LAYLA – An InteRRaP Extension for Layered Learning in Repeated Games

Michael Rovatsos
(rovatsos@dfki.de)

Diplomarbeit

nach einem Thema von Prof. Dr. Jörg H. Siekmann,
Fachbereich Informatik der Universität des Saarlandes

Betreuer: Dipl.-Inform. Jürgen Lind,
Deutsches Forschungszentrum für Künstliche Intelligenz

Eidesstattliche Erklärung.

Hiermit versichere ich an Eides statt, die vorliegende Arbeit selbständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

Saarbrücken, den 29. September 1999

Michael Rovatsos

Acknowledgements.

I would firstly like to thank Prof. Jörg Siekmann for accepting my proposal for this thesis and Jürgen Lind for his willingness to supervise a rather theoretical, exploratory thesis – not knowing whether it would work or not. The multi-agent systems group at DFKI was always a very pleasant environment to work in, and people there should be thanked for tolerating that I clogged up filespace and CPUs with seemingly endless experiments for this work.

Furthermore, the anonymous reviewers of the *Workshop on Learning About, From and With other Agents* held at *IJCAI'99* deserve my gratitude for pointing at various theoretical inconsistencies of earlier versions. Countless discussions with Michael Schillo have contributed to a deeper understanding for the problems of social interaction (for both sides, I believe) and he deserves my special thanks for this. Also, I would like to thank Oliver Plaehn, Alexander Koller and Hiroko Nishuichi who reviewed parts of this work and spent long hours cross-checking references, correcting spelling mistakes and checking formulae and proofs.

Above all, I am indebted to my family for their relentless material and immaterial support which has always been reassuring during the past years (and for not getting too worried about not knowing what *exactly* it is that I've been doing).

Finally, I want to thank Susanne Jörgens for bearing with me for the last months, while my head was full of utility functions, MDPs and Recursive Belief Models. Without her love and understanding much of this would have not have been possible.

Abstract.

We present a layered learning architecture for adaptive behaviour in repeated games as an instance of a generic hierarchical, distributed view of multi-agent learning. This architecture builds on the hybrid InteRRaP architecture and extends it naturally by adding learning components to each of the InteRRaP layers. The three learning layers are responsible for managing sub-tasks of learning how local actions can be coordinated effectively with those of other agents. We identify three essential determinants of interaction which will be learned by these components and devise appropriate learning algorithms for them. We claim that such an architecture can be used to achieve cooperation amongst self-interested, non-benevolent agents in the absence of an external authority without assuming that they are able to communicate, that they have any prior knowledge of the underlying games or any information about the welfare of their opponents.
The adequacy of our approach is tested in the context of a resource-sharing scenario, and first empirical results prove that the learning architecture enables agents to do "better than equilibrium" even in large games which represent very hard coordination problems, while they converge to truly optimal behaviour for smaller problem sizes.

Contents

Chapter 1

Introduction

Preserve me from the enemy that has something to gain;
and from the friend that has something to lose.
— T. S. Eliot, Choruses from 'The Rock'

1.1 Motivation

Interaction, the process by which entities induce an effect on each other by their actions, is ubiquitous. Any natural or artificial agent who is not completely isolated or completely indifferent toward its environment is inevitably confronted with the need to interact with that environment in order to attain its goals. This is due to the fact that, at least in the environments humans reason about, the resources, i.e. the material, informational or emotional factors on which goal attainment depends, are limited. Furthermore, the access to these resources usually involves some kind of "goal-driven" effort, which can range from simple sensory or motoric actions to highly complex cognitive and communicative activities. In worlds inhabited by multiple agents, these efforts inevitably overlap and cause interdependencies between the actions of the agents with respect to their effectiveness. This is to say that the outcome of some agents' actions is inherently dependent on other agents' actions whenever they concern the same limited resource. Moreover, every agent itself becomes something like a "resource" for others in such worlds: its abilities and its behaviour (which are both limited as well, unless the agent is omnipotent and utterly altruistic at the same time) map to the value and the accessibility of some external resource.

Consider, for example, pollution problems caused by car traffic in an urban area. Drivers may be tempted to make excessive use of their cars in times of relatively low air pollution, because the resources they are using (air quality) seem virtually unlimited to them. However, such behaviour may lead to smog alarms and temporal governmental restrictions may be imposed regarding the use of the public road system. Ultimately, even those drivers who urgently need to use their cars might be unable to do so because of the uncoordinated behaviour of the driver community, and thus every other driver's behaviour affects one's own situation.
If every driver had considered the global resource allocation before deciding when and how often to drive, pollution highs would have been prevented and she could freely access all roads at all times. Clearly, rational decisions with this respect always depend on the expected actions of co-actors and have to be coordinated with these.

So whenever there is no abundance of resources or total absence of others wishing to access that same resource, interaction is inevitable. Taking into account that almost any real-world situation involves resource contingencies, one could argue that *any* cognitive problem-solving ability developed by agents to improve their standing in a non-solitary environment is concerned with interaction in that wider sense.

It is therefore not surprising that interaction has been studied for a long time by many, very distinct areas ranging from psychology, sociology and politics to economics, organisation and management science (cf. Pfeffer and Koller (1997) and the references therein). For psychology, especially for the sub-field of social psychology, the effects of encounters with others on the self are of major importance when analysing how personality structures come about. Also, the behaviour of interacting groups is studied to investigate why individuals exhibit a certain behaviour in particular interaction situations and how this influences group dynamics. At a more abstract and global level, such analysis is also carried out by sociologists in order to find out how societies evolve. To a certain degree, politics research assumes a similar stance, except that in this field, *normative* interaction concepts are sought for (rather than *descriptive* ones), in order to improve political decision-making in interaction situations (e.g. in conflict resolution or international relations). Economics and organisation and management science, finally, do indeed nothing else than to study the interdependence amongst humans and their environment, and to devise methods that are likely to improve the standing of the *homo oeconomicus* in a social context characterised by the scarcity of the resources it provides, and by a multitude of co-actors that seek to optimise their own profit.

But what is the essence of interaction? How is it possible, for any given collection of intelligent beings, to *coordinate* their behaviour with that of others effectively? Should they care about how well the community fares, or should they only be concerned with how they can profit from the situation? How much information is required to be able to defend oneself against attempts of exploitation made by others? How should one behave toward others not being sure about what actions they will take in the future? What are, if any, reliable generic strategies to use for interaction? Under what conditions can cooperation emerge in a society?
These are only a few unsolved questions in interaction and coordination research, and it is some of these questions that we confront in this work. More specifically, we ask whether it is possible to *learn* optimal interaction behaviour by employing certain reasoning techniques. We suggest a method for *layered coordination learning* in multi-agent systems, that allows agents to adapt to a particular interaction situation , and present an application example to illustrate its practical relevance. Our main goal is to identify the essential determinants of interaction settings and to design an agent architecture that is capable of handling the dependencies and contingencies that agents have to face in repeated abstract multi-agent interactions without previous knowledge in the absence of a central authority. We restrict the scope of our analysis to a very specific, well-defined and well-understood area of *repeated games*, because it provides us with the necessary abstraction needed to develop a generic, domain-independent framework for learning interaction.

The novelty of our approach lies in the fact that we attempt to capture all essential aspects of interaction (in its game-theoretic, abstract form) in a single, integrated learning architecture. This architecture could, in principle, be "added" to virtually any multi-agent system to help optimise the coherence of agent interactions, and thus increase

system performance. And while it will, most probably, increase the long-term system performance in purely cooperative or collaborative systems only marginally, it might have a dramatic impact on the performance of agents in heterogenous, anonymous agent societies consisting of purely selfish individuals. Such societies become increasingly interesting if we consider Internet agents and Electronic Commerce applications, in which agents often have to interact with possibly non-benevolent agents, about which they have little knowledge and with whom they don't share goals. Hence, we believe that studying how effective interaction behaviour could be implemented in such systems is an issue that deserves our attention.

1.2 Learning interaction in multi-agent systems

1.2.1 The case for interaction in Distributed Artificial Intelligence

For Artificial Intelligence (AI), the field that "strives to *build* intelligent entities as well as to understand them" (Russell and Norvig, 1995, p. 3), the "interaction perspective" of cognition becomes interesting whenever those intelligent entities are seen as parts of "agent societies". The advent of Distributed Artificial Intelligence (DAI) lead to the development of a multitude of theoretical frameworks and practical applications that focus around this view. DAI, "the study and design of collections of interacting systems, or agents, which in some sense can be called intelligent" (Seel, 1991) is based on the concept of agents, that can be traced back to Carl Hewitt's concurrent Actor model (Hewitt, 1977), and in which Hewitt proposed "the concept of a self-contained, interactive and concurrently-executing object" (Nwana, 1995).
More than twenty years of DAI research have yielded a splendour of definitions of what an agent is. We believe the one given by Hewitt above provides a "greatest common denominator" for most of these definitions (cf. for example Sundermayer (1993) for an overview), at least at the computational level. At the conceptual level, the attributes most often ascribed to agents are *situated*, *autonomous*, *rational*, *intelligent* and *social*[1], and it is the last property of agents that makes interaction a key issue in the study of distributed intelligent systems: concurrently operating agents will inevitably *have to* interact with each other in a common environment.
According to Bouron and Collinot (1992)

> "a social agent is characterised by its capabilities to be integrated in a social context and above all by its capabilities to take an active part in the system organization."

There has been much dispute about whether these "capabilities" have to fulfill certain criteria for the agents to deserve being called "social" (such as, e.g., "helpfulness", "cooperativeness", "knowledge of the social context") or whether agents are *per se* social by

[1] This is not an exclusive, exhaustive or undisputed list of possible criteria for "agenthood". Wooldridge and Jennings (1995), for example, distinguish between a *weak* notion of agency (which requires that agents be autonomous, reactive and pro-active) and a *strong* notion, in which it is additionally required that agents have certain mental attitudes such as belief, knowledge, intention, commitment, desire, goal etc. The distinction between these two notions reflects part of the dispute between research on *reactive* agents initiated by the works of Brooks (1991) and Agre and Chapman (1987) (that complies only with the weak agent notion) and the traditional *deliberative* school of DAI (in which it is assumed that agents use symbolic knowledge representation and reasoning).

virtue of their participation in the social context, i.e. by merely co-existing and interacting with other agents (Sundermayer, 1993). The stance one assumes toward this issue is essential to the design of agent systems: a proponent of the first definition is very likely to design systems of interacting agents that work together to solve a common problem (Kalenka and Jennings (1995) call this the *reductionist view*), whereas someone in support of the latter definition would devise agents that operate to further their individual needs and expect the system behaviour to emerge from the microscopic interplay between agents (in the *constructivist view*).
We might even argue that it is precisely this distinction that lead to the two sub-fields of DAI, Distributed Problem Solving (DPS) and Multi-Agent Systems (MAS). DPS is a discipline concerned with solving difficult tasks by dividing them among cooperative agents, while research in MAS focuses on "coordinating the behaviour of autonomous intelligent agents, how they coordinate their knowledge, goals, skills and plans jointly to take action or solve problems" (Müller, 1993, p. 10). Thus, while the primary reason for building DPS systems is to solve a particular problem by decomposing it into easier sub-tasks, MASs concentrate on the notion of autonomy and self-organization of societies.

It has been broadly acknowledged in both areas that the way in which agents interact with each other is crucial to system behaviour, and a great deal of work in DAI focuses on this issue. According to Jennings (1996), coordination, i.e. "the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure the community acts in a coherent manner" is perhaps the key problem in DAI, and Durfee (1991) states that "research in distributed artificial intelligence concentrates on understanding the knowledge and reasoning techniques needed for intelligent coordination, and on embodying and evaluating this understanding in computer systems."
At this point we should maybe introduce what exactly we mean when we talk about interaction and coordination. A very broad definition (Malone and Crowston, 1991) states that

> "An interaction can be viewed as a formalization of a concept of dependence between agents, no matter on whom or how they are dependent. Coordination is a special case of interaction in which agents are aware how they depend on other agents and attempt to adjust their actions appropriately."

As Stirling has pointed out, coordination is a neutral concept, and "cooperation, negotiation, competition, and any other form of behavior short of complete indifference and isolation between agents will involve some form of coordination" (Stirling, 1994, p. 167). We will adhere to this view throughout this work, and we also adopt Stirling's definition of *coordinated decision*:

> "Any decision by an agent that uses information concerning the existence, decisions, or decision-making strategies of other agents is a coordinated decision. Information used for coordination is obtainable in many ways. It may be known *a priori*, it may be communicated between agents directly, it may be obtained via sensory input, and it may be learned by the agent as it functions in its environment."

This is, in fact, one of the few definitions that stresses the importance of *information* for coordinated decision-making and it is precisely the focus on the *epistemic* aspects of interaction that constitutes the starting point for our analysis. In particular, we investigate the possibilities of learning the information necessary to achieve effective coordinated decision-making.

1.2.2 A layered learning approach to interaction

In this work, we try to answer two questions: Firstly, is it possible to endow agents with social reasoning capabilities that enable them to *learn* to coordinate their behaviour effectively with that of other agents without explicitly designing desirable interaction patterns at the system level? And secondly, can we provide generic methods that realise such "coordinative" capabilities on a domain-independent level?
Of course, these questions are not new to the DAI community. What is new, however, is the coverage of our approach and the method we employ to prove that the above questions can be answered in the positive.
As concerns the coverage, it can be claimed that our approach is generic in that it does not assume that agents have any prior knowledge of the "interaction setting" (we shall specify more precisely what we mean by that shortly) except that they know who they are interacting with and what their own action options are. Furthermore, it suffices for an agent to be given information about (a) what other agents do and (b) how "satisfied" it should be with the current situation in the course of the interaction. Agents need not be able to communicate with other agents, they need not care for the "common good", and it is not necessary for them to know anything about how successful other agents are. Finally, agents do not need to rely on perfect information and action, i.e. their ability to learn efficient coordinated behaviour is (within certain bounds) robust to noise in perception and action.
These assumptions are substantially distinct from those put forward in most of current research on coordination mechanism design within DAI: most of the approaches so far are based on explicit negotiation between agents, on the investigation of settings in which global system goals have to be balanced with agents' individual goals, or on the exploitation of domain-specific knowledge to determine efficient coordination strategies (Section 2.2 gives a number of examples).

The novel contribution we make with respect to the method we use is that we suggest a generalised form of layered learning to explore the possibilities of emergent coordination. This means that we first decompose the task of "learning to interact effectively" into a number of simpler learning tasks which concentrate on certain essential aspects of interaction. We then combine the learning results of these components to devise an agent architecture that is capable of handling interactions effectively.

Determinants of Interaction Situations

We have identified three concepts that have to be learned in order to conduct effective coordinative reasoning, the *essential determinants* of interaction, three components that suffice to characterise the essence of an "interaction setting": *interdependence modalities*, *opponent behaviour* and *social potential*.

By *interdependence modalities* we mean the way in which the outcomes of agents' actions depend on each other, or alternatively, the dimensions and entries of the payoff matrix, in game-theoretic models[2].

[2]Compared to the straightforward mathematical properties of a payoff function, the term "interdependence modalities" may seem a bit blown-up. We use it, however, in an attempt to make clear that the *quality* of an interaction setting is essential to understanding it, and even if it can be modelled by a simple real-valued matrix, there is usually a multitude of individual characteristics of the interaction, such as power, influence, ability, knowledgeability involved in how "these numbers" come about.

These modalities are given by the action capabilities of agents (the number and properties of the actions they can perform), the mutual impact of different agents' actions on each other when performed synchronously, and the subjective view the agents have of the outcomes of joint actions (i.e. how high or low they value their outcomes), which of course depends on the goals agents pursue or the tasks they have to complete.
Seen from a more pragmatic perspective, they simply describe the effects of co-acting in a common environment in pursuit of maximising individual profits. In a blocks world scenario, for example, in which two robots have to perform two different tasks, determining these modalities would involve asking some of the questions: What physical actions can the robots perform? What sensing capabilities do they have? Do they employ some form of explicit planning? Can robot A threaten the fulfillment of B's task or vice versa, and if so, is this inevitable due to A's/B's task? Do the agents know anything about (the existence) of each other? Are there possibilities for the robots to "help" each other? How would they react if conflict arose?
In other words, knowledge of the factual interdependence modalities is knowledge about "what would happen to agents $p_1, p_2, \ldots, p_n$ and their environment, if they concurrently performed actions $a_1, a_2 \ldots, a_n$ in situation S".

Opponent behaviour is a concept complementary to that of interdependence modalities in that it determines what the concrete interaction *will* be like for the agent when enacted, rather than what it *could* be like. Quite naturally, it is important for an agent not only to know how its opponents[3] might affect its own standing in theory, but to be able to infer in one way or another what choices they will make in the future – knowing that the agent's own success depends on the future actions of opponents. And for an outside observer of the interaction this way in which the interaction process is actually conducted is indispensable when it comes to describing that process[4].

The *social potential* or *cooperation potential* of an interaction situation, finally, is the most complex of the three proposed concepts. Basically, it combines the knowledge of the interdependencies and opponents' anticipated behaviour to reason about whether and how the interaction situation can be exploited to the benefit of all participants, i.e. whether there are ways to achieve fruitful cooperation. To show that the existence of such a potential depends on the other two parameters, consider a case in which the interdependencies that govern the current situation result in a strictly non-cooperative (zero-sum) payoff structure, i.e. one player always loses what the other wins. Such a situation does not allow for any cooperation, it can be expected that both actors will selfishly try to make the other lose. Or, as another situation with zero cooperation potential (caused by opponent behaviour this time), consider a case in which a fair distribution of payoffs is possible, but one agent constantly tries to exploit the other – the exploited has no reason not to "strike back". If we assume, however, that the exploiter's strategy is not completely fixed, cooperation might be possible.
Combining the two previously introduced concepts could lead to the assumption that this social potential does not add anything to their meanings. However, discovering the social potential *is* different from experiencing the interrelation of action outcomes or observing some opponent's behaviour, in that it can be used to reason about what the interaction

[3]We will use the terms opponents, co-actors, partners, peers and adversaries as synonyms henceforth; from an individually rational, distrustful standpoint it doesn't really make any difference.

[4]In game-theoretic models, such behaviour could be formalised as a (pure or mixed) strategy (cf. Section 3.1) or as some mathematical decision rule, such as the maximin rule.

should be like. And this is the essence of what is new to our approach on the theoretical side: we believe that it has been neglected by most authors to develop agents who reason about the process of interaction *itself* (some of the works reviewed in Section 2.2 constitute exceptions), rather than agents who just follow utility-maximising rules (which can lead to deadlocks in undesirable states, leaving possibilities for mutually profitable cooperation undiscovered) or act according to some pre-designed interaction patterns (which can be suboptimal and inflexible). Therefore, learning this social potential is the most challenging aim of our work.

Layered learning

The techniques we actually employ to learn these concepts are well-known from the field of machine learning (ML) (Mitchell (1997) provides an excellent introduction) and there is a growing body of research on *multi-agent learning* (Weiß and Sen, 1996) that investigates the possibilities of incorporating ML techniques to agents in MASs and vice versa. We contribute to this research by proposing a generalised view of hierarchical learning, in which agent societies (*learning communities*) consist of several agents with individual learning capabilities (*learners*) which, in turn, comprise a number of *learning layers*[5]. It is the task of each of these levels of learning units to "delegate" learning sub-tasks to its sub-components and "integrate" their learning results.
This view seems adequate for breaking down difficult learning tasks, and we present a simple instantiation of it for the problem of coordination learning. It will be the task of the learning community to ensure effective interaction at the *system level*, while the agents learn to coordinate their behaviour with that of others at the *individual* level, which is, in turn, achieved by combining the results of learning the three proposed aspects of interaction at the *sub-individual* level.
This layered learning methodology borrows from the works of Stone (1998), Stone and Veloso (1999), in which they have proposed a hierarchical machine learning paradigm consisting of several learning layers responsible for learning sub-tasks of the global learning goal. However, they restrict the possible relationships between these learning layers to three specific types, so that a sub-layer provides either

1. the learning inputs,
2. the training examples or
3. the learning outputs

to its super-layer. In contrast to this intuition, we allow for hierarchical structures in which layers consist of several learning units, which adds a *horizontal* dimension to the distribution of the learning process. As an effect of this, we obtain a wider range of possibilities in coupling the learning units than that of the three classes just mentioned. More precisely, we see a learning architecture more like an *organism* built from both hierarchical and non-hierarchical structures. In it, the results of particular learning units may be combined, selected from, forwarded to "higher-order" learners, used for controlling actions or integrated in any other thinkable way to the end of coping with learning tasks intractable for "monolithical learning".

[5] In the case of distributed representations such as neural nets or genetic algorithms these layers may again consist of a number of *learning cells* that encode and handle parts of the concepts to be learned.

Assumptions

In its most general sense, as we have already mentioned, interaction is a very complex phenomenon, whose characteristics depend, amongst others, on the *knowledge* and *abilities* of the agents, on the *availability* and *accessibility* of resources, and on the *scarcity* and *demand* for these resources. Furthermore, interaction has *temporal* and *spatial* dimensions so that the duration, frequency and regularity of agent encounters as well as the physical effect of agents' actions on their environment and on other agents have an influence on how coherent global behaviour can be achieved. Of major importance is also the *dynamic* aspect of interaction: the concurrent operation of multiple agents effects changes to the environment, so that the future situation usually depends on past interactions, a fact which makes it even more difficult to devise appropriate reasoning mechanisms that deal with interaction effectively.
Therefore, it cannot be our aim to design coordination mechanisms capable of handling all these issues. Instead, we will concentrate on a very specific sub-class of problems to prove that we can solve such problems under certain assumptions by using the proposed methodology.

More specifically, we will concentrate on interaction situations that can be modelled as repeated games (we introduce the necessary game-theoretic concepts in Section 3.1), i.e. as stateless repeated interactions, in which the interdependencies among agents are characterised by the same payoff matrix in each round. It is assumed that action choices are made synchronously in discrete time, and that there is no planning of action sequences[6], since agents are faced with the same action options in each move.
Furthermore, we will assume that agents are *individual utility maximisers*, whose ultimate goal will be to maximise the payoff they receive after each round from the environment, and ask "how high-level, autonomous, independently-motivated agents ought to interact with each other so as to achieve their goals" (Genesereth, Ginsberg, and Rosenschein, 1986), if the attainment of these goals can be measured by the numerical payoffs periodically assigned to agents. We claim that if the payoff function captures the behaviour of the system adequately it is not necessary to incorporate explicit *social goals* that the agents pursue to achieve effective coordination amongst them. In the assumed scenario, we do not endow agents with any *a priori* knowledge of the payoff matrix or of the behaviour they should expect from their opponents in order to prove that these can be learned from observations, and this makes our approach substantially distinct from most previous work on the subject.
Finally, and maybe most importantly, we choose to restrict ourselves to no-communication settings. Firstly, this is because we claim that communication is not essential to achieve effective interaction behaviour. We agree with Genesereth et al. (1986) in that "communication is a powerful instrument for interaction", that it can help to establish cooperative behaviour more easily, but that it is not a prerequisite for rational behaviour in interactions, as they have proven in their now famous article "Cooperation without Communication". In fact, the approach presented here resembles their view to some degree, except that they pre-assume a lot about what agents know about their opponents' decision situation and reasoning mechanisms (as pointed out in Müller (1993)) to alleviate the restrictions imposed (on how agents can model each other) by not allowing communication.

[6]It should be remarked that this "statelessness assumption" is probably the most severe restriction imposed on the scope of our analysis. In Chapter 7 we review this issue, explain why the assumption was made, and present alternatives.

The stance we adopt here is orthogonal to theirs in that our agents will be capable of *learning* something about their opponents' decision situation and reasoning mechanisms. A second reason for excluding the possibility of communication is that, in the settings we examine, the existence of communication would force us to deal with issues such as trust, deception and fraud, which would clearly exceed the scope of this work. We refer the interested reader to (Schillo, Funk, and Rovatsos, 1999) as an example of work on these problems.
As Gmytrasiewicz, Durfee, and Wehe (1991) have pointed out, decisions about performing communicative acts can be treated just like decisions about any other actions. They introduce a *decision-theoretic pragmatics* for communication, which assigns to each communicative act some utility that corresponds to the value of the information obtained by that communication. In short, they argue that communication between agents induces changes on the models they have of each other. If the rational choice an agent can make with respect to the model it has of its opponent *after* the communicative act yields a higher payoff than that *before* the communication, performing the communicative act was a rational decision by itself, for which reason it is adequate to assign some payoff to it. This can be seen as a third reason for excluding communication from our consideration, because it means that, ultimately, communication is nothing else but *another* activity that has to be coordinated. So if certain communicative acts are included in the action choices agents have, and if their effects are adequately captured by the payoff function, learning to communicate can be reduced to learning general action coordination[7].

All these assumptions admittedly restrict the scope of our analysis significantly. Yet they still allow for the modelling of various real-world situations, especially in MAS consisting of non-benevolent, highly autonomous agents. Since we can only model repeated identical interactions, our approach will be most suitable for the long-term optimization of distributed systems devoid of centralised control. As an example for such systems, the next section introduces *multiple-access resource-load balancing*, a combination of two problems, one widely studied in distributed systems research and the other motivated by a recent publication on Web-searching agents. We shall use this problem as a practical application example throughout the rest of this work to validate the adequacy of our approach.

1.3 Application example

1.3.1 Two problems in one

As mentioned before, the availability, accessibility and demand for resources determine the behaviour of a social system to a great degree as soon as there are several individuals trying to access them and if the amount of available resources is limited.
In many real-life situations, the issue of resource *divisibility* imposes additional constraints on how agents have to behave to thrive in their environment. This issue arises because dividing the resource is often not possible (i.e. if it is accessed by one individual it becomes unreachable for others), or, at least, causes some cost. This cost represents the efforts

[7]Having said that, we do not question the achievements of research on communication protocols (e.g. work on the Contract Net protocol by Davis and Smith (1981)) or on agent communication languages (such as KQML, cf. Finin, Fritzson, McKay, and McEntire (1994)). But dealing with the issues such research focuses on is inappropriate in the context of our abstract, game-theoretic treatment.

necessary to divide the resource and to distribute it, an *overhead* in coordination and communication costs that is not directly related to the process of accessing the resource itself.
There are countless examples for this effect: companies engage in cost-intensive administrative activities to guarantee that resources are correctly allocated within their organizational structure, nations sustain expensive governments to redistribute the wealth produced by the national economy, and so on.
Thus, if a resource is divided to make parts of it available to each individual, some cost is associated with the act of "splitting" the resource: *Cutting the cake will always make you loose some crumbs.*

This observation is the starting point for the application scenario we will use to demonstrate how our approach works in practice: multiple-access resource-load balancing, a combination of the *multiple access* problem as introduced in (Bicchieri, Pollack, and Rovelli, 1996) and the classical problem of *distributed load balancing*, to which much work has been devoted in the fields of distributed computer systems, organization theory, management science and DAI (cf. Schaerf, Shoham, and Tennenholtz (1995) and the references therein).

The multiple-access problem

The multiple-access problem focuses around intelligent network agents on the Internet, "who are capable of understanding a user's specification of information goals, and deciding upon and carrying out the tasks necessary to achieve those goals." (Bicchieri et al., 1996, p. 6). The problem as such is given by the following questions:

> "If an agent seeking a piece of information knows of several sites that have, or might have, that information, how many queries should it issue, and when? Should several queries be issued, rather than in the sequential fashion that is typical today?"

In the assumed setting, agents issue queries to a fixed number of sites that are capable of handling only one query at a time, so that the utility of accessing the resource first is higher than that of accessing it second, accessing it second is better than accessing it third, and so on, assuming that the utility decreases with the response time of the site (which, quite naturally, increases with "request load"). This means that a divisibility limitation as defined above exists, whenever the sum of utilities distributed to accessors increases sub-linearly, i.e. if the sum of utilities $n + 1$ agents receive provides a lower individual reward for each of them than if n accessors were accessing it.
Bicchieri and colleagues examine cases in which the position of an individual in the row of agents accessing the site is determined by a stochastic process and analyse under what conditions cooperative behaviour can emerge if agents are tempted to greedily access all sites at all times (after all, even coming last in the query queue yields some positive payoff).

Resource-load balancing

The aspect of load balancing (disregarded by the above authors) comes in at the point where we ask "if a collection of independent, individually rational agents exhibits a certain behaviour in accessing the available resources, how efficient will the use be that they

make of the resource?" This is important, because at the system level, we are also interested in how the query load is distributed among the available resources. Agents might, for example, converge very quickly to an equilibrium in which every agent accesses every resource, which would lead to a situation in which resources are hopelessly *overloaded* and fail to operate efficiently.
An example of work that is concerned with exactly this issue (and which we use as another starting point for constructing an appropriate application scenario) can be found in (Schaerf et al., 1995). There, a *multi-agent multi-resource stochastic system* is defined, which involves a set of agents and a set of (passive) resources with changing resource capacities. New jobs with probabilistically changing job sizes are probabilistically assigned to agents over time, and it is the task of agents to select a resource for each new job. The efficiency with which these jobs are handled depends on the current capacity of the resource and on the number of other jobs handled by that same resource over that period of time. The approach presented to learn appropriate resource-selection rules uses reinforcement learning based on *purely local information*, i.e. agents are only imparted information about how efficiently the submitted job was handled by the chosen resource (they know nothing about the load of other resources). Although both information and decision procedure are purely local, this approach aims at "globally optimizing the resource usage in the system while ensuring fairness (that is, a system should not be made efficient at the expense of any particular agent), two common criteria for load balancing" (Schaerf et al., 1995, p. 475).
What we find particularly interesting about this problem is the combination of decentralised decision-making and the need for a globally coherent system behaviour that makes optimal use of the available resources, and the variety of real-world situations in which this connection between individual rationality and global optimality becomes relevant. Thomas and Sycara (1998, p. 293) underline its importance by stating that

> "the growing interest in decentralised systems evinced by the fields of Multi-agent Systems and Distributed Artificial Intelligence brings along with it a concern for the stability of such systems. When agents lack explicit coordination mechanisms, or act on incomplete or delayed knowledge, actions that may appear locally optimal may create global instability at the system level. This can be a particular problem in systems where agents allocate resources among themselves with no central control. Problems with this characteristic include load balancing over multiple processors, the allocation of Internet traffic on multiple network routes, and market-like control systems."

1.3.2 The repeated multiple-access resource-load balancing game

Guided by these preliminary considerations, we present the following application scenario: A set of agents has a fixed set of resources at their disposal. Each agent may choose to access an arbitrary subset of these. The utility (the value of the service provided by the resource) that is distributed to the agents that have accessed the resource depends on the capacity of the resource and on the number of the agents competing for access to the resource (we only account for cases in which the efficiency of the resource decreases with the number of accessors). After each agent has decided which resources to use, the individual utilities are computed and distributed as payoffs to each agent. Agents neither receive information about the payoffs their peers receive, nor do they know *how* their own payoff computes. However, they are given (more or less accurate) information about their

peers' action choices. This process is iterated *ad infinitum*, and the performance measure applied to evaluate the success of an agent is the sum of payoffs that an agent has received so far.

The intuition behind this model is that we have a set of resources capable of executing certain tasks, such as processors in a multi-user machine, WWW search engines, or application servers in a LAN environment. We assume that agents synchronously and discreetly access these resources to further their own needs, and that these needs (as opposed to the scenario proposed in Schaerf et al. (1995)) remain constant over time, which means that, in each round, agents need to access the same set of resources (for repeatedly executing the same tasks).
It is reasonable to assume that even in decentralised systems agents will probably have some information about which and how many other agents access the same resources like them. In the search engine example this is of course less probable, but in the multiprocessor and LAN domains it is usually possible to monitor which users run which processes on the CPUs and servers.
In such systems, information about the agent's own payoff is indirectly imparted to it by the response time of servers or by the execution speed of processes. However, it is unlikely that agents would know something about how well other users fare, since it is, for example, unlikely that some greedy user who found out that accessing all available resources pays would reveal this knowledge to her ignorant co-users.
It is also realistic to model the users of such systems as *selfish* agents concerned only with their own success in the environment. Firstly, this is because the individuals competing for resources in distributed computer systems usually form a body of heterogenous, independent agents that pursue different goals, anonymous actors (e.g. in the WWW domain) in an "increasingly distributed world", who will not cooperate with others unless cooperation is beneficial to them.
Secondly, and much more importantly, we claim that it is not *necessary* for them to care for how well their opponents fare. If the available resources can be divided in a fair and optimal way[8], then truly rational agents, i.e. agents that do not fall into "egoist traps", who are not prone to submit to short-sighted selfish behaviour patterns will identify possibilities for profitable cooperation. We will show that, under certain conditions, cooperation can even be achieved among purely selfish individuals.

We should still have a brief look at the alternative view of things. This alternative view is best summarised by the notion of *socially responsible agents*, according to which "a decision function can be designed which enables agents to exploit interactions for their own gain, but which means that they are sometimes willing to do things for the greater good (to improve system coherence)" (Kalenka and Jennings, 1995, p. 2). The principle underlying such reasoning is the *Principle of Social Rationality*, as proposed in (Jennings and Campos, 1997):

> "If a member of a responsible society can perform an action whose joint benefit is greater than its joint loss, then it may select that action."

The joint benefit is a combination of the individual benefit some agent obtains for executing some action and the benefit obtained by the society in which that agent is situated,

[8]Whether this is the case of course depends on the nature of the resources: if, e.g. a resource is indivisible, there is nothing more rational for an agent to do than to try and get hold of it irrespective of what others do. It is only fair to point out that in such situations, cooperation cannot evolve.

and a "responsible" society is a society that is concerned with balancing both individual and system goals. The argument for using a combination of individual and social rationality is that with it a hybrid form of agent systems can be devised that lies somewhere *between* DPS and MAS and combines the advantages of both views (Hogg and Jennings, 1997).
In our view, this view is rather inadequate for the study of *open systems* (Hewitt, 1991), for practical reasons given above, but also because we think that it is precisely "the top-down imposition of fictitious coordination mechanisms" (Tesfatsion, to appear, p. 1) that has to be avoided if we want to design flexible, autonomous, adaptive agents able to survive in a multi-agent world.

In Chapter 3 we will provide an abstract formulation of our multiple-access resource-load balancing game which will be used throughout the remaining work to evaluate our learning architecture. We will show that it exhibits certain properties that make it adequate for analysing essential problems that arise in coordinated decisions-making. Thus, apart from representing a class of practically relevant real-world problems, it is suitable for our purpose of evaluating a learning architecture for adaptive behaviour in games.

1.4 Preview of other sections

The following chapters are organised as follows: In Chapter 2, we briefly explain how our work can be seen as a combination of three fields of research and review some examples of research that is concerned with problems similar to the ones we analyse.
Chapter 3 presents the abstract formalisation of the multiple-access resource-load balancing game, for which we also determine optimal solutions.
The subsequent chapter serves as a detailed introduction to the architecture we propose, which is an extension of the InteRRaP paradigm for layered learning in repeated games.
Chapter 5 provides an extensive treatment of an instantiation of that architecture, a proof-of-concept prototype with readily applicable learning algorithms embedded in a concrete game simulation system. A further chapter is devoted to the empirical evaluation of this prototype that proves its adequacy and gives insights to the strengths and weaknesses of our approach.
Finally, Chapter 7 rounds up with central conclusions, a review of achieved goals and open issues and a critique of the work with respect to its theoretical value and its practical relevance.

Chapter 2

Related work

The introductory chapter already hinted at various lines of research that our work is related to. The most prominent of these were machine learning, game theory and load-balancing research in the field of distributed computing.
In this chapter, we take a closer look to (a) three rather general fields of DAI research amidst which our work can be placed and (b) some concrete examples of coordination mechanism design research that are quite similar to our approach.

2.1 Related fields of research

In general terms, our work can be seen as a combination of three strands of DAI research, *multi-agent learning*, *agent-based computational economics* and *social simulation*, in the sense that we borrow techniques from these fields, but also some theoretical positions they assume.

