dynamic environments make it difficult to evaluate a particular change of the policy. Although there are no theoretical guarantees that RL converges to stable policies for non-stationary environment, they have the advantage that they rely on a single decision policy which could in principle use its own interaction with the environment to track the "environmental drift".

Furthermore, local search may have difficulties with communication, since there is no standard way of evolving communication acts which may also require function approximators. They also have problems with continuous environments, since in this case it is hard to make single policy changes. Finally, it often assumes that agents are static, since navigating an agent in some environment demands some kind of behaviors or controller. Game theory can not be used for continuous state spaces,[2] since in this case negotiations between agents can be quite troublesome (although prices for trading could be continuous, in general settings prices are sliced into discrete values, see e.g. Vidal and Durfee, 1998). Finally, game theory cannot be used for navigating dynamic agents, since it does not provide tools for this. It does provide useful tools for designing and analyzing communication, and could also be used for dynamic environments since all agents constantly remain inside the environment.

Although we have not mentioned it before, partial observability of the environment may also be an interesting feature for choosing between RL and EC. Although (Moriarty et al., 1999) discuss a comparison between an EC method to Q-learning, and show that Q-learning does not work for non-Markovian problems, TD(1) approaches based on Monte Carlo simulations come much closer in spirit to EC algorithms and do not suffer that much when evaluating a policy (Wiering and Schmidhuber, 1997). The most interesting difference therefore may be that EC algorithms usually solve the structural and temporal credit assignment at the same time, e.g. by evolving a recurrent neural network. The question then remains whether similar things cannot be done using RL.

4 MAS problems

In this section we will review a set of problems and research work to solve them.

4.1 Network routing

Network routing problems consist of a set of nodes connected in a network by transmission edges. At each time-step a set of packages are transmitted to the nodes, and these are transported over the network until they reach their destination address. The goal is to send packages to those neighboring nodes which will minimize the overall traveling time. For decision making, the destination address of the package and information which describes how busy neighboring nodes are may be used. A problem which resembles network routing is traffic control where we have to switch traffic lights in order to minimize the average waiting time for the cars. Algorithms for solving network routing problems are the ant-system (Di Caro and Dorigo, 1998a; Di Caro and Dorigo, 1998b), Q-routing (Littman and Boyan, 1993), predictive Q-routing (Choi and Yeung, 1995) and TPOT-RL (Stone and Veloso, 1999).

4.2 Stock supply

In stock supply problems, we have a set of vehicles which drive around in an environment and have the goal to fetch objects and transport these to their destination address. Note that the big difference with network routing problems exists in the fact that in stock supply problems the vehicles are physical objects driving around in an environment which makes navigation and path-planning issues important problems. There are static stock supply problems such as delivering fuel from a fuel deposit to a set of fuel stations with the goal to minimize the overall traveling distance and these can be modeled by vehicle routing problems where we list the set of addresses to which fuel has to be sent, and evaluate whether all constraints are fulfilled (such as tank-vehicle i can carry sufficient fuel to supply tank stations x_1, x_4, and x_6). For such problems the ACS, EC, or TS can be used. There are also very dynamic problems such as stock supply in a supermarket where requested items should be taken of the shelves and a continuous stream of items can be requested.

[2]Although we always have the option to discretize the space first.

For such problems we may prefer to use RL to learn reactive policies which can immediately react to changes to the state of the environment. Another example of a stock supply problem is elevator dispatching where the goal is to bring a set of passengers in an elevator building to their destination addresses using multiple elevators. An RL algorithm for solving this problem is described in (Crites and Barto, 1996). Schneider et al. (1999) describe different RL methods using different reward rules for distributed control of a simulated power grid. The goal is to direct power from a set of producers through a number of distributors to a set of consumers (cities). The comparison between four different local learning rules (using global reward, local reward, local reward and rewards of neighboring nodes, and local reward and value functions of neighboring nodes) shows that learning a local value function which is not only based on the local reward but also on value functions of neighboring nodes is the most promising variant.

4.3 Explorative detection problems

In explorative detection problems (distributed measurement) the goal is to generate a set of trajectories in order to maximize the probability that an event of interest will be detected. Examples are forest fire detection (Kourtz, 1994) where the goal is to generate a set of airplane trajectories over a 2-dimensional discretized map of a forest with probabilities that in each region a forest fire may have started. Other problems are pollution detection (detect a pollution belt in the air), land-mine detection and food foraging.

Massios and Voorbraak (1999) describe an application of decision theoretic planning for surveillance robots. The goal is to minimize the cost of particular events such as an undetected fire or flooding in a room of a building. For this, the agent has to find out in which order to visit rooms. Although the present application involves a single agent it could easily be generalized to a multi-agent problem.