The relation to multi-agent learning[1] (MAL) is the most obvious of these, because the focus of our work is on learning to coordinate with other agents for which fact it stands in contrast to most work in classical game theory. There, agents are assumed to be capable of analysing the constraints governing the interaction situation by conducting a rigorous mathematical analysis of the underlying game (we make references to work in DAI that follows this tradition in Chapter 3). We oppose to this position because we consider a complete mathematical analysis of the underlying abstract games inadequate for realistic situations, in which we are inevitably confronted with the issue of *bounded rationality* (Simon, 1982).
According to this concept (which is enjoying increasing popularity in the DAI community), agents have only limited reasoning abilities, or at least, even if they can in theory solve many highly-complex problems, in practice they have to "trade off the quality of a solution versus the cost of invested computation" (Jung, 1998). In the games we will typically consider, the number of possible action combinations (typically, thousands or millions) makes the application of exact solution methods on the side of the agents (without the designer's knowledge of the payoff function, that is) impossible.
Thus, we believe that a learning approach is much more appropriate in which agents gradually gather information about their opponents and about the interplay between their actions.

[1] Cf. (Weiß, 1996) for an introduction

It should also be pointed out that our approach shares with MAL intuitions the focus on designing and evaluating the adaptive capacities of agents rather than on acting itself. This means that we will not go into any reasoning about how (physical) actions are actually performed, how sensoric data is processed or how actuators should be controlled.

Work in the field of agent-based computational economics (ACE) is in two ways relevant to ours: firstly, we borrow from it the use of game-theoretic abstraction for modelling interactions and its underlying assumptions and, secondly, its focus on evolutionary, emergent aspects of complex system behaviour.
According to Tesfatsion (to appear, p. 1), ACE can be roughly defined as the "computational study of economies modelled as evolving decentralised systems of autonomous agents". He sees as its main purpose "to understand the apparently spontaneous formation of global regularities in economic processes [...], to explain how these regularities arise from the bottom up, through the repeated local interactions of autonomous agent, [...], rather than from the top-down imposition of fictitious coordination mechanisms" (ibid.). This statement might, in fact, serve as a "statement of purpose" for our work, and ACE in general describes most suitably our efforts because it stresses the fact that we are talking about agent-based computer simulations and that the decision-making principles applied are taken from classical utility theory, which lies at the heart of the economists' standpoint.
The main reasons for which we adopt the position of economics are

- its use of game-theoretic models (which provides us with the necessary abstraction to handle complex interaction situations and to gain insights into the essence of interaction, and also offers mathematical solution methods to evaluate our own results) and

- the fact that in economics agents are modelled as individual utility maximisers (this provides a clear and simple working definition of rationality, makes the performance of different agents comparable and allows for the construction of *heterogenous* agent societies beyond the realm of collaborative MAS).

In his argument for a closer interdisciplinary connection between the fields of AI and economics, Shoham (1996) names these two aspects of economics research as two principles from which AI could benefit, and we believe that they are essential to developing an understanding of how worlds inhabited by autonomous, individually rational intelligent agents will evolve.

Finally, it should be remarked that ACE can be seen as a sub-branch of the field of social simulation (also called the artificial societies approach), in which researchers (predominantly social scientists) make use of computer simulations to study the social processes that come about through the local interactions of a great number of agents. As Doran (1998) has pointed out, the novel aspect of such research is that it "enables studies to be made of the relationship between agent-level and society-level phenomena in a way not previously possible."
We stress the importance of this aspect here, because when talking of interaction modelling, it is of twofold relevance. On the one hand, as we have mentioned before, our aim is to endow agents with simple reasoning mechanisms for interaction, and to "see what happens" at system level, i.e. to explain complex system behaviour from scratch. It is precisely this "agent-society leap" that social simulation in its constructivist tradition is

about. On the other hand, we believe that the principles of interaction are crucial to the understanding of this "leap", and that the emergence of interaction patterns is of vital importance for the evolution of the society.

2.2 Coordination mechanism design

In a more specific sense, i.e. as regards the problems that we are concerned with, our work is closely related to research on *coordination mechanism design*. "Coordination mechanism design"[2] can be used as general term for any efforts to conceive, formalise and validate adequate coordination techniques for MASs. The design of such mechanisms in fact constitutes part of almost any MAS realisation, though sometimes (especially in the realm of reactive agent architectures, cf., e.g., Agre and Chapman (1987), Brooks (1991), Dagaeff, Chantemargue, and Hirsbrunner (1997), Drogoul and Ferber (1994)) the focus is more on *emergent* coordination rather than on explicit social capabilities that agents should have.

The most common approach to dealing with coordination issues in MAS called *Multi-Agent Planning* (MAP) is the extension of classical AI planning systems to multi-agent systems (cf., e.g.,von Martial (1993) for an introduction to the field).
However, work from this field can hardly be compared to ours, because it relies on the use of explicit communication (as von Martial (1993, p. 95) puts it, "MAP = Planning + Communication")[3], it contradicts our "no sequential planning"-assumption and, very often, assumptions concerning benevolence or the existence of some central authority (Georgeff, 1984; Lansky, 1989) are made.

Yet there also exist (relatively few) approaches which, in the spirit of the already mentioned work (Genesereth et al., 1986), exclude the possibility of communication and operate in game-theoretic, "non-planning" settings. Here, we review some examples of work on *learning* in *games* that are quite similar to ours.

In (Claus and Boutilier, 1998), the differences between *individual learning* (IL) and *joint action learning* (JAL) are studied. Individual learners learn values for their own actions, thereby ignoring the existence of others, whereas JAL learners learn action values for joint actions, i.e. for action combinations of all players. However, the scope is restricted to strictly cooperative games (unlike in our work), so that the same payoff is distributed to all agents for some particular action combination, i.e. no conflict of interests can arise. The main result is that JAL enables agents to choose the optimal equilibrium (if several equilibria exist), since they have access to the action values of all joint actions once these have been learned, and can simply choose that action that yields the maximal action value.

[2]The term is borrowed from (Fischer, Ruß, and Vierke, 1998), where it is used in a much more specific and *normative* sense, as research concerned with identifying interaction rules for the efficient assignment of goods and tasks to agents, "so that when agents use them [...], desirable social outcomes follow" (Fischer et al., 1998, pp. 57–58). Employing it here is simply due to the absence of a more generic and established term.

[3]Examples for "coordination through communication" include market-oriented approaches based on negotiation (e.g. the contract net protocol proposed by Davis and Smith (1981)), simulated trading (Bachem, Hochstättler, and Malich, 1996) and auction-like mechanisms (cf. Chapter 4 in Fischer et al. (1998) for an overview), voting mechanisms, the exchange of individual plans and collaborative planning (Grosz and Snider, 1988).

Freund and colleagues approach our setting and objective pretty closely, in that they examine "the problem of *learning* to play optimally against an adversary whose precise strategy is unknown" (Freund et al., 1995, p. 1). They also hint at an issue that resembles our considerations about the social potential of a game (cf. Section 1.2.2) by pointing out that an adversary's future behaviour, and with it his willingness to cooperate, will depend on the agent's past actions.
Therefore, there argument continues (ibid., p. 1),

> "[...] it is not enough to simply predict the adversary's actions in order to play optimally; we must also discover how to massage the adversary into his most cooperative state."

They examine three classes of *computationally bounded* adversaries (one of these is similar to the class of adversaries we model in the Strategy Engine, cf. Section 5.3) and present theoretical results on strategy learning algorithms. Most importantly, the authors prove that there exist *efficient strategy learning algorithms* for all three classes of adversaries. Unfortunately, their approach is limited to 2×2-games – games with only two players and two action options – and this is clearly too restrictive for our purposes.

Modelling opponents as finite automata is the object of research in the paper of Carmel and Markovitch (1996). There, it is argued that

> "When looking for an efficient strategy for interaction, an agent must consider two main outcomes of its behavior. First, the direct reward for its action during the current encounter with others. Second, the effect of its behavior on the expected future behavior of other agents."

This is an alternative description of the determinants of an interaction situation to the one we proposed that essentially captures the same aspects of interaction.
To the authors, solving the problem of finding an optimal strategy can be reduced to learning the opponents' automata (for which they present a heuristic algorithm). They account for n-player environments, in which the interactions are carried out between pairs of agents only, so-called 2-player *encounters* (this is a setting that is commonly used in *evolutionary game theory* cf., e.g., Smith (1982), Weibull (1995)).

Finally, the work of Shoham and Tennenholtz (1997) on *social conventions* is notably similar to ours, because its focus is on endowing agents with simple game-learning abilities in settings where no previous knowledge of the game and of others' strategies is available, in order to analyse whether and how social compromise can emerge in the long run and whether it will be stable.
Starting from a system level perspective, they first define *social laws* as restrictions of the action sets available to the players, i.e. a social law, if enforced by some central authority, would prohibit certain actions otherwise available. If such a law restricts the agents' behaviour to one particular strategy, it is called a *social convention*. Switching to the agent level, such conventions would only be accepted by the individuals, if they deemed them *individually* rational. Since the focus is on decentralised systems, the authors do not assume the existence of some central authority and ask how such conventions can emerge in societies of purely selfish agents. For a specific sub-class of two-player games, it is shown that, using very simple action-selection rules, a social convention (that is both individually and collectively desirable) will be reached after some bounded number of

iterations with arbitrarily high probability, and, once reached, it will never be left. This is an important result, because it proves that cooperation can in principle be learned, and this result is strengthened by the authors' extensive empirical results which show that conventions emerge in reasonable time.

For our approach, the main importance of all these works lies in the fact that they give examples for systems in which particular aspects of those crucial to interaction dynamics were learned effectively. This not only shows that it is sensible to look for a more generic learning methodology for interaction, but also that the task is, at least in principle, manageable.

2.3 Summary

This chapter provided an overview of the research areas that our work relates to and a survey of recent publications on similar research. First, three sub-fields of DAI[4] were presented and some central aspects were pointed out that make these fields particulary relevant to our work.
After that, we discussed some contributions that are essentially concerned with solving the same class of problems, even though the assumptions they make are substantially distinct from ours. We believe that the combination of a complete lack of prior knowledge, the absence of communication and the suitability for games with more than two players form a set of assumptions that have not been previously attacked using a hierarchical learning architecture, and this is the novelty of our approach.

In the following chapter, we introduce a game-theoretic formalisation of the application scenario within which our approach will be evaluated, determine optimal solutions and discuss the central interaction problems that it models.

[4]We deliberately avoid going into discussions about whether social simulation of computational economics are sub-fields of DAI or vice versa here.

Chapter 3

Game-theoretic model of the application example

If you play a game of chance, know, before you begin,
If you are benevolent you will never win.
— William Blake, *On Friends And Foes*

In this chapter we introduce some basic concepts of game theory and present a game-theoretic model of our resource-load balancing scenario. We devote a separate chapter to these, because they constitute the mathematical framework upon which we formalise the interaction setting and in which agent's coordination reasoning capabilities will operate. Furthermore, the exact solutions for the resource-load game which we can determine using the tools of game theory serve as a basis for the analysis of the performance of the proposed system architecture: the solutions for the game underlying the interaction simulation provide the benchmarks against which the performance of our agents will be measured.

The chapter is structured as follows: first, only some essential game-theoretic notions are introduced that will be used throughout the remaining chapters[1]. The mathematical model of the application scenario will be provided subsequently, together with a presentation of its optimal solutions. A discussion of the results of this analysis will then point at the complexity of the underlying games to give a flavour for the nature of the tasks agents will have to solve.

3.1 Basics of game theory

Modern game theory saw its advent with the publication of the "Theory of Games and Economic Behaviour" by John von Neumann and Oscar Morgenstern in 1944, in which economic decision situations were modelled within the mathematical framework of games for the first time[2]. The clarity and mathematical rigour with which it enabled the mod-

[1]The reader unfamiliar to the more elaborate concepts of classical game theory, such as dominance relations, equilibria, pareto-optimality and the kernel solution concept may find an extensive introduction to these in Appendix B.

[2]To be more precise, that was the first time such treatment of games received adequate attention – the analysis of games of chance that initiated the development of probability theory actually goes back to as early as the 17th century, and first contributions to the theory of strategic games were made by

elling and analysis of interaction situations and strategic decision-making principles contributed largely to its popularity, not only among mathematicians and economists: it has also been applied to fields as different as psychology (cf. the work of Luce and Raiffa (1958)), biology (Smith, 1982), governmental policy and military analysis (e.g. nuclear disarmament strategies, Aumann and Maschler (1996)) and more.
In computer science, game theory was employed in work on computer security, on network routing protocols and on discourse understanding to name just a few examples (cf. the references in Pfeffer and Koller (1997)), while games have been one of the central areas of interest in (especially early) AI (the Bibliographical Notes to Chapter 5 in Russell and Norvig (1995) provide a good overview of such work). In the last years, it has become *the* predominant interaction modelling tool in DAI (where it was first introduced by Zlotkin and Rosenschein (1989)).

What is game theory? It is essentially that sub-branch of decision theory that is concerned with describing and analysing decision situations in which the actions taken by a number of agents have a mutual impact on each other, and to provide solution methods for these. General decision theory strives to build mathematical formalizations of decision situations and of notions such as "solution" and "rationality" for such situations. Game theory inherits these aims, while trying to give adequate answers to these problems for a particular subclass of decision problems, namely that in which the consequences of actions depend on *several* agents' actions.

In the following, we introduce some notions essential to an understanding of abstract games, which will be illustrated by two very popular 2-player games, the Prisoners' Dilemma (PD) and the Coordination Game (CG). These examples provide some very interesting insights to central game-theoretic problems, and we will show later that these problems are also relevant in the games we use to model our application scenario.
Then, we will have a look at some points of critique and we will describe how they explain the stance we have assumed for our research (which is somewhat different from the game-theoreticians' standpoint, as has been pointed out before).

3.1.1 Basic definitions – games and strategies

Games are abstract formalisations of the interdependencies between co-acting agents. Formally, they are given by the set of participating agents, by the sets of actions that players can perform and by the outcomes of collective action combinations for each player (expressed as numerical payoffs). In this work, we will restrict ourselves to n-player games in *normal form*, i.e. games that consist of a single move of all players and a subsequent distribution of the respective payoffs to the players. Games which consist of several intermediate steps and in which the payoffs are only distributed after the last step (so-called *extensive form games*) can be shown to be equivalent to normal form games[3], so that the solution concepts presented in Appendix B also hold in the extensive form case. We begin by defining games in normal form and repeated games.

Zermelo at the beginning of the 20th century.

[3]This is done by viewing the extensive form game as a normal form game: every possible combination of players' decision sequences (every path in the game tree) defines *one* joint action combination that leads to a final state (a leaf node) which has a payoff distribution for all players associated with it. Thus, a payoff matrix can be defined over these player decision sequences and the respective final outcomes

Definition 3.1 (Normal Form Games, Repeated Games)
*An n-***player game in normal form** $\Gamma = \langle N, S, u \rangle$ *is defined by:*

- *the* **set of players** $N = \{1, \ldots, n\}$,
- *a finite, non-empty* **set of strategies** $S_i = \{s_{i1}, \ldots, s_{im_i}\}$ *for each player i. The collection $\{S_i\}_{i \in N}$ of all players' strategy sets is called the* **strategy space** *of the game, and a conjoint strategy selection of all players $s = (s_1, \ldots, s_n) \in S = \times_{i \in N} S_i$ is called a* **joint strategy**[4].
- *a* **payoff function** $u : \times_{i \in N} S_i \to \mathbf{R}^n$ *which assigns a payoff vector $u(s) = (u_1, \ldots, u_n)$ to each joint strategy s, such that u_i is the payoff that player i will receive.*

*A t-***repeated n-player game** $\Gamma^t = \langle s^{(1)}, \ldots, s^{(t)} \rangle \in S^t$ *is a finite sequence of t* **rounds**, *i.e. a series of joint strategy choices in a game $\Gamma = \langle N, S, u \rangle$.*

Note that these definitions already make some important assumptions: firstly, the payoff one player receives does not rely on anything else than the simultaneous actions of its adversaries (in particular, it does not depend on current world states); secondly, this same dependency holds for every enactment of the game, if several rounds are played. Furthermore, repeated games only allow for carrying out discrete encounters of synchronously executed actions from finite action pools, which means that they do not provide means to model continuous or asynchronous interactions. While this may seem inadequate for modelling realistic interaction situations, it is only these restrictions that make a tractable analysis of game-theoretic models possible. This also explains why we adopted these assumptions for our architecture in the first place (cf. Sections 1.2.2 and 1.3.2).
Since we will make use of that term later, we should also introduce the notion of *mixed strategies.*

Definition 3.2 (Mixed Strategy, Expected Payoff)

- *A* **mixed strategy** σ_i *of a player i in a n-player game $\Gamma = \langle N, S, u \rangle$ is a probability distribution over the (pure) strategies $s \in S_i$ of player i, i.e.*

$$\sigma_i : S_i \to [0; 1], \qquad \sum_{s \in S_i} \sigma_i(s) = 1.$$

  The set Σ_i of such mixed strategies is called the **mixed strategy space** *of i.*

- *We write e_i^k for the k-th pure strategy of i seen as a mixed strategy (that assigns probability 1 to strategy s_{ik} and probability 0 to all other pure strategies)*[5].

[4] We shall use s_i as a short form for player i's strategy in a strategy tuple s by virtue of its position in the cross-product S. We will also make use of the common abbreviation $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$ for the joint strategy of all players other than i and in writing $s = (s_i, s_{-i})$ we will ignore the proper ordering of tuples in S for reasons of convenience (sometimes we will use the same method for abbreviating joint strategies of "remaining players" given some subset of player ($M \subset N$) by writing s_{-M} in a similar fashion).
More generally, we use capital letters A, B, C for sets and their small letters a, b, c for elements of these sets (function names will be small letters for which respective sets have not been introduced). If a set A is a n-long cross-product, we will write A_i for the i-th set in this cross-product, and A_I for some index set $I \subseteq \{1 \ldots, n\}$. We use the notation a_i (or $a[i]$) and a_I in the same way for some tuple $a \in A$.

[5] The intuition behind using the letter e is that e_i^k can be interpreted geometrically as the unit vector of the k-th dimension in $|S_i|$-dimensional hyperspace.

- *A joint probability distribution $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma$ of all players is called a* **mixed joint strategy** *and $\Sigma = \times_{i \in N} \Sigma_i$ is the* **mixed joint strategy space** *of all players.*

- *The* **expected payoff** *of some mixed strategy σ_i is computed as the sum of the respective pure joint strategies weighted by the probabilities that these will actually be played:*

$$u_i(\sigma) = \sum_{s \in \times_{j \in N} S_j} u_i(s) \cdot P(s),$$

where

$$s = (s_1, \ldots, s_n) \Rightarrow P(s) = \Pi_{1 \leq j \leq n} \sigma_j(s_j)$$

is the joint probability that all players (including i) will play s.

3.1.2 Two illustrative examples

The two games we present as examples are both two-player games in which both players may choose from two action alternatives, i.e. $|S_1| = |S_2| = 2$, and both are so-called *non-constant sum games*, which means that the sums of payoffs received by both players under two different joint strategies differ, or, more formally,

$$\exists s, s' \in S. \sum_{i=1}^{2} u_i(s) \neq \sum_{i=1}^{2} u_i(s')$$

holds. In contrast to constant-sum two-player games (where one player inevitably loses what the other wins), this class of games is not strictly non-cooperative, because there is no "complete clash of interests" (Fischer et al., 1998, p. 32). This means that utility pairs can be preferred by both players conjointly from which at least one player will profit, while the other will not be at a disadvantage.
It is for this reason that *cooperative*[6] games offer a much wider spectrum of possible solution concepts and deeper insights into aspects of rational behaviour, especially when carried out repeatedly.
The Prisoner's Dilemma (PD) due to Luce and Raiffa (1958) is probably the most famous such game. In it, each player has the option to cooperate (C) or defect (D), and the key property of the game lies in the fact that if both cooperate they will receive a lower payoff than they would have obtained if they had defected while the other cooperated. The actual "dilemma" comes about, because if both defect (which they will, given the condition just mentioned) they both receive a lower payoff than if they had *both* cooperated.
The following table shows a typical PD payoff matrix. In each matrix position, the payoff for player one (the "row" player) is followed by the payoff for player two (the "column" player).

player 2 / player 1	cooperate	defect
cooperate	(3,3)	(0,5)
defect	(5,0)	(1,1)

[6]Many authors (see, e.g., Fischer et al. (1998)) require that two further conditions hold for a game to be called cooperative. Firstly, that players have some means of communication to make obligatory commitments and that there exists some "higher authority" which is capable of enforcing these commitments (or punishing their violation) in the second place. We will use the term here for any non-constant sum game, taking as a key property of cooperative games the existence of mutually beneficial possibilities of *coordination* rather than the possibility of establishing reliable agreements.

It can be seen that it only pays for a player to cooperate if her adversary cooperates as well, and therefore in a situation devoid of prior agreement, each player is sure to choose the defect strategy. This is because – under the worst circumstances – that strategy will still yield a higher payoff than the option of cooperating.

Using this "worst-case" argument as a basis for action selection is commonly known as the *maximin* principle. Formally, it is the action selection rule by which i chooses to play

$$s_i^* = \arg\max_{s_i \in S_i} \min_{s_{-i}} u_i(s_i, s_{-i}),$$

i.e. that action which ensures the highest payoff if all other players act in the most undesirable way (and is hence a very *pessimistic* rule).

The Coordination Game first mentioned by Lewis (given by the matrix below) addresses an entirely different problem of joint action selection. Here, neither of the players can be exploited by cooperating while her opponent defects, but both players only profit from situations in which they simultaneously pick the same strategy. Thus, players may obtain low payoffs by making the wrong choice, although there is no conflict of interests.

player 2 / player 1	A	B
A	(1,1)	(-1,-1)
B	(-1,-1)	(1,1)

These two games, albeit very simple, already hint at some central issues in coordination mechanism design. The CG may seem to depict a much simpler coordination problem than the PD, but looking at repeated versions of the two games shows that this is not really the case: in the Iterated PD (IPD), fairly simple strategy-selection rules such as TIT FOR TAT[7] often suffice to force the opponent into a cooperative stance by punishing her ambitions to exploit; for the CG, simple strategies like "playing what the other played the round before" may fail hopelessly if coordination is not achieved in the very first round.

3.1.3 Shortcomings

Despite the undisputed value of game theory as a tool for the mathematical modelling of interaction situations, many points of criticism have been put forward against it. Here, we will address only those that are significant for our work and some open questions that our methodology attempts to investigate (pointers to more comprehensive critical works can be found in Lomborg (1994)). We assume that the reader is familiar with the concept of the Nash equilibrium and with the kernel solution method for cooperative games (Appendix B provides an introduction to these and to other simple solution concepts that we make reference to in the following).

One problem of classical solution concepts for games is that of *selecting* the stable state of behaviour that will actually converge. When there exist various equilibria or

[7] TIT FOR TAT is an action-selection rule that requires the player (i) not to be the first to defect and (ii) to consistently play whatever the opponent played in the last round. evolution (1984) proved by conducting a round robin tournament of various strategies that this is a very robust strategy (i.e. a strategy that is very successful against a great number of other strategies).

the kernel comprises more than one imputation, it is very hard to predict which of these will actually emerge. Moreover, it is not even clear whether the players' behaviour will converge to any of them, or, as stated by Lomborg (1994, p. 2),

> "[...] in having more Nash equilibria, one not only loses the possibility of precise prediction, but the players themselves will no longer know what to choose, and thus the resulting outcome need not even be a Nash equilibrium."

This is a particular problem if there is no explicit communication among players and only reactive or very risky initiatives to form coalitions are possible rather than enforceable agreements (just remember the CG example). It follows from this observation that most of the work in classical game theory heavily relies on players being able to communicate, so that it overlooks a large portion of real-world problems in which this is not possible.

Another issue that has been largely ignored is that of bounded rationality. In realistic settings, it is rather unlikely that agents will have the computational resources to solve games mathematically, especially in games with many players and large strategy sets. Solution concepts such as the kernel assume that actors can compute solutions of simultaneous equations in several unknowns, e.g. by using linear optimisation methods. Apart from the fact that agents might not have the time and reasoning capacities to employ such methods, is also appears rather counter-intuitive that humans go about solving interaction problems with the use of these. They are much more likely to assume the *decision-analytic* stance, which stands in contrast to the game-theoretic view as Myerson (1991) pointed out:

> "The decision-analytic approach to player i's decision problem is to try to predict the behaviour of the players other than i first, and then to solve i's decision problem last. In contrast, the usual game-theoretic approach is to analyze and solve the decision problems of all players together, like a system of simultaneous equations in several unknowns."

This brings us directly to another shortcoming of classical game theory, namely that it is too often restricted to a too *normative* or too *descriptive* view. The primary goal of normative decision theory is "to provide instructions for an agent to act in a rational manner when the premises of the decision are given" (Fischer et al., 1998, p. 13) while pre-assuming that agents are fundamentally rational. As Fischer et al. point out (ibid.), "the models used do not consider how to obtain the premises of decisions, how to obtain the necessary informations or how the decision process is influenced by the environment", and are therefore referred to as *closed models* which contradicts the open systems of situated and adaptive agents which are the object of research in MAS. Descriptive decision theory, on the other hand, "starts with the description of real decision behaviour of people in decision situations and analyses the consequences of decisions" (ibid., p. 14). But although the models used in descriptive decision theory are *open models*, they are equally inadequate for our purposes, since they fail to provide any concrete guidance for agents in interaction situations.
Therefore, these authors argue that there is a need for a synthesis of these, a *prescriptive* decision theory that will enable agents to solve their own decision problem, and this observation is in fact a key motivation for developing our own theory.

In the context of such prescriptive models, *adaptivity* is also an issue that has been neglected by most game-theoretic approaches. In fact, the possible need for adaptation

has (deliberately) been excluded *a priori* by defining games as fully *static* models of interaction, thereby avoiding such questions as "what if the payoff function changes?" or "what if opponents suddenly change their strategies?" We believe that these questions play an important role when it comes to devising effective interaction mechanisms and therefore adaptivity is at the focus of our methodology.

It should be remarked that this last point of criticism is only one facet of what is probably the most important shortcoming of game theory: the fact that it has largely overlooked the *epistemic* aspects of interaction – knowledge, information and uncertainty. In most of the game-theoretic literature, for example, full knowledge of the payoff matrix (even of the opponents' payoff function!) is pre-assumed. Some authors account for games in which there is uncertainty about *which* of several possible games is being played (cf., e.g., Aumann and Maschler (1996), Pfeffer and Koller (1997)), but there has been no attempt to answer the question "what if players know nothing about the payoff structure of the game?" Yet knowing the payoff function (which implies knowing what payoffs other agents obtain during the game, if their actions can be recognised) seems to be a very strong assumption that does not hold in most real-life situations.

A similar constraint is imposed by the proposed *postulate of rationality*, which asserts that agents (have enough information to) behave rationally and that they know that their opponents are rational, too.
Finally, it should be mentioned that within game-theoretic research, the issue of uncertainty and *noise* in information has received too little attention. Lomborg (1994, p. 6) remarks that all of the available literature focuses on noise as *misimplementation* (i.e. the failure of players to carry out selected strategies), whereas *misperception* (the false perception of opponent actions or received payoffs) has been completely neglected. However, noisy data is ubiquitous in natural (and artificial) environments, so this adds yet another unrealistic restriction to the ones mentioned before.

All these critical arguments show that "something is utterly amiss" (Lomborg, 1994, p. 2) in game theory. Recent developments in the already mentioned works on evolutionary game theory (Smith, 1982) and the experimental approach taken by evolution (1984) but also some work on bounded rationality in games (Rubenstein, 1985; Freund et al., 1995) have succeeded in partially alleviating these problems. Our learning architecture can be seen as another contribution to this line of research, as we question virtually all of the assumptions of classical game theory that lead to the presented problems.

3.2 Game-theoretic abstraction of the application example

In this section we introduce the resource-load game as it has been discussed in Section 1.3.2 formally and provide optimal solutions for it in order to measure the performance of the agents we will devise. Unless where stated otherwise, using it as an example does not affect the generic character of our approach.
The payoff function we have constructed is very simple, yet it exhibits some very interesting properties, by which it combines the difficulties of games like the CG, in which equilibrium selection and action *synchronisation* play an important role, and those of games belonging to the class of the PD, where *egoist traps* tempt the players to act in

an uncoordinated and inefficient way, even though the games as such are cooperative. In that, the simplicity of our game allows for a mathematically tractable straight-forward analysis, so that optimal solutions (the benchmarks for agent performance) can be easily determined.

3.2.1 Formal model

The multiple-access resource-load game (MARLOG) is a repeated n-player game in normal form, in which the possible strategies of every agent are given by the subset of resources it accesses. This subset can be determined arbitrarily by the agent in each round by choosing from a finite pool of abstract resources (representing e.g. Web search engines, application servers in local area networks (LANs), or CPUs in a multi-user machine). Each of these resources is assumed to have a particular value for the agent and accessing it produces some cost (e.g. the computational cost of a HTTP request, of connecting to a socket in the network, or that of spawning a process to the process queue).
The most important feature of these resources is that the value they distribute to their accessors decreases with the number of accessors; each resource is associated with some access time for each agent, which represents the "response delay" induced by communication links (caused by the Internet routing procedure, packet transport in local networks, or disk and RAM access times respectively). This access times obviously increases if other agents access that same resource in the same round; typically, it will increase non-linearly, i.e. accessing three resources with three users yields less payoff than accessing two resources with two users each – the effects of the previously discussed resource divisibility limitations (cf. Section 1.3.1).
We will now define the non-repeated version of this game in a more general form than the one we will use later on to give an idea of the intuition that stands behind it.

Definition 3.3 (The General Multiple-Access Resource-Load Balancing Game)
A **general multiple-access resource-load balancing game** $RL = \langle n, k, v, c, T, \varepsilon \rangle$ *is defined by:*

- *the* **set of agents** $\mathcal{P} = \{1, 2, \ldots, n\}$,
- *the* **set of resources** $\mathcal{R} = \{1, 2, \ldots, k\}$, *the power set* $2^{\mathcal{R}}$ *of which denotes the strategy set (a particular strategy is given by the subset of* $\mathcal{R}$ *that represents the resources accessed by the agent);*
- *a* **set of actions** $A = \{0, 1, \ldots, 2^k - 1\}$ *can be defined as the strategy set of any player* i *accordingly by virtue of the mapping* $f : 2^{\mathcal{R}} \to A$ *where*

$$\forall Q \subseteq \mathcal{R}. f(Q) = \sum_{r=1}^{k} 2^{r-1}.$$

We also use $\beta : A \to \{0; 1\}^k$ *as the* k*-bit* **binary encoding function**[8] *defined by*

$$\sum_{r=1}^{k} \beta(a_i)[r] \cdot 2^{r-1} = a_i$$

[8] $\beta(a)$ is the binary representation of $a \in A$ that is one for all resources $r \in \mathcal{R}$ that are accessed by choosing that action, and 0 for all other resources

for each[9] $a_i \in A$.

- *A* **resource value function** $v : \mathcal{P} \times \mathcal{R} \to \mathbf{R}^+$, *an* **access cost function** $c : \mathcal{P} \times \mathcal{R} \to \mathbf{R}^+$ *and an* **access time function** $T : \mathcal{P} \times \mathcal{R} \to \mathbf{R}^+$.
  For any agent $i \in \mathcal{P}$ *and resource* $r \in \mathcal{R}$, *the non-negative values* $v(p,r)$, $c(p,r)$ *and* $T(p,r)$ *are called the* **value**, **access cost** *and* **access time** *of resource* r *for agent* i, *respectively.*

- *A* **payoff function** $u : \mathcal{A} \to \mathbf{R}^n$ $(\mathcal{A} = \times_{i \in \mathcal{P}} A = A^n)$ *given by*

$$u_i(a) = \sum_{r=1}^{k} \beta(a_i)[r] \cdot \left(\frac{v(i,r)}{\left(\sum_{j=1}^{n} \beta(a_j)[r]\right)^{\varepsilon(i)} \cdot T(i,r)} - c(i,r) \right),$$

  where $\varepsilon : \mathcal{P} \to \mathbf{R}^+$ *is an* **accessor-resource delay factor**.

Let us briefly explain how the rather complex calculation of the payoff vectors comes about. If i plays a_i, then the binary encoding $\beta(a_i)$ of that action will be 1 in all positions of that binary vector that correspond to the resources that are in the subset $Q \subseteq \mathcal{R}$, i.e. the resources i chooses to access, and 0 for all non-accessed resources. Player i assigns some positive value to each resource, and specific access costs and access times are associated with each $r \in Q$ for i. The actual access time that results from the attempt of several other agents to access that same resource is computed by taking the number of those accessors (including i) to the power of some delay factor $\varepsilon(i)$ (representing the degree to which the overhead of distributing the resource delays the actual access time in the concrete environment) and multiplying the access time constant $T(i,r)$ by it. The maximal value $v(i,r)$ of the resource is then divided by this product and $c(i,r)$ (the basic cost of accessing the resource) is subtracted from the result.
Finally, quite naturally, the agent only receives a payoff for those resources it has actually accessed (this is expressed by multiplying the *resource payoffs* (the quantity in round braces) by the bit at position r in the binary encoding of a_i).

This definition allows for the modelling of a great number of resource-sharing applications, although it is subject to severe restrictions. Most prominently, the fact that in the repeated form of this game only identical interaction situations can be modelled, and that these must consist of simultaneous joint agent actions, seems very unrealistic, especially because it assumes that agents have identical *needs* for resources in each round, so that no notion of task-orientation can be incorporated in our analysis. This can be partly alleviated by changing the game (from outside) arbitrarily at some point and to see whether agents will be able to adapt to new circumstances, and we will investigate this issue in our architecture, but this does not constitute a satisfactory solution to the fundamental problem.
A second, more subtle problem is that the resource distribution limitations we assume are completely theoretical and do not relate in any way to real-world distributed systems. With this respect, we can only remark – in defense of our model – that other authors, e.g. Thomas and Sycara (1998), have proceeded in a similar fashion guided by the need

[9] In the same manner as s and σ were used to distinguish pure strategies from mixed strategies, we shall use α_i, α_I and α to denote mixed strategies of individuals, subsets of players and of all players, respectively. Although A is identical for every player i, we often write e.g. $a_i \in A$ to stress that we are talking about the action of player i.

for a simple mathematical model of resource-load balancing problems.

If we let $\mathcal{P} = N$ and $\forall i.S_i = A$ such that $S_1 = S_2 = \ldots, S_n$ in the definition of normal form game, then the MARLOG represents a finite n-player game in normal form, and expanding it to a repeated game can be done in the usual way.
In the following analysis, we will make some simplifying assumptions for the game just introduced, to make the mathematical arguments more concise and simpler to follow. To this end we define the notion of the *simple* MARLOG.

Definition 3.4 (The Simple Multiple-Access Resource-load Balancing Game)
A general multiple-access resource-load game $RL = \langle n, k, v, c, T, \varepsilon \rangle$ *is called* **simple**, *if the following conditions hold:*

- *The value, access cost and access time functions can be reduced to constants* $v, c, T \in \mathbf{R}^+$, *and the access time delay factor is 1 for all players and resources. Then we can rewrite the payoff function* $u : \mathcal{A} \to \mathbf{R}^n$ *as*

$$u_i(a) = \sum_{r=1}^{k} \beta(a_i)[r] \cdot \left(\frac{v}{\sum_{j=1}^{n} \beta(a_j)[r] \cdot T} - c \right),$$

  and the game is reduced to a structure $RL = \langle n, k, v, c, T \rangle$.

- *the minimal payoff per resource and player is always positive:*

$$\frac{v}{n \cdot T} - c > 0.$$

Apart from the fact that replacing the functions v, c and T by constants yields a much simpler formulation of the game, all these conditions make the analysis of the game much easier, and we will assume that they are fulfilled in the arguments to follow.

Before going into the specific properties of the game, let us present an example (of its simple version) that will give an intuitive feeling for them. The matrix below shows the payoff function for a two-player variant of the game with $k = 2$ where $v = 30$, $c = 2$ and $T = 3$. To make the computation of the individual payoffs more obvious, the actions 0, 1, 2, and 3 have been replaced by their binary encodings, so that it can be seen from these which resources are being accessed.

player 2 / player 1	00	01	10	11
00	(0,0)	(0,8)	(0,8)	(0,16)
01	(8,0)	(3,3)	(8,8)	(3,11)
10	(8,0)	(8,8)	(3,3)	(3,11)
11	(16,0)	(11,3)	(11,3)	(6,6)

The matrix entries are basically predetermined by two values of the payoff of one resource: the one that corresponds to one accessor (=8) and the one that is assigned to an agent if both agents access that resource (=3). From these two values all matrix positions can be deduced by combining the respective resource payoffs for every joint action combination. The first observation that can be made is that the '00' strategy is strictly dominated by all other pure strategies; '01' and '10' are strictly dominated by the '11' strategy while neither of these dominates the other. This leads to the observation that ('11','11') is the

only (strict) Nash equilibrium, because '11' is the best reply to any opponent strategy for both players. However, as in the PD game, this equilibrium is not pareto-optimal: ('01','10') and ('10','01') provide for both players a payoff higher by two units than that obtained when playing the equilibrium. A further difficulty springs from the fact that there are *two* such optima, thus confronting players with the same problem as in the CG. The example shows that by using a rather simple game definition we have succeeded in designing an interaction setting in which actors are faced with a problematic decision situation.