Ye and Tsotsos (1997) consider a collective search task for a team of robots. In a 3D environment, some target object is hidden and the goal of the team is to cooperatively explore the environment in order to detect the object as soon as possible. Finding trajectories which maximize the probability of detecting the object is a NP-hard problem. Their proposed method works with a probability grid representing the probability that each of the discrete cells contains the target. By navigating through the environment and sensing the space with cameras (which are not accurate enough to be able to always detect the object when it is in the surroundings), the probability grid is updated and this influences the search. It is also important to coordinate the agents, since it may happen that all agents believe that the target object is in some area and will cluster in the same working space. Therefore a term for spreading the agents over the environment is used while planning agent movements.

4.4 Transportation problems

Transportation problems are common real world problems. Examples are harbor container problems (Gambardella et al., 1998) where we have a dock where containers are loaded and unloaded from ships with their specific destination addresses and a big yard (depot) is used for stocking containers which should be shipped to some destination. There are a set of cranes which can be used for loading and unloading ships and for building container piles on the container yard and a set of transport trains which carry the containers between the quay and the container yard where they have to be loaded/unloaded. The goal is to find efficient schedules for the cranes storing the containers in the yard, for loading and unloading the ships, and for the trains transporting containers between the yard and quay so that the cost of the whole operation is minimized. Particular problems which have to be solved are the following: containers which leave quickly should not be stored under a large set of containers which should then be moved on the yard to different places which would cost a lot of time. Other problems involve deadlock situations in which two cranes are assigned to the same area of the yard. Another transportation problem is the train container problem where trains are used to carry containers to their destination addresses and for which containers have to be transported on railway stations to change trains.

4.5 Traffic light control

In traffic light control, we have an environment consisting of an infrastructure, cars, and traffic lights. The cars drive over the roads and have a particular destination address. Traffic lights are modeled as agents which can switch between red and green signals. The goal of the system is to minimize the overall waiting time in front of the traffic lights which also minimizes traveling times. Thorpe and Anderson (1996) use SARSA (Sutton, 1996) with replacing eligibility traces (Singh and Sutton, 1996) to learn traffic controllers on a simulated traffic control problem consisting of a network of 4×4 traffic light controllers modeled in a grid. They modeled average speed, queueing and acceleration/ deceleration of cars. They optimized the traffic controller on a single intersection after which they copied it to the other intersections (so they use policy sharing). Results showed that using their best state representation (which indicated which partitions of each road segment contained cars) they were able to outperform an algorithm which used fixed waiting times or an algorithm which allowed the largest queue to go first. This latter algorithm suffers from switching lights too often on crowded traffic nodes which costs a lot of time, since cars need to decelerate and accelerate again and again.

Taale et al. (1998) compare using evolutionary algorithms (a (μ, λ) evolution strategy) to evolve a traffic light controller for a single simulated intersection to using the traditional traffic light controller used in the Netherlands (the RWS C-controller). They found comparable results for both systems.

4.6 Multi-machine scheduling

In multi-machine scheduling, jobs have to be assigned to machines so that certain performance demands like time and cost effectiveness are optimized. Brauer and Weiss (1997) discuss a RL approach using local agents to optimize the global behavior of the system. They consider two types of time constraints: the time that one machine needs for completing its production step and the time needed for transferring one job from one machine to a successor (the machines are represented by a network in which each job goes through a machine at all levels). By learning an estimation of the remaining time for transferring one job from one machine to each successor machine, the successor machine can be chosen leading to the lowest expected cost. Their algorithm outperformed a random assignment policy and a method which assigns a job to the machine with the lowest cost for a one-step lookahead step. It can also re-adapt itself to changes in the environment, e.g. when a machine breaks down.

4.7 Quadratic assignment problem

The quadratic assignment problem (QAP) provides (like many other NP-complete problems) a suitable model for many real-world problems such as: campus planning, typewriter keyboard design, hospital layout, minimizing average job completion in machine scheduling and others (Colorni et al., 1993). In a QAP there is a flow matrix and a cost matrix, e.g. the average number of times the director visits the secretary and the distance between their rooms. The goal is to place all objects in such a way to minimize the product of the flow and cost matrices. A number of algorithms can be used to solve the QAP such as Tabu search, simulated annealing, genetic algorithms, immune networks, sampling and clustering, the ant colony system and others. These algorithms can be used to find good solutions to difficult QAP problems as shown in (Colorni et al., 1993; Gambardella et al., 1997).

4.8 Robot soccer

In robot soccer, the goal is to find team soccer strategies. Each agent follows a particular action selection policy and the goal is to optimize that policy so that the team will win against other teams with the largest possible score difference. This is a challenging problem combining a dynamic environment, cooperation, competition and communication. Soccer recently received much attention by various multi-agent researchers (Sahota, 1993; Asada et al., 1994; Littman, 1994a; Stone and Veloso, 1996; Matsubara et al., 1996). Most early research focused on physical coordination of soccer playing robots (Sahota, 1993; Asada et al., 1994). There also have been attempts at *learning* low-level cooperation tasks such as pass play (Stone and Veloso, 1996; Matsubara et al., 1996). Littman (1994) describes a 5×4 grid world with two single opponent players. Luke et al. (1997) describe an approach to learning soccer team strategies in more complex environments. Some novel research on learning soccer strategies is described in (Sałustowicz et al., 1998; Wiering et al., 1998). Stone et al. (1999) describe their soccer team "CMUnited-98" which became the Robocup-98 simulator league champion. CMUnited-98 keeps an accurate global world model representing the positions, velocities of all moving objects and the confidences of these parameters. Then it uses the global world model for local decision making. For this skills are constructed such as: Kicking, Dribbling, Ball Interception, Goal-tending, and Defending. These skills take the predicted world model and predicted effects of future actions into account to determine the optimal primitive action. Furthermore, machine learning techniques are used for choosing the action of the agent possessing the ball. Finally, CMUnited uses clever strategic positioning of the team players using a procedure maximizing a mathematical expression consisting of attraction (to the ball and goal) and repulsion (from passive teammates and opponents) terms.

4.9 Forest fire fighting

In forest fire fighting, the goal is to control forest fires with a set of agents. The forest fire is modeled and we can control a set of bulldozers to cut firelines. These firelines have to ensure that the propagation of the forest fire stops. The goal is to learn optimal strategies for each bulldozer which uses local information in order to create firelines. In (Wiering and Dorigo, 1998) a method is described which has the aim to solve this problem. It relies on using function approximators to solve the local navigation problem and uses RL and cooperative search to learn a policy which dynamically constructs optimal firelines given the current state of the forest fire.

4.10 Cooperative robot tasks

In robotic domains, using multiple robots to speed up a task like object gathering (Mataric, 1994) or to make pushing a heavy box possible (Mataric, 1997) is a natural extension to single agent problem solving methods. With robot teams, new features become possible such as communication, carrying heavy objects with multiple robots, and task specialization. Murciano and Millan (1995) describe a RL method which uses prewired behaviors for a team of agents in which the goal is to learn to signal to other agents where the objects of interest are. E.g., when there is a group of objects clustered together, agents may use light sources to provide other agents with the information that they found a spot from which objects may be gathered. Using this signal-based communication, each agent can learn to become a signaling agent or an exploring agent. The results show that the team can learn to cooperate well. A similar way of communicating messages was done by Mataric (1997), where robots had to collectively collect pucks and robots could send messages to each other which would influence where other robots would go to.

Balch and Arkin (1994) discuss communication in reactive multi-agent robotic systems. The tasks involved are foraging (go and bring objects to the homebase), consuming (go to objects and consume them), and grazing (e.g. lawn mowing). They provide the robots with the ability to send state (which behavior are they currently following) or goal information (which place may be interesting) among agents. The results show that for the foraging and consume task, communication helps to reduce the time needed to find or consume all objects in the environment. For grazing no improvement was found, however. The authors explained that in grazing, there is already implicit communication since agents can observe the lawn and infer where other agents are working.

4.11 Load balancing

In load balancing we want to map a process to a parallel machine. The process which has to be parallelized can be modeled as a network where nodes represent subtasks and edges represent communication between tasks. The goal is to fit the process as well as possible on the machine so that communication overhead is minimized. This is also a NP-complete problem. Schaerf et al. (1995) discuss an approach to adaptive load balancing using reinforcement learning. The problem they consider is a multi-agent multi-resource stochastic system: there is a set of agents, a set of resources with capacities, and probabilistic job sizes. An agent must select a single resource for each job which will process it. If a resource is working on multiple jobs at the same time, it spreads its production activity over all jobs. Thus, the goal is to let agents learn to assign the best resource to each job and to distribute the jobs in the right way. They use a simple RL scheme, where each agent estimates the time each machine needs to process their jobs. By having all agents learn simultaneously, some agents will prefer to use particular resources whereas others prefer other resources. In this way the jobs are spread. The results show good performance of the learning system compared to non-adaptive decision systems, and that the learning system can also effectively handle changing capacities of resources.