3.2.2 Optimal solutions

It can be shown that the properties of the payoff function just observed for a simple example hold in general: there always exists a single, strict Nash equilibrium that is sub-optimal compared to a family of pareto-optimal solutions. This family of collectively rational behaviours ensures a much higher payoff to *every* player than the equilibrium. Here, we will only present the respective theorems for the general case and discuss their consequences informally. An extensive account of the rather lengthy mathematical proofs can be found in Appendix C. We have chosen not to include them here, since their technical details are not of any conceptual relevance for the methods we develop, given that the optimal solutions will only be used to measure the performance of our agents.

The first two preliminary observations that can be made for any simple MARLOG concern the existence of a single, strictly dominant strategy for each player and the fact that accessing no resource is always a "dumb-ass" alternative, since by choosing to do *nothing* the agent invariably sacrifices some positive payoff.

Lemma 3.2
For any player i,
(1) the strategy $all_i = e_i^{2^k-1}$ strictly dominates all other strategies and
(2) the strategy $none_i = e_i^0$ is strictly dominated by all other strategies.

Proof: cf. Appendix C, Section C.2.

■

It is straightforward to derive the following corollary by using observation (1) of this lemma:

Corollary 3.2
The set of Nash equilibria of the simple multiple-access resource-load game $RL = \langle n, k, v, c, T \rangle$ is given by

$$\Theta(RL) = \{all_{\mathcal{P}}\} \quad , \tag{3.1}$$

that is the singleton set containing the joint combination of the '$2^k - 1$'-th pure strategy of all players (the joint strategy in which all players access all resources).

Proof: cf. Appendix C, Section C.2.

■

Now we have shown for the general case what we had already observed in the 2-player example: that the greedy strategy of all players accessing all resources at the same time is

the only strict Nash equilibrium. This means that, at least when using maximin agents, we can expect no better system performance than the one which occurs in a situation in which resources are hopelessly overloaded. To find better solutions, let us present a theorem that determines kernel payoff vectors, i.e. individually and collectively rational payoff distributions.

Theorem 3.2
Let the payoff vector $u^* = (u_1^* \ldots, u_n^*) \in \mathbf{R}^n$ be defined as

$$\forall i \in \mathcal{P}.u_i^* = \frac{k}{n} \cdot \left(\frac{v}{T} - c\right) \tag{3.2}$$

The vector u^* is in the kernel of the game $CF(RL)$ where RL is an arbitrary simple MARLOG in normal form.

Proof: cf. Appendix C, Section C.3. ∎

Let us look more closely at these payoff vectors. What they imply, is that

1. each resource provides the highest value to the society if it is accessed by *one* player (this can be inferred from the fact that k quantities $\left(\frac{v}{T} - c\right)$ are distributed amongst players – the denominator T shows that each of the resources is accessed by exactly one player) and that

2. the total payoff the global coalition $\mathcal{P}$ earns is distributed *evenly* among its participants (dividing it by n assigns the same amount of payoff to every i), for which reason we will call this vector the *fair* payoff distribution.

So in any simple MARLOG there exist pareto-optimal solutions in the global coalition that are more beneficial to each player than their own selfish strategies[10]. However, a major problem arises due to the two basic assumptions made by cooperative game theory. One of these is that players are able to communicate with each other and thus able to make agreements and to form coalitions. A second is that the distribution of coalition payoffs is "decoupled" from the obtainment of the profit, i.e. payoffs can be transferred between players. These assumptions are necessary, because in the context of cooperative game theory, "solutions" are seen as payoff *distributions* rather than as the payoff *outcomes* of particular joint actions (that are pre-determined by the payoff function) – sharing profit reasonably *after* it is actually obtained by coalition members is considered equivalent to achieving the same payoff distribution by virtue of the payoff function.
In the communication-less settings we examine, the fact that payoff distribution is *not* decoupled from the payoff function implies that if the proposed pareto-optimal u^* vectors are going to be realised, then they will have to be the payoff vectors of joint actions, because there is no way of transferring utilities between agents and no communication to make agreements about such transfer.

[10] These are not the only solutions in the kernel of the game – in fact, there exist many payoff vectors that distribute a total sum of

$$v(\mathcal{P}) = k \cdot \left(\frac{v}{T} - c\right)$$

to the society and are still individually and collectively rational. However, the solutions presented in Theorem 3.2 alone have the property that each agent receives the *same* payoff. Since all agents are equal in "power" in our setting, it is unlikely to assume that they would participate in coalitions by which they would obtain lower payoffs than some of their opponents.

So the question that remains to be answered is whether there exist such joint strategies. If we consider pure strategies only, then – in the non-repeated version of the game – this is only possible if the resource set can be distributed among the agents in a fair way, i.e. if

$$\exists k_0 \in \mathbf{N}. \frac{k}{n} = k_0 \quad .$$

In that case, every agent can access k_0 resources, such that each resource in $\mathcal{R}$ has load one, and thus the sum of all agents' payoffs will equal $\sum_{i \in \mathcal{P}} u_i^*$. If the resources cannot be divided amongst agents (especially if $k < n$), then obviously some agent will receive less payoff than its adversaries, and since the "access all" strategy is dominant, it is very unlikely that this agent will obey the agreement.
Allowing additionally for mixed strategies, the picture changes. In that case, even if the number of resources is not divisible by the number of agents, it can be left to every agent to access (solely) all resources in every n-th round and to access none in all other rounds (this is just one way of achieving the maximal coalition profit in every round), so that such joint strategies always exist.
In both cases, however, there will be a multitude of possible action combinations that have the above properties. In the pure strategies case with divisible resource sets for example, there are

$$\binom{k}{k_0} \cdot \binom{k-k_0}{k_0} \cdot \quad \ldots \quad \cdot \binom{k_0}{k_0}$$

possibilities to achieve the pareto-optimal resource allocation (even for the small example of $k = 12$ and $n = 3$, this yields 34650 combinations!). It is therefore a highly non-trivial problem for agents to settle on one of them, especially if they are unable to communicate.

3.3 Summary

This chapter provided a mathematical model of the application scenario which we use to validate the systems we will design. A brief introduction to some of the necessary game-theoretic definitions was given together with a discussion of certain drawbacks of classical game theory that justifies why we aim at devising a *learning* architecture to achieve effective behaviour in repeated games. Most importantly, we attempt to tackle the problems inflicted by the strong assumptions made by game theory concerning knowledge, information and rationality.
The formal definition of the multiple-access resource-load balancing game was presented subsequently that serves as a stylised mathematical model for resource-sharing environments. We believe that – given the properties we provided rigorous proofs for – it captures many essential problems that can occur in game-theoretic decision-making, especially the existence of multiple optimal solutions that are non-equilibrium states. In addition to these mathematical properties, the absence of communication makes the emergence of effectively coordinated behaviour seem even more unlikely, and it will be a major challenge to overcome these difficulties with the methodology we propose.

In the following chapter, we lay out the foundations for this methodology. Learning tasks are determined whose attainment will lead to effective behaviour in games and they are brought in relation with each other by means of a generic layered learning theory. Then, starting from the InteRRaP architecture, we propose learning extensions for these tasks and integrate these in an adaptive hierarchical agent architecture.

Chapter 4

A layered learning architecture for adaptive behaviour in games

The ability to learn is usually considered one of the key characteristics of intelligent beings. People learn from their observations and from knowledge that is communicated to them by others; they learn from mistakes, by "trial-and-error", by imitating others, from practice and by habit. They generalise over past observations, evaluate their past decisions and draw previously undiscovered conclusions from their knowledge. And these adaptive capacities have certainly contributed largely to the "success" of the species.
Of course, the manifold ways in which humans learn, ranging from "the trivial memorization of experience [...] to the creation of entire scientific theories" (Russell and Norvig, 1995, p. 525) do not directly translate to a single computational concept, but defining learning by the *effects* it should typically have on the learner allows for a simple working definition for machine learning as put forward by Mitchell (1997, p. 2):

> A computer program is said to *learn* from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

One of the foremost aims of DAI research is to build *situated* agents capable of operating without external guidance. Faced with the contingencies of dynamic environments, such agents will need to adapt to changes, to reason on incomplete knowledge and they should degrade gracefully when they encounter difficulties in carrying out their tasks.

In this chapter, a general model of hierarchical learning is introduced that builds upon the mechanisms provided by multi-agent learning and layered learning. The aim of this model is to enable the decomposition of the complex learning problems that occur in MAS in a very flexible way so that learning hard problems can (a) be reduced to learning a number of simpler problems and (b) distributed among several learners to improve accuracy or to achieve a speed-up of the learning process.
The presentation of this abstract distributed learning paradigm has to be thought of as an attempt to describe the intuition that stands behind the actual layered learning architecture for interaction learning. The specific instantiation of the layered learning methodology we present is an extension of the hybrid InteRRaP agent design paradigm for learning adaptive behaviour in games and constitutes the main subject of this chapter, as it will be used throughout the rest of this work.

4.1 Layered learning agents

Weiß (1996) distinguishes between three classes of multi-agent learning mechanisms: *multiplication*, *division* and *interaction*. These provide three basic categories for how distributed learning can be organised *horizontally* in a multi-agent system: learners either manage the same learning task in parallel independently, they are responsible each for a different sub-problem or interact to influence each other's learning process or to find a common learning strategy.
Layered learning as proposed by Stone (1998), on the other hand, extends monolithical learning in a *vertical* dimension, such that super-layers receive either the training examples, the learning inputs or the learning outputs by their sub-components. As pointed out in Section 1.2.2, we consider these three very specific patterns of interaction between layers as too restrictive and we hence prefer to view the relationship between layers of learning units in terms of *control*, i.e. that super-layers delegate tasks to their sub-components, integrate the results of these components, select from their results or use their training data, inputs or outputs (thus this definition encompasses the three specific cases just mentioned), and the term "layered learning" as used in the following will always refer to this definition.
The learning methodology we propose here is the combination of these two dimensions. Assume a learning component L has to be devised whose (global) learning goal is G. In our framework, L will typically consist of some set of learning units $L_1, \ldots, L_n$, and for any two L_i, L_j one of the following relationships holds:

- L_i is a *sub-component* of L_j: L_j may use the internal data of L_i (including its stored training data, input and output representations, hypotheses and performance measures); in more extreme cases of "inclusion", L_j may control L_i (it sets it learning parameters, controls its training process (training iterations, available training time) or delegates learning problems to it).

- L_i is a *co-learner* of L_j: in that case, neither of the two is a sub-component of the other, but there is some connection between them via a third component or because they interact (by negotiation or simply by exchanging data).
  If L_i and L_j are connected to a common super-component L, they either compete for selection of their learning results (if they have the same learning goal, i.e. if $G_i = G_j$) or contribute different sub-results to the learning process of L (if they have different learning goals).
  If they have a common sub-component l, then they either both control this sub-component (or compete for its control) or both receive data from it.

- L_i is *independent* of L_j: they neither share super- nor sub-components and don't communicate in any way (the trivial case).

These three relations provide a set of construction methods for hierarchical learners: the hierarchy is a result of the "sub-component of"-relation, so that the units of one layer are the building blocks of a learning unit at the higher layer. The super-layer may simply bring together the results of the sub-learners. Alternatively, it can act as a meta-layer by deciding which of the sub-layer results to choose or by directly controlling the sub-processes.
The framework also allows for simple multi-plied learning by allowing for the co-existence of learners that are not sub-components of each other, so that their learning processes

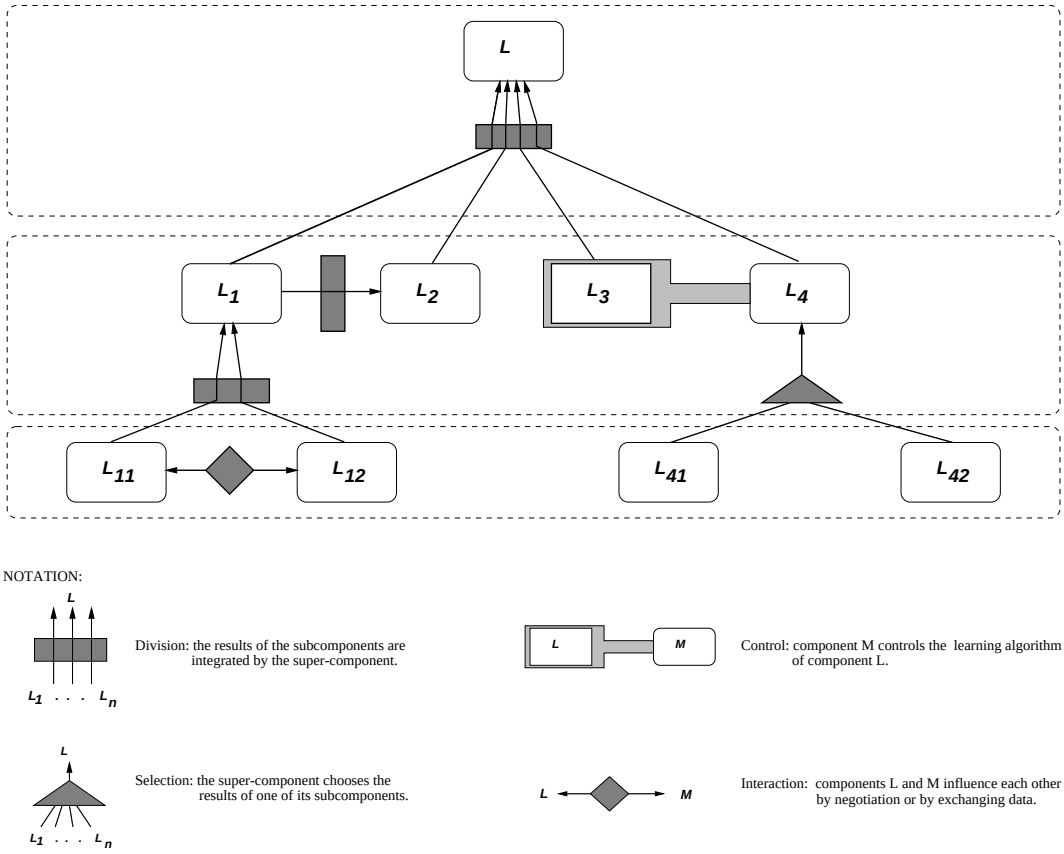


Figure 4.1: Sketch of a layered learning model for a "route selection" learning component L. Each learning unit is a node in the learning graph. The layers resulting from the hierarchical structure are shown by the dotted lines.

are independent[1]. This *parallelism* in learning adds the mentioned horizontal view to the framework.
Note that it is *not* necessary for one component to be a super-component of the other in order to control it (in fact, control is just a special, extreme case of the interaction mechanisms). Similarly, sub-components can be part of a super-learner *without* being controlled by it, e.g. when they are autonomous agents which simply forward their learning results.

To illustrate the possible relationships between learners in such an adaptive system, consider, as an example, the design of an adaptive driver-less vehicle control robot, whose learning goal is to find a mapping from any given traffic situation to an optimal action. Apart from learning components that are responsible for the vehicle's vision system, its motoric steering and driving capabilities etc., this robot will certainly have to learn how to choose appropriate routes when driving, and to make things simpler, we concentrate on this last learning task for our example. Figure 4.1 shows a graphical model of the learning architecture that we propose for this example and which we will now briefly explain.
It is assumed that the "route selection learning algorithm" is responsible of learning the sub-tasks of (a) how to avoid obstacles, (b) whether and how to overtake, (c) to select appropriate lanes and (d) to select a global route from the current position to the des-

[1] In that sense, the "sub-component of"-relation formalises a concept of much stronger dependence, because the super-unit actually *consists* of the lower layer components.

tination. The respective four components L_1, L_2, L_3 and L_4 share the super-component L, the route selection learning algorithm, and since their learning goals are different from each other, L's learning task is divided among L_1, L_2, L_3 and L_4.
Now L_1 would probably be a sub-component of L_2, because avoiding obstacles is essential to learning how to overtake, so L_2 might want to use the results of L_1. L_1 may in turn divide its learning task into two sub-components L_{11} and L_{12}, one responsible for avoiding moving objects and one for driving around stationary obstacles. The learning tasks of L_{11} and L_{12} have some common elements, e.g. deciding when to change direction, and there might be general rules that govern this issue, so it is possible for these two learning units to interact in order to find a learning strategy that will yield good results for both.
Assume further that L_4 uses two sub-learners: one of these, a case-based reasoning algorithm (L_{41}) tries to construct optimal routes by consulting similar routes experienced before, while the other, e.g. a genetic algorithm (L_{42}) encodes routes as genes and tries to minimise their length (L_4 might employ such a multiplication strategy to ensure better results). Thus L_{41} and L_{42} have the same learning goal and compete for selection.
Another interesting connection is the one between L_3 and L_4. It might be useful to hand the control of L_3 over to L_4, for example, if the problem of selecting lanes (in the actual setting) has not yet occurred due to the routes selected by L_4 and L_3 is not needed, so that no computational effort should be wasted on its learning algorithm.
This is a very simple example of a learning architecture in a concrete application, but it gives a feel for the multitude of design decisions that have to be made when constructing such "societies of learners".

We now present an instantiation of this general methodology for learning coordinated behaviour in games which will be based on the InteRRaP architecture. We first define the concrete learning problem that has to be solved to develop intelligent interaction behaviour. Then, InteRRaP is briefly introduced, and the remainder of this chapter gives a detailed treatment of the concrete adaptive agent architecture we use.

4.2 The learning problem

In the introductory chapter the concept of the three determinants of interaction was presented informally. It was claimed that learning them is the key to achieving optimal behaviour in games. To prove this claim, we have to build learning algorithms which learn these concepts, and it has to be shown that agent societies, in which a learning component that integrates these algorithms is employed, achieve a more efficient interactive behaviour than those devoid of such learning capacities.

4.2.1 A well-posed formulation of the "Coordination Problem"

The problem of "how to interact effectively" has only been described very vaguely in the previous chapters, so we still lack the precise formulation of a well-posed learning problem, the definition of a class **T** of tasks that our agents should be able to solve with increasing performance according to some performance measure **P** using experience **E**.
Since we have restricted the scope of analysed interaction situations to repeated abstract n-player games, the task **T** is defined by "achieving optimal behaviour in playing games". However, as has been explained in the chapter on game-theoretic solution concepts, the term "optimal behaviour" is somewhat ambiguous, because it depends on the assumed

point of view: from some single agent's standpoint, only individual utility maximising behaviour can be considered optimal, but from a system designer's point of view this is only true of behaviour that ensures high payoffs to the whole agent society.
It turns out that assuming agents to be selfish and not necessarily benevolent, which has been the stance we adopted until now, forces us to adhere to the first definition to start with. Yet it is clear that (as the examples in Chapter 3 have shown) individual rationality does not always ensure that some agent makes the best of its situation, so *socially coherent* global behaviour is what we actually have to look for, i.e. behaviour that is both individually rational and collectively rational. Such behaviour would typically be characterised by joint strategy selections in which agents only sacrifice potential payoffs which they would not be able to gain anyway, given the effects of mutual greed. This means that the "sacrifice" is ultimately to their own good, so that they can still be seen as individual utility maximisers, with the difference that they don't "indulge in their own egoism".
These considerations motivate the following informal definition of the task **T**:

> **T**: *The task of an agent society (a group of recognizable individuals) to achieve socially coherent, i.e. individually and collectively rational global behaviour in a repeated n-player game* Γ *whose players are the individuals of the society.*

There is a subtlety in this definition that deserves some attention, namely that we have required optimal collective behaviour to be collectively rational for the task to be considered "solved". While this appears to stand in contrast with our previous remarks, note that if the behaviour is not collectively rational, then some agent is faring well at the expense of some other agent, so that this other agent has failed in defending itself against exploitation. This would mean that the agent cannot be acting individually rational itself, so that collective and individual rationality actually coincide at the system level.
It should be remarked, for the sake of completeness, that such a task is only well-defined if the optimal global behaviour(s) are known in advance, so that the learners' performance can be compared to them, which is not necessarily the case because the games might not be solvable or too complex to be analysed mathematically[2].

The definition of **T** directly provides the performance measure for the society: if reaching behaviours that are individually and collectively rational is the learners' goal, then the payoffs the agents receive will reflect the degree to which the socially coherent interaction pattern has been achieved. Hence, the performance of the society increases as the (geometric) distance of the current payoff vector $\vec{u}$ to the optimal payoff $\vec{u}_{opt}$ vector decreases.

> **P**: *The inverse of the distance between the current vector of payoffs received per round* $\vec{u}$ *and the payoffs of a socially coherent action* $\vec{u}_{opt}$:
>
> $$\mathbf{P} = |\vec{u}_{opt} - \vec{u}|^{-1}$$

Note that this performance measure will only yield high values if *all* agents approach pareto-optimal payoffs, so that neither exploitation nor sacrifice are considered successful behaviours, and if one of them occurs, than at least one agent is not coordinating its behaviour optimally with that of its opponents.

[2]Fortunately, we have been able to provide those solutions for the class of games we will examine in the previous chapter.

This leaves us with the task of defining the experience **E** from which the society should learn the optimal behaviour. This is a somewhat difficult undertaking, because we imposed the constraint that agents' percepts should consist only of the currently performed joint action combination and their *own* received payoffs. This means that we enforce the decomposition of the "society-level learner" L into *non-interacting*[3] "agent learners" $L_1, \ldots, L_n$ that are not given any information about the payoffs other agents receive. Hence the following, somewhat awkward, definition.

> **E**: *The experience from which the society is supposed to learn is given by a sequence of previous interactions, i.e. pairs of performed joint strategies and the respective payoff vectors such that*
>
> $$\mathbf{E} = \left\langle (s^{(1)}, u(s^{(1)})), \ldots, (s^{(t)}, u(s^{(t)})) \right\rangle$$
>
> *It is additionally required that the society consists of n independent learners (as defined in Section 4.1)* L_i $(1 \leq i \leq n)$ *who learn only with experience*
>
> $$\mathbf{E_i} = \left\langle (s^{(1)}, u_i(s^{(1)})), \ldots, (s^{(t)}, u_i(s^{(t)})) \right\rangle$$
>
> *and that no further learning other than that conducted by* $L_1, \ldots, L_n$ *takes place.*

This constraint of *private payoff notification* prevents us from adopting a centralised view of the learning problem. It is, in fact, this very aspect – together with the premise of having selfish agents – that makes a multi-agent learning approach to solving interaction problems reasonable.

This completes the presentation of the learning problem as such, but it also constitutes a first decomposition step (from society-level learning to agent-level learning), and this implicitly suggests that if the individual learners $L_1, \ldots, L_n$ solve their learning problems, then the society as a whole will have solved its initial problem.
Because we have assumed that there are collectively and individually rational joint strategies in the games we examine this proposition holds, and the task and performance measure for the agent-level learning algorithms can be derived from those of the society.
We now turn to the agent-level view of the problem and analyse how it can be further decomposed into simpler sub-problems for which standard machine learning algorithms can be devised.

4.2.2 Agent-level problem and further decomposition

Seen from the perspective of one of the individual learners L_i, the problem of learning how to interact optimally with its peers in a common environment has three aspects that are best summarised as the answers to the following questions:

1. "In which way do the outcomes of my actions depend on those of my opponents?"
2. "How can I act in a way that will prevent me from being exploited?"
3. "How can I, by my own action choices, influence the action choices of my co-actors in a way that is most beneficial to me?"

[3] In the sense that the learning algorithms are not interconnected (or *independent*, in terms of the relations defined for the layered learning model), not the agents themselves.

Behaviours that implement the answers to these questions aim at the maximal welfare of the agent and disregard the common good, but the third question already implies that the agents will be willing to compromise, if they can profit from it.
It is evident that these issues map directly to the "determinants" put forward in Section 1.2.2: 1. is nothing but the quest for the *interdependence modalities* as reflected by the payoff function, the answer to 2. is given by determining the future *opponent behaviour* and choosing those actions which will ensure (in a maximin fashion) the greatest minimal payoff. The third question, finally, asks how the agent can exploit further mutually beneficial *potentials for cooperation*, how it can participate in and initiate the formation of implicit coalitions which make "better-than-maximin" behaviours possible.
But how can these determinants be turned into computational concepts, such that learning algorithms for them can be designed? The task of learning a payoff function can be formulated in mathematical terms in a straightforward manner: under the assumption that the strategy sets of all agents are known in advance, it simply consists of constructing an approximation $\pi_i : \times_{i \in N} S_i \rightarrow \mathbf{R}$ of the actual (unknown) payoff function u_i, and the performance of such an approximation can be measured in the mean error it produces on the entire set of possible joint strategies. More precisely, the learning problem of an *interdependence modality learning component* L_i^{IM} can be defined as follows:

Definition 4.1 (Interdependence Modalities Learning Problem)
The learning problem $G(L_i^{IM}) = \langle \mathbf{T}, \mathbf{E}, \mathbf{P} \rangle$ *of an interdependence modality learner* L_i^{IM} *belonging to an agent learner* L_i *is defined by:*

- $\mathbf{T}$*: For any joint action* $s \in S$ *predict the value of player* i*'s payoff function by using a (self-constructed) approximation function* π_i*, such that*

$$\forall s \in S . \pi_i(s) = u_i(s)$$

- $\mathbf{E}$*: A sequence of past joint action/payoff pairs*

$$E_i = \left\langle (s^{(1)}, u_i(s^{(1)})), \ldots, (s^{(t)}, u_i(s^{(t)})) \right\rangle$$

- $\mathbf{P}$*: The inverse of the mean error at predicting the payoff of joint actions in the combined strategy set* S*:*

$$\mathbf{P}(\pi_i) = \left(\frac{1}{|S|} \cdot \sum_{s \in S} |u_i(s) - \pi_i(s)| \right)^{-1}$$

The task of learning to predict opponent behaviours is somewhat harder to formalise, because the problem arises of whether functions that predict whole *sequences* of joint opponent actions should be learned, or whether it suffices for the learned functions to predict the next opponent action only.
From a *perfect rationality* standpoint, only actions should be chosen that ensure a high payoff for all of the remaining game rounds, which – given the assumption we have made about agents not knowing the duration of the game – would mean that an asymptotic analysis of the expected future payoff should be conducted, and that therefore arbitrarily long opponent action sequences should be predicted (we explain why this is the case in our discussion of Markov Decision Processes in Section 5.3.1).
Although we strongly support the realisation of *boundedly rational* reasoning capabilities in agent design and hence the opponent behaviour prediction (OBP) abilities of the agents

we design will be far from being perfectly rational, we define the behaviour prediction learning problem here in its rigorous version to explain what the task would be "in theory", if it could be achieved in realistic situations. As concerns the performance measure, a measure for how closely a predicted opponent action matches the actually occurring action combination will certainly depend on more specific properties of the game being played[4], and therefore we will provide only very general conditions for it.

Definition 4.2 (Opponent Behaviour Prediction (OBP) Learning Problem)
The learning problem $G(L_i^{OBP}) = \langle \mathbf{T}, \mathbf{E}, \mathbf{P} \rangle$ *of an opponent behaviour prediction learner* L_i^{OBP} *belonging to an agent learner* L_i *is defined by:*

- **T**: *For any number* t_2 *of future rounds, predict the sequence of joint opponent moves that will occur in the next* t_2 *moves, given a* t_1*-long sequence of past joint actions and a* t_2*-long sequence of action choices of* i*. This means that the learner's goal is to construct a prediction function* $f : S^{t_1} \times S_i^{t_2} \rightarrow S_{-i}^{t_2}$ *that meets the following condition:*

$$f(s^{(1)}, \ldots, s^{(t_1)}, s_i^{(t_1+1)}, \ldots, s_i^{(t_1+t_2)}) = (s_{-i}^{(t_1+1)}, \ldots, s_{-i}^{(t_1+t_2)}) \iff$$
$$\Gamma^{t_1+t_2} = \left\langle s^{(1)}, \ldots, s^{(t_1)}, \quad (s_i^{(t_1+1)}, s_{-i}^{(t_1+1)}), \ldots, (s_i^{(t_1+t_2)}, s_{-i}^{(t_1+t_2)}) \right\rangle,$$

  if $\Gamma^{t_1+t_2}$ *is the game that is actually being played (including the following* t_2 *rounds).*

- **E**: *A sequence of past joint action/payoff pairs as before (cf. Definition 4.1).*

- **P**: *The performance* $\mathbf{P}(f)$ *is proportional to some similarity function* $sim : S_{-i}^{t_2} \times S_{-i}^{t_2} \rightarrow [0; 1]$ *defined on pairs of the future opponent action sequence, such that the following conditions hold:*

  1. $\forall . s^{t_2} = \langle s^{(1)}, \ldots, s^{(t_2)} \rangle \in S_{-i}^{(t_2)} . sim(s^{t_2}, s^{t_2}) = 1$
  2. $\forall . s^{t_2} \neq t^{t_2} \in S_{-i}^{t_2} . sim(s^{t_2}, t^{t_2}) < 1$

  (apart from the condition of optimal similarity being reached only by equal action sequences it is left to the system designer to define the desired similarity function).

This leaves us with the problem of defining a learning problem for the component of L_i that is supposed to learn the cooperation potential of the game. Such a component should, for any current game situation, suggest a sequence of actions that will make the opponents converge to a behaviour which, together with the agent's own coordinated action selection, will result in the formation of the optimal coalitions within a minimal number of additional sub-optimally played rounds. A formal definition of this concept is given by the following:

Definition 4.3 (Cooperation Potential Learning Problem)
The learning problem $G(L_i^{CP}) = \langle \mathbf{T}, \mathbf{E}, \mathbf{P} \rangle$ *of a cooperation potential learner* L_i^{CP} *belonging to an agent learner* L_i *is defined by:*

[4] Such a measure for action similarity would typically be proportional to the payoff difference between the two actions who are being compared, so that errors in predicting the opponent action would be punished proportionally to the error in predicting the respective payoff that results from mis-predicting the opponent action.

- **T**: *Construct an action selection rule* $h_i : S^{t_1} \rightarrow S_i^{t_2}$ *that suggests for any past joint action sequence* S^{t_1} *the shortest sequence of future actions for player* i *which will result in the convergence of the society to strategy combinations in the set of individually and collectively rational (cf. Appendix B for definitions of these terms) joint strategies*[5] OPT*, so that*

  $$h_i(s^{(1)}, \ldots, s^{(t_1)}) = (s_i^{(t_1+1)}, \ldots, s_i^{(t_1+t_2)}) \iff \forall j > t_1 + t_2 . \exists opt \in OPT . s_{-i}^{(j)} = opt_{-i}$$

  holds. (If the n *following actions of* i *can ensure that the opponents will "play their part" in the optimal coalition (*$\forall j \geq t_1 + t_2 . opt_{-i} = s_{-i}^{(j)}$*) forever after* t_2 *more rounds, then agent* i *can choose the appropriate action* opt_i *to participate in that coalition.)*

- **E**: *A sequence of past joint action/payoff pairs as before (cf. Definition 4.1).*

- **P**: *Since the agent has no access to the global payoff vector error* $|\vec{u}_{opt} - \vec{u}|$*, it is only fair to judge its performance by the difference between its own mean payoff per round after the* n *actions suggested by function* h_i *and the payoff-per-round it would receive under* OPT*:*

  $$\mathbf{P}(h_i) = \left(\frac{1}{|S^{t_1}|} \sum_{1}^{\infty} |u_i(s^{(t_1 + length(h_i(s_1^t)) + j)}) - u_i(opt)| \right)^{-1}$$

  (The infinite sum (actually a limit in the upper bound) measures the total error between the actually received payoffs and the optimal payoff for an infinite duration (after the $t_2 = length(h_i(s^{t_1}))$ *initial rounds computed for a past action sequence* s^{t_1} *that are necessary to reach the optimum according to* h_i*). This error is computed for the whole set* S^{t_1}*, and the inverse of the average error is taken to be the performance of* h_i*.)*

These definitions provide us with well-posed learning problems that the agent should solve. Yet, it is quite unrealistic that they can be solved in the way they have been presented and are hence of rather theoretical value. This is due to the fact that

1. L^{IM} will need to be trained on the whole joint strategy space S in the worst case, which would require an exponential game duration ($|S| = \prod_{i \in N} |S_i|$),

2. predicting the precise future opponent behaviour implies that L^{OBP} has a complete understanding of all opponents' reasoning mechanisms and action selection rules and, in the case of mixed opponent strategies, can even act as an oracle and

3. L^{CP} cannot (even if it is using the results of L^{OBP}) construct a perfect hypothesis unless it knows the precise payoff structure of all other peers (which is rather improbable, given that it is not imparted any explicit information about the payoffs its adversaries receive).

Therefore, adequate approximative solutions for these problems will have to be sought for in practice, and the experimental prototype presented in Chapter 5 is an attempt to design simple components which implement such approximate learning.

[5] For reasons of notational convenience, we assume that (as in the resource-load game) collectively and individually rational *pure* strategy combinations exist which form the set OPT. The definitions can be extended in a straightforward way for mixed strategies by replacing u with the expected payoff (because the learning problem requires that an equilibrium situation arises that is not left ever after, the mean payoff per round will be the same under an mixed optimal strategy as it is for a pure optimal strategy).

4.2.3 Learning and acting

If we were to provide a solution for the discussed learning problems, it would normally suffice to find a machine learning algorithm that can be trained with hypothetical action sequences, and which would ideally be provably able to improve its predictive capacities over time.
Unfortunately, this is not possible, because in the game simulations that would be needed to provide the training samples, decision processes and learning processes are *interleaved*: the input data for the learning algorithms is given by previous agent actions and the action selection rules will be based on what the agent learned before (otherwise there is no reason to conduct any learning in the first place).

This observation brings with it the need for an *active learning* architecture which builds its world model by observing the activities of the society and acts according to its current state of belief. We have found the InteRRaP[6] architecture to be a good starting point for building such an agent architecture, since it provides three layers of decision and knowledge modules which can be easily extended by learning components, such that each layer is responsible for one of the above learning tasks. Furthermore, the learning problem that is assigned to each layer is in concordance with the abstraction levels that the layers are supposed to represent in InteRRaP.

Before going into the details of the final layered learning framework we will employ, we first outline the foundations of InteRRaP to explain how and why our theory builds upon it.

4.3 The InteRRaP architecture

The dispute between the *reactive* and *deliberative* school of DAI has partly found its reconciliation in research on *hybrid* architectures which try to combine the strengths of both philosophies. The key hypothesis of such approaches is the belief that "the benefits accrued from having a combination of philosophies within a singular agent is greater than the gains obtained from the same agent based entirely on a singular philosophy" (Nwana, 1995, p. 33). Clearly, the main advantages of reactive agent design are robustness, faster response times and adaptability, while deliberative approaches are better suited for handling long-term goal-oriented issues and using symbolic knowledge and reasoning.