4.12 Dynamic channel allocation

In dynamic channel allocation for cellular telephone systems, calls have to be mapped to channels. Each channel can be used for multiple calls, but only if the calls using them are spatially distributed. Singh and Bertsekas (1996) discuss a RL approach to learning an assignment policy for incoming calls. The goal is to learn decision policies which minimize the number of calls which are blocked since no channel is available for them. They use feature extraction, a linear neural network and TD(0) (Sutton, 1988) learning a value function for telephone network configurations. In this way when a call has to be assigned, all next configurations can be evaluated and the one with the lowest estimated number of (future) blocked calls is selected. The algorithm uses a single controller, and the system is therefore not really a MAS, although it could be decentralized as well. The algorithm has been shown to outperform fixed assignment policies and the best heuristical assignment policies on a large cellular system with approximately 70^{49} states.

4.13 Game theoretical problems

Arthur's El Farol Bar problem (Wolpert et al., 1999b) is a fairly difficult problem to solve with multiple agents. There is a bar and a set of agents. Each agent has to select a night for attending the bar. Since the bar may be empty or overcrowded, the agents have to cooperate to maximize the global payoff. E.g., during a night a payoff is obtained of: $xexp(-x/6.0)$ which is maximized when $x = 6$. Therefore if there are 168 agents, the optimal solution is to have 6 agents in the bar on 6 nights and to have the remaining 132 agents in the bar on the last night. Such problems need cooperative agents and good reward functions to learn this solution. E.g., in (Wolpert et al., 1999b), it is described that just learning from the global reward does not work well, but different reward functions such as the wonderful life utility lead to much better results.

4.14 Predator-prey problems

Multi-agent RL has also been used to study predator-pray problems, such as the problem described in (Ono and Fukumoto, 1997; Stone and Veloso, 1997) where 4 predators have to close in a prey on a grid and their actions are to stand still or to go to each of the four directions. The predators need to cooperate whereas the prey has to escape the predators. In (Ono and Fukumoto, 1997) predators use Q-learning and obtain information from the environment using local visual fields. The problem is that since there are a total of 5 agents (including the prey), the number of possible inputs grows exponentially with the number of agents. Therefore the authors use a modular approach in which different modules are used to detect prey and single predator positions within the local fields. In this way, agents can learn to cooperate by locating and following other agents. The experiments show that the team of agents are able to learn cooperative behaviors such as herding (agents stay together which facilitates enclosing the prey), and altruistic behaviors (agents stay close to the prey so that other agents are able to either detect the prey or other agents and in this way the chances to detect and close in the prey increase).

Tan (1993) studies how independent agents behave when compared to cooperative agents. The cooperative agents can communicate their perceptual fields, policies and learning episodes to another agent. They have to solve a 2 predator, 2 prey task (locating and hitting the prey) in a discrete grid-world. The results show that communicating local perceptual fields by a (third) scouting agent helps to reduce the number of steps for catching the prey. Policy sharing and presenting other agents with episodic memory (the learning trial of another agent) also help. Thus, in these ways cooperating communicating agents outperform independent agents.

Schmidhuber and Zhao (1996) study three co-evolving agents which are simultaneously prey and predator. E.g. Agent A tries to catch agent B, but has to escape from agent C. They use the success-story algorithm (Schmidhuber, 1996; Schmidhuber et al., 1997), an algorithm which keeps policy updates if they result in long term reward acceleration. The policy updates are performed by the policy itself, so that the policy is self-modifying. The results show that each agent learns interesting pursuit-evasion behaviors.

5 Conclusion

This paper described multi-agent systems (MASs), systems consisting of multiple individual agents working in the same environment. Since these agents share resources and may have their own individual goals, conflicts may arise when we want to co-optimize the agents. Therefore an important topic in dealing with MASs is to coordinate agents. Coordination can be done using special multi-agent architectures. Different architectures of special interest are global world models, hierarchical systems (management agencies), and explicitly communicating agents. Some of these are already used for multi-agent applications such as global world models for robot soccer (Stone et al., 1999). Most recent multi-agent applications, however, rely on local agent architectures (e.g. Crites and Barto, 1996) or on systems using limited communication possibilities. Since larger applications such as RoboCup rescue, the new RoboCup challenge requiring many agents to cooperatively deal with disaster situations, will require good coordination skills, much more research in developing and testing more advanced architectures is required.

Optimization algorithms for solving multi-agent problems are another important topic for MAS researchers and this topic has received more attention during the last years. In this paper, we made the distinction between microlearning and macrolearning for optimizing MASs. Microlearning is used to optimize the behavior of single agents — a single agent interacts with its environment (including other agents) and learns to maximize its long term reward intake. Macrolearning is used to optimize the global behavior of the system and can be used to optimize the architecture, reward functions, communication protocols, or other global decisions such as choosing how many agents will be used for solving the problem. We discussed using reinforcement learning (RL), evolutionary computation (EC), local iterative search, and game theory. RL and EC can be used for microlearning and macrolearning. Just as in single agent RL, there are tradeoffs whether to use RL or EC (see (Moriarty et al., 1999) for a discussion of relative advantages/disadvantages). In multi-agent RL, problems are often non-Markovian since local agents only perceive partial state information, and non-stationary. For such problems both methods may profit from different, specialized, architectures. Therefore experimental comparisons between both approaches would be helpful to find their relative strengths. Local search and game theory are useful for coordinating multiple agents using either hierarchical systems (local search) or communication (game theory). Furthermore, game theory is interesting for studying the social outcomes of interacting self-interested agents.

We discussed previous research in solving a set of multi-agent problems. For most problems, multi-agent RL with local independent agents have been used. Still, although simple, these approaches have often been shown to be quite effective. In future research, we would like to see the advantages of going beyond local agents. For this, we could use communication, hierarchical systems where agents get roles given global strategies, global world models, and task structures for dividing tasks into sequences of subtasks. It has already been shown effective to expand the state information of an agent by some more global information. E.g. Stone and Veloso (1999) find good solutions to a network routing problem by using information whether a neighboring node is busy or not to decide to which node a package should be sent. In this way, local decision making can be improved. When we look into the future, we expect that more systems will be constructed combining local information and global information in which the global information may be obtained in different ways, e.g. by communication. Another interesting topic is how reward rules can be set up so that optimizing a single agent's behavior is for the common good. For now, many interesting multi-agent problems are still waiting to be studied and lots of exciting work has to be done to construct algorithms solving them.

Acknowledgments

Thanks to Stephan ten Hage, Nicoletta Fornara, and Mehdi Dastani for many helpful comments.

A Vickrey-Groves-Clark mechanism

The Vickrey-Groves-Clark (VCG) mechanism can be used to let agents reveal the truth so that the problem to be optimized can be solved. The mechanism uses payment to agents in a way that this payment minus the cost for an agent is maximized when the agent tells the truth (Nisan, 1999). E.g. consider the following problem from (Nisan, 1999): A cache is shared by many processors. When an item is entered into this cache, all processors get faster access. Each processor i saves t^i having a certain item in the cache. The cost of loading the item in the cache is C. Thus, we want to load the item if $\sum_i t^i > C$.

If we divide the cost by all participating agents, we set $p^i = -\frac{C}{n}$ for n agents. However, if an agent has $t^i > p^i$, it could announce its valuation as greater than C, and ensure that the item is loaded. If agents have to pay a fragment of their (mentioned) valuation, they will try to declare the lowest possible valuation, resulting in a free rider problem which can lead to no item being loaded.

The solution is to let an agent pay the minimal declaration needed from her to load the item in the cache, thus: $p^i = \sum_{j \neq i} t^j - C$. Thus, the payment is only non-zero if $\sum_{j \neq i} t^j \leq C \leq \sum_j t^j$. If agents declare lower utility, the item may not be loaded so that the agent cannot profit by having $t^i > p^i$. If agents declare a higher utility, the item may be loaded and the agent may have to pay for it.

References

Aarts, E. and Korst, J. (1988). *Simulated Annealing and Boltzmann Machines.* John Wiley.

Albus, J. S. (1975). A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61.

Andersson, M. and Sandholm, T. (1998). Contract types for satisficing task allocation: II experimental results. In *AAAI 1998 Spring Symposium: Satisficing Models.* Stanford University.

Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., and Hosoda, K. (1994). A vision-based reinforcement learning for coordination of soccer playing behaviors. In *Proceedings of AAAI-94 Workshop on AI and A-life and Entertainment*, pages 16–21.

Axelrod, R. (1984). *The evolution of cooperation.* Basic Books, New York, NY.

Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E. D. (1997). A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research-C*, 5:109–122. http://www.idsia.ch/~eric/articles.dir/articles.html.

Balch, T. and Arkin, R. (1994). Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1:27–52. http://www.cc.gatech.edu/grads/b/Tucker.Balch/papers/comm.ps.Z.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138.

Bellman, R. (1961). *Adaptive Control Processes.* Princeton University Press.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-dynamic Programming.* Athena Scientific, Belmont, MA.

Boutilier, C. (1998). Sequential optimality and coordination in multiagent systems. Technical report, University of British Columbia. http://www.cs.ubc.ca/spider/cebly/Papers/seqCoord.ps.

Boyan, J. A. and Moore, A. W. (1997). Using prediction to improve combinatorial optimization search. In *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, page 14.