The InteRRaP agent architecture due to Müller (1996) is a typical example for such hybrid agent design. Essentially, it is based on the idea that each agent consists of three layers, the lowest of which implements low-level situation-action patterns of behaviour and also manages the agent's sensing and action, while the intermediate layer encapsulates the agent's long-term planning capabilities and individual goals. The topmost layer is concerned with negotiation, communication and coordination, i.e. any social activities of the agent.
The motivation for this layering is obvious: it allows for a combination of the reactive stimulus-response behaviour (necessary to cope with time-critical decision situations and to degrade gracefully under critical conditions) with deliberative reasoning mechanisms (so that a long-term rational behaviour can be implemented through explicit planning

[6] **Inte**gration of **R**eactive Behaviour and **Ra**tional **P**lanning

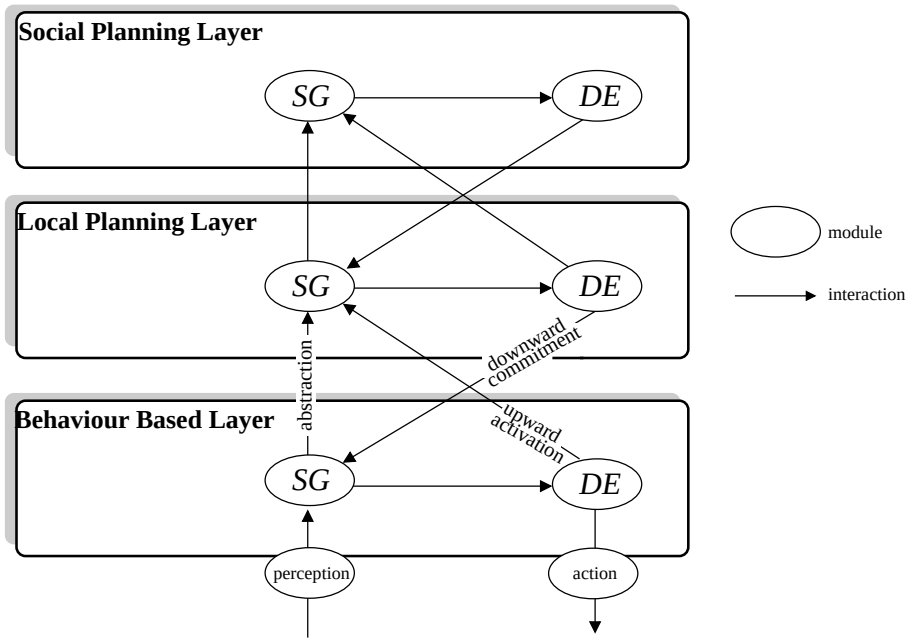


Figure 4.2: The InteRRaP architecture.

and inference). Moreover, it incorporates the more computationally expensive and elaborate capabilities required to communicate effectively, so that inter-agent conflicts can be resolved and the community can act in a socially coherent fashion.
Since these activities can be conducted concurrently and also because they are interleaved due to certain interactions between them, the architecture offers high flexibility, resource-adaptability and also an elegant way of bringing together different levels of (sub-)cognitive capabilities and knowledge abstraction.

Figure 4.2[7] provides an overview of the layers and their internal structure. Each layer consists of two modules, the *Situation Recognition and Goal Activation Module* (SG) and the *Decision Making and Execution Module* (DE). The SG spawns new goals out of the maintenance of belief by combining the information obtained from percepts with its knowledge base, while the DE decides how to meet these goals, builds plans that implement these decisions and spawns their execution.
The *Behaviour-Based Layer* (BBL) is located at the lowest level of the architecture. It serves as an interface to the agent's environment as it encapsulates the agent's sensing and acting capabilities and implements *patterns of behaviour*, procedural routines that describe the agent's immediate reaction to certain percepts. Such patterns need not necessarily be simple (they can in fact be very complex), but they can be seen as the primitives of the overall behaviour because they cannot be further decomposed, i.e. they represent *black-box* action sequences. Since this layer is the only component that maintains a connection to the "outside world", it will have to communicate the information it has about the performed actions and their effects to the other layers.
On top of the BBL, the *Local Planning Layer* (LPL) reasons about long-term goals that lie beyond the scope of simple reactivity, goals that require the explicit planning of future activities. The SG builds a knowledge base by abstracting from the concrete information imparted to it by the BBL to extract more general properties of the current situation, to

[7] Illustration taken from Jung (1998), used by kind permission of C. G. Jung.

identify world states and operators that can be used to solve its planning problem. Once goals have been determined (very often these will be sub-goals in the sense of hierarchical planning), the DE decides on how they can be attained. The resulting new patterns of behaviour are then spawned to the BBL and affect the situation context of the BBL's SG, thus influencing the concrete action choices made by the DE.
However, the individual (local) plans of the LPL may not suffice to successfully (or optimally) complete the agent's task in a societal context. Conflicts between the agent's own actions and those of others can arise, and cooperation potentials between agents may remain undiscovered, so that the overall system behaviour is incoherent or inefficient. To accommodate the needs of inter-agent coordination, the *Social Planning Layer* is added to the framework as the topmost component that is concerned with social reasoning and decision-making.
In it, the SG employs further abstraction on the local information obtained from the LPL in order to produce social goals, and once these goals are turned into decisions, their execution affects the functioning of the lower layers (the LPL might have to modify its plans to fit the goals of the SBL, and the BBL will have to carry out the communicative acts that are necessary to enact negotiation).

As Jung (1998, p. 4) points out, this layering "models the smooth transition from sub-symbolic reactivity to symbolic deliberation and even social capabilities". The fact that the internal control flow represented by the SG and DE inside each layer is identical for all of them adds to the simplicity of the theory, and the hierarchical view of cognitive processes makes them particularly appealing as a concept for modelling intelligence.

We have already hinted at some of the interactions that take place between the layers, but we have not presented the general concept behind them. In the original definition of InteRRaP (Müller, 1996), layers interact via *upward activation* and *downward commitment* (as shown in Figure 4.2), i.e. the sub-layer calls its super-layer as a subroutine whenever this is necessary and the super-layer alters the behaviour of its sub-component by the intentions it spawns. Thus, layers represent optional "paths of computation" and thus their decisions have the same status and are arbitrated in between.
In the extended specification (Jung and Fischer, 1997), the interaction between layers is seen more like a meta-object relationship between components that realise the *whole* functionality of an agent. The BBL, LPL and SBL are individual *control processes* that manage their own sub-components (sub-*processes*), and lower layers are supported by their super-layers (rather than subsumed by them) in that they are being monitored, reconfigured and reasoned about by the higher-level components. So the inter-layer relationship is rather one of *cooperation* than one of *control*.
To keep things simple (and especially to reduce computational complexity), we have adopted the first view for the learning architecture we propose. The more elaborate version certainly offers an interesting extension, though.

The structure of InteRRaP allows for a very intuitive mapping of the three layers to the three learning problems which constitute the task of interaction learning. The next subsection explains how InteRRaP can be extended in a very natural way by learning modules to accommodate the needs of interaction learners and also introduces the simulation environment that will be used to conduct experiments to evaluate the adequacy of our approach.

4.4 LAYLA – an extension of InteRRaP for game-learning agents

Reconsidering the agent-level learning problems discussed in Section 4.2.2, a striking similarity between them and the InteRRaP layers can be observed.
Firstly, both consist of three parts, a fact which would not deserve further attention if it were not the case that there is a hierarchic dependence between those layers in both cases. This is because, although this has not been mentioned so far, the learning results of L^{IM} will have to be made available to L^{OPB} and L^{CP} will need to access the results of both L^{IM} and L^{OPB} when it comes to making *decisions*, choices concerning future actions (we give reasons for this in the detailed overview of the components below). In an *active* learning system decision making is indispensable, and as we will see, the dependencies between the learning problems will require a control flow between the respective decision processes that is very reminiscent of the interactions between the InteRRaP layers.
Secondly, there is a close connection between the abstraction levels of knowledge, situation recognition, goal activation and decision making at each of the InteRRaP layers and the learning problems of the three proposed learning modules: L^{IM} learns the immediate effects of the agent's actions and the way in which its own obtained payoffs depend on peer actions, or alternatively, it learns the nature of certain patterns of behaviour. And since carrying out action primitives is the task of the BBL, it only seems natural to locate a component that learns something about these at the very same layer. Quite similarly, the learning conducted by L^{OBP} is focused around predicting future opponent actions so that optimal strategies can be picked accordingly, and such strategies are nothing but individual plans that help realise the ultimate (and sole) goal of utility maximisation. This implies that such learning should be associated with the LPL[8].
Learning to exploit cooperation potentials, finally, is just a positive rephrasement of "resolving conflicts" between local strategies (plans) and – in the context of repeated games – socially coherent joint strategies are the equivalent of feasible multi-agent plans. Therefore, incorporating the L^{CP} in the SPL seems a reasonable choice.

All this provides a justification for our choice of InteRRaP as a framework within which we construct our learning architecture. At the same time it suggests a possible way of extending InteRRaP as a generic architecture (beyond the game-theoretic context, that is) by learning modules, such that the learning layers *learn* to act, to plan and to coordinate just like the InteRRaP layers act, plan and coordinate.
The following paragraphs provide detailed descriptions for the system components and present the resulting integrated LAYered Learning Agent architecture, the extension of InteRRaP to an game-learning layered architecture.

4.4.1 Overview

An exhaustive presentation of multi-agent simulation systems has to specify the properties of three major system components: the *agents*, in particular their sensing, acting and reasoning capabilities, the *environment* within which they evolve and from which they

[8]It could be argued that choosing actions according to predictions of peer behaviour already implements some kind of coordination and should therefore be considered a *social* activity. However, opponent behaviour prediction views other players much more as mechanical system variables that exhibit a certain behaviour than as rational agents. The advantages of this perspective are given in more detail in Section 5.3 .

receive their percepts, and the *simulation* algorithm that describes the environment-agent and agent-agent control flow.
For systems that simulate repeated multi-player games with layered-learning InteRRaP agents as we design them, these components translate to the following:

- Agents, consisting of three layers, each of which comprises a *Learning Module*, a *Knowledge Base* and a *Decision Module*. The learning modules are responsible for the three sub-problems as discussed. The knowledge bases are equivalent to the SG modules introduced before, but we prefer to view them as simple data storage facilities because both situation recognition and goal activation can be reduced to trivial concepts in this abstract simulation[9]. Decision Modules, the layers' DEs are responsible for making action choices. Given that the agent is only playing an abstract game, the execution of these actions can actually be reduced to symbolic messages to the environment, i.e. it is detached from the agents. In a similar way percepts are reduced to notifications about opponent actions and about the agent's own received payoff by the environment.
  The layers that result from extending the BBL, SPL and LPL by learning modules are called the *Utility Engine*, *Strategy Engine* and *Social Behaviour Engine* respectively.

- A *Simulation Engine* that represents the environment in which the agents are situated. It receives their action choices, computes the respective payoffs and notifies them of all other agents' actions. If the agents are realised as concurrent processes, it has the additional task of synchronizing the effectuation of their actions, i.e. the payoff computation has to be suspended until all agents have decided what to play in the next round (this puts the "discrete and synchronous execution"-assumption of game theory into practice).
  To simulate the action and perception failures that occur in more concrete environments, we equip the Simulation Engine with three sub-components: the *Execution Engine*, which simulates the transition from decisions to actually effected actions, the *Payoff Generator* that computes the agents' payoff on the basis of the actually executed actions and the *Perception Generator* that turns the results of payoff computation and the executed joint actions into the actual percepts the environment provides. Noise generators between these sub-components can manipulate the data exchanged between these three components, so that *misimplementation* and *misperception* (cf. Section 3.1.3) can be simulated adequately.

- The (rather trivial) *Game Algorithm*: After each round, the agents output some action choice from their Utility Engine Decision Module to the Execution Engine in the Simulation Engine. The Utility Generator computes some payoff for each agent and communicates it together with information about all agents' actions via the Perception Generator back to the agents' Utility Engine Learning Modules. Using this information, the agents employ their Learning Modules to update their world models and modify their Knowledge Bases accordingly. Then, the Decision Modules consult the updated knowledge to compute some action choice for the next round

[9]The "situations" are given by past joint action choices and obtained payoffs and spawning goals is a trivial matter, since there is only *one* fixed goal, namely to maximise future payoffs (finding out *how* this goal can be met certainly requires that the layers interact, but this is the task of the DE, not the SG). Therefore the SG only needs to be capable of storing action/payoff sequences and the current learning hypotheses.

Entity	*Components*	*Subcomponents*
Agent	Learning Module	Utility Learning Module
		Strategy Learning Module
		Social Learning Module
	Knowledge Base	Utility Model
		Strategy Model
		Social Model
	Decision Module	Action Generator
		Strategy Generator
		Social Behaviour Generator

Table 4.1: Agent components and subcomponents (vertical view).

(by applying methods yet to be specified). This process is repeated for a finite number of rounds (that is unknown to the agents).

The detailed system architecture is presented in Figure 4.3. A discussion of each of the components is given in the following sections.

4.4.2 Agent architecture

LAYLA agents consist of the three layers and their subcomponents, as presented in Table 4.1. In the following, we discuss the precise functionality of each of these, but this treatment does *not* include the learning algorithms that will actually be used to solve the particular learning tasks (this will be left to Chapter 5). Instead, we present the interactions between them as results from the interleaved nature of the learning problems at a more abstract level by describing the data and control flow between the modules as well as their internal data structures.

Behaviour-based Layer: The Utility Engine

At the reactive level, agents control the data flow to and from the Simulation Engine, thereby realising a simple action-perception structure. Furthermore, the Utility Engine learns and stores a payoff function estimate and maintains the necessary links to the other layers to exchange information with them. Finally, it controls the exploration/exploitation "attitude" of the agent by deciding to gain more information about particular actions (while neglecting payoff maximisation) if this is necessary or, if the payoff function has been adequately modelled, leaving action decision to higher layers.
The Learning Module, Knowledge Base and Decision Module of the Utility Engine (UE) are the *Utility Learning Module*, the *Utility Model* and the *Action Generator*.
Controlling the interface to the Simulation Engine is handled by the Action Generator, which dispatches the actual action choice to it after each round and by the Utility Learning Module which receives payoff and action information from it and propagates it to the learning modules of the super-layers. Those super-layers can then use it to update their strategy or social model.
The Utility Learning Model also updates an approximation of the payoff function with the information received after each round, and the current payoff function estimate is stored in the Utility Model. It can be consulted by the Action Generator (to determine whether further exploration is necessary or if the action choice can be left to the other two layers)

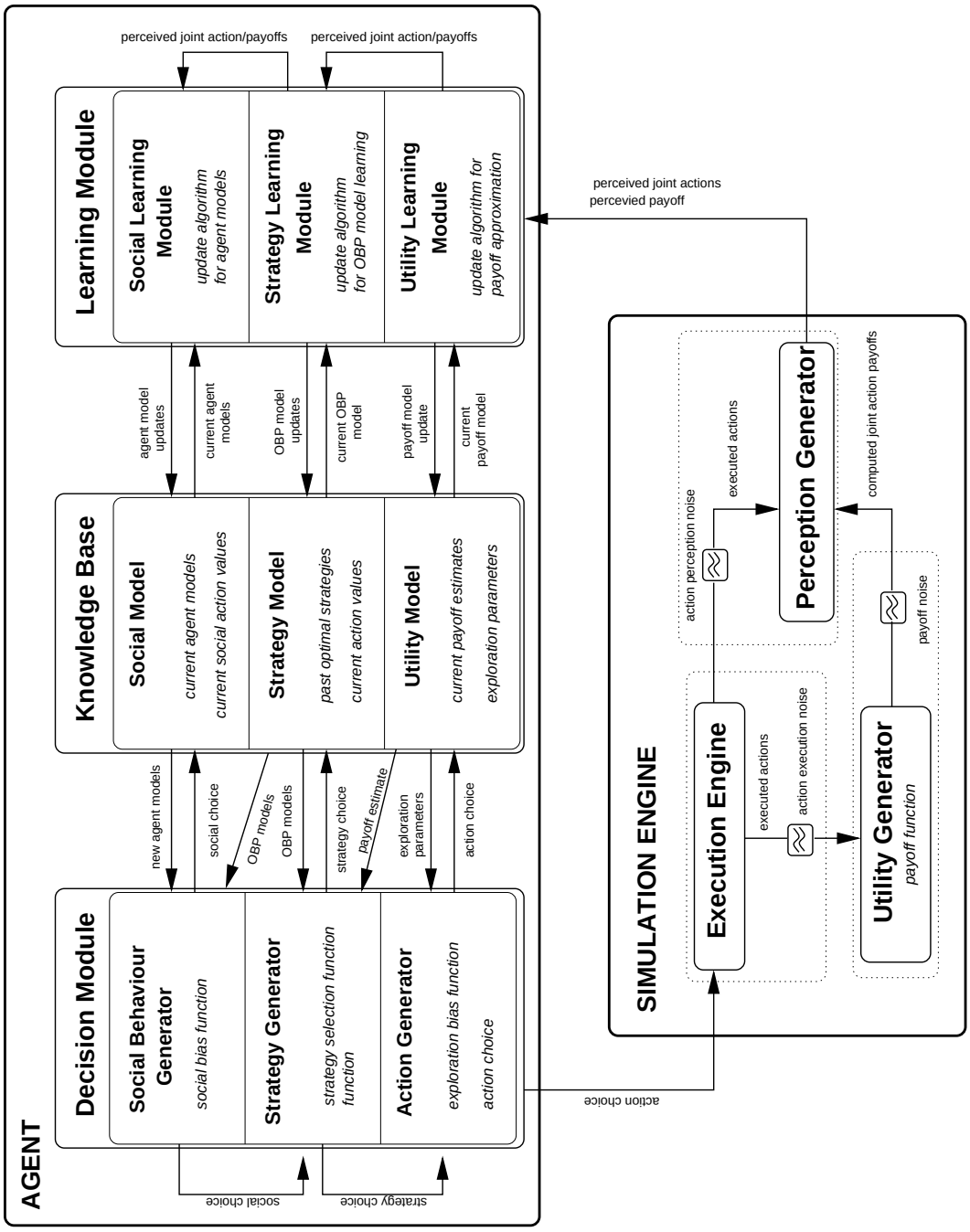


Figure 4.3: Detailed view of the system architecture with layered agents.

Subcomponent	Utility Learning Module
Inputs	perceived joint action and payoff (from Perception Generator) current payoff estimate (from Utility Model)
Internal Parameters	update algorithm for payoff estimate
Functions	update payoff estimate with current joint action and payoff
Outputs	new payoff estimate (to Utility Model) perceived joint action and payoff (to Strategy Learning Module)
Subcomponent	Utility Model
Inputs	payoff estimate update (from Utility Learning Module) last chosen action (from Action Generator)
Internal Parameters	current payoff function estimate current exploration parameters
Outputs	current payoff function estimate (to Utility Learning Module)
Subcomponent	Action Generator
Inputs	mixed strategy to be played (from Strategy Generator) exploration parameters for actions (from Utility Model)
Internal Parameters	exploration bias function
Functions	bias action choice by exploration factors update exploration parameters choose action according to strategy and exploration bias
Outputs	action choice (to Execution Engine and Utility Model)

Table 4.2: **Utility Engine** subcomponents and their functions.

as well as by the super-layer components, if their learning and reasoning process depend on the current payoff estimate.
The Action Generator, as suggested, either executes the (exploitative) decisions of the Strategy Engine (which can in turn be influenced by the Social Behaviour Engine) in a *downward commitment* fashion or conducts further exploration based on its own state of beliefs. In the latter case, there is no reason to engage in any higher-level reasoning, so the super-layers are not used (*upward activation*).

A detailed description of the inputs, outputs and functions of the Utility Engine subcomponents is given in Table 4.2.

Local Planning Layer: The Strategy Engine

The local planning layer is concerned with devising a *personal* agenda for how the agent should act. The need to integrate decision and learning in such a layer springs from the fact that regardless of how inaccurate the agent's model of payoffs or of others' future behaviour might be, the agent nevertheless *has* to act.
Thus, the Strategy Engine (SE) has to combine the OBP model with knowledge about the payoffs that can be obtained under the expected opponent behaviours and to output a strategy that promises the highest possible payoff under the predicted future circumstances.
In practice this means that the agent maintains a model of the future behaviour of adversaries (in the knowledge base of the *Strategy Model*) and that it updates this model with incoming knowledge about opponent actions (via the *Strategy Learning Module*). Then, using the payoff estimate of the UE, the *Strategy Generator* reckons which of the agent's

Subcomponent	Strategy Learning Module
Inputs	agent percepts (from Utility Learning Module) current OBP models (from Strategy Model)
Internal Parameters	update algorithm for behaviour prediction
Functions	update OBP models
Outputs	prediction updates (to Strategy Model) current joint action and payoffs (to Social Learning Module)
Subcomponent	Strategy Model
Inputs	updated OBP models (from Strategy Learning Module) action-value function (from Strategy Generator)
Internal Parameters	OBP models current action values
Outputs	current OBP models (to Strategy Learning Module) action values (to Social Behaviour Generator)
Subcomponent	Strategy Generator
Inputs	social value function (from Social Behaviour Generator) current OBP models (from Strategy Model)
Internal Parameters	action-value determining function
Functions	choose strategy according to action-value function and social value function
Outputs	(mixed) strategy to be played (to Action Generator)

Table 4.3: **Strategy Engine** subcomponents and their functions.

own actions will yield the highest expected payoff given the predictions of the Strategy Model and generates the corresponding decision as a pure or mixed strategy.
More specifically, the Strategy Learning Module receives payoff and action information from the Utility Learning Module (and propagates this information to the *Social Learning Module*). It updates current behaviour prediction models obtained from the Strategy Model with the new information and returns updated models to the Strategy Model, which are then forwarded to the Strategy Generator.
The Strategy Generator constructs an action-value function by comparing the expected payoffs of each of the agent's action options under the predicted opponent behaviour. To this end, it uses the Utility Model and the updated Strategy Model. (This action-value function is also stored in the Strategy Model so that it can be consulted by the Social Behaviour Model.) Unless overruled by the decisions of the Social Behaviour Engine, the action-value function is transformed into a mixed strategy and propagated to the Action Generator. Otherwise, a *social value function* is applied to the action-value function and the resulting modified strategy is output.

As before, we summarise the details in Table 4.3. Quite differently from the UE, the SE conducts real *strategic reasoning*. It reasons about the value of particular strategies and makes rational decisions based on those values (hence the name of the layer). In contrast to that, the UE's "payoff-blind" decision-making is solely guided by the wish to explore the effects of unseen joint action combinations.

Subcomponent	Social Learning Module
Inputs	perceived joint action/payoffs (from Strategy Learning Module) opponent models for all opponents (from Social Model)
Internal Parameters	update algorithm for opponent models
Functions	update opponent models
Outputs	updated opponent models (to Social Model)
Subcomponent	Social Model
Inputs	updated opponent models (from Social Learning Module)
Internal Parameters	opponent models
Outputs	current social value function current opponent models
Subcomponent	Social Behaviour Generator
Inputs	current agent models (from Social Model) action-value function (from Strategy Model)
Internal Parameters	social bias function
Functions	compute social value function
Outputs	social value function (to Strategy Generator)

Table 4.4: **Social Behaviour Engine** subcomponents and their functions.

Social Planning Layer: The Social Behaviour Engine

So far our agents take the existence of their opponents into account only very implicitly by reasoning about how they will behave in the future and by acting according to these expectations. A complete understanding of interaction dynamics, however, requires that the *social context* be analysed, that potentials for future cooperation are discovered and that actions are taken to achieve such cooperation. To facilitate the accomplishment of these goals, the *Social Behaviour Engine* (SBE) is installed on top of the BBL and LPL, which, just like the "lower" layers, consists of a *Social Learning Module*, the *Social Model* and the *Social Behaviour Generator*. Those three subcomponents interact in order to provide a *social value function* that is incorporated in the agent's strategies and that reflects expectations concerning future compromise and how it can be reached.
The Social Model maintains *opponent models* that approximate the adversaries' reasoning mechanisms, so that the future *reasoning* of other agents can be simulated beyond simple behaviour prediction. Acquiring such knowledge is necessary to find how peers can be "massaged" into the state that is most desirable for the agent.
The Social Learning Module receives payoff and action information from the Strategy Learning Module and the current agent models from the Social Model. It updates the agent models accordingly and returns them to the Social Model.
It is the primary task of the Social Behaviour Generator to construct the *social action-value function* and to propagate it to the Strategy Generator. Such a function will typically alter the values of the individual strategies provided by the Strategy Generator (by applying a *social bias function* to them), if deviating from the individually most rational strategy (as reflected in the Strategy Generator action-value function) seems promising to achieve a collectively rational stance on the opponents' side. To this end, it will need to access the current action-value function of the Strategy Model, because a trade-off between "greedy" exploitation and the risky initiation of vague implicit agreements will be necessary if an overtly compromiseful stance of the agent is to be prevented.

Table 4.4 summarises the functions, inputs and outputs of the Social Planning Layer. Having completed the presentation of the individual agent components we are left with the task of presenting a simple functional model that specifies how LAYLA agents will actually *behave* in game simulations.

Integration

At any stage of the game, the training data available to agent i is the *history* of the game so far, i.e. a sequence H_i^t of past joint actions and associated payoffs for i filtered through the components of the Simulation Engine. Let π_i be the current payoff approximation of the Utility Learning Module and $e_i : (\mathcal{A} \to \mathbf{R}) \times [0;1]^A \to [0;1]^A$ be the exploration bias function of the Action Generator, that outputs for each current payoff estimate π_i and mixed strategy m_i returned by the Strategy Generator some m_i' (that is only different from m_i, if "exploration needs" require the agent to deviate from utility-maximising decisions). Further, let m_i be the Strategy Generator decision function which takes π_i and the OBP function f_i (cf. Definition 4.2 in Section 4.2.2) as inputs and computes a mixed strategy out of the resulting action values.
We also define the social bias computing function of the Social Behaviour Generator l_i as a function that transforms the individually rational strategies of the Strategy Generator into *socially feasible* strategies (strategies that are in concordance with the social values of the Social Learning Module). Then the mixed strategy that will be played by i in round $t+1$ can be defined as

$$\alpha_i^{(t+1)} = e_i\left(\pi_i, l_i\left(m_i(\pi_i, f_i), h_i\right)\right) \tag{4.1}$$

where h_i is the current hypothesis of L^{CP} as in Definition 4.3, which, as has been stated before, is used together with the action-value function m_i to compute the social value function l_i.
It should be remarked that π_i, f_i and h_i, the hypotheses of the three learning modules depend on the agent's past percept history H_i^t, while the decision functions are static throughout the game (we do not account for cases in which the decision functions themselves are learned, that is).
Another detail that has been left out in the above equation it that the Action Generator is required to draw some (pure) strategy out of the given strategy probabilities, i.e. some function $d : [0;1]^A \to A$ needs to be applied to $\alpha_i^{(t+1)}$ to obtain the pure strategy $a_i^{(t+1)}$ that will be played in the next round, i.e.

$$a_i^{(t+1)} = d(\alpha_i^{(t+1)}) \quad . \tag{4.2}$$

Including the agent's history of percepts from which the agent builds its learning hypotheses explicitly, we introduce the *transition* function δ_i that computes the next action choice of i on the basis of the current states of π_i, f_i and h_i:

$$\delta_i(H_i^t) = a_i^{(t+1)} = d\left(e_i\left(\pi_i(H_i^t), l_i\left(m_i\left(\pi_i(H_i^t), f_i(H_i^t)\right), h_i(H_i^t)\right)\right)\right) \quad . \tag{4.3}$$

By dropping the i-indices and taking the cross-products of the learning and decision functions of all agents, the above constructions allow for extending the functional definition of $a_i^{(t+1)}$ into a formula that predicts the next joint action $a^{(t+1)}$ by applying the global transition function δ to the previous joint action.

It now remains to specify how the action and perception histories H_i^t come about given a sequence of joint action decisions, how agent decisions are turned into actions, how the payoffs are computed and how percepts are generated from these that constitute the agent's learning input. These are the tasks of the *Simulation Engine* which will be described next.

4.4.3 The Simulation Engine

As mentioned before, the purpose of the *Simulation Engine* is to simulate action and perception in the system, so that they are conceptually detached from the agents themselves. By virtue of this *externalisation*, agents can neglect the details of handling sensors and actuators and concentrate on reasoning processes.
Moreover, it embodies the interdependencies between the actions of the individuals (resource limitations, etc.) and plays an important part in the learning process: it is the only system component that has full knowledge of the payoff function, the only source of training data in the form of action and payoff information (it also provides *noise* for the samples) and the only reliable *critic* of the agents' performance.
Its three main tasks (action execution, utility computation and information distribution) are handled by its three components: The *Execution Engine* brings together the action choices received from all agents' Action Generators and applies the *action execution noise* functions to them in order to simulate misimplementation (execution failure) on the side of the agents. The resulting (actually performed) actions are propagated to the *Utility Generator* and to the *Perception Generator*.
The Utility Generator receives the executed action tuples from the Execution Engine and computes the resulting payoffs for all agents using the payoff function u. Before the payoff values are passed on to the Perception Generator, they are filtered through a *payoff perception noise* generator which belongs to the Utility Generator as well and which models the misperception of payoffs.
The Perception Generator, finally, distributes the action and payoff information received from the Utility Generator and the Execution Engine to all agents' Utility Learning Modules after applying the *action perception noise* functions to them, a noise function that simulates the erroneous perception of action choices due to sensory failure.

Table 4.5 captures the functionality of the Simulation Engine in detail. This approach to designing an "environment simulator" in the proposed way is quite advantageous: it encapsulates all the uncertainty factors in the "simulation side", thus allowing for a neat separation of agent and environment, such that agents are reduced to the cognitive processes they embody, while all system properties are part of the environment.
Another important feature is that one uncertainty component is associated with each Simulation Engine component. Thus, we ensure that every possible point where uncertainty might come in has been considered.
Furthermore, the Utility Generator is the sole component of the whole system that has an explicit representation of the exact payoff function. This ensures that agents have no other possibility whatsoever to reason about the interaction than to learn with the information that is revealed to them by the system.

4.4.4 Simulation algorithm

Before presenting a pseudo-code version of the rather simple top-level algorithm that controls the game simulations, we need to make some additional formal constructions

Component:	Execution Engine
Inputs:	agents' action choices (from Action Generator)
Internal Parameters:	action execution noise
Functions:	compute actually executed actions
Outputs:	executed agent actions (to Utility Generator) executed agent actions (to Perception Generator)
Component:	Utility Generator
Inputs:	executed actions (from Execution Engine)
Internal Parameters:	actual payoff function payoff noise function
Functions:	compute agents' payoffs
Outputs:	payoff values (to Perception Generator)
Component:	Perception Generator
Inputs:	agent payoff values (from Utility Generator) executed agent actions (from Execution Engine)
Internal Parameters:	action perception noise function
Functions:	compute agents' percepts
Outputs:	agents' percepts (to all agents' Utility Learning Modules)

Table 4.5: **Simulation Engine** components and their functions.

concerning the way the Simulation Engine provides the input for the agents and receives their output, i.e. a formal model of how the *history* of the game evolves.

For this purpose, let δ_i be the action transition function as defined in Equation 4.3. We define the following probabilistic noise functions for the Simulation Engine: a family $\{n_i^{ae}\}_{i\in\mathcal{P}}$ of action execution noise functions, such that $n_i^{ae} : A \to A$ maps each action of i to another (or the same) action with some fixed probability; a second family $\{n_i^{pp}\}_{i\in\mathcal{P}}$ of payoff perception noise functions, such that $n_i^{pp} : \mathbf{R} \to \mathbf{R}$ alters the payoff i perceives from the payoff it actually obtained. Similarly, action perception noise functions $\{n_i^{ap}\}_{i\in\mathcal{P}}$ are provided which affect the perception of joint actions for i ($n_i^{ap} : \mathcal{A} \to \mathcal{A}$).
Then the game history on which δ_i operates after round t is given by the previous history H_i^{t-1} and percepts and actions of round t:

$$H_i^t = \left(H_i^{t-1}, \left(n_i^{ap}(n_i^{ae}(a_i^{(t)})), n_i^{pp}(u_i(n_i^{ae}(a^{(t)})))\right)\right) \tag{4.4}$$

Thus, the history at round t is given by concatenating the agent's experience before the last round H_i^{t-1} with the joint action/individual payoff pair, where the perceived joint action is the (noisy) executed joint action as filtered through the action perception generator, and the payoff noise is additionally applied to the actually performed joint action to supply the payoff that is actually perceived by i. Using the joint transition function δ of all agents, we can reformulate Equation 4.4 as

$$H_i^t = \left(H_i^{t-1}, \left(n_i^{ap}(n_i^{ae}(\delta_i(H_i^{t-1}))), n_i^{pp}(u_i(n_i^{ae}(\delta(H_i^{t-1}))))\right)\right) \tag{4.5}$$

which provides us with a recursive formula, the reverse of which (a sequence starting with some initial joint action $a^{(0)}$) can be used to construct a simple iterative algorithm as presented in Table 4.6.

GAME-SIMULATION($duration, \mathcal{P}, \mathcal{A}, u, \delta, n^{ae}, n^{pp}, n^{ap}$)
  /*$\mathcal{P}$,$\mathcal{A}$, and u determine the game being played, *duration* is the number of rounds,
    δ is a family of initial action transition functions and n^{ae}, n^{pp}, n^{ap}
    are the families of currently applied noise functions */
  FOR $t = 1$ TO *duration* DO
    IF $t = 1$ THEN
      pick a randomly from $\mathcal{A}$
    ELSE
      FORALL $i \in \mathcal{P}$ DO
        /* update the transition function agent according to the new percepts */
        $\delta_i \leftarrow \delta_i(H_i^t)$
    END;
    /* the agents pick the next action according to the updated transition function */
    $a \leftarrow \delta(H^t)$
    /* the new payoff/action information is added to the game history */
    $H^{t+1} \leftarrow cat(H^t, (n^{ap}(n^{ae}(a)), n^{pp}(u(n^{ae}(a)))))$
  END;
END

Table 4.6: The Game Simulation algorithm for finite iterations of an n-player normal form game $\Gamma = \langle \mathcal{P}, \mathcal{A}, u \rangle$.

4.5 Summary

This completes the presentation of the layered interaction-learning architecture, and it also ends the rather theoretical *conceptual* part of the work, in which we have presented a discussion of the game-theoretic background of the analysed interactions (particularly as concerns the resource-load balancing scenario), and an intuitive outlook on the general learning methodology we employ. Furthermore, we have provided a rigorous treatment of the learning problems we confront, and starting from these we introduced the LAYLA architecture, the alleged answer to the research questions stated in Chapter 1.
What we would like to have, however, is a concrete system that proves the validity of our line of argumentation, and this is the subject of the remaining chapters: building an experimental prototype whose design follows the proposed principles and evaluating its adaptive capabilities in the repeated multiple-access resource-load balancing game.

Chapter 5

An experimental prototype

Presenting a prototypical system that realises the methodology laid out in the previous chapter involves a detailed depiction of sample instances for the learning algorithms and decision rules that can be used to build operational LAYLA agents, so that ultimately a full functional description of the agents' behaviour can be provided (bearing in mind the top-level architecture as it has been discussed in the previous chapter).
The system we devise has to be thought of as a proof-of-concept implementation that claims by no means to optimally exploit all the possibilities of complex hierarchical learning. Instead, it will be considered sufficient to build an adaptive agent system that demonstrates how coordinated behaviour can evolve with very little a priori knowledge. Such a demonstrator should typically be characterised by the fact that system performance provably increases over time and that its performance is better than that of a comparable non-adaptive system. We will not assume that agents can *perfectly* learn the concepts necessary to behave optimally in an interactive environment, so we will refer to optimal performance (as achieved by enacting the optimal solutions for the underlying games provided in Chapter 3) only as a *measure* for the successful completion of our task, not as the task itself.

The upcoming sections are organised as follows: we first provide a concise overview of the system that summarises the employed learning algorithms. Then we present, one after another, the detailed design of the *Utility Engine*, the *Strategy Engine* and the *Social Behaviour Engine*. For each of these, we reconsider their learning tasks, discuss adequate learning algorithms for them and justify our own design decisions. The presentation of the respective algorithm, which constitutes the central part of each of these expositions, is followed by examples and preliminary empirical results that explain how the components can be fine-tuned and why they work in practice (a thorough empirical validation of the entire system will be carried out in Chapter 6).

5.1 Overview

In designing learning systems, various choices have to be made that pre-determine to a great degree the adequacy, performance and transparency of the final design.
Mitchell (1997, p. 13) provides a simple pattern for the main choice points in the ML design process consisting of four major steps. The first step is to determine the type of training experience that will be made available to the learner, the data that will be used to update the algorithm's current hypothesis.

In the second step, the target function has to be defined that specifies exactly what type of knowledge will be learned and how it will be used by the performance system.
This is followed by choosing an appropriate representation for this target function. The representation defines the inputs and outputs of the function, and it determines the *hypothesis space* that will be searched by the learning algorithm. Ultimately, the learning algorithm itself has to be determined, the component that updates the learner's current hypothesis on incoming training data.
As concerns our own learning architecture, the first two steps have already been completed in the previous chapter, where we explained that the training experience can only consist of past action/payoff percepts and defined target functions π_i, f_i and h_i formally for the learning layers.
Thus, we can basically concentrate on presenting the function representations and the employed learning algorithms, even though the layered character of our learning system will additionally require us to explain how the decision-making modules of the layers interact, i.e. we will have to define the concrete instances of e_i, m_i and l_i that we employ.