Brafman, R. and Tennenholtz, M. (1996). On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research*, 4:477–507. http://www.cs.washington.edu/research/jair/contents/v4.html.

Brauer, W. and Weiss, G. (1997). Multi-machine scheduling — a multi-agent learning approach. Technical report, Technische Universität München.

Carver, N., Cvetanovic, Z., and Lesser, V. (1991). Sophisticated cooperation in FA/C distributed problem solving systems. In *Proceedings of the Ninth National Conference on Artificial Intelligence.* http://www.cs.siu.edu/~carver/ps-files/tr91-23.ps.gz.

Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 726–731. Morgan Kaufman.

Choi, S. and Yeung, D.-Y. (1995). Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. Technical report, Hong Kong University of Science and Technology.

Colorni, A., Dorigo, M., and Maniezzo, V. (1993). Algodesk: an experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem. *European Journal of Operations Research.*

Crites, R. and Barto, A. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge MA. MIT Press.

Di Caro, G. and Dorigo, M. (1998a). An adaptive multi-agent routing algorithm inspired by ants behavior. In *Proceedings of PART98 - Fifth Annual Australasian Conference on Parallel and Real-Time Systems.*

Di Caro, G. and Dorigo, M. (1998b). Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365. http://www.cs.washington.edu/research/jair/contents/v9.html.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M. and Di Caro, G. (1999). The ant colony optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization.* McGraw-Hill.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation*, 1(1):53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41.

Finin, T., McKay, D., and Fritzson, R. (1992). An overview of KQML: A knowledge query and manipulation language. Technical report, University of Maryland.

Fogel, D. B. (1993). Evolving behaviors in the iterated prisoner's dilemma. *Evolutionary Computation*, 1:77–98.

Fritzke, B. (1994). Supervised learning with growing cell structures. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 255–262. San Mateo, CA: Morgan Kaufmann.

Gambardella, L., Rizzoli, A., and Zaffalon, M. (1998). Simulation and planning of an intermodal container terminal. Technical Report IDSIA-41-98, IDSIA, Lugano.

Gambardella, L. M., Taillard, E., and Dorigo, M. (1997). Ant colonies for the QAP. Technical Report IDSIA-4-97, IDSIA, Lugano, Switzerland.

Glover, F. and Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers.

Gmytrasiewicz, P. and Durfee, E. (1992). A logic of knowledge and belief for recursive modeling: Preliminary report. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 396–402.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA.

Hardin, G. (1968). The tragedy of the commons. *Science*, 162:1243–1248.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kourtz, P. (1994). Advanced information systems in Canadian forest fire control. Technical report, Wildfire Management Systems Inc.

Koza, J. R. (1992). Genetic evolution and co-evolution of computer programs. In Langton, C., Taylor, C., Farmer, J. D., and Rasmussen, S., editors, *Artificial Life II*, pages 313–324. Addison Wesley Publishing Company.

Littman, M. and Boyan, J. (1993). A distributed reinforcement learning scheme for network routing. In Alspector, J., Goodman, R., and Brown, T., editors, *Proceedings of the First International Workshop on Applications of Neural Networks to Telecommunication*, pages 45–51, Hillsdale, New Jersey. http://www.cs.duke.edu/˜mlittman/docs/routing-iwannt.ps.

Littman, M. L. (1994a). Markov games as a framework for multi-agent reinforcement learning. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 157–163. Morgan Kaufmann Publishers, San Francisco, CA. http://www.cs.duke.edu/˜mlittman/docs/ml94-final.ps.

Littman, M. L. (1994b). Memoryless policies: Theoretical limitations and practical results. In Cliff, D., Husbands, P., Meyer, J. A., and Wilson, S. W., editors, *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 297–305. MIT Press/Bradford Books. http://www.cs.duke.edu/~mlittman/papers/sab94.ps.

Luke, S., Hohn, C., Farris, J., Jackson, G., and Hendler, J. (1997). Co-evolving soccer softbot team coordination with genetic programming. In *Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence (IJCAI-97).*

Massios, N. and Voorbraak, F. (1999). Hierarchical decision-theoretic robotic surveillance. In *Proceedings of the IJCAI'99 Workshop on Reasoning with Uncertainty in Robot Navigation*, pages 23–33.
http://carol.wins.uva.nl/~massios/.

Mataric, M. (1997). Using communication to reduce locality in multi-robot learning. In *American Association for Artificial Intelligence.* http://www.cs.brandeis.edu/~maja/pubs/aaai97-my.ps.gz.