Our basic design choices regarding the learning algorithms can be summarised as follows: to approximate the payoff function we use multi-layer neural networks with standard back-propagation and binary input representation. This is is the simplest of the the employed algorithms, and the section on the Utility Learning Module will mainly consist of a description of how to fine-tune the nets to suit our purposes.
Learning OBP models requires somewhat more elaborate concepts. We have settled on a combination of genetic algorithms and instance-based learning, which uses individuals in a population of "opponent behaviour predictors" to anticipate the next opponent action. Again, standard machine learning algorithms are employed, but a new possibility of combining them is shown.
The suggestion of optimal-coalition-yielding actions is certainly the most challenging problem of the three, so the design of the Social Learning Module requires several steps of abstraction from the learning task to get where we want to. Our approach is based on techniques that stem from the field of *user modelling*, especially on nested models of preference structures (so-called *recursive belief models*). Agents will use these to construct hypotheses about their opponent payoff functions based on the knowledge of opponents' past actions. We have designed new formalisms to represent the payoff dependencies between agents (*gain models* and *probabilistic ordering models*), update algorithms for these and, also, a method to simulate the opponents' reasoning process with the current knowledge. Finally, a "cooperative" action selection rule is devised that aims at suggesting those future actions that will make the opponent act in the most desirable way. This is the most experimental part of our architecture, and it therefore deserves being treated with more detail than the other two.

Unlike the learning algorithms which require a considerable amount of sophistication, the decision-making functions that integrate their results will be rather simple. Their design mostly relies on certain *error margins* of learners which are used to "switch super-layers" off and on and on the existence of *socially feasible* actions which implement compromiseful behaviours whenever this is possible. If such actions don't exist, greedy Strategy Engine choices are turned into action. Admittedly, this is a rather naive way of connecting the three layers, but since it affects their *decision rules* only (the *learning algorithms* themselves will be updated even while their results are not being used for decision-making purposes) we think of it as a means to keep things reasonably simple.

We first describe the Behaviour-Based Layer of our agents, the Utility Engine.

5.2 Learning the payoff function: the Utility Engine

The learning module of the Utility Engine approximates the agent's own payoff function by training an artificial neural network (ANN) with the pairs of joint action and associated payoff received from the Simulation Engine after each round of the game. The purpose of this payoff approximation is to provide the Strategy Engine (more precisely, the Strategy Generator) with a means to reason about future payoffs so that it can choose utility-maximising actions.

5.2.1 Designing a payoff learning component

There are probably countless ways an agent could go about constructing an approximation[1] $\pi_i : \mathcal{A} \to \mathbf{R}$ of the actual (unknown) payoff function u_i given a number of joint action/local payoff pairs[2]

$$\langle a, u_i(a) \rangle \quad .$$

The most naive approach would be to simply store these pairs as entries of a payoff matrix so that simple table-lookup can be used to predict the payoffs of particular action combinations. Constructing such a payoff matrix is unfortunately highly infeasible in n-player game due to the "combinatorial explosion" in the number of possible action combinations that agents are faced with in non-trivial applications. If we take the action sets to be equal for every player (as in the resource-load game), then $\mathcal{A} = A^n$ if n players are involved, so that $|\mathcal{A}| = |A|^n$, a number exponential in the number of players. Even in the rather small example of ten players and five resources, this yields

$$2^{kn} = 2^{5^{10}} \approx 1.13 \cdot 10^{15}$$

as the total number of possible joint actions, and it is clear that an agent cannot be expected to wait more than 10^{15} rounds[3] to model its payoff function accurately.
Thus, a learning algorithm is required that can *generalise* from a relatively small number of samples to predict the payoff values of unseen combinations.

As Mitchell (1997, p. 81) points out, neural network learning methods "provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions", and for many tasks they have proved to be one of the most effective currently available learning methods, especially for tasks that require learning to interpret complex, noisy sensor data.

[1] We shall use the notation introduced for the resource-load balancing game in Section 3.2 throughout the following chapters to underline the fact that many of the particular design decisions use certain properties of the class of games we examine. In particular, we will use $A/\mathcal{A}$ instead of S_i/S as the set of (joint) actions and we will make frequent use of the resource set $\mathcal{R}$ and binary representations $\beta(a)$ of actions.

[2] Actually, these pairs should include the Simulation Engine noise functions, i.e. they should read $\langle n_i^{ap}(n_i^{ae}(a)), n_i^{pp}(u_i(n_i^{ae}(a))) \rangle$, but for reasons of notational convenience we take a and $u_i(a)$ to be the *perceived* joint action and payoff.

[3] Note also that this is actually a very optimistic lower bound for the minimum number of rounds required to fill the matrix, because it can only be guaranteed that some actions don't occur repeatedly if exploration is perfectly coordinated among agents: only if they agree on a way to "enumerate" all joint actions can the set $\mathcal{A}$ be completely explored in $|\mathcal{A}|$ rounds.

Neural networks are densely interconnected sets of simple units that apply some simple function to their inputs (which are possibly the outputs of other units) to compute a local output (which, again, possibly provides the input for other units). Two fixed subsets of the set of units, so-called *input nodes* and *output nodes* are used to feed the input values (according to the chosen representation) of the training samples into the net and to determine the output predicted by the net for the (possibly unseen) sample. The computation of outputs is done by propagating the inputs from unit to unit while computing the results of the unit functions and weighing them with *weights* associated with the links between units.
Updating the current hypothesis of the net (which is reflected by the current values of all links' weights) is achieved by computing the output of the training sample input according to the current net, determining the error between this predicted output and the real output (as given by the sample) and updating the weights according to the computed error, so that the predicted output matches more closely the true output with respect to the new sample.

There are several reasons that suggest the use of ANNs for payoff approximation. Firstly, the training sample inputs, joint action tuples of the form $(a_1, \ldots, a_n)$, consist of integer vectors and the sample outputs are real-valued payoffs. This rules out the possibility of using *symbolic* sample representations since it requires the approximation of a real-valued function.
Secondly, neural computation is considered a *sub-symbolic* learning method for which fact it perfectly suits the intuition behind the Behaviour-Based Layer that is supposed not to decompose the representations of actions and percepts further and to implement rather sub-cognitive reasoning and learning capabilities.
A third reason is the expressiveness of neural networks. Theoretical results have shown that if an appropriate network structure is used, *any* boolean function (Russell and Norvig, 1995, p. 583) and any *continuous* function (Cybenko, 1988) can be represented by such networks. This means that, even though we choose the specific network structure according to the properties of the resource-load game here, the *method* as such is suitable for a very wide range of possible payoff functions.
Another reason is that the most prominent drawback of neural networks, namely their lack of *transparency* can be completely ignored in our learning architecture. Instead of using explicit representations to formulate a learning hypothesis, neural networks encode their hypothesis in many real-valued weights of network links which makes the analysis of the learned concept in an explanatory fashion impossible. As Russell and Norvig (1995, p. 584) put it, "even if the network does a good job of predicting new cases, many users will still be dissatisfied because they will have no idea *why* a given output value is reasonable".
However, this disadvantage bears no effect on our agent learners, since they won't *reason* about the results of the neural network but use it only as a utility-predicting device. It is therefore of no relevance to them how the interdependence of utility values comes about as they are only interested in the effects of these on their own welfare.

In devising the concrete networks that will be used in our system, choices have to be made regarding input and output representations of the training examples, the network structure, the propagation functions of the network units and the weight-updating rule. Furthermore, the internal parameters of the networks, such as the learning rate and the initial weights have to be fine-tuned to yield optimal results. Another important issue is

the training strategy, the way in which the available examples will actually be used to train the net. These issues are the subject of the next section.

5.2.2 Constructing payoff-learning neural networks

Network type

Multi-layer feed-forward networks with sigmoidal units are a commonly used form of ANNs. Such networks have an acyclic structure, i.e. there is no path from any unit to itself. The *input layer* and *output layer* are thus clearly separated from each other, and a fixed number of intermediate, so-called *hidden layers* each of which consists of a number of *hidden units* connects the input layer to the output layer. We adopt this class of networks because their representational power is much greater than that of feed-forward networks without hidden layers (so-called *perceptrons*) and also because a simple and well-understood training algorithm called the BACKPROPAGATION algorithm can be applied to them. This algorithm aims at minimizing the mean squared error between the true and the predicted output of some sample input, i.e. the quantity

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

where D is a set of training examples, *outputs* are the output nodes of the network, t_{kd} is the kth component of the *true output* of the training example and o_{kd} is the output of the kth output node in the network (the kth component of the *predicted* sample output). This mean squared error is a function of the network weights $\vec{w}$ since they determine how the predicted outputs are computed.
We will not explain in full detail the algorithm as such and refer the interested reader to Chapter 4.5 in (Mitchell, 1997) for an extensive discussion. Here, we will concentrate on the decisions we have to make in constructing networks for our application: input and output representation, the structure of hidden layers, the range of values from which we choose the initial weights, the learning rate and termination criteria that will apply for the training procedure.

Input representation

For our purposes using binary input representations has been found to be much more effective than decimal inputs, i.e. we prefer to use

$$\langle \beta(a_1), \ldots, \beta(a_n) \rangle$$

as the input representation of training examples rather than

$$\langle a_1, \ldots, a_n \rangle \quad .$$

This has been proven through extensive testing (Section 5.2.3 provides the details), but it is also intuitively clear that u_i can be much more precisely expressed in terms of the resource access that is realised by all agents' actions rather than by the decimal "black-box" equivalents of these actions.
Unfortunately, this result forces us to pre-assume that the agents have some additional *a priori* knowledge of system properties in that they know that payoffs depend on the way resources are accessed by the society (a similar assumption will be made by the Strategy Engine learning algorithm). A fact that partly alleviates the restriction imposed by

this assumption is that decimal-input neural networks are also able of approximating the function, albeit their performance being worse.
Using binary input representations implies that the network has $n \cdot k$ input nodes, one for each player and resource. Compared to decimal nets which only need n input nodes, this results in additional space requirements and computation time needed for back-propagation.

Output representation

As far as output representation is concerned, there are essentially two alternatives: a single, real-valued output unit could be used or several units whose outputs are combined to compute $u_i(a)$.
One possibility of using vector outputs would be to derive the output complexity from the number of agents and resources. One of the results of the resource-load balancing game definition is that there is only a finite number of utility values that an agent can obtain for *one* resource depending on the *opponent load* that the resource has to manage. These payoffs form the set U defined as

$$U := \left\{ \left(\frac{v}{T} - c\right), \quad \left(\frac{v}{2T} - c\right), \qquad \ldots, \left(\frac{v}{(n-1)T} - c\right) \right\} \quad .$$

The total payoff $u_i(a)$ can then be represented by a boolean function $b(a) : \mathcal{R} \times U \rightarrow \{0; 1\}$ which yields one if the payoff received for resource $r \in \mathcal{R}$ is $u \in U$ under the joint action a. If the set U is known, the sum of all resources whose b-values are 1 can be taken to compute $u_i(a)$.
This example for a vector-valued output representation illustrates why we choose to equip the networks with a *single* output node: firstly, knowledge of the set U would require far more in-depth knowledge of the payoff function than what was assumed in the previous paragraph – it would involve (a) knowing that resource utilities decrease with increasing opponent load and (b) knowing that the total payoff is the sum of the individual resource payoffs. In fact, any output representation that is based on decomposing the computation of payoffs[4] involves further knowledge of system properties and this would force us to further depart from the initial motivation as presented in Chapter 1.
Secondly, we believe that such decomposition may very well be *implicitly learned*, in that hidden layers may, for example, effectively split the payoff computation into several resource payoffs that are summed, only that this property of the target function is *learned* rather than pre-assumed. We therefore leave these intermediate steps to the hidden layers and use a single output unit that brings together the results of possibly distributed internal representations.

When using a single real-valued output unit, an additional issue arises due to the *sigmoid* nature of the network units. The function by which they compute their outputs always yields some value between 0 and 1, i.e. it maps the whole range of real numbers to a very small interval (it is therefore also called the "squashing function").
If n is a sigmoid unit that output values between $[0; 1]$, we have to find some mapping to this range from the range $[0; u_{max}]$ of actually obtained payoffs. In order to avoid implicit knowledge of the maximal payoff, we use the heuristic of agents dividing the obtained

[4]Vector representations that don't follow this "decomposition" intuition, such as the possibility of having several integer output units to represent the *digits* of the payoff value, have not been examined.

payoff by the *currently known* maximal utility. The estimate for u_{max} will be constructed by simply keeping track of the maximal perceived utility value, i.e. if the current percept history of agent i is H_i^t (the action/payoff pairs from the game history H^t that consist of the performed joint action and the payoff for i for each round) we define

$$u_{max} := \max_{\langle a, u_i(a)\rangle \in H_i^t} u_i(a)$$

and use as an output representation of training samples $\langle a, u_i(a)\rangle$ the single-entry vector

$$\left\langle \frac{u}{u_{max}} \right\rangle,$$

a value that is obviously always smaller than one.
Approximating the maximal payoff value implies that the agent will inevitably make mistakes in training the net at the beginning of the game, but it also means that it always trains the network with respect to the *currently relevant* range of outputs, which can be advantageous if certain action combinations never occur.

Hidden layer structure

As regards the issue of choosing an appropriate hidden layer structure, it can only be stated that determining a reasonable hidden layer structure is solely based on the experience we have gained by testing hidden layers of various sizes (Section 5.2.3 gives an account of these tests), where it was found most effective to use *two* hidden layers of size $2n$ each. It is not surprising that the optimal hidden layer structure depends on the number of players n, so we will obey this rule for all test sets in the following.

Training strategy

Defining a termination criterion is a subtle issue in active learning architectures as the one we propose: while in passive learning environments some margin can be defined so that training terminates as soon as the prediction accuracy of the network lies above that margin, active learners have to trade off accuracy against action. If, for example, after each sample the agents trained the net until it reached an accuracy of, say, 99%, then the required computation time would not only lead to a delay in the agent's response but this would also mean that the agent will have to wait very long until it obtains the next sample, so that it would effectively make a big effort to "learn a lot with very little experience". Considering that generalizing from a large number of samples is one of the key aims of learning algorithms, this would certainly not be a very effective learning strategy. There is a huge variety of possible training strategies, ranging from very *lazy* to highly *eager* learning. For payoff learning networks, an extremely lazy way of learning would be to feed only the new action/payoff pair into the network and to perform one back-propagation iteration. At the other end of the spectrum we can find extremely eager strategies, e.g. a strategy that not only re-feeds *all* past samples into the nets but repeats the back-propagation for every example until the mean prediction over all known samples falls below 10% of the previous mean error.

We choose a compromise solution between these two extremes that respects the issue of bounded rationality while making most out of the current experience at the same time, a strategy we will call the *constrained size feed-all-once* strategy. It consists of feeding

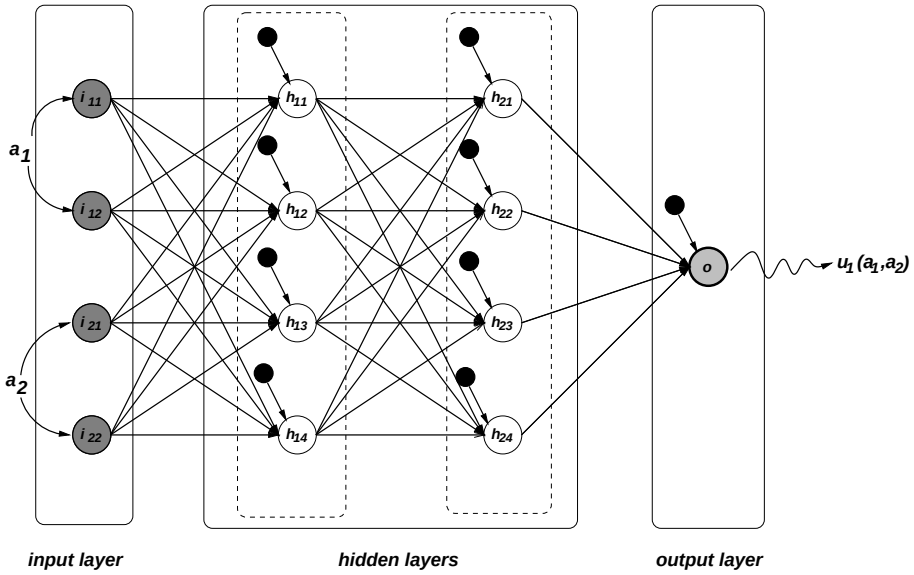


Figure 5.1: Network for a two-player two-resource game with two hidden sigmoid unit layers of size $2 \cdot n$. The four binary components of a_1 and a_2 are fed into the input layer units ($\beta(a_i)[j] = i_{ij}$) and propagated through the hidden layer units h_{ij} to the output unit u. The black circles represent the *threshold units* that provide some input-independent basic activation to each sigmoid unit (the weights of the links between these and their target units are also updated during back-propagation).

all known samples into the net in each round, but conducting only one back-propagation update step for each samples. Also, it uses "constrained" training sets, i.e. it starts replacing the most obsolete samples in a first-in first-out manner as soon as the *maximum training set size* (some positive integer constant) is reached.
This strategy has two major advantages: Firstly, it realises *oblivious* learners, who are not perfectly rational and concentrate on currently relevant experience gained within a reasonably recent interaction history. Secondly, and much more importantly with respect to evaluating the architecture, it makes the *learning progress* directly measurable in terms of game duration. The idea behind this (which also serves as a principle for the learning strategies of the other algorithms that will be introduced) is that, if we know that one learning iteration is performed with every round of the game, then the number of rounds required to achieve the learning results necessary to behave in a coordinated fashion will provide a measure for how difficult the overall learning task is.
Figure 5.1 shows an example for the final network design in a two-player game with two resources.

In the next section, first empirical results with the nets for various problem sizes are reported as well as the resulting fine-tuning decisions concerning initial weights and learning rate.

5.2.3 Preliminary results

Testing and fine-tuning the neural networks is different from testing the learning algorithms we suggest for the Strategy Engine and the Social Behaviour Engine: The most distinguishing property of payoff approximation is that we have *perfect knowledge* of the concept to be learned, given that the payoff function u is part of the definition of the

resource-load game. This implies that, at any stage of the learning process, training samples can be generated and the exact prediction error of the networks to them can be determined. This is not possible for the OBP problem and cooperation potential learning, because the opponents' reasoning processes that are supposed to be learned are *not* pre-determined since they depend on their own learning results. Our network evaluation strategy will therefore consist of generating a set V of validation samples $\langle a, u(a)\rangle$ by using the real payoff function every t rounds, and we will measure for each network the mean difference between network output and actual payoff, i.e. the mean prediction error $E(V)$ of p_i's network with respect to V:

$$E(V) = \sum_{\langle a,u(a)\rangle \in V} |\pi_i(a) - u_i(a)| \quad .$$

For small problem sizes (if $|V| \geq 2^{k^n}$), this mean error is actually identical to the performance measure $\mathbf{P}(\pi_i)$ defined for the L^{IM} problem in Section 4.2.2, while for larger problem sizes it is certainly a reasonable approximation of $\mathbf{P}$.
Secondly, the *parameter space* is much larger than for the other two learning problems. In order to completely evaluate all possible design decisions, we would have to consider every possible hidden layer structure, the whole range of possible initial link weight distributions and learning rates and to compare binary input networks to decimal input networks. Clearly, we cannot explore this parameter space exhaustively, so we need to constrain it to a local search by fine-tuning one parameter after the other in a best-first manner. This is to say that we first compare binary to decimal nets, then different hidden layer structures for the binary nets (which we choose after the first step) and finally optimise the choice of initial weights and of the learning rate.
In that, it is of course impossible to conduct these experiments for *any* problem size and therefore we restrict our analysis to two specific *test sets*: a game G_I with two players and two resources and a game G_{II} in which ten players compete for access to five resources[5]. It is our aspiration that experiments with these will hint at parameter choices that are in some way a function of the problem size $\langle n, k, v, T, c\rangle$. If this is the case, this mapping can be used for any other MARLOG to provide more general guidelines for decent, if not optimal, network design.

We start our analysis by comparing networks that use a binary input representation to decimal-input networks. Figure 5.2 shows plots of the mean error of both types of networks in G_I computed over 500 validation samples every ten rounds (V was separately randomly generated for each evaluation step). As in all of the following simulations, agents' individual action choices were made at random, so that exploration was maximal. The training sets were not constrained in size, so that agents always stored all past samples and fed them once into the net together with the newly obtained sample in each round.

We have chosen to conduct all experiments with full a priori knowledge of the maximal payoff value because we intend to measure the minimum number of rounds after which a reasonable prediction accuracy has been reached by the network as such, i.e. not affected by the initial errors due to mis-estimating the maximal payoff (in the system-level experiments of Chapter 6 we will lift this assumption again).

[5]We often will only present plots of the learner's performance for one of these games; in that case the results have always been validated for the other game.

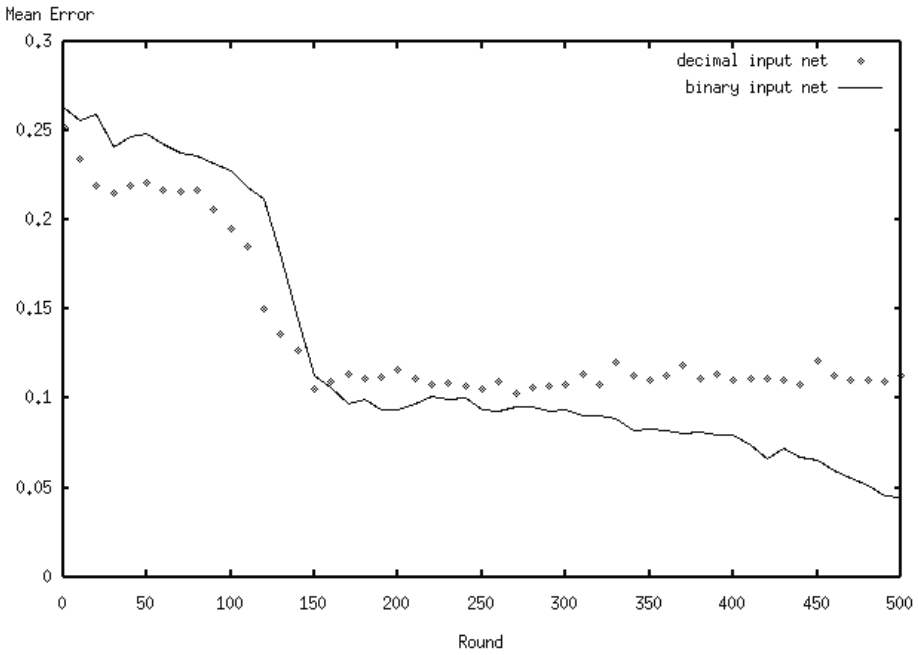


Figure 5.2: Mean error of a binary and decimal input representation employing networks in G_I. The superiority of the binary input networks carries over to the larger test set G_{II} and is independent of hidden layer structure, learning rates and initial weight distributions.

It can be seen that the learning accuracy of the binary network clearly exceeds that of the decimal input net. In fact, the decimal input seems unable to improve its performance further after a certain number of rounds, which suggests that there is a natural limit to approximating u as a function of decimal inputs. This is a rather impressive result, given that in G_I only 16 ($= 2^{2^2}$) function values have to be learned. This effect is even more dramatic for the "larger" game G_{II} and we therefore adhere to the use of binary input representations for the remaining experiments.

Next, the hidden layer structure of the networks is subject to our analysis. Experiments with a single hidden layer showed that, although differences were often only marginal, optimal hidden layer sizes are related to the problem sizes n and k, i.e. the smoothest convergence curves were produced by using hidden layers of size n, $n \cdot k$, $2 \cdot k$, $2 \cdot k \cdot n$ and so on. This suggests a regularity that seems quite intuitive, namely that the optimal hidden layer structure is some function of the problem dimensions.
To investigate this issue further, tests were conducted with various "hidden layer vectors" $\langle h_1, \ldots, h_s \rangle$ so that each h_j was the size of the j-th hidden layer, typically k, n or some simple function (product or sum) of these values. Figure 5.3 shows a comparison between the performance of some of the resulting networks for G_I and G_{II} which suggests that neither vectors more complex than $\langle 2n, 2n \rangle$ provided a further increase in performance nor hidden layer vectors simpler/smaller than the two-layer $2n$-sized version. This provides us with a simple hidden layer construction rule.

The last step in fine-tuning the networks, which consists of selecting appropriate initial weights and the right learning rate η, turns out to be much harder than one might be lead to think. A first notable result is that for G_I learning rates much larger than the small fractions usually employed (a textbook treatment suggests $\eta = 0.05$ as a typical value) are

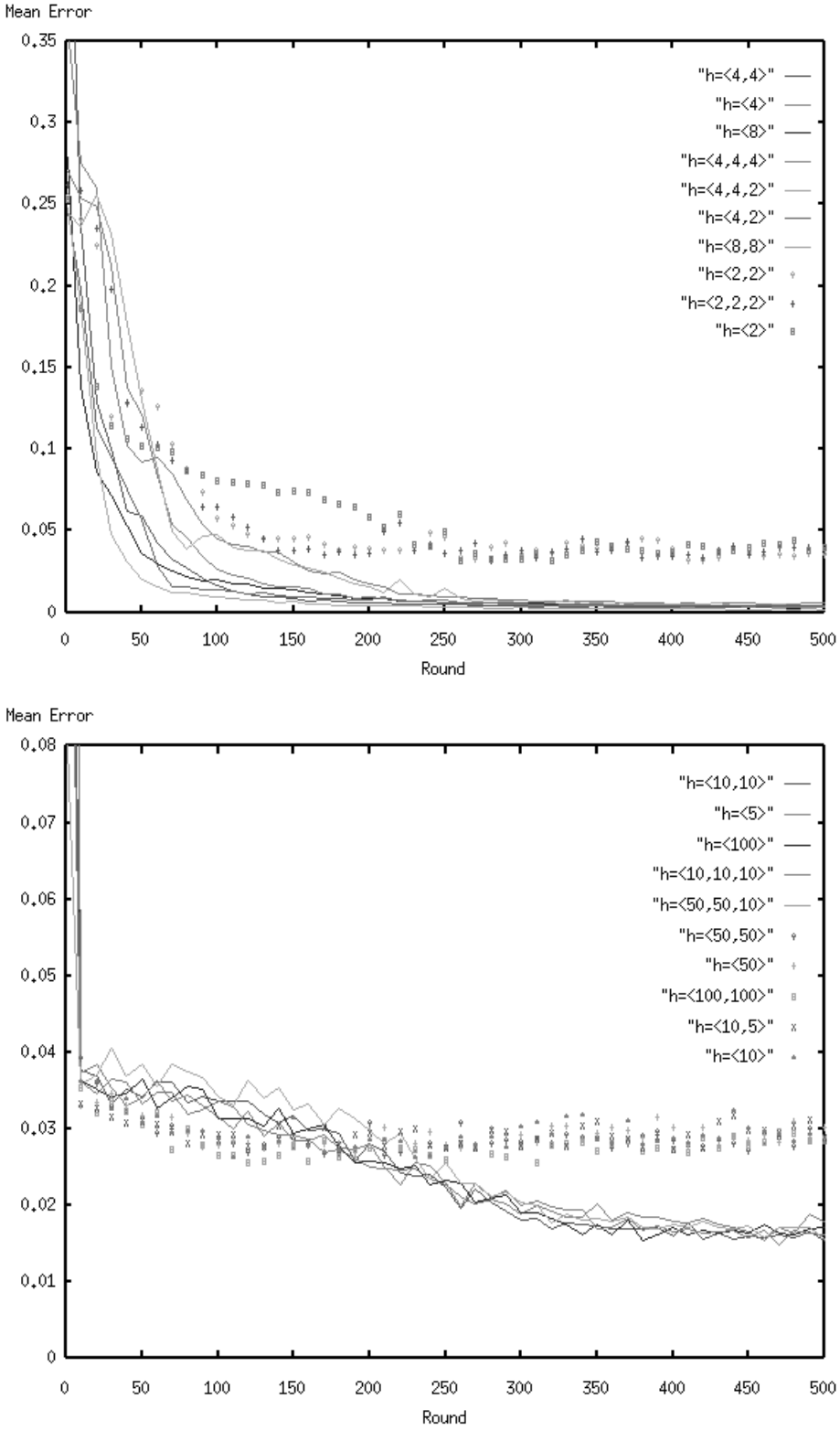


Figure 5.3: Experiments with various hidden layer structures. The simplest hidden layer structure that yields high performance for both games follows the pattern $\vec{h} = \langle 2n, 2n \rangle$ (dotted curves denote the performance of those hidden layer structures whose performance was judged inacceptable).

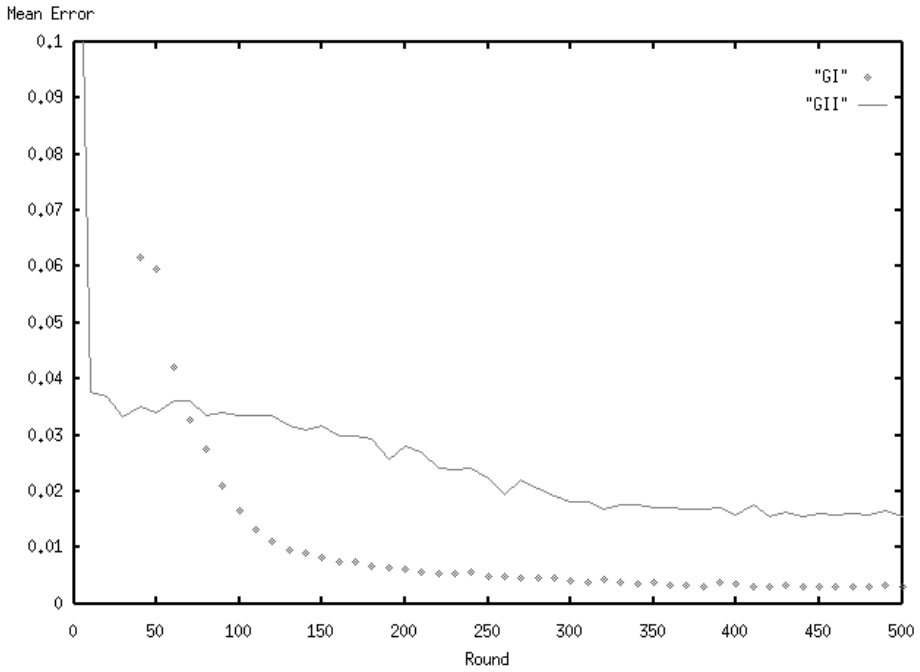


Figure 5.4: Performance of the optimally tuned networks for G_I (dotted curve) and G_{II}. In the smaller game, the average error over 500 samples has fallen by 76% between round 100 and 300, while for G_{II} it only decreases by 47% in the same time span.

much more effective than smaller η-values. The most extreme difference that occurred was that the mean error was 0.9% under $\eta = 10.0$ and 24% under $\eta = 0.1$ after 100 rounds of training (with initial weights chosen randomly from $[-0.5; 0.5]$). Unfortunately, the converse holds for the other test set, i.e. small learning rates yield much better results. After extensive testing with both test sets and learning rates from 0.01 to 50.0 we settled on the use of

$$\eta = \frac{v}{k^2 \cdot n}$$

as a rule of thumb, so that we applied $\eta_I = 2.5$ and $\eta_{II} = 0.4$ as learning rates in the two games. Additionally, a learning rate update rule is used to slowly decrease the learning rate after each iteration by taking

$$\eta \leftarrow 0.997 \cdot \eta$$

after each round.
By cross-testing all explored learning rates we also determined the optimal range of initial link weights as

$$[-\eta; \eta]$$

This completes the presentation of preliminary results for the payoff-learning neural networks. Figure 5.4 shows plots for the resulting network payoff performance in G_I and G_{II}. It can clearly be seen that learning progress decreases with increasing learning problem size – which is not particularly surprising, given the increased complexity of the payoff function – and it is up to the overall system evaluation in Chapter 6 to validate whether the learning performance of the Utility Engine suffices to learn the agent-level task successfully.

5.3 Learning to (re)act optimally: the Strategy Engine

The task of the Strategy Learning Module is to approximate the expected future opponent behaviour, to solve the learning problem L^{OBP} as introduced in Section 4.2.2. Such a model can then be used by the Strategy Generator to generate an action-value function

$$m_i : A \rightarrow \mathbf{R}$$

that models the desirability of performing a particular action $a_i \in A$ for agent i. The idea behind this combination of learning and action selection is that if the next action(s) of opponents can be predicted and the Utility Engine provides an accurate model of the agent's own payoff function, then that action can be chosen that ensures the highest payoff for the expected opponent action.

The strategic learning algorithm that we propose combines the techniques of *Genetic Algorithms* (GAs) and *Nearest-Neighbour Learning*, a very simple instance-based learning paradigm often employed in case-based reasoning (Mitchell, 1997, Chapter 8). It uses genes to encode opponent *action transition rules* and selects by nearest-neighbour search those genes that match the current situation most closely. These are then consulted to predict the next opponent action, so that m_i can be constructed accordingly.

We start the discussion of the employed learning methodology by some preliminary considerations regarding the nature of the learning problem and how we go about solving it.

5.3.1 Designing a strategy learning component

As in the previous chapter, let us start with a naive, *ad hoc* opponent behaviour predicting learning algorithm. Since OBP models aim at constructing a function that reflects the expected payoffs that can be ensured, we can – recalling Definition 3.2 of Section 3.1.1 – take $m_i(a_i)$ to be the expected payoff of $a_i \in A$, so that

$$m_i(a_i) = u_i(a_i) = \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \cdot P(a_{-i}),$$

where

$$P(a_{-i}) = \Pi_{1 \leq j \leq n, j \neq i} \alpha_j(a_j)$$

is the probability of the joint opponent action a_{-i}, given the (mixed) strategies $\alpha_j(a_j)$ currently played by i's peers.
Obviously $\alpha_j(a_j)$ can be approximated by keeping track of all agents' action frequencies, the ratio between the number of rounds in which agent j played a_j divided by the total number of rounds so far. Further, the payoff approximation π_i can be used as a substitute for u_i. Wouldn't this provide us with a perfectly simple and yet effective strategy learning algorithm?
The answer is *no*. Firstly, such an algorithm would require, just like the naive "matrix memorization"-method mentioned in the previous chapter, an exponential number of payoff and probability computations to obtain the m_i-values for all actions (a number of $|A^{n-1}| = 2^{k^{n-1}}$ in this case) which places it beyond the realm of computationally

tractable algorithms. Secondly, and perhaps conceptually more important, such an action-value computation suggests that the *future* opponent strategies are identical to their *past* strategies, because it is not capable of modelling dynamic behaviour changes (the action frequency ratio implies that a single mixed strategy is approximated over time) and this is quite clearly what we do *not* want. Thus, once more, we have to look for something more elaborate than that.

Strategic learning and reasoning is very different from social reasoning, despite the fact that it cannot be conducted without considering knowledge an agent has about other agents (since the own action outcomes of i inevitably depend on its peers' behaviour). The latter observation could lead to the assumption that strategic reasoning *is* social reasoning in a way, which raises the question of whether it can or should be separated from the Social Planning Layer at all, i.e. whether the very existence of the Strategy Engine is justified.
To justify the separate view of the two layers, we should explain more precisely what we mean by "social" and "strategic" reasoning: their key difference is that the strategic component views peers as a "mechanical" part of the environment, i.e. as variables that exhibit a certain behaviour according to certain rules and not as intentional entities that are capable of managing such concepts as compromise and exploitation.
The social reasoning component, on the contrary, assigns to peers such capabilities as knowledgeability and rationality, a fact which bears very different implications for the agent's own decision-making from the assumptions underlying the strategic reasoning mechanism.
This does, however, *not* mean that strategic reasoning excludes the possibility of having rational co-actors, because after all we design it for environments in which all agents can in one way or the other be called rational. It rather aims at decoupling opponents' *behaviour* from their *reasoning* processes from the modelling agent's point of view, so that the behaviour they exhibit is seen as the effect of their reasoning, but not as reasoning *per se*.
The most important consequence of this view for the design of the respective learning components is that the strategic learners will try to adapt to changes in opponents behaviour while assuming that the grounds opponents base their reasoning on remain constant throughout the interaction. The dynamics of opponent reasoning will only be considered at the social reasoning level because it is the Social Behaviour Engine that aims at finding out how the opponent can be made to alter its behaviour and this can only be achieved if the opponent's reasoning process is modelled.

This rather theoretical discussion of the essence of OBP learning serves as a starting point for specifying crucial properties of the L^{OBP} problem that justify the design of the proposed learning algorithm, which are presented in the following paragraphs.

Opponent "automata"

One of these properties results from the observation that the Strategy Learning Module ignores the existence of reasoning mechanisms behind the effected opponent behaviour. If some peer's behaviour is not the implementation of some rational strategy, what is it then? If the peer behaves as some unintentional "mechanical" component of the environment, as a simple system variable, what are the system parameters that spawn its actions, the ultimate reason for its behaviour?

Since the only dynamic parameters that the environment provides are the actions effected to it by the choices of all agents, this allegedly fixed behaviour of the peer can only be a function of the past joint action sequences of the game; if it uses any features of the current situation at all to determine its behaviour, this must consist of what describes this "current situation" – the joint action choices of all agents so far.[6]
This implies a state-oriented view of the interaction environment because it is implicitly assumed that the sequence of past joint actions defines a state in which opponents are in when they select their actions for the next round, i.e. that joint actions are *causally linked* with each other so that the next joint action is always the effect of its predecessor(s). This resembles very much the idea of modelling opponents as *deterministic/probabilistic finite automata* (DFA), a method that was used, amongst others, by Freund et al. (1995) and Carmel and Markovitch (1996) (these works were already reviewed in Section 2.2). In such automata, the opponent enters some initial state at the beginning of the game and transitions between states are triggered by the most recent action combination of the remaining agents (which is the input to the transition function) and accompanied by the agent's own selected action (and by the change to some new state, of course). By having arbitrarily large sets of states, any opponent behaviour can be modelled that operates on histories as regular expressions over finite "alphabets" of joint actions. If stochastic functions determine the actually occurring state transitions (in the case of probabilistic finite automata) additional flexibility is added that allows for modelling mixed strategies thereof.

Following this intuition, we reduce the complexity of modelled opponent behaviour even further by assuming that each opponent j has only $|A|$ "mental" states, each of which determines the action j is going to perform. More specifically, j will play a_j whenever it enters state $\langle a_j \rangle$ (since this is only an auxiliary construction we refrain from introducing new notations or formalisms for it), and a transition between these states

$$\langle a_j \rangle \overset{a_{-j}}{\longrightarrow} \langle a'_j \rangle$$

is effected by the previous joint action a_{-j} of all agents other than j.
For the modelling agent i in question, each opponent j is a system variable that behaves on the basis of such simple action transition rules. So from i's point of view, the joint opponent strategy set A_{-i} defines the set of system (i.e. opponent) states and the transitions between these states are spawned by i's own action choices.
This simple OBP model is illustrated for a three-player example by the state transition diagram in Figure 5.5. In this example, agent 1 models the behaviour of the remaining system, i.e. of its opponents (agent 2 and agent 3). We assume that $A_1 = A_2 = A_3 = \{0, 1, 2, 3\}$, so that the set of system states is given by the cross-product of $A_2 \times A_3$ of the individual strategy sets of agent 2 and agent 3. Furthermore, the diagram is assumed to capture the opponent action transitions for some fixed behaviour a_1 of player 1 (e.g. $a_1 = 2$) so that actually another 3 such diagrams would be needed to fully describe opponent behaviour. The edges capture the transitions from previous to next joint actions. The edge from node (2,1) to node (3,2) for example has the semantics of the transition

$$\langle 2, 1 \rangle \overset{2}{\longrightarrow} \langle 3, 2 \rangle \quad ,$$

[6] This is not to say that it *must* depend on the game history. The only conclusion we draw is that all a sub-intentional agent *can* use as an input for its behaviour-determining function is the data obtained from the environment.

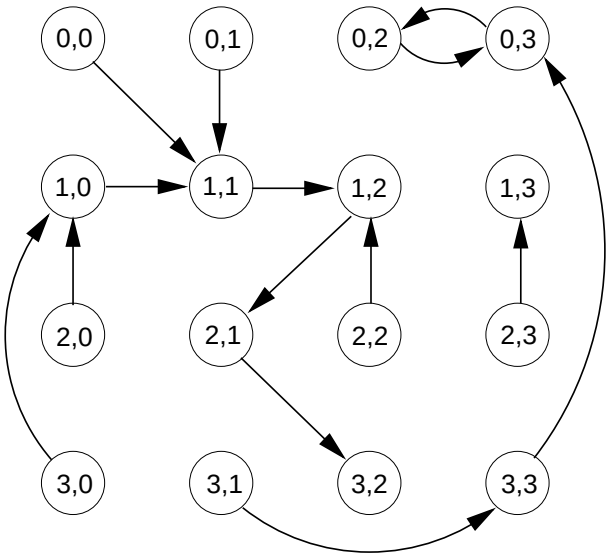


Figure 5.5: Simple state-oriented view of a game with three players and four individual action choices for a *single* action of the modelling agent 1. The vertices denote states, labeled by the joint action (a_2, a_3) performed by the opponents of 1. Each edge represents a deterministic transition from one action tuple to the next, so that a "source" opponent action is always followed by a "target" opponent action.

which means that if (2,2,1) was played in the last round, then (X,3,2) will be played next, were action X of player 1 remains unpredicted, of course, because it is agent 1 that is modelling the remaining peers' behaviour: its own decisions are seen as the *input* to the system behaviour function and are thus subject to further reasoning, so they are not determined by the model.
Note that this view of the system is based on the idea that (a) future actions are an immediate consequence of the most recent action combination and (b) this action-to-action transition function is purely *deterministic* – if the first joint action is known, any future (sequence of) opponent action(s) can be exactly determined.
To alleviate the second restriction, we extend the world model by *state transition probabilities*, i.e. the state transitions are now weighted by probabilities $\Delta(a_{-i}, a_i, a'_{-i})$ that reflect how likely it is that the next joint opponent action will be a'_{-i} if the previous joint action was (a_{-i}, a_i).
More formally, the probabilistic state transition function $\Delta : A^{n-1} \times A \times A^{n-1} \to [0; 1]$ is defined as

$$\Delta(a_{-i}, a_i, a'_{-i}) = P\left(\forall t.a^{(t)} = (a_{-i}, a_i) \Rightarrow a_{-i}^{(t+1)} = a'_{-i}\right)$$

for any round t of the game.
This leads to state transition models in which edges are labeled with Δ values, so that the probabilities of all outgoing edges of every node sum to one. Figure 5.6 shows a modified version of the previous example with such edge labels.

Under the assumptions that the next opponent action only depends on the previous action, that the rules that govern this action-to-action behaviour remain constant and that the transition rules are deterministic, learning $|A|$ such graphs is obviously equivalent to the problem L^{OBP} as defined in Section 4.2.2: after any round t, the joint actions performed in rounds 1 to $t-1$ can be neglected (because future actions will depend on the joint action tuples starting with $a^{(t)}$ and for any sequence of future actions of i the resulting actions $a_{-i}^{(t+k)}$ can be determined by following the respective edges in the

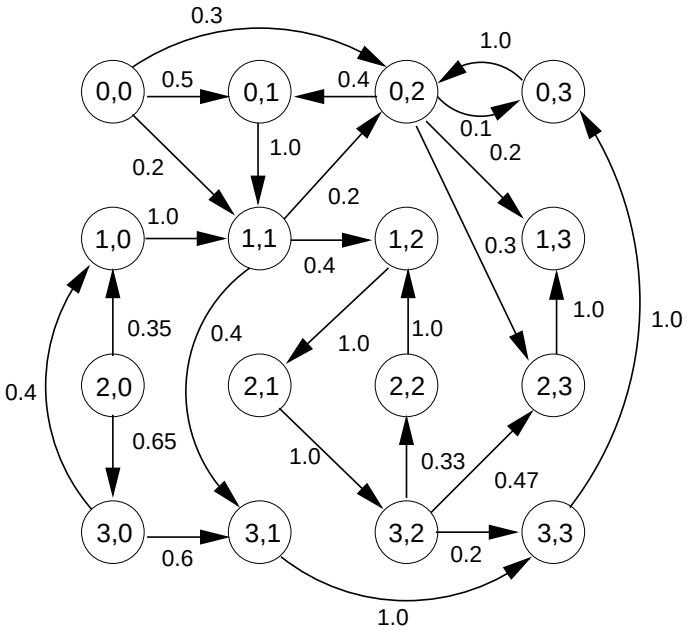


Figure 5.6: Non-deterministic view of the game in Figure 5.5: transition edges are now labeled by probabilities. Such a graph can be used by player 1 to represent the remaining players' behaviour for a constant action of itself in terms of mixed strategies.

transition diagrams.
For non-deterministic state transitions an *oracle* would be needed as stated in Section 4.2.2 to determine the actual opponent choices if they are using mixed strategies, but a probability-weighted opponent behaviour tree could be constructed so that the likelihood of each possible future opponent action sequence can be computed.

Prediction, control and Markov Decision Processes

The above model of opponent behaviour focuses around the notion of describing the opponents' action selection rules in terms of one's own actions by determining to which degree and in which way the agent influences peer decisions, or, alternatively, by describing how fixed behaviour patterns of other agents are *controlled* by the modelling agent's choices. However, such an analysis, even if it succeeds in predicting opponent behaviour accurately, is useless unless it is exploited to maximise the agent's own payoffs over time. What is actually needed is a solution to the problem of *learning sequential control strategies*, which can in most general terms be formulated as a *Markov Decision Problem*[7] (MDP).

An MDP is defined by a set of states $\mathcal{S}$ an agent can perceive, and a set of actions A it can perform. At each (discrete) time step t, the agent senses the current state s_t and has to perform some action a_t. According to this action choice the environment produces some successor state $s_{t+1} = \delta(s_t, a_t)$ and the agent receives some reward $r(s_t, a_t)$. The functions r and δ depend only on the previous state s_t and on the agent's action and are not necessarily known to the agent. We assume for the moment that r and δ are deterministic functions.
The task of the agent is to find a *policy*, a mapping $\varphi : \mathcal{S} \rightarrow A$ that outputs for each current state the optimal action to be taken. The ways to specify what exactly should be

[7] We follow the lines of the introduction of MDPs provided in (Mitchell, 1997, Section 13.2).

learned as an *optimal* policy φ^* are manifold. An obvious choice is to choose

$$\varphi^* = \arg\max_{\varphi} V_\varphi(s)$$

for all $s \in S$, the policy that maximises some function V_φ of the rewards obtained from state s on in the future states reached from s by employing φ^*.
Thereby, common choices for V are the *cumulative reward* , the *average reward* that will be received in each future state and the *finite horizon reward*, the sum of future rewards for a fixed number of transitions.
In the case of non-deterministic state transitions and/or rewards (e.g. in games with dice rolls), the quantity V_φ is simply redefined as the *expected value* $\tilde{V}_\varphi$ of one of the previous measures, or, more formally,

$$\forall s_t \in S.\tilde{V}_\varphi(s_t) = E[V_\varphi(s_t)].$$

In the light of the L^{OBP} problem, it is quite evident that the problem of optimal action selection is nothing but the optimal policy learning problem for MDPs: if we let the set of states be the states we derived from opponent action combinations ($\mathcal{S} := \{\langle a_{-i}\rangle\}_{a_{-i}\in A_{-i}}$), define $r(s_t, a_t) := u_i(a_{-i}^{(t)}, a_i^{(t)})$ as a (deterministic) reward function and take $P(\delta(s_t, a_t) = s_{t+1}) = \Delta(a_{-i}^{(t)}, a_i^{(t)}, a_{-i}^{(t)})$ to be the probability of every state transition, the non-deterministic version of V_φ can be used as a future cumulative utility to be maximised by the optimal policy – viz, strategy.

Since the convergence of so-called *Q-learning* algorithms (a sub-class of *Reinforcement Learning* methods) has been shown for non-deterministic MDPs (the original theorem is due to Watkins and Dayan (1992)), we have an algorithm at hand which could be readily used to learn the required action-value function m_i (while implicitly solving L^{OBP}).
Unfortunately, things are not that simple, due to, once more, combinatorial explosion in the number of joint opponent actions which would produce a state space that is practically impossible to explore in reasonable time.
This observation explains why *Reinforcement Learning* methods cannot be employed: firstly, even if we had the time to explore the whole state set, this would initially lead to the exploration of very many states that will not be relevant for future actions. Secondly, as strategies become more and more elaborate, the agent will only be interested in transitions among states in a very small partition of the state space (especially those states close to equilibria and dominant strategies), for which fact a lot of the "learning work" done before will be in vain. Finally, a complete description of the state space will result in a failure to generalise amongst them, because the response rules each opponent uses might be in fact much simpler and easier to represent than they appear.
At the same time, the above discussion gives a feel for the concepts that have to be approximated by OBP learners and directly motivates the construction of the learning algorithms we will employ.

5.3.2 Genetic nearest-neighbour learning of best-response strategies

The learning algorithm we design for the Strategy Engine provides only very approximative versions of the above concepts: it rests on the assumption that opponents pursue some fixed behaviour and that this behaviour can be expressed by an action selection

function that depends on nothing but the past joint action. Additionally, it only allows for predicting a *single* next opponent action, not probability-weighted distributions over several opponent action combinations. The optimal policy is only computed with respect to the *immediate* payoff the agent hopes to receive in the next round – it does *not* take (discounted) future rewards into account.
On the other hand, it offers a very pragmatic way of handling non-determinism and is able to generalise from past observations by discovering similarities between situations, so that it is not necessary to explore the entire set of possible opponent actions.

The idea behind the algorithm is rather simple. Transition rules of the form

$$\langle a_{-i} \rangle \xrightarrow{a_i} \langle a'_{-i} \rangle$$

are encoded as genes in $|A|$ evolving populations and reproduce with the goal of producing specimen that match a high number of transitions perceived so far. After each round, those individuals that match most closely the current joint action (the *nearest neighbours*) are chosen to predict the next opponent action. The fitter such a neighbour is, the more seriously is its prediction taken.
Such a prediction can obviously be made for each action a_i separately, so that the agent can feed it into the Utility Engine to obtain the π_i-estimate of the respective payoff value. Finally, $m_i(a_i)$ is taken to be precisely this predicted payoff value.
Because the learner predicts just one future opponent action and suggests to choose that action that maximises the payoff under the given prediction, this can be seen as a form of *best-response* learning. Looking at Definition B.2 of Appendix B (Section B.1.1) it can be seen that what the learner actually does is to pretend knowing what other agents are going to do next and to choose its own action accordingly.

Genetic Algorithms that learn transition rules

Genetic Algorithms (GAs) are a learning method that is loosely based on simulating biological evolution. Hypotheses are encoded as individuals in a *population* and the repeated reproduction of this population searches through the hypothesis space by re-combining the properties of the best solutions so far. Additionally, new features are generated randomly every now and then, so that getting trapped into locally optimal hypotheses is avoided.
As with neural networks, we provide only a very sketchy description of how GAs work (cf., e.g., (Mitchell, 1997, Chapter 9) for a detailed treatment) to explain how we construct Best-Response GAs.

GAs consist of a set of learning hypotheses $\mathcal{H}$, the so-called population and a real-valued *fitness function* that yields for each hypothesis in $\mathcal{H}$ a measure for its "goodness" (which depends on the application for which the GA is designed). A *crossover operator* is provided that defines how any two hypotheses in $\mathcal{H}$ can be combined to produce *offspring* (i.e. two new hypotheses). Furthermore, a *replacement ratio* $rr \in [0;1]$ has to be defined that specifies what fraction of the population is replaced by the offspring of previous individuals in each *generation* and some small *mutation rate* $mr \in [0;1]$ is used to determine the probability with which some attribute of the individual is randomly altered during crossover.

To describe the design of the GAs we will use, we need to determine a representation for the learning hypotheses, to design an appropriate fitness function and to specify the other parameters just mentioned.
Recalling the extensive discussion of transition rules as the basic building blocks of OBP learning in the previous section, we can see that there is a very natural way of turning these into binary strings by using the β-mapping introduced in Section 3.2.1. This consists of representing the rule

$$\langle a_{-i} \rangle \overset{a_i}{\longrightarrow} \langle b_{-i} \rangle$$

as a bit-string

$$\Big(\beta(a_1), \beta(a_2), \ldots, \beta(a_{i-1}), \beta(a_{i+1}), \ldots, \beta(a_n)\ ,\ \beta(b_1), \beta(b_2), \ldots, \beta(b_{i-1}), \beta(b_{i+1}), \ldots, \beta(b_n)\Big)$$

of length $2k(n-1)$. Such a bit-string has the semantics of the rule

$$\forall t. IF\ a_1^{(t)} \wedge \ldots, \wedge a_{i-1}^{(t)} \wedge a_{i+1}^{(t)} \wedge \ldots, \wedge a_n^{(t)}\ \ THEN\ b_1^{(t+1)} \wedge \ldots, \wedge b_{i-1}^{(t+1)} \wedge b_{i+1}^{(t+1)} \wedge \ldots, \wedge b_n^{(t+1)}$$

i.e. for any round t in the game, if the previous joint opponent action was a_{-i} and i itself played a_i, then the opponents will play b_{-i} in the next round.
The reader may have noticed that a_i appears neither in the pre- nor in the postcondition of the rule (bit-string). The reason for this is that we actually plan to use $|A|$ populations of such hypotheses $G(0), \ldots, G(2^{k-1})$ (remember that we defined the set A of individual actions as a set of integers $\{0, \ldots, 2^{k-1}\}$) so that each population "specialises" on the opponent behaviour under one specific action of i[8]. The advantage of this design is that it enables us to construct *symmetric* bit-strings consisting of a $k(n-1)$-long precondition and a postcondition of the same length and thus to define very simple crossover operators[9].

This representation is expressive enough to capture rules of the form $a_{-i} \rightarrow a'_{-i}$ for some action a_i of i, but it does not allow for a formalization of *generalized* rules such as

"player 3 will play action 4 regardless of the previous joint action".

To account for such rules, we adopt the common method of allowing a *wildcard* value "#" at each bit position with "don't care"-semantics, i.e. a rule that contains this symbol at the l-th position suggests that the rule holds regardless of whether the l-th bit is '0' or '1'.

Before defining the fitness function and the other parameters of our algorithm, we should present an example to illustrate our constructions so far.
Suppose agent 2 participates in a game of four players and four resources. In round $t-1$ of the game, the joint action was $a^{(t-1)} = (5, 3, 4, 11)$ and in the subsequent round t the agents played $a^{(t)} = (7, 2, 5, 1)$. Since agent 2 played 3 in the previous round, it uses the two consecutive adversary actions to create a new sample for $G(3)$ (the third of sixteen populations it maintains). This sample has the form

$$\underbrace{0101}_{5}\ \underbrace{0100}_{4}\ \underbrace{1011}_{11} \quad \longrightarrow \quad \underbrace{0111}_{7}\ \underbrace{0101}_{5}\ \underbrace{0001}_{1}\ .$$

[8]For reasons of readability we drop the index i from these populations which should actually read $G_i(a_i)$.

[9]Although a_i is conceptually a part of the precondition for the next opponent action, it makes more sense to exclude it from the representation of transition rules, because i, as we have explained, aims at modelling the behaviour of the *remaining* system.

Now assume that some hypothesis $h \in G_1(3)$ is of the form

$$\#10\# \quad 0100 \quad \#0\#1 \qquad \longrightarrow \qquad \#111 \quad \#\#\#\# \quad \#\#01$$

then this h matches the new sample perfectly, being a short form for a set of $2^{11} = 2048$ fixed rules (since h has 11 wildcard positions).

This example already hints at the question of how to devise an appropriate fitness function for the populations $G(a_i)$, because it illustrates that the only knowledge the agent can use to assess the quality of its own hypotheses is given by the samples the Simulation Engine distributed during the sequence of interactions enacted so far.
Since we have assumed that the set of rules that governs each opponent's behaviour is *fixed*, the history of the game H_i^t as perceived by agent i after t rounds can be seen as the trace of the application of those rules. Hence, a *sample set* $D(a_i)$ can be constructed for each $G(a_i)$ which maintains the transitions performed whenever i's action was a_i:

$$D(a_i) = \left\{ (\beta(a_{-i}), \beta(a'_{-i})) \mid \exists t_0 < t. \quad a_{-i} = a_{-i}^{(t_0)} \in H_i^t \wedge a'_{-i} = a_{-i}^{(t_0+1)} \in H_i^t \wedge a_i^{(t_0)} = a_i \right\}$$

These sample sets are then used to guide the search for appropriate hypotheses. Let $h \in G(a_i)$ be a member of the a_i-th population, i.e. a $2k(n-1)$-long vector of "bits" $h_l \in \{0, 1, \#\}$ as in the above example. We define the *match value* $mv(h, d)$ for any sample $d \in D(a_i)$ as follows:

$$mv(h, d) = \frac{1}{2k(n-1)} \left(\sum_{l=1}^{k(n-1)} I_{pre}(h_l, d_l) + \sum_{l=k(n-1)+1}^{2k(n-1)} I_{post}(h_l, d_l) \right)$$

where

$$I_{pre}(h_l, d_l) = \begin{cases} 1 & \text{if } h_l = d_l \vee h_l = \# \\ 0 & \text{else} \end{cases} \qquad I_{post}(h_l, d_l) = \begin{cases} 1 & \text{if } h_l = d_l \\ 0 & \text{else} \end{cases}$$

are two indicator functions that measure whether the hypothesis matches the sample in the l-th bit. For the postcondition bits wildcards do *not* count as matches, so that wildcards in postconditions (which only predict some *set* of possible next opponent behaviours) are punished. Since wildcards always represent disjunctions of several hypotheses it is reasonable to favour bit-strings that discover similarities in the conditions that must hold for an opponent to behave in some way (which are possibly much simpler than completely specified descriptions of previous joint actions). Hypotheses with "don't care"-features in their *post*conditions are quite useless, because they only express *uncertainty* about future opponent actions.
The match function can now be used to define the fitness function which for any hypothesis $h \in G(a_i)$ reflects how many of the samples in $D(a_i)$ were *classified* correctly and to which degree:

$$fitness(h) = \left(\frac{1}{|D(a_i)|} \sum_{d \in D(a_i)} mv(h, d) \right)^2$$

This means that the fitness will be measured as the squared average match value of h (squaring the mean value puts additional pressure on the individuals). This constitutes a compromise between fostering hypotheses that account for a wide range of seen examples and supporting the development of "specialised" individuals that reflect precise opponent

behaviour for special cases.

As a crossover operator we choose standard *one-point random crossover*, an operator that randomly selects some position l in the binary representation and copies all positions from both parents cross-wise. This is shown in the following example, in which two ten-bit parents are subject to crossover at position 4 (counting from 1):

$$\begin{array}{l} 1011|101100 \\ 0100|111011 \end{array} \quad \longrightarrow \quad \begin{array}{l} 1011|111011 \\ 0100|101100 \end{array}$$

As concerns mutation, the standard method of selecting some random bit and flipping it from its current value $h_l \in \{0, 1, \#\}$ to one of the other two values with equal probability is employed.
Instead of using a fitness threshold as termination criterion (which is the training strategy usually employed), just one iteration of the training algorithm will be performed after each round (for the same reasons of balancing learning and decision efforts laid out for neural network learning in the previous section).

All this provides us with a method for updating the populations given new incoming joint action transitions. Two more details should be pointed out concerning this update procedure. Firstly, note that although $|A|$ populations will be maintained, only *one* of these has to be updated after each game round, because the just performed joint action only provides a transition sample for the a_i that was played by i in the previous round. Secondly, it should be mentioned that this update is done with a delay of one round, because the consequence of the joint action in question has to be awaited to construct a sample $(\beta(a_{-i}^{(t)}), \beta(a_{-i}^{(t+1)}))$.

Next we discuss the use of the generated hypotheses for behaviour prediction.

Nearest-neighbour prediction

The MDP view of the decision situation requires that the probabilities for state transitions be predicted and that the rewards which can be obtained by taking a particular action in the resulting states are compared, so that an optimal policy can be chosen accordingly.
Given a set of populations of state transition hypotheses obtained after some number of rounds this raises the question of how the transition probabilities can be computed. The method employed here is based on a very simple version of *instance-based learning* called *nearest-neighbour learning*.
Instead of constructing general hypotheses for the target function, instance-based learning methods simply store the training examples. Each time a new query instance is encountered, its relationship to previously stored examples is examined, and a target value is assigned to the new instance according to the values of previous examples. In the case of $\mathcal{K}$-nearest neighbour learning, samples are assumed to be represented as points in n-dimensional Euclidean space $\mathbf{R}^n$. To compute the target value of a new instance, its $\mathcal{K}$ (geometrically) nearest neighbours are consulted and their outputs are combined to obtain the value of the target function for the new instance. The "nearer" neighbours are, the more will their target function values be taken into account (we refer the reader again to Mitchell (1997, Chapter 8) for an extensive treatment of these methods).
The foremost motivation for using this technique for OBP is that, no matter how large the populations of hypotheses are, they cannot be guaranteed to include samples that match

any possible current state. Hence, we will be forced to use rules whose preconditions *resemble* the current situation. Furthermore, using several individuals as "predictors" in a given situation can be used to overcome the problems that arise when agents use non-deterministic transition rules. By assessing the traits of postconditions of several predictions a model of *expected properties* of the next state can be obtained, if a precise prediction of the next state is impossible.

The nearest-neighbour method is applied to the GA populations in the following way: given a particular population $G(a_i)$ for i's action a_i, we interpret the preconditions of hypotheses as points in $k(n-1)$-dimensional hyperspace (or sets of points, if wildcards are involved). The *distance* from any hypothesis rule h to the current joint opponent action a_{-i} can then be computed as

$$distance(h, a_{-i}) = \sqrt{\sum_{l=1}^{k(n-1)} \chi(h_l, \beta(a_{-i})[l])^2}$$

where

$$\chi(h_l, \beta_l) = \begin{cases} 0 & \text{if } h_l = \beta_l \vee h_l = \# \\ 1 & \text{else} \end{cases}$$

is the distance between two bit values, which is zero if they are both equal or if the value of h in bit l is '#'. Thus, the *distance* function computes nothing but the geometric distance between the current opponent action and the hypothesis precondition in a $k(n-1)$-dimensional binary hypercube (in which preconditions with w wildcards represent 2^w edges, so that one of these will always be equal to $\beta(a_{-i})$ in those components). This distance function makes the computation of sets of $\mathcal{K}$ nearest neighbours $N(\mathcal{K}, a_i, a_{-i})$ possible for arbitrary $\mathcal{K} \leq |G(a_i)|$. We refrain from presenting a formal construction of these neighbour sets here; it should be clear that by computing the distance to a_{-i} for any $h \in G(a_i)$ the $\mathcal{K}$ nearest neighbours of this opponent action can be obtained by sorting them appropriately.
These sets represent the gene sub-populations that most closely match the current joint action, for which reason they should be used to compute the most probable next opponent action. The prediction is computed bit-wise by counting the postcondition bits of nearest neighbour hypotheses and weighing the "votes" for 1 and 0 by the fitness of the respective neighbour. We define

$$predict(a_i, a_{-i}, l) = \begin{cases} 1 & \text{if } Q(N(\mathcal{K}, a_i, a_{-i}), l) > 0 \\ 0 & \text{else} \end{cases}$$

where Q is the fitness-weighted sum of the neighbours in N that predicted a '1' in the l-th postcondition bit diminished by the weighted sum of those neighbours that predicted a '0' at the same position:

$$Q(N(\mathcal{K}, a_i, a_{-i}), l) = \textstyle\sum_{h \in N(\mathcal{K}, a_i, a_{-i})} I(h_l) \cdot fitness(h), \qquad I(h_l) = \begin{cases} 1 & \text{if } h_l = 1 \\ -1 & \text{if } h_l = 0 \\ 0 & \text{else} \end{cases}$$

So the decision of whether bit l in the next joint opponent action will be set or not is made by taking a weighted majority vote among all $\mathcal{K}$ neighbours.
This definition can easily be extended to compute, bit by bit, the full binary version of the next joint opponent action a'_{-i} by taking

$$predict(N(\mathcal{K}, a_i, a_{-i})) = a'_{-i} \iff \beta(a'_{-i})[l] = predict(N(\mathcal{K}, a_i, a_{-i}), l + k(n-1))$$

for all $1 \leq l \leq k(n-1)$, so that every bit at position l of the bit-string representation of a'_{-i} has the neighbour-predicted value of bit $l + k(n-1)$ (the index offset $k(n-1)$ is necessary to access the neighbour postcondition bits). In the following we will write $predict(a_{-i}) = a'_{-i}$ whenever it is clear which and how many neighbours are used.

Best-response action values

Now we are left with the task of turning action prediction into an action-value function that reflects to what degree $a_i \in A$ is considered to be an implementation of the optimal policy in the next round. As opposed to the cumulative reward, average reward and finite horizon reward value functions mentioned in Section 5.3.1, our agents only take into account the *immediate reward* received from entering a new state, i.e.

$$V_\varphi(s_t) = r(s_t, a_t)$$

for any policy φ, which translates, in terms of our constructions, into

$$V_\varphi(a_{-i}^{(t)}) = u_i(a_{-i}^{(t+1)}, a_i^{(t+1)}) \quad .$$

Since an optimal policy φ^* maps each "state" $\langle a_{-i}^{(t)} \rangle$ to an action that maximises the quantity on the right-hand side of the above equation we obtain

$$\varphi^*(a_{-i}^{(t)}) = \arg \max_{a_i^{(t+1)} \in A} u_i(a_{-i}^{(t+1)}, a_i^{(t+1)})$$

which is nothing but the best-response strategy with respect to the next opponent action $a_{-i}^{(t+1)}$. By virtue of our nearest neighbour algorithm and the assumptions concerning opponent transition rules, we can claim that this next opponent action can be deterministically anticipated and thus define the *individual action-value* function m_i for any round t of rounds played so far as

$$m_i(a_i) = \| \pi_i(predict(a_{-i}^{(t)}), a_i) \| \tag{5.1}$$

(π_i is once more i's payoff function estimate). Thereby, $\| \cdot \|$ is a *normalizing* function such that for any function $f : X \to \mathbf{R}$

$$\| f(x) \| = \frac{f(x)}{\sum_{x' \in X} f(x')}$$

which can be used to turn any action-value function (in fact *any* real-valued function with countable domain) into a mixed strategy.
Thus, m_i can be seen as a mixed strategy with selection probabilities for each action that correspond to the individual action values of the pure strategies[10].

Note that the computation of the predicted opponent action requires consulting the neighbours of $G(a_i^{(t)})$, because the next opponent action is seen as the effect of i's last action choice. Thus, apart from only *updating* only one GA in each round, only one of the 2^k GA populations has to be used to *predict* the next opponent action. Apart from matters concerning fine-tuning decisions about the replacement ratio, the mutation rate, population sizes and "neighbourhood" sizes, the design of the strategy learning and decision algorithm is now completed. These will be dealt with in the following section, together with a preliminary evaluation of the algorithm.

[10]Using such a mixed strategy profile is an alternative to choosing always the optimal policy $\arg\max_{a_i} m_i(a_i)$. In Chapter 6 we will investigate whether the pure action selection rule is more effective than the mixed strategies output by m_i.

5.3.3 Preliminary results

As opposed to the task of payoff approximation, strategic learning does not provide for an explicit representation of the concept to be learned: even an external observer cannot *a priori* determine how agents' strategies will evolve in the course of a simulated interaction This is due to the fact that an agent's strategy is the result of reasoning about the behaviour of its opponents and vice versa.
Therefore no sample action transitions can be generated in a similar fashion as the action/payoff pairs that were used to validate the neural networks of the Utility Engine, so that only an *a posteriori* assessment of the performance of the strategic learners can be achieved. This will consist of comparing their opponent action predictions to the opponent actions actually performed in later rounds.
Since the Strategy Engine only performs a one-step lookahead, the *prediction error* E which will be used as the performance measure throughout the analysis to follow can only be calculated in terms of the actually occurring opponent action. The *distance* function used for the definition of the fitness function complies with the conditions put forward for a "similarity-measuring" function in Definition 4.2 of Section 4.2.2 and provides a simple method for computing the prediction error L_i^{OBP} made in round t:

$$E(t) = distance\left(predict(a_{-i}^{(t-1)}), a_{-i}^{(t)}\right)$$

To illustrate how this error-per-round evolves over time, the average error μ_i in all rounds so far will be shown in the plots of this section. It computes as the mean of all past prediction errors:

$$\mu_i = \frac{1}{t}\sum_{j=1}^{t} E(j)$$

In order to evaluate the performance of the Strategy Engine as such it seems reasonable to test it without the Utility Engine, so that it remains unaffected by the errors in payoff approximation. In computing the action-value function, we will therefore use the real payoff function u_i instead of π_i in Equation (5.1) so that the "blame" for sub-optimal action choices can be purely ascribed to the predictive capabilities of the Strategy Engine.
Another issue that should be remarked is that the results we present here are very much influenced by the properties of the resource-load balancing game. More specifically, the existence of a single strict equilibrium (the joint action in which every agent accesses every resource simultaneously) implies that for every agent i action all_i "=" $111\ldots1$ is the single best response to *any* opponent strategy. Hence this action will yield the highest action value no matter which joint opponent action is predicted, and irrespective of how accurate this prediction is, choosing the greedy action will always be optimal. This explains the use of prediction accuracy measuring methods instead of utility-measuring methods, because they don't provide a means of assessing the learners' performance.

This property bears another implication for agents that consist of the Utility and Strategy Engines only, namely that the maximal performance that can be expected of them is the evolution of the Nash equilibrium. As soon as their payoff-approximating function is accurate enough to infer that acting greedily constitutes the best response to any opponent behaviour, they will converge to the equilibrium. If this actually happens, they will have proven able of adapting successfully to an environment of selfish co-actors in the sense that they can at least avoid being exploited by others and learn to act individually rational, even though they are unable to do any better as a whole.

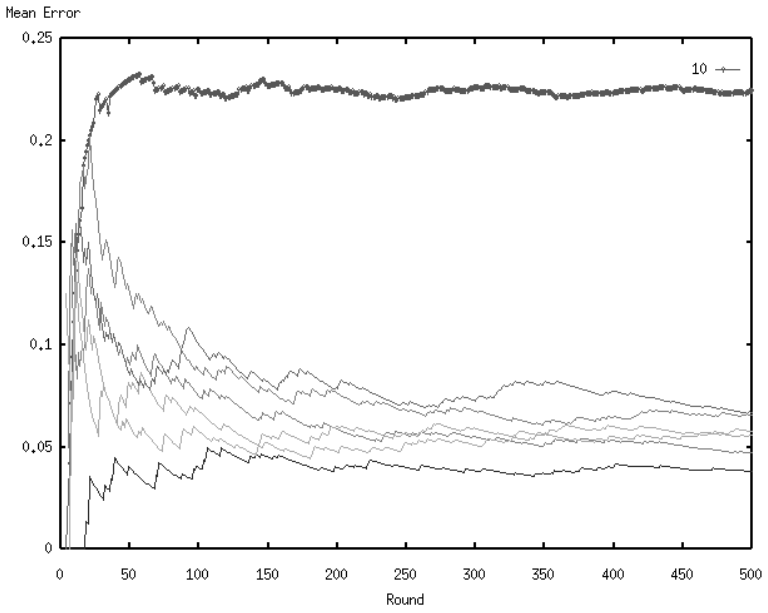


Figure 5.7: Various population sizes ($|\mathcal{H}| \in \{10, 50, 100, 250, 500, 750, 1000\}$) for strategic learners in the game G_I. Apart from the divergent curve ($|\mathcal{H}| = 10$) performance is roughly comparable. The curve with lowest mean error represents $|\mathcal{H}| = 100$.

The first series of experiments we review is concerned with determining appropriate population sizes and the number $\mathcal{K}$ of nearest neighbours that will be used for behaviour prediction. Both these problems are actually harder than one might think. For obvious reasons, an increasing number of hypotheses in each population leads to a greater diversity and using more neighbours certainly adds to the complexity of the decisions that are made during action prediction, because each of them might contribute a different part of acquired knowledge to the final decisions. Thus, optimal population sizes and numbers of nearest neighbours will probably depend on the *complexity* of opponent transition rules. Unfortunately, we do not know anything about this complexity, so all we can do is to compare different population and neighbourhood sizes and pick the best ones.
In the resource-load balancing game, it turns out that changing population sizes above a certain minimal number of hypotheses does not have great effects on learning performance, even though the curves become smoother and the difference between several learners using the same population size becomes smaller. For reasons of keeping space requirements on a decent level (consider that even test sets as small as G_{II} require a total of $10 \cdot 2^5 = 320$ GA populations), we choose to equip each agent with no more than a total of 3000 hypotheses. In the case of G_I this means that each agent has four populations of size 500 at its disposal, while for G_{II} each population will consist of 100 individuals. Figures 5.7 and 5.8 provide an overview of curves for various population sizes where $\mathcal{K} = 10$ was applied throughout. Action choices were made by the agents according to an "almost purified" version of the action-value function m_i, where $\arg\max_{a_i} m_i(a_i)$ was played with probability $1 - 0.01 \cdot (|A| - 1)$ (any sub-optimal strategy was played with probability 0.01).