Mataric, M. J. (1994). *Interaction and Intelligent Behavior.* PhD thesis, Massachusetts institute of Technology. http://www.cs.bham.ac.uk/~sra/People /Mno/Mataric/#Interaction.

Matsubara, H., Noda, I., and Hiraki, K. (1996). Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In Sen, S., editor, *Working Notes for the AAAI-96 Spring Symposium on Adaptation, Coevolution and Learning in Multi-agent Systems*, pages 63–67.

McCallum, R. A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 387–395. Morgan Kaufmann Publishers, San Francisco, CA.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130.

Moriarty, D. and Langley, P. (1998). Learning cooperative lane selection strategies for highways. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-'98).* http://www.isi.edu/~moriarty/mypapers.html.

Moriarty, D., Schultz, A., and Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276. http://www.isi.edu/~moriarty/mypapers.html.

Moriarty, D. E. and Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7:195–209. http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html.

Moriarty, D. E. and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32.
http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html.

Murciano, A. and Millan, J. (1989). Learning signaling behaviors in cooperative agents. Technical Report COINS 89-89, University of Massachusetts, Amherst MA 01003.

Nisan, N. (1999). Algorithms for selfish agents: Mechanism design for distributed computation. In *STACS'99*. Springer Verlag, Berlin Heidelberg.

Ono, N. and Fukumoto, K. (1997). A modular approach to multi-agent reinforcement learning. In Weiss, G., editor, *DAI Meets Machine Learning*, Lecture Notes in Artificial Intelligence, pages 25–39. Springer-Verlag.

Prescott, T. (1994). *Exploration in Reinforcement and Model-based Learning*. PhD thesis, University of Sheffield.

Rechenberg, I. (1971). Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Dissertation. Published 1973 by Fromman-Holzboog.

Rechenberg, I. (1989). Evolution strategy: Nature's way of optimization. In Bergmann, editor, *Methods and Applications, Possibilities and Limitations*, pages 106–126. Lecture notes in Engineering.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

Sahota, M. (1993). Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. Master's thesis, University of British Columbia.

Sałustowicz, R. P. and Schmidhuber, J. H. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141.

Sałustowicz, R. P., Wiering, M. A., and Schmidhuber, J. H. (1997). On learning soccer strategies. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Proceedings of the Seventh International Conference on Artificial Neural Networks (ICANN'97)*, volume 1327 of *Lecture Notes in Computer Science*, pages 769–774. Springer-Verlag Berlin Heidelberg. http://carol.wins.uva.nl/˜wiering/publications.html.

Sałustowicz, R. P., Wiering, M. A., and Schmidhuber, J. H. (1998). Learning team strategies: Soccer case studies. *Machine Learning*, 33(2/3). http://carol.wins.uva.nl/˜wiering/publications.html.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229.

Sandholm, T. (1996). *Negotiation among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts at Amherst.

Sandholm, T. (1999). Distributed rational decision making. In Weiss, G., editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. MIT Press.

Sandholm, T. W. and Crites, R. H. (1995). On multiagent Q-learning in a semi-competitive domain. In Weiss, G. and Sen, S., editors, *IJCAI'95 Workshop: Adaption and Learning in Multi-Agent Systems*, pages 164–176. Springer-Verlag.

Schaerf, A., Shoman, Y., and Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500. http://www.cs.washington.edu/research/jair/contents/v2.html.

Schmidhuber, J. H. (1996). A general method for incremental self-improvement and multi-agent learning in unrestricted environments. In Yao, X., editor, *Evolutionary Computation: Theory and Applications*. Scientific Publ. Co., Singapore.

Schmidhuber, J. H. and Zhao, J. (1997). Multi-agent learning with the success-story algorithm. In Weiss, G., editor, *Distributed Artificial Intelligence meets Machine Learning*, pages 82–93. Springer, Berlin. http://www.idsia.ch/~juergen/onlinepub.html.

Schmidhuber, J. H., Zhao, J., and Wiering, M. A. (1997). Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130. http://carol.wins.uva.nl/~wiering/publications.html.

Schneider, J., Wong, W.-K., Moore, A., and Riedmiller, M. (1999). Distributed value functions. In *Proceedings of the International Conference on Machine Learning (ICML'99)*. http://www.cs.cmu.edu/~schneide/papers.html.

Singh, S. and Bertsekas, D. (1996). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*. MIT Press, Cambridge MA.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Steels, L. (1997). Constructing and sharing perceptual distinctions. In van Someren, M. and Widmer, G., editors, *Machine Learning: Proceedings of the ninth European Conference*, pages 4–13. Springer-Verlag, Berlin Heidelberg.

Stone, P. and Veloso, M. (1996). Beating a defender in robotic soccer: Memory-based learning of a continuous function. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 8*, pages 896–902. MIT Press, Cambridge MA.