Next, appropriate choices for $\mathcal{K}$ were sought for. Apparently, very few neighbours suffice to achieve optimal behaviour prediction (Figure 5.9 shows some examples for the performance of learners under various sizes of $\mathcal{K}$ in G_{II}), but it should be remarked, once again, that this may only be due to the properties of the resource-load balancing game just mentioned – ultimately most surviving hypotheses will probably predict

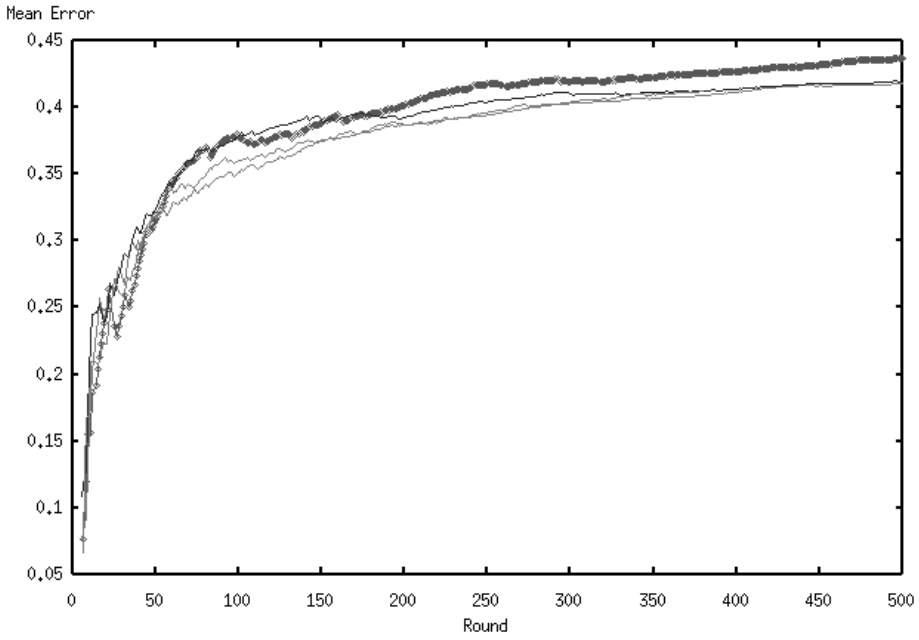


Figure 5.8: Plots for $|\mathcal{H}| \in \{10, 50, 100, 500\}$ in G_{II}. Differences between learners are even smaller than for G_I, except that for $|\mathcal{H}| = 10$ (the bold curve) performance is again slightly worse than that of the other agents.

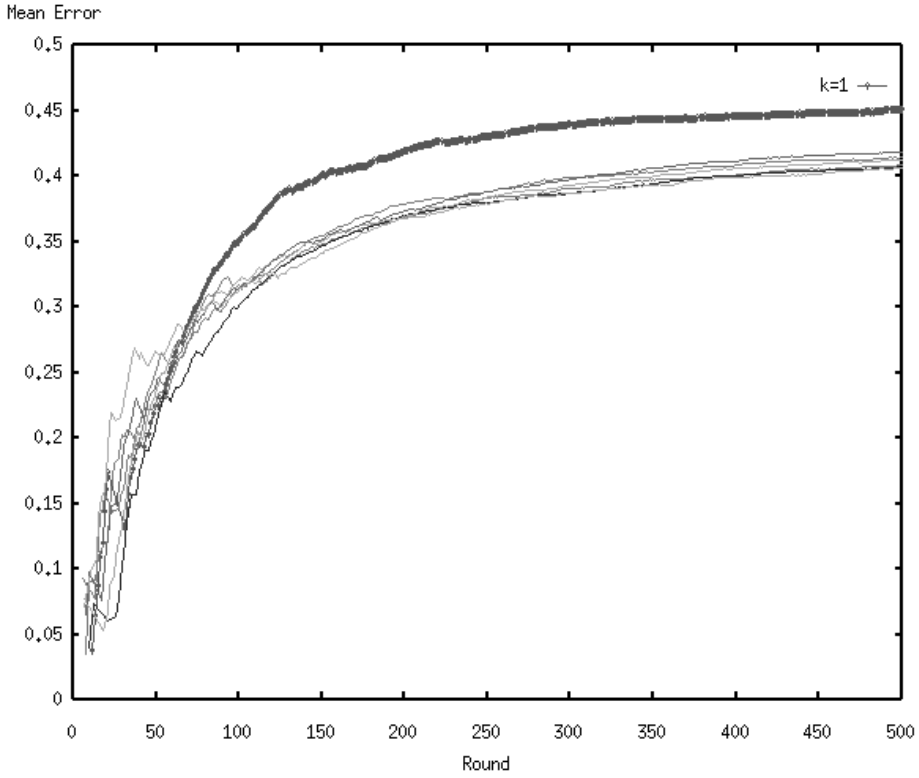


Figure 5.9: Experiments with $\mathcal{K} \in \{1, 5, 10, 25, 50, 75, 100\}$ for G_{II}-learners applying a population size of 100 for each population. Except from agents using one neighbour for action prediction all others do equally well.

"each opponent behaves greedily regardless of what was played the previous round"

which is a concept that can be adequately modelled by using a single hypothesis

$$\#\#\#\#\ldots\#\#\#\#\#\# \longrightarrow 111111\ldots1111$$

It is actually worthwhile to check whether such hypotheses actually occur in the populations. In fact, the fittest individual (with $fitness(h) = 0.8298$) after 1000 rounds was

$$1011\#\#011100\#11\#0\#1111110\#1\#0\#11101001\#0011\#1 \longrightarrow$$
$$111111111\#11111111110101111110\#11\#11110101111$$

whose postcondition consists almost entirely of '1'-bits while its precondition allows for 10 degrees of freedom. Given that the actually played mixed strategies implemented the best-response strategy around 40% of the time, this hypothesis constitutes a reasonably accurate opponent behaviour approximation.
We decided to settle on using $\mathcal{K} = 0.1 \cdot |\mathcal{H}|$ for both games since all available evidence suggests that this is a sufficient number of nearest neighbours for the resource-load balancing game.

Finally, replacement ratio and mutation rate had to be determined. Tests were conducted on both G_I and G_{II} for mutation rates ranging from 0.03 to 0.3 and a replacement ratio between 0.1 and 0.9. Again, results were hardly elucidating, because even after testing the performance of 100 agents using the same combination of parameters, the variance between their performances was almost as big as between different mutation and replacement rate choices. However, the following observations seem to give a few hints for appropriate parameter choices: mostly, strategic learners which used replacement rates between 0.3 and 0.7 outperformed those using the remaining values and, for mutation rates above 0.2, some of them did considerably worse than others (Figures 5.10 and 5.11 show the performance of agents in G_I with varying replacement ratio for a mutation rate of 0.03 and 0.3 respectively).
We therefore adhere to the use of a replacement rate of 0.5 and a mutation rate of 0.03, which represents the final design choice in building the Strategy Engine.

Comparing the plots for G_I and G_{II}, a major discrepancy between learning performance in those two games becomes visible, namely that the mean prediction error does never *decrease* for G_{II}. To investigate this issue further, longer simulations were conducted, but even after 3000 rounds, the curves remain stable around 0.4-0.5 (which implies that the error per round is still around 0.01%). Fortunately, this is only due to the fact that agents employ mixed strategies in our simulations, by which the play any of the 31 sub-optimal strategies with probability 0.01 – when using *pure* best-reply strategies, the performance *does* converge, as shown in Figure 5.12 . So this apparent "deficiency" can be simply explained by observing that the approximation of non-deterministic transition rules is, of course, much harder than that of deterministic behaviours. It should also be mentioned that by using pure strategies agents converge to the equilibrium combination within less than 100 rounds and never depart from it ever after.

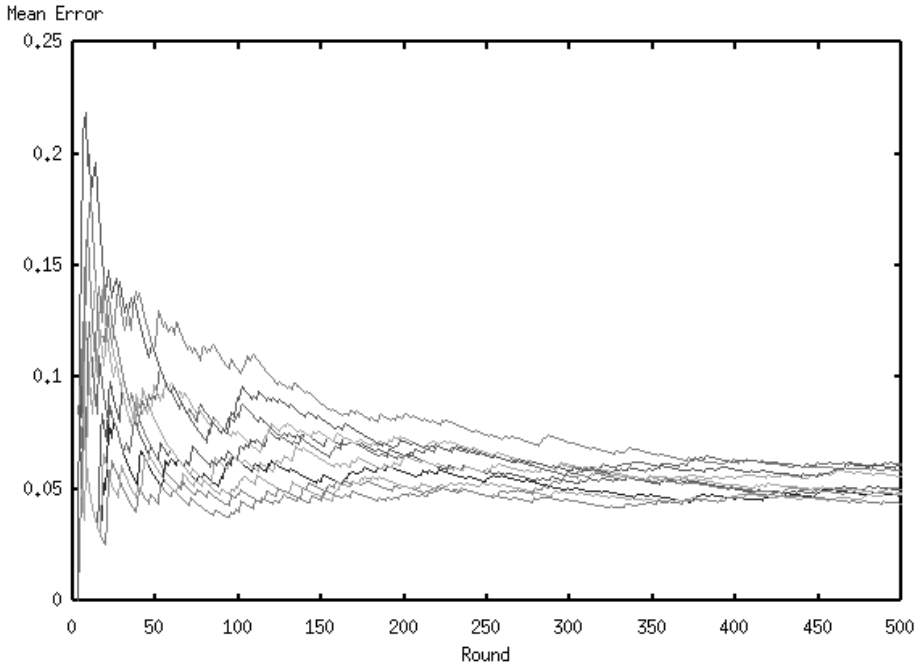


Figure 5.10: Plots for agents employing a mutation rate of 0.03 and a replacement ratio from $\{0.1, 0.2, \ldots, 0.9\}$ in G_I.

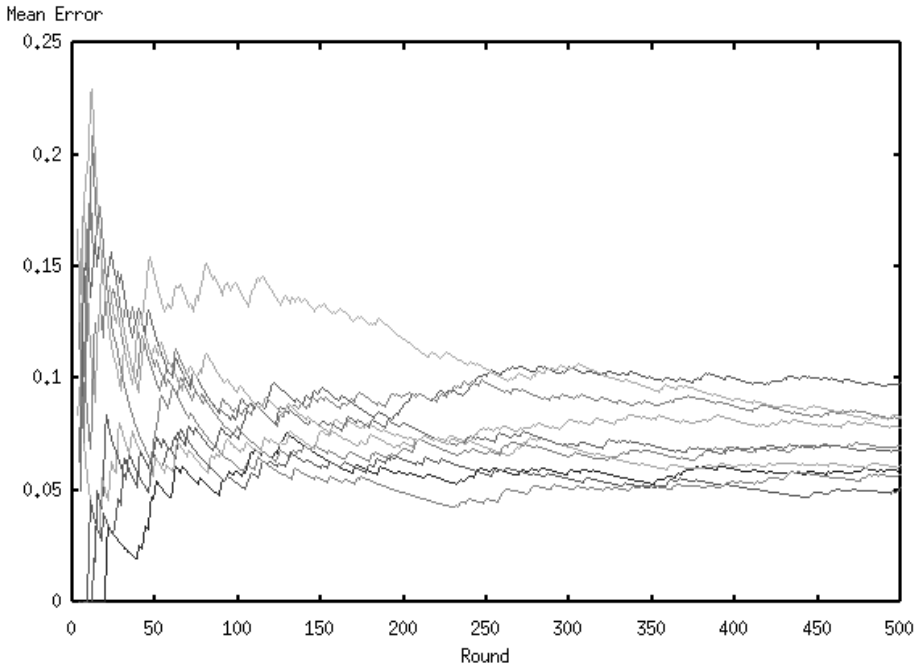


Figure 5.11: Same scenario as in Figure 5.10 with a mutation rate of 0.3. It can clearly be seen that variance between the mean errors of agents has increased without any agent doing better than before.

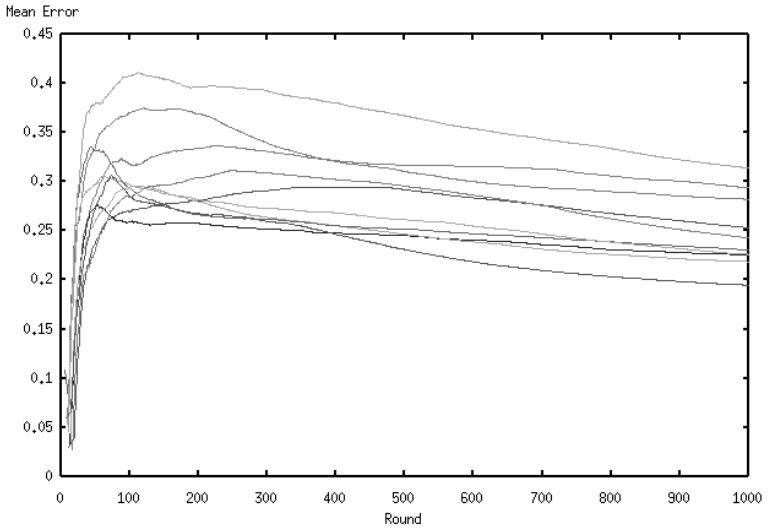


Figure 5.12: G_{II}-simulation with pure best-response strategy choice. As opposed to the mixed strategy variant, mean error starts decreasing after only about 100 rounds.

These experimental results suggest that, at least for the resource-load balancing game with its simple best-response structure, agents are able to learn the safe equilibrium strategy within reasonable time provided they have an accurate picture of the payoff function. Next, we discuss the design of the Social Behaviour Engine prototype to complete the presentation of the individual learning components.

5.4 Learning the "social potential" of games: the Social Behaviour Engine

The Social Behaviour Engine (SBE) is the component that is supposed to extend the architecture presented so far by *social* learning and reasoning capabilities that facilitate the evolution of coordinated interaction patterns beyond the greedy, short-sighted utility maximisation aimed at by the decision-making component of the Strategy Engine.
In order to discover and exploit the cooperation potentials an interaction situation offers, agents have to reason about the possibility of implicit agreements and since this depends on the global constraints that govern the payoff functions of all agents, agents will have to gather information about the *preferences* of their opponents. Based on this analysis of opponent preferences, the modelling agent will try to simulate those opponents' reasoning processes to find out how they can be brought into the state of behaving in a cooperative manner.

The following sections introduce a prototypical SBE design based on *recursive belief model* learning and a simple social decision-making principle for social learners that we employ.

5.4.1 Designing a social learning component

Preliminary considerations

Unlike the learning tasks of L^{IM} and L^{OBP}, the learning goal specified for cooperation potential learners in Definition 4.3 of Section 4.2.2 does not translate directly into a concept

that can be learned with standard machine learning methods. This is partly due to the fact that the hypotheses of the Utility Engine and the Strategy Engine don't provide a measure for whether globally optimal "agreements" have been reached or not. Ignorance of adversaries' payoff functions makes it impossible for agents to discover potentials for the establishment of collectively beneficial joint action choices.
Another problem is that agents are ignorant of *how long* the process of achieving cooperation is supposed to take. If, for example, the agents tried several action selection rules to make peers converge to cooperative action combinations, they would not know when to evaluate the employed hypothesis, because there is no criterion that guarantees to assess in a verifiable way whether "continuing with the current hypothesis cannot yield good results in the future if it has not done so until now".
This is, in fact, just one aspect of a much more severe restriction, namely that learning data as such is not readily available. Agents wouldn't know when and whether optimal agreements have been reached, but even if they deem their peers' current actions cooperative, they cannot verify whether they are going to *persist* in the future and whether other agents are behaving only by coincidence in a cooperative way or by rational deliberation. Thus, no past joint behaviours can be directly transformed into training samples to validate or falsify the usefulness of alleged "cooperation-evoking" action sequences the agent is trying to determine.
Finally, developing hypotheses of the form *"if I play a certain action sequence in the next t rounds my opponents will converge to optimally cooperative behaviour patterns after these"* will be very time-consuming and initially probably lead to very poor performance results for the learner, as the evaluation of hypotheses will take a very long time due to the problems just mentioned.

Reconsidering social reasoning

We propose a view of the social learning task that leads only very indirectly to the socially optimal action-selection rules of the L^{CP} definition. The basic idea of this approach is that modelling the *reasoning principles* of adversaries allows for a much more coherent view than learning cooperation-evoking action sequences directly. This is because knowing what other agents "think" is, quite naturally, more reliable than knowledge about their reactions to certain behaviours: while behaviours which depend on a multitude of parameters of current and past situations, may change or be perturbated by faulty implementation and perception, the reasoning processes of individuals can be expected to be stable. This does certainly not hold for humans whose reasoning capabilities as such evolve with knowledge and experience, or for task-oriented domain MAS in which "rationality" may mean different things at different times. For our game-theoretic environments, however, characterised by the pervasive element of utility maximisation as the essence of all agents' reasoning, it can be assumed that the grounds on which agents reason remain constant throughout interaction processes.

What actually matters about the adversary's reasoning process is determining how it can be used to "massage" that agent into a cooperative stance, i.e. an action selection mode from which the modelling agent can benefit. How this can be achieved will certainly depend on the nature of two-player payoff dependencies between the two agents: for a given (arbitrary) behaviour of third parties, certain action combinations of these two agents will be more beneficial for the modelling agent, others will be preferred by the modelled agent and some may be considered "reasonable compromise". It is precisely

this last class of binary action tuples that count when it comes to establishing mutually beneficial agreements, joint action selections by which neither of the agents is exploited while at least one of them profits from the compromise.
Guided by this intuition, the following sub-tasks of L^{CP} can be specified for the learner i attempting to achieve individually and collectively rational agreements with some other agent j: first, find out how j can contribute to your own success regardless of the behaviour other agents exhibit and vice versa. If neither of you can "help" the other, discard social considerations in action selection and choose the "exploitation" strategies proposed by the Strategy Engine. Else, be willing to "sacrifice" some potentially possible payoff to the degree that the other agent can be expected to act to your own advantage. Attempt to implement that compromise and if it seems to evolve look for further cooperation potentials. If not, go back to the "greedy" action-selection rules of the Strategy Engine.
If these tasks can be learned successfully for every peer j, L^{CP} will be solved indirectly, because the evolution of possible coalitions can be predicted on the grounds of basic opponent rationality and actions can be taken to initiate it. We now discuss the principles upon which we build the learning algorithms that will be used to solve the individual tasks.

Social learning and decision-making principles

In order for an agent i to model the reasoning mechanisms of adversaries, it maintains (recursive) models of peers' payoff functions for all opponents in its neighbourhood[11]. More precisely, not the opponents' payoff functions are approximated, but the binary dependencies between their payoffs and the payoffs i itself might receive for given action combinations are approximated as 2-player utility pairs (so-called *gains*) in n-player strategy spaces. Agent i updates these models with information about past moves and payoffs (using the information provided by the UE), and decides on whether there are *socially feasible* action alternatives which ensure a high payoff while making a cooperative stance of the neighbours probable.

Adopting the viewpoint of agent i, the decision situation when trying to assess the potential for cooperation and what it would do to achieve such cooperation is the following: agent i is basically aware of the fact that any other agent's actions affect its own payoffs. Therefore, it must try to make other agents act to its own favour, and, in turn, it should be willing to compromise and sacrifice some of its own possible benefits to initiate or keep up such cooperation if that seems promising (as e.g. in the Prisoner's Dilemma).
Informally, such correlation between the two agents' actions is governed by the equation

$$\begin{aligned}(1)\ j\text{'s propensity to help } i \times (2)\ \text{value of } j\text{'s help for } i \\ = (3)\ i\text{'s willingness to compromise towards } j\ (\star)\end{aligned}$$

where (1) reflects the assessment of j's behaviour in the course of the game with respect to how beneficial it is for i, (2) is a measure of how high i values j's possible help, i.e. the "power" of j towards i, the degree to which j is able to improve i's standing. These two values together determine the degree to which i is willing to sacrifice some of its own payoff in order to achieve compromise with j to the benefit of both.
Note that by "sacrificing some of its own payoff" we mean choosing some alternative that is "safer" than the previous action in the sense that it ensures a possibly sub-maximal,

[11] A neighbourhood $\mathcal{N}_i$ of player i is an arbitrary, non-empty subset of i's peers ($\mathcal{N}_i \subseteq P - \{i\}$).

yet more reliable payoff[12]. This notion does *not* bear any implications of "common" or "global" goals, however; we still view an agent as an individual utility maximiser.
How can j obtain a model of (1) and (2)? A peer's propensity to help i will certainly depend on whether actions that i thinks of as "helpful" are desirable for the peer j; it will also depend on whether i would help j in return, and whether it has the *power* to help j at all, given that j's standing also depends an the remaining agents $\mathcal{P} - \{i, j\}$. This "power" is nothing but what is expressed by (2), seen from the opposite point of view (i.e. the value of i's help for j).
It becomes clear that this mutual modelling of preferences and "values of peers" is inherently recursive in a society of rational agents reasoning about their opponents, so we have to decide on a method of coping with this recursion.
The position we assume is that we see rational agents as entities which are aware of their opponents' rationality and which also know that they are being modelled as rational entities by their opponents (and know nothing more than that). This may seem a confusing way to put it, but more simply described, the nesting of models goes down to "level 3" meaning that in how i behaves towards some neighbour $j \in \mathcal{N}_i$, i will consider:

what i knows about j,
what i thinks j knows about i and
what i thinks j knows about what i knows about j.

This view bears strong resemblance to Gmytrasiewicz' (Gmytrasiewicz, 1996; Gmytrasiewicz and Durfee, 1995) Recursive Modelling Method, but its limited maximum nesting depth makes it much easier to handle than recursive models with infinite nesting[13].
Presenting these reasoning principles for two agents i and j already hints at our strategy of decomposing the analysis of neighbourhoods of arbitrary size into many binary modelling steps, so for the moment we shall only focus on a single agent i and on the model it tries to build of some peer j. Later, we shall return to $|\mathcal{N}_i|$-sized neighbourhoods by combining the results of several binary interdependency analyses.

In the following sections we introduce formal frameworks for modelling (a) the *value*[14] and (b) the *preference structure* of an arbitrary peer with respect to the modelling agent's action alternatives in a multi-player environment: *Gain Models* and *Probabilistic Ordering Models*. Simple update algorithms will be provided for these and a social decision-making principle based on the principles informally expressed in Equation ($\star$) will be proposed that turns the nested models into *social action values* which can then be used by the Social Behaviour Generator to generate a social bias that may alter the individual utility-maximizing strategies of the SE. In that, most of the concepts will be different from previous work on recursive modelling methods (Gmytrasiewicz, 1996; Vidal and Durfee, 1998a). Therefore, a more extensive treatment of the technical details will be necessary in contrast to the algorithms presented in the previous sections which were based on "off-the-shelf" machine learning methods.

[12] As an example, the option to "cooperate" in the Prisoner's Dilemma produces a lower payoff than "exploiting" the opponent, but attempts to "exploit" will not go unnoticed in the long run, thus yielding the poor results of the "defect/defect" joint action most of the time.

[13] Another (admittedly vague) reason for choosing this level of nesting is that we don't expect humans to go into "deeper" recursive reasoning, and humans can still infer "what one likes from what one does".

[14] "Value" of a peer and "value of a peer's help" shall be used as synonyms henceforth.

5.4.2 Formal framework

The formal framework underlying the learning and decision-making activities of the SBE consists of three parts – the approximation of two-player utility dependencies in n-player interactions, a learning algorithm for recursive models of these and the computation of social action values to bias agents' strategic decision-making toward socially feasible action choices.

Modelling binary payoff interdependencies in n-player games

The first thing that is required to find out which actions of a peer j can be considered cooperative and which not is an approximation of the value of j's actions for i, given the payoffs actually received by i after each round. It is part of the nature of the game that those payoffs depend on the actions of agents other than j, therefore our approach will be that we introduce those values for a constant behaviour of all agents except i and j first and then "approximate those other agents away".
Loosely speaking, if i intends to play $a_i \in A$, the value v_{a_i} of j's action $k_1 \in A$ is larger than that of $k_2 \in A$, if the distance of the respective payoffs to the minimal payoff for i under j's actions is larger for k_1 than it is for k_2. Thus, if $u_i : \mathcal{A} \rightarrow \mathbf{R}$ is once more i's payoff function,

$$v_{a_i}(k_1) > v_{a_i}(k_2)$$

is equivalent to

$$u_i(a_i, k_1, a_{-\{i,j\}}) - \min_{a_j \in A} u_i(a_i, a_j, a_{-\{i,j\}}) > u_i(a_i, k_2, a_{-\{i,j\}}) - \min_{a_j \in A} u_i(a_i, a_j, a_{-\{i,j\}})$$

if we assume some constant behaviour $a_{-\{i,j\}}$ for the remaining agents. So whenever j chooses an action that is better than some other action as compared to the "worst" alternative it could have chosen, this action is considered more valuable for i. Diminishing the payoffs of the two action combinations by the minimal worst payoff $u_i(a_i, a_j, a_{-\{i,j\}})$ j could induce on i in the current situation (given that i plays a_i and the remaining agents play $a_{-\{i,j\}}$) is necessary to put the benefits of k_1 and k_2 in relation to the worst outcome j can cause.
If we were to compute the exact values for v_{a_i} for the whole spectrum of $a_{-\{i,j\}}$, the above inequality would have to be replaced by

$$\sum_{a_{-\{i,j\}} \in A_{-\{i,j\}}} P(a_{-\{i,j\}}) \cdot v_{a_i}(k_1) > \sum_{a_{-\{i,j\}} \in A_{-\{i,j\}}} P(a_{-\{i,j\}}) \cdot v_{a_i}(k_2) \tag{5.2}$$

where $P(a_{-\{i,j\}})$ is the joint probability of the remaining agents' actions. Such values are, of course, highly infeasible to compute, since even the update of the probabilities requires exponential space and an exponential game duration to yield reliable values. We therefore simplify this model to the easy-to-update approximation of *gain*-values defined as follows for any two players i and j:

Definition 5.1 (Gains)
Let $\mathcal{P}$ be the set of players in an n-player game with strategy space $\mathcal{A} = A^n$. For any two players $i, j \in \mathcal{P}$ we define the **gain values** *of j's actions for i's action $a_i \in A$ as the values of the function $gain_{a_i} : A \rightarrow [0; 1]$ computed as*

$$gain_{a_i}(a_j) \quad = \quad \frac{\lambda \cdot maxgain_{a_i}(a_j) + (1 - \lambda) \cdot mingain_{a_i}(a_j)}{risk(a_i)} \tag{5.3}$$

for any **optimism parameter** $\lambda \in [0;1]$.
Thereby, the **maximal gain** *and* **minimal gain** *values of j's action a_j for i's action a_i and the* **risk** *value of action a_i are defined by the following equations:*

$$mingain_{a_i}(a_j) = \min_{a_{-\{i,j\}} \in A^{n-2}} \left(u_i(a_i, a_j, a_{-\{i,j\}}) - \min_{a'_j \in A} u_i(a_i, a'_j, a_{-\{i,j\}}) \right) \quad (5.4)$$

$$maxgain_{a_i}(a_j) = \max_{a_{-\{i,j\}} \in A^{n-2}} \left(u_i(a_i, a_j, a_{-\{i,j\}}) - \min_{a'_j \in A} u_i(a_i, a'_j, a_{-\{i,j\}}) \right) \quad (5.5)$$

$$risk(a_i) = \max_{a_{-i} \in A^{n-1}} u_i(a_i, a_{-i}) - \min_{a_{-i} \in A^{n-1}} u_i(a_i, a_{-i}) \quad (5.6)$$

Equations (5.4) and (5.5) provide measures for the maximum/minimum "gain" in payoffs measured as the distance to the minimum payoff with respect to j's actions under all joint actions performed by the remaining agents. The respective $a_{-\{i,j\}}$ for which these gains are achieved represent the most/least favourable action combinations of third parties with respect to the "help" j can provide to i if it plays action a_j.
Equation (5.6) computes the overall risk for i in taking action a_i, i.e. the difference between the absolute minimum and maximum payoff under a_i. The actual gain values defined in Equation (5.3) are computed by combining the best and worst possibly obtainable gains weighted by some "optimism parameter" λ that expresses how likely the implementation of the respective remaining agents' joint action is considered by i. Then, the gain of j playing a_j is determined as that weighted gain combination in relation to the risk value of action a_i.

To illustrate the properties of such gain models, consider the following example of a four-player resource-load game in which agents share two resources (the parameters of the game are $v = 20$, $T = 4$ and $c = 1$). The following table shows a matrix of gain values of the actions of agent 2 for agent 1, i.e. player 1 is the agent that models the help that 2 can provide irrespective of others' actions, and agent 1 uses an optimism parameter of $\lambda = 0.5$.

*player*1 \ *player*2	00	01	10	11
00	0	0	0	0
01	0.39	0	0.39	0
10	0.39	0.39	0	0
11	0.42	0.21	0.21	0

If agent 1 plays '00', obviously none of agent 2's actions can improve agent 1's standing, because it will always receive zero payoff. Likewise, if agent 2 plays '11' this cannot be of any help for agent 1 because that action always maximises the "damage" done to agent 1 by agent 2. When the first player accesses only one of the resources, then any action of agent 2 by which it either accesses none of the resources or the opposite resource of the one agent 1 has chosen is considered equally helpful for agent 1. If agent 1 accesses both resources, the influence agent 2 can have on agent 1's standing is smaller then before, except in the optimal case in which agent 2 refrains completely from resource access (thereby "stepping out of i's way").

Thus, gain functions provide us with simple approximations of binary payoff dependencies in n-player environments. Updating such a model is a rather trivial matter; in

the course of the game, agent i simply uses its payoff estimate π_i instead of u_i and the joint actions perceived so far instead of the whole remaining agents' strategy space A^{n-2} to compute the quantities from Definition 5.1. We will call the estimates agents build of the gain values *gain models* (GMs) to distinguish between the exact values defined above and the picture agents have constructed of them from their experience.
Bearing in mind, however, that i is supposed to model the gain values it induces on j, the picture j has of the gains it is inducing on i and so on, we need a formalism to approximate some *other* agent's gain models, which will have to make do *without* information about the opponent's payoff function. This is presented in the next paragraph.

Probabilistic ordering models

A Probabilistic Ordering Model (POM) is a probabilistic approximation of the orderings that govern j's preferences concerning the combinations of its own actions and the actions of i, i.e. it is a model one agent has of its peer's gain model. While other authors (especially Gmytrasiewicz (1996)) represent hypotheses about opponent preferences as distributions over a set of possible payoff matrices, we encapsulate the uncertainty here in *rank probabilities*. This means that we are only interested in the probability that the value of some (i,j) action combination (l,k) in j's GM[15] takes on a certain ranking position among all (a_i, a_j) rather than in the probability with which it has a concrete numerical value. We believe that such a probability distribution over combined actions and ranks yields a more compact representation of all possible matrix entry orderings than a (probability-weighted) enumeration of all payoff matrices that correspond to those orderings. We first define POMs formally before going into the details of their use.

Definition 5.2 (Probabilistic Ordering Models)
A **Probabilistic Ordering Model** *is a structure* $s = \langle A, M, R, \rho, p\rangle$ *where*

- A *is the action set of both agents* i *and* j,
- $M : A^2 \to \mathbf{R}$ *is a real-valued matrix,*
- $R = \{1, 2, \ldots, |R|\}$ *is a* **set of ranks**,
- $\rho : A^2 \to R$ *is a* **ranking function**, *such that*

  1. $\forall l, k, l', k' \in A. \quad \rho(l', k') < \rho(l, k) \iff M(l', k') > M(l, k)$,
  2. $\forall l, k \in A. \quad \rho(l, k) = r \Rightarrow \forall r' \neq r. \rho(l, k) \neq r'$ *and*
  3. $\forall l, k \in A. \quad \exists r. \rho(l, k) = r$

  hold [16].

- $p : A^2 \times R \to [0; 1]$ *is the* **rank probability function**, *that assigns to each pair* (l, k) *a non-negative probability with which it assumes position* $r \in R$ *in the ordering of all matrix entries.*

[15] When talking about models of opponent gains, we will use letters l, k to denote the two agents' actions instead of a_i and a_j.

[16] These conditions require that each matrix entry (l, k) (i) holds exactly one rank in the ordering and (ii) that this rank is lower than that of some other entry, if the value of the first entry is higher (i.e. the maximal entry in M has rank 1).

The idea behind having such POMs is that if we take M to be j's actual GM matrix and choose R large enough to represent any ordering in GM ($|R| \geq |A^2|$), then POMs allow for the representation of any probability distribution over the orderings in that GM. Hence, i can use evidence obtained by action observations to infer changes in the probability of certain GM orderings by updating the individual rank probabilities of the respective binary action tuples (locally). More specifically, the observations will be used by i to increase one of $P((l',k') > (l,k))$ or $P((l',k') < (l,k))$ by some constant $\delta \in [0;1]$ which we call the *POM update factor*.

Using elementary transformation rules for probabilities and the conditions put forward for ρ in the POM definition, these two probabilities can be computed as follows for any $l,k,l',k' \in A$:

$$\begin{aligned} P((l',k') > (l,k)) = P(\rho(l',k') < \rho(l,k)) &= P\left(\bigvee_{r'<r} (\rho(l',k') = r' \wedge \rho(l,k) = r)\right) \\ &= \sum_{r'=1}^{|R|-1} \left(p(l',k',r') \cdot \sum_{r=r'+1}^{|R|} p(l,k,r)\right) \end{aligned}$$

and

$$\begin{aligned} P((l',k') < (l,k)) = P(\rho(l',k') > \rho(l,k)) &= P\left(\bigvee_{r'>r} (\rho(l',k') = r' \wedge \rho(l,k) = r)\right) \\ &= \sum_{r'=2}^{|R|} \left(p(l',k',r') \cdot \sum_{k=1}^{r'-1} p(l,k,r)\right) \quad . \end{aligned}$$

This calculation leads to the definition of the update operation for POMs, the add-operation. It is based on the intuition that i starts with the "empty" POM $s_\emptyset$ (whose p-function maps every (l,k,r) to the initial probability $\frac{1}{|R|}$ for every rank $r \in R$ (implying that each matrix entry holds every position in the total ordering with the same probability). On incoming evidence in support of either of the propositions $(l',k') > (l,k)$ and $(l',k') < (l,k)$, it updates the rank probabilities for both (l,k) and (l',k') according to the following definition:

Definition 5.3 (The add-operation for POMs)
Let $s = \langle A, R, \rho, p\rangle$[17] be a POM. The **add-operation** *is defined by*

$$\mathsf{add}(s, (l',k'), (l,k), \delta) = s'$$

where $s' = \langle A, R, \rho, p'\rangle$ and $\delta \in [0;1]$ is the fraction by which $P((l',k') > (l,k))$ should be increased.
Then

$$\forall r \in R.p'(l',k',r) = \frac{1}{1+\delta} \cdot \left(p(l',k',r) + \delta \cdot \frac{p(l',k',r) \cdot \sum_{r'=r+1}^{|R|} p(l,k,r')}{P((l',k') > (l,k))}\right)$$

and

$$\forall r \in R.p'(l,k,r) = \frac{1}{1+\delta} \cdot \left(p(l,k,r) + \delta \cdot \frac{p(l,k,r) \cdot \sum_{r'=1}^{r-1} p(l',k',r')}{P((l,k) < (l',k'))}\right)$$

[17] From now on, we can drop the M argument of the POM specification, because we are talking about i's *model* of j's GM, and i is ignorant of the real structure of M.

r	1	2	3	4
$(0,0)$	0.25	0.25	0.25	0.25
$(0,1)$	0.40	0.29	0.19	0.12
$(1,0)$	0.12	0.19	0.29	0.40
$(1,1)$	0.25	0.25	0.25	0.25

(a) After two add(s,(0,1),(1,0),0.3) operations; the probabilities of the higher ranks for (0,1) have increased to a greater degree than those of low ranks, with opposite effects on the rank probabilities of (1,0). Thus, the algorithm assigns larger probability shifts to those ranks who can cause δ most easily.

r	1	2	3	4
$(0,0)$	0.96	0.02	0.01	0.01
$(0,1)$	0.02	0.92	0.04	0.02
$(1,0)$	0.02	0.04	0.92	0.02
$(1,1)$	0.01	0.01	0.02	0.96

(b) After 20 iterations of adding 0.3 to all binary relations in in the chain $(0,0) > (0,1) > (1,0) > (1,1)$; the POM captures the transitive nature of ">" by clearly distributing all ranks among the matrix entries. This is a very interesting property, because we get the transitivity conclusions "for free".

r	1	2	3	4
$(0,0)$	0.44	0.16	0.14	0.26
$(0,1)$	0.03	0.92	0.04	0.01
$(1,0)$	0.01	0.04	0.92	0.03
$(1,1)$	0.33	0.12	0.11	0.44

(c) This is the result of adding 0.3 twice to $(1,1) > (0,0)$ in the POM shown in (b); the model uncovers the violation of the antisymmetricity of ">" and reduces/increases the probabilities of extreme ranks for (1,1) and (0,0) much quicker than it changed them in (a). (1,0) and (0,1) remain unchanged.