Stone, P. and Veloso, M. (1997). Multiagent systems: A survey from a machine learning perspective. Technical report, Carnegie Mellon University.

Stone, P. and Veloso, M. (1999). TPOT-RL: Team-partitioned, opaque-transition reinforcement learning. *Submitted to Machine Learning*.

Stone, P., Veloso, M., and Riley, P. (1999). CMUnited-98 champion simulator team. In Asada, M. and Kitano, H., editors, *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Artificial Intelligence*, pages 61–76. Springer-Verlag.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1045. MIT Press, Cambridge MA.

Taale, H., Bäck, T., Preuß, M., Eiben, A. E., de Graaf, J. M., and Schippers, C. A. (1998). Optimizing traffic light controllers by means of evolutionary algorithms. In *EUFIT'98*.

Taillard, E. (1990). Robust taboo search for the quadratic assignment problem. Technical Report IORWP 90/10, DMA, Swiss Federal Institute of Technology, Lausanne.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal of Computing*, 6:108–117.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124. http://www.cs.washington.edu/research/jair/contents/v7.html.

Tambe, M., Adibi, J., Alonaizon, Y., Erdem, A., Kaminka, G., Marsella, S., and Muslea, I. (1999). Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*. http://www.isi.edu/teamcore/tambe/.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337.
http://www.cs.brandeis.edu/~aeg/papers/tan.ML93.ps.

Tesauro, G. (1992). Practical issues in temporal difference learning. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 4*, pages 259–266. San Mateo, CA: Morgan Kaufmann.

Thorpe, T. and Anderson, C. (1996). Traffic light control using SARSA with three state representations. Technical report, IBM coorporation.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence Journal*, 99(1):21–71.

Turner, R. M. (1993). The tragedy of the commons and distributed AI systems. Technical Report 93-01, University of New Hampshire, Durham.

Turner, R. M. and Turner, E. H. (1998). Organization and reorganization of autonomous oceanographic sampling networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Versino, C. and Gambardella, L. M. (1997). Learning real team solutions. In Weiss, G., editor, *DAI Meets Machine Learning*, volume 1221 of *Lecture Notes in Artificial Intelligence*, pages 40–61. Springer-Verlag.

Vidal, M. and Durfee, E. (1998). Learning nested models in an information economy. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):291–308.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Wiering, M. A. (1999). *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam.
http://carol.wins.uva.nl/~wiering/publications.html.

Wiering, M. A. and Dorigo, M. (1998). Learning to control forest fires. In Haasis, H.-D. and Ranze, K. C., editors, *Proceedings of the 12th international Symposium on "Computer Science for Environmental Protection"*, volume 18 of *Umweltinformatik Aktuell*, pages 378–388, Marburg. Metropolis Verlag.

Wiering, M. A., Sałustowicz, R. P., and Schmidhuber, J. H. (1998). CMAC models learn to play soccer. In Niklasson, L., Bodén, M., and Ziemke, T., editors, *Proceedings of the 8th International Conference on Artificial Neural Networks (ICANN'98)*, volume 1, pages 443–448. Springer-Verlag, London. http://carol.wins.uva.nl/˜wiering/publications.html.

Wiering, M. A. and Schmidhuber, J. H. (1997). HQ-learning. *Adaptive Behavior*, 6(2):219–246. http://carol.wins.uva.nl/˜wiering/publications.html.

Wolpert, D., Turner, K., and Frank, J. (1999a). Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems 11*. MIT Press.

Wolpert, D. H., Tumer, K., and Frank, J. (1999b). An introduction to collective intelligence.

Wooldridge, M. (1999). Intelligent agents. In Weiss, G., editor, *Multiagent Systems: A modern approach to Distributed Artificial Intelligence*, pages 27–77. MIT Press, Cambridge, Massachusetts, London, England.

Yamauchi, B., Schultz, A., Adams, W., and Graves, K. (1998). Exploration and spatial learning research at NCARAI. In *Proceedings of the Conference on automated learning and discovery (CONALD'98): Robot Exploration and Learning.* Carnegie Mellon University, Pittsburgh.

Ye., Y. and Tsotsos, J. (1997). On the collaborative object search team: A formulation. In Weiss, G., editor, *DAI Meets Machine Learning*, Lecture Notes in Artificial Intelligence, pages 94–116. Springer-Verlag.

Zlotkin, G. and Rosenschein, J. (1996). Mechanisms for automated negotiation in state oriented domains. *Journal of Artificial Intelligence Research*, 5:163–238. http://www.cs.washington.edu/research/jair/contents/v5.html.