Figure 5.13: The POMs shown in (a)-(c) show the results of several (series of) operations on a simple POM with $|A^2| = |R| = 4$; the rows show the probability of each entry of the matrix $A \times A$ assuming rank r (represented by the columns). Initially, all POM-entries are 0.25.

and the values $P((l',k') > (l,k))$ *and* $P((l,k) < (l',k'))$ *are calculated as presented above using the* original *POM s (we shall write* $s((l',k') > (l,k))$ *for* $P((l',k') > (l,k))$ *henceforth, if that probability is calculated using the POM s).*

At first look, this definition seems quite complex, but it is necessary to keep the rank probabilities p consistent with the axioms of probability theory. It is rather difficult to explain the mathematical details of this construction in words, so we suggest observing Figure 5.13 which shows some examples of POMs and operations performed on them to illustrate their behaviour.

One of the interesting features of the formalism is that in performing add-operations, we need not worry about decreasing the δ added to some POM entry with every new piece of evidence. This is normally necessary, because if $s((l',k') > (l,k))$ is already very large due to past observations, it is reasonable not to weigh any further evidence for this statement as heavy as before. Fortunately, add-operations have precisely this effect on probability values even if a constant δ is employed throughout.
Another important aspect is shown in table (b) of Figure 5.13, namely that if evidence for transitive "chains" of >-relationships is provided, then the POM distributes the ranks neatly among matrix entries. Table (c) shows that the POM is "aware" of this rank distribution being fragile, so that a violation of the previously assumed chain immediately has very strong effects on the values affected by the contradiction.

This makes POMs a very practical means of approximating the orderings that govern matrices and thus particularly useful in settings in which knowledge of the precise quantities of matrix positions is not required. We next discuss the way POMs will be employed by the Social Behaviour Engine.

Recursive belief modelling

We now have shown how an agent i can update a model of its own gains with respect to some peer j's actions and how the models it has of j's GM will be constructed. As mentioned before, we intend to conduct this modelling down to the third level of nesting. We therefore equip agent i's SBE with the following four components for each agent

$j \in \mathcal{N}_i$:

Model	*Function*	*Type*
$s^0(l,k)$	i's model of its own gains wrt j	GM
$s^1(k,l)$	i's model of j's gains	POM
$s^2(l,k)$	i's model of j's model of i's gains	POM
$s^3(k,l)$	i's model of j's model of i's model of j's gains	POM

The GM s^0 of i will be constructed as described above, i.e. $\forall a_i \in A, a_j \in A.\quad s^0(l,k) = gain_{a_i}(a_j)$. The update algorithm for it, UpdateGm will not be described in detail, since it is clear that it can be conducted by simply keeping track of the maximal/minimal payoffs and the corresponding joint actions during the game.
The remaining models will be POMs with an initially uniform distribution over all ranks for every matrix entry (i.e. $s^1 = s^2 = s^3 = s_\emptyset$). These POMs will be updated after each round of the game using two different algorithms UpdatePom_unprofiled and UpdatePom_profiled, depending on whether the behaviour of the agent whose gains are being modelled is profiled or not, i.e. whether it exhibits some preference for specific actions or not.

UpdatePom_unprofiled

This algorithm is employed whenever a SBE component other than s^0 has to be updated and the opponent of the agent whose gains are being modelled exhibits an "unprofiled" behaviour, i.e. it shows no preference for any action or, to put it another way, does not show any *intentionality* in its actions. Such an update mechanism is necessary for two reasons: firstly, during an early phase of the game agents might still follow an exploration strategy to gather information about the nature of their payoff function. For their opponents, such a "playing around"-attitude does not reveal any information about the preferences of those agents but they know that they might be modelled – if acting intentionally themselves – as intentional entities by their sub-intentional behaviour exposing peers. A second reason is that agents might encounter opponents which *are* sub-intentional non-deliberative components of the system lacking any rationality. A possible condition to decide whether some peer behaviour[18] $\Pr_j : A \to [0;1]$ is profiled or not is given by

$$profiled(j) = \texttt{true} \iff \sum_{a_j \in A} \left(\Pr_j(a_j) - \frac{1}{|A|} \right)^2 > \vartheta$$

where $\vartheta \in [0;1]$ is a *variance threshold* (which is exceeded, if the variance from the supposed mean probability $\frac{1}{|A|}$ becomes too high).
We now explain UpdatePom_unprofiled informally for a situation in which j acts in an "unprofiled" way and i acts intentionally.
Given this situation, how should i's GM be modelled by j? Since j reveals no preferences through its actions (and is aware of that), it knows that i's $s^1 = s_\emptyset$. Therefore j assumes i to act irrespective of j's own actions, i.e. i will choose that action which provides the highest gain in s^0 regardless of what j plays. If i just played l, the expected gain for i was

$$\sum_{k \in A} \Pr_j(k) \cdot s^0(l,k)$$

[18] $\Pr_j$ is the posterior probability of the peer to play $k \in A$ for every k, i.e. the frequency with which it has chosen that action in the past.

because for i j's action probabilities are only distributed according to $\Pr_j$. Assuming that i is a rational player, if i played l_0 in the previous round, it must have tried to improve its expected payoff when choosing l, which means that

$$\sum_{k \in A} \Pr_j(k) \cdot s^0(l,k) \quad > \quad \sum_{k \in A} \Pr_j(k) \cdot s^0(l_0,k)$$

must hold. Given that this inequality might have been caused by any of the $s^0(l,k) > s^0(l_0,k)$ and considering that j's actions are almost equally distributed, we add some small (constant) quantity δ to the probabilities of all such relations. So in any round, if l_0 and l are i's subsequent action choices, we perform the operations[19]:

$$\forall k \in A. \quad \mathsf{add}(s^2, (l,k), (l_0,k), \delta)$$

UPDATEPOM_PROFILED

The last section covers the cases in which at least one of the two agents exhibits an unprofiled behaviour. If both act like deliberative, rational players, things get much more complicated, because action choice depends on the opponent's expected behaviour, the opponent's behaviour depends on the first agent's behaviour, and so on.
Before going into the details of the update algorithm devised for such situation, the auxiliary definition of *flattened POMs* is necessary, because the probabilities $P((l',k') > (l,k))$ that we have used so far only provide a means of making *comparative* statements between the values of two action tuples and no method of deducing *absolute* values for (l,k) action combinations has been introduced.

Definition 5.4 (Flattened POM) *Let s be any POM. Then the* **flattened POM** $\tilde{s}$ *corresponding to s is a matrix $\tilde{s} : A^2 \to \mathbf{R}$ such that*

$$\forall l,k \in A. \tilde{s}(l,k) = \sum_{r=1}^{|R|} c_r \cdot p(l,k,r)$$

and $\vec{c} = (c_1, \ldots, c_{|R|})$ is a real-valued **rank-flattening vector** *with $\forall r' > r.\ c_{r'} < c_r$, i.e. some vector that assigns the highest value to the lowest rank and decreases with increasing rank.*

Flattening a given POM compiles it into a binary payoff-matrix, which seems to contradict our initial considerations about discarding quantitative matrix properties and restricting ourselves to an analysis of the orderings of opponents' gain values. Yet this transformation is indispensable for the following analysis of the agents decision situation, so we will use it in the following without specifying how the $\vec{c}$-vector will be determined in practice. For reasons of conventional notation we shall write $s(l,k)$ for $\tilde{s}(l,k)$ henceforth for any POM s, thereby pre-assuming that the models have been converted into flat matrices where necessary.

Since we have decided to consider this nesting only up to the third step, we can assume the following rational choice model for players:
Player i will base its choice upon the gain it expects for any action $l \in A$ given the expected behaviour of its peer j. The expected gain $g_i(a_i)$ can be used as a measure for the probability with which i will choose the respective action and it can be computed as

$$g_i(a_i) = \| \sum_{k \in A} s^0(a_i,k) \cdot P(a_j = k) \| \tag{5.7}$$

[19]The same operations would be performed on s^1 and s^3 if i was acting in an unprofiled way, and j exhibited a profiled behaviour.

if $P(a_j = k)$ is the probability with which i expects j to play $k \in A$ ($\| \cdot \|$ is again a normalizing function, so that the g_i-values always sum to one). The next step is to determine the peer action probabilities $P(a_j = k)$. Assuming that j is a rational player, too, i has to consider that the peer follows a similar logic in selecting its own actions. So if j estimates the probability with which i might play l as[20] $\varepsilon_i(l)$ we obtain

$$P(a_j = k) \approx \| s^1(k, l) \cdot \varepsilon_i(l) \| \tag{5.8}$$

as a selection probability estimate for all $k \in A$, because the picture i has of j's gain model is reflected in the POM s^1(again, the values are normalised to obtain a valid probability distribution).
Furthermore, i assumes that j considers also what it knows about i's own GM to compute $\varepsilon_i(l)$ in Equation (5.8). Knowledge of what i has revealed about its own preferences is stored in s^2, so these $\varepsilon_i(l)$-values compute as

$$\varepsilon_i(l) = \| s^2(l, k) \cdot \varepsilon_j(k) \| \tag{5.9}$$

by virtue of the same argument as before. Taking the recursion one step further, s^3 (i's knowledge of what it has revealed about its knowledge of j's GM) can be used to obtain the values for $\varepsilon_j(k)$ in Equation (5.9):

$$\varepsilon_j(k) = \| s^3(k, l) \cdot \varepsilon_i(l) \| \tag{5.10}$$

This time, however, i assumes that no further recursive models are maintained by its opponent, so that all j can use to determine the action probabilities of i are the posterior probabilities with which the actions have been selected so far:

$$\varepsilon_i(l) = \Pr_i(l) \tag{5.11}$$

Taking equations (5.8) to (5.11) together and replacing the ε-variables where required, we can construct an explicit representation of the *expected gain* $g_i(a_i)$ for any action of i that makes use of all available models.

Although this choice model will only be used later in the social decision-making rule, we have introduced it here to motivate the construction of UPDATEPOM_PROFILED and, in fact, the above equations map directly to appropriate update operations for the POMs s^1, s^2 and s^3 after each round.
These operations basically reflect the propagation of the ε-probabilities from s^3 to s^1. Assume that if $(l_0, k_0) \rightarrow (l, k)$ is a (i, j) two-move sequence (i.e. for two consecutive rounds t and $t+1$, $a^{(t)}_{\{i,j\}} = (l_0, k_0)$ and $a^{(t+1)}_{\{i,j\}} = (l, k)$ hold). For each such consecutive action pair, we perform the following steps:

1. Let $\bar{s}^3 := s^3$.
2. Perform $\mathsf{add}(\bar{s}^3, (k, l), (k_0, l_0), \delta)$.
3. For all $l' \in A$, let

$$d_{l'} := \frac{1}{|A|} \cdot \sum_{k' \in A} \Big(\bar{s}^3((k', l') > (k_0, l_0)) - s^3((k', l') > (k_0, l_0)) \Big)$$

be the mean of the differences in the probabilities of all affected orderings induced on s^3 by Step 2.

[20] The use of ε instead of P for these probabilities is due to the fact that ε-values are derived from recursive *models* of preferences, while probabilities P always denote the *exact* likelihood with which some proposition holds.

4. Let $\bar{s}^2 := s^2$.

5. For all $l' \in A$, perform
$$\mathsf{add}(\bar{s}^2, (l', k), (l_0, k_0), d_{l'} \cdot \delta)$$
so that s^2 is adapted to the changes in Step 2.

6. For all $k' \in A$, let
$$d_{k'} := \frac{1}{|A|} \cdot \sum_{l' \in A} \Big(\bar{s}^2((l', k') > (l_0), k_0) - s^2((l', k') > (l_0, k_0)) \Big) \quad .$$

7. Let $\bar{s}^1 := s^1$.

8. For all $k' \in A$, perform
$$\mathsf{add}(\bar{s}^1, (k, l'), (k_0, l_0), d_{k'} \cdot \delta)$$
to propagate the changes in Step 5. to s^1.

9. Finally, adopt $\bar{s}^1$, $\bar{s}^2$, and $\bar{s}^3$ as the new POMs for i's SBE.

Briefly summarised, all UPDATEPOM_PROFILED does is to update the affected probabilities in s^3 according to the action observation, and to propagate the resulting probability changes from s^3 to s^2 and from s^2 to s^1, as if, loosely speaking, the changes of s^1, s^2 and s^3 directly represented the changes in the ε-values in equations (5.8)-(5.12) above.

In the next section we provide a social action-value function computing rule, which suggests a method of realising the right-hand side of equation ($\star$) presented in Section 5.4.1 by making use of the formalisms introduced in this section.

Determining social action values

Intuitively, an agent is willing to compromise or cooperate if it would give away some proportion of the gain it expects for itself for the sake of achieving that compromise. In terms of our formal framework this translates into the principle

> *"instead of choosing that action which promises the highest utility for i it will also consider other, sub-optimal actions if the expected behaviour of j given those actions promised a gain at least as big as the amount of utility sacrificed"*

In our recursive gain model system, this can be restated as follows: if i has computed some predicted behaviour $P(a_j = k)$ for j then i's expected gain is $g_i(a_i)$ for any action $a_i \in A$ as defined in Equation (5.7). Then, if we let $\gamma \in [0; 1]$ be a *compromise factor* that expresses how big a percentage of its expected maximal gain i is willing to sacrifice, and m_i is i's (individual) action-value function obtained from the Strategy Engine, we can construct the decision function SOCDECIDENEXT for i for round t in the following way:

1. For every $a_i \in A$ compute the normalised expected gain $g_i(a_i)$ as defined in Equation (5.7).

2. Let $\mathcal{L}_j := \{a_i \in A \quad | \quad m_i(a_i) + \gamma \cdot g_i(a_i) \geq \max_{a_i' \in A} m_i(a_i')\}$.

3. While $\mathcal{L}_j = \emptyset$ decrease γ and try constructing $\mathcal{L}_j$ as before, until either $\mathcal{L}_j \neq \emptyset$ or $\gamma = 0$.

In step 2. we construct the set $\mathcal{L}_j$ of all actions for which the sum of action value $m_i(a_i)$ and the γ-weighted expected gain induced by j's behaviour is greater than the maximal action value (the expected relative payoff of the most "greedy" action). We call $\mathcal{L}_j$ the set of *socially feasible actions* towards j. If this set is empty, we decrease γ until $\mathcal{L}_j$ becomes non-empty, which means that we look for actions which enact less compromise, if "we can't be as nice as we would like to be". If γ becomes zero and $\mathcal{L}_j$ is still empty, the algorithm terminates and returns the empty set; else, a non-empty set of socially feasible actions is returned.

Extending this rule to arbitrarily-sized neighbourhoods $\mathcal{N}_i \subseteq P - \{i\}$ can be done in a straightforward manner by proceeding in the following way: for each agent j in i's neighbourhood $\mathcal{N}_i$, i constructs the set of socially feasible actions $\mathcal{L}_j$ separately and determines the set of *neighbourhood-feasible* actions as $\mathcal{L} := \cup_{j \in \mathcal{N}_i} \mathcal{L}_j$. On the basis of these constructions, the *social action value* function l_i of Section 4.4.2 can be defined as follows:

$$l_i(a_i) = \| \lambda_i(a_i) \|, \quad \lambda_i(a_i) = \begin{cases} \sum_{j \in \mathcal{N}_i} \sum_{k \in A} s_j^1(k, a_i) & \text{if } a_i \in \mathcal{L} \\ 0 & \text{else} \end{cases} \tag{5.12}$$

i.e. that action in $\mathcal{L}$ is chosen with highest probability that seems most desirable for the largest possible number of agents in the neighbourhood (after all, their preferences should also be taken into account). If $\mathcal{L} = \emptyset$, l_i should exhibit no preference for any action, so we choose to define

$$\forall a_i \in A. l_i(a_i) = m_i(a_i)$$

to account for cases in which compromise with any neighbour is impossible.
The separate computation of $\mathcal{L}_j$-sets implies that we allow for various degrees of cooperation with each neighbour, a fact which makes the implicit deal-making highly flexible and adaptive to peers with different "power" and "helpfulness".
The construction of $\mathcal{L}$-sets constitutes the central step in the employed social reasoning methodology necessary to construct the social action-value function l_i. This can be used to overrule the greedy "strategic" choices implemented by playing m_i.

At this point one might be tempted to infer from the design of this function that we have circumvented the problem of antagonism by endowing agents with a *cooperative bias*, which inevitably will make them cooperate in practice, because they are designed to do so. However, this is not the case, since our agents still do not pursue any other goal than to maximise their individual utility. Their propensity to cooperate springs solely from their ability to detect the potential for cooperation to the benefit of both themselves and their opponent. It is now time to conduct a first evaluation of this learning component as was done for the other two parts of the architecture. For various reasons which we soon discuss, such preliminary testing cannot be easily organised for the Social Behaviour Engine, so we will actually restrict this analysis to the presentation of a simple example.

5.4.3 Preliminary results

The formal framework used by the social learning and decision-making layer in the LAYLA agent architecture offers a variety of parameters for which appropriate values should be

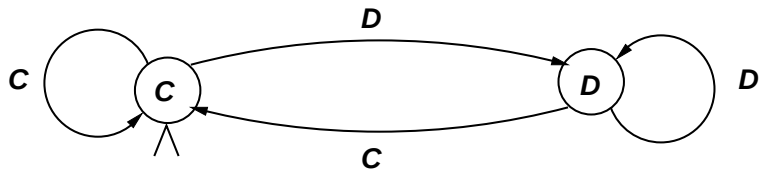


Figure 5.14: TIT FOR TAT state transition diagram. The arcs represent the actions of the modelling agents, the nodes those of the modelled opponent. 'C' is marked the initial state, so that the opponent would never defect first. Whenever the modelling agent defects, the opponent will defect as well in the following round and the converse holds for cooperating.

determined through empirical evaluation. These include the *optimism parameter* λ employed to calculate gain values, the *rank-flattening vector* $\vec{c}$ used to compile POMs into real-valued matrices, the POM *update factor* δ, the *compromise factor* γ that determines which actions will be considered socially feasible and the size and structure of *neighbourhoods* $\{\mathcal{N}_i\}_{i\in\mathcal{P}}$.
Unfortunately, we are not in the position of testing the performance of the SBE independently from the other components as was done for the Utility Engine and the Strategy Engine in previous chapters. The payoff function estimate π_i could in principle once more be replaced by the precise payoff function u_i. The action-value function of the Strategy Engine, however, can *not* be substituted by the respective precise concept, because, as was previously remarked (cf. Section 5.3.3) the opponent behaviour on which action values are based is not fixed. Thus, no fine-tuning decision of the SBE can be conceived of that can be proved to be valid regardless of the SE results.

For this reason, we deviate from the methodology of making parameter choices through extensive testing and present a simple, very stylised example of the behaviour of social learners. This example will be based on the Iterated Prisoner's Dilemma (IPD) as introduced in Section 3.1.2. The reason for choosing the IPD is that the simple TIT FOR TAT strategy provides a reliable, robust meta-strategy that can be used as a fixed behaviour pattern to simulate the OBP models of the Strategy Engine. TIT FOR TAT requires the player (i) not to be the first to defect and (ii) always to perform whichever action the opponent has performed in the previous round. Using the notational conventions of the "opponent automata" (cf. Section 5.3), TIT FOR TAT translates for either player into the state transition diagram shown in Figure 5.14. Assuming that both players are TIT FOR TAT-compliant, the action value function m_i yields for any player i in round t

$$m_i(C) = \begin{cases} 0.325 & \text{if } a_i^{(t-1)} = C \\ 0.0 & \text{else} \end{cases} \qquad m_i(D) = \begin{cases} 0.675 & \text{if } a_i^{(t-1)} = C \\ 1.0 & \text{else} \end{cases}$$

Quite naturally, defecting is considered more effective if the agent has been defecting before and cooperating seems slightly more promising if the agent has been cooperating before (which, by the rules of TIT FOR TAT forces the opponent to cooperate in the next round as well).
Although these action values take opponent behaviour into account, they still favour selecting the maximin "defect" strategy, so our task will be to show that the (C,C)-agreement can be established when using the SBE.

A last preliminary step that has to be taken to provide all the necessary inputs to the SBE is to construct the two players' private gain models s_1^0 and s_2^0 which can be

computed using the exact payoff functions u_1 and u_2. The matrix obtained by performing the calculations of Definition 5.1 is for both players

player 2 / player 1	C	D
C	1.0	0.0
D	1.0	0.0

and is for this special simple case independent of the choice of λ.

Using $\delta = 1.0$ and assuming an initially equal posterior probability distribution ($\forall i \in \{1,2\}.\forall a_i \in \{C,D\}.\Pr_i(a_i) = 0.5$), cooperation on both sides emerges *immediately* and remains stable throughout the game if and only if $\gamma > 0.5$. This observation is subject to the condition that both agents play 'C' in the first round, which is understandable given that the computation of m_i depends on the previous action and the construction rule for $\mathcal{L}_j$ cannot include 'C' in these sets if $m(C) = 0$.
In a second test, we decrease γ_2 to 0.4 after 50 rounds. Quite remarkably, it took player 1 only 4 rounds in which it was exploited by its opponent to realise that it should alter its behaviour to 'D'. The choice of γ_1 is a very sensitive issue here; while for values of 0.505/0.508 it takes agent 1 6/13 rounds to discover the exploitation, it will fail to do so for any $\gamma_1 > 0.509$. For all remaining rounds, the (D,D)-combination was played, since player 2 cannot go back to 'C' due to its low compromise factor.
Changing γ_2 back to 0.5 again after 10 more rounds yields a permanent (D,D) behaviour which means that agent 1 does not "forgive" agent 2's egoism – unless both agents play 'C' in some future round "by accident", that is.
This is again an effect of the action values resulting from the TIT FOR TAT strategy, and it is not clear whether such effects carry over to the resource-load balancing games in which OBP models are much fuzzier and in which a certain degree of randomization will always influence the way in which the POMs will develop over time. We therefore choose not to make any further decisions concerning SBE parameters and leave this task to the evaluation chapter, in which all components will be tested together.
However, we have obtained a very essential result for our social learning and reasoning components with this "toy example", namely that under certain conditions it *can* make agents move away from short-sighted greedy action choices to collectively rational behaviours. The fact that all used components were constructed one by one according to the formalisms of the UE, SE and SBE prototypes suggests that the integrated reasoning and learning architecture is reasonable, because the result cannot simply be explained with coincidence.

To complete the design of our prototype, we next present the top-level algorithm that summarises the behaviour of LAYLA agents. It encapsulates the control flow between the components and implicitly defines the decision-making function of the agent.

5.5 Integration – the SOCCER algorithm

The top-level LAYLA algorithm called the SOCial Cooperation-Enabling Reinforcement algorithm is rather simple and relies on the notions of "exploration needs" and "profiled behaviour". "Exploration needs" reflect the necessity for any agent to try out actions at the beginning of the game until it has a satisfactory picture of its own payoff function. We assume that as long as those needs are not satisfied, the agent doesn't go into any

strategic/social reasoning at all, thus behaving seemingly sub-intentionally for its peers. We shall use *exploration_needs_fulfilled* as a testable predicate for this (the testing condition might be, e.g. "has the error in payoff prediction fallen below some threshold?").
This notion is crucial to the construction of a social reasoning algorithm, because it provides a clear condition of when to switch the social decision-making component "on" and "off".

The SOCCER algorithm for some player i in round t is shown in Figure 5.15. Since the update mechanisms for the SBE are somewhat complex in that they depend on whether the opponent or the agent itself are behaving intentionally, we have chosen to include the update operations in the top-level algorithm.

The algorithm takes as inputs a neighbourhood $\mathcal{N}_i$, the current joint action $a^{(t)}$ and associated payoff $u_i(a^{(t)})$ and collections of the recursive models s^0-s^3 for every neighbour $j \in \mathcal{N}_i$ (where $\mathcal{S}_j$ denotes the respective s-model for opponent j). Implicitly, the SBE has, of course, a Utility Engine UE and a Strategy Engine SE at its disposal.
The first two function calls UPDATESE and UPDATEUE are meant to update the ANNs and GAs of the lower layers as discussed in earlier sections. Additionally, they output strategies α^{UE} and α^{SE} as exploration/exploitation strategies according to the exploration needs of the UE and the action values of SE.
As far as the Strategy Engine is concerned, the strategy α^{SE} it returns will simply be the (mixed-strategy) action-selection rule m_i derived from individual action values:

$$\forall a_i \in A. \alpha^{SE}(a_i) = m_i(a_i) \tag{5.13}$$

The Utility Engine takes α^{SE} and combines it with an exploration function z that reflects how much exploration is still needed for particular actions, defined as

$$z_i(a_i) = \| \left(\frac{\#(a_i)}{t} \right)^{-1} \| \tag{5.14}$$

for any $a_i \in A$, where $\#(a_i)$ is a function that counts how many times a_i has been played so far, so that those actions that have been played less often will be assigned greater selection probabilities.
If i is not satisfied with the performance of its neural network it will attempt to bias the action selection rule toward actions with higher z_i-values. So whenever the payoff prediction error exceeds the threshold ζ, the strategy α^{UE} returned by the UE (which implicitly defines the UE strategy e_i introduced in Section 4.4.2) will be a mixture of α^{SE} and z_i. Whenever payoff prediction is accurate enough, the action-value maximizing strategy will be chosen:

$$\alpha^{UE}(a_i) = \| e_i(a_i) \|, \quad e_i(a_i) = \begin{cases} z(a_i) + \alpha^{SE}(a_i) & \text{if } |u_i(a^{(t)}) - \pi_i(a^{(t)})| > \zeta \\ \alpha^{SE}(a_i) & \text{else} \end{cases} \tag{5.15}$$

Thereby we use "$|u_i(a^{(t)}) - \pi_i(a^{(t)})| > \zeta$" also as a testable condition for the predicate *exploration_needs_fulfilled*.
UPDATEACTIONPROBS is a simple action-frequency updating function that keeps track of how often i and j have performed particular actions in the past; UPDATEGM() simply updates the $|\mathcal{N}_i|$ s^0-models according to new payoff/action pairs and the results of the

SOCCER$(i, \mathcal{N}_i, a^{(t)}, u_i(a^{(t)}), \mathcal{S}^0, \mathcal{S}^1, \mathcal{S}^2, \mathcal{S}^3)$
  $\alpha^{SE} \leftarrow$ UPDATESE$(a^{(t)})$;
  $\alpha^{UE} \leftarrow$ UPDATEUE$(a^{(t)}, u_i(a^{(t)}), \alpha^{SE})$;
  UPDATEACTIONPROBS$(a^{(t)})$;
  FORALL $j \in \mathcal{N}_i$ DO
    UPDATEGM$(\mathcal{S}_j^0, a^{(t)}, u_i^{(t)})$;
    IF *profiled*(j) = `false` THEN
        IF *profiled*(i) = `true` THEN
          UPDATEPOM_UNPROFILED$(\mathcal{S}_j^2)$;
        END;
    ELSE
        IF *profiled*(i) = `false` THEN
          UPDATEPOM_UNPROFILED$(\mathcal{S}_j^1)$;
          UPDATEPOM_UNPROFILED$(\mathcal{S}_j^3)$;
        ELSE
          UPDATEPOM_PROFILED$(\mathcal{S}_j^1, \mathcal{S}_j^2, \mathcal{S}_j^3)$;
        END;
    END;
    $\mathcal{L}_j \leftarrow$ SOCDECIDENEXT(j, α^{SE});
  END;
  IF *exploration_needs_fulfilled*(i) = `true` THEN
    $\alpha^{(t+1)} \leftarrow$ SOCCOMBINE$(\{\mathcal{L}_j\}_{j \in \mathcal{N}_i})$;
  ELSE
    $\alpha^{(t+1)} \leftarrow \alpha^{UE}$;
  END;
  RETURN $\alpha^{(t+1)}$;
END

Figure 5.15: The SOCCER algorithm.

Utility Engine.
Integration of the SBE, whose recursive gain models are updated with the UPDATE_POM_PROFILED and UPDATE_POM_UNPROFILED algorithms is achieved by performing SOCDECIDENEXT for every peer. This function constructs a (possibly empty) $\mathcal{L}_j$-set for each neighbour by using the action values α^{SE} of the Strategy Engine as described in the previous paragraphs. SOCCOMBINE implements the computation of a social value function l_i by creating a mixed strategy that outputs the most peer-desirable action contained in the union of $\mathcal{L}_j$-sets.
Finally, the strategy to be played in the next round is selected according to whether the UE demands further exploration or the social action values can be safely used (which can be identical to the SE action values, if compromise is not possible).

This provides us with simple instantiations of the functions e_i, m_i and l_i that were proposed to organise the decision-making control flow between LAYLA components, thus enabling the realisation of game simulations with integrated agents.

5.6 Summary

This chapter introduced a prototypical design for game-learning LAYLA agents that is in concordance with the general architecture introduced in Chapter 4, as illustrated in Figure 5.16. The choice of the particular algorithms was partly motivated by the learning tasks, as e.g. in the case of the payoff-learning neural networks and the transition-rule genetic algorithms used for opponent behaviour modelling, and partly derived from extrinsic intuitions, such as the best-response argument used for deriving action values and the "compromise" argument employed by the social reasoning component.
An interesting side-effect of these choices is that the smooth transition from sub-symbolic methods to explicit symbolic reasoning was realised for the learning components in the same way as it exists in the generic InteRRaP architecture. While the Utility Engine uses the entirely sub-symbolic formalism of ANNs, the Strategy Engine employs a mixture of both principles since it combines logical rules with evolutionary methods and instance-based learning. The Social Behaviour Engine, finally, uses an explicitly symbolic decision-rule and an inference mechanism for re-organizing knowledge about preference structures that is similar to symbolic/numerical methods for *probabilistic reasoning*, e.g. Bayesian learning.

Amongst the most important characteristics of the prototype is that it employs very little knowledge of the system environment: agents only need to know about their action alternatives and those of others and they need to be informed about the payoffs they receive and about the currently performed joint actions. At some points we also assumed that agents were aware of the fact that their payoffs depend on global resource access, but we believe that in a resource-load balancing scenario this is not an overtly restrictive assumption. Apart from this assumption, the designed components can easily be adapted to any other n-player game.
For the specific class of resource-load balancing games that we account for, preliminary results proved that agents can learn the individual concepts at least in principle. Whether the learners' performance is sufficient and whether we can successfully combine the learning components to ensure the emergence of coordinated behaviour remains to be shown in the next chapter.

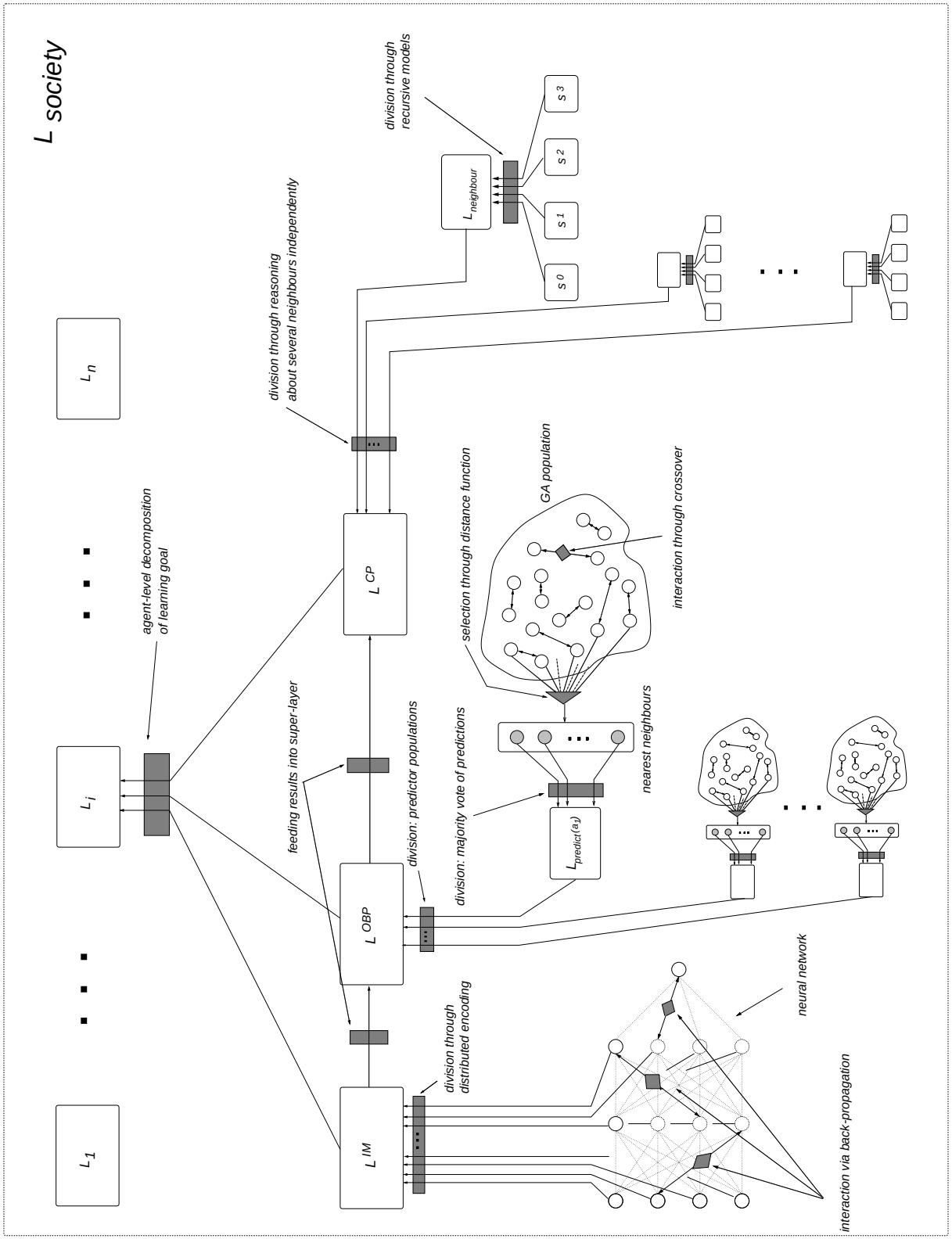


Figure 5.16: Layered learning view of the LAYLA prototype. Each agent "consists-of" three learning layers for different learning tasks; learning results are propagated through these in a hierarchical way. L^{IM} can be decomposed further at a "learning cell" level: the ANN units interact via back-propagation. L^{OBP} combines the results of 2^k GA populations; individuals in these populations interact through crossover, and they compete for selection in the nearest-neighbour algorithm. L^{CP} is decomposed into several neighbour analyses, each of which is built of the recursive belief models. The society-level layer $L_{society}$ consisting of n agent learners is only sketched (by a dashed frame) as it is not implemented explicitly.

Chapter 6

Evaluation

Each new program that is built is an experiment. It poses a question to nature, and its behaviour offers clues to an answer.

— A. Newell and H. A. Simon,
Computer Science as Empirical Enquiry

So far, we have explained the general class of problems for which our approach was designed, we presented an abstract architecture that is supposed to tackle these, and we have provided an extensive treatment of a possible concrete agent design that embodies the properties of the general framework.
Yet these undertakings can only be justified by proving their practical adequacy, be it through exact mathematical proof methods or by empirical validation. As mentioned before, we choose to pursue the latter alternative, mainly because an exhaustive theoretical analysis of the employed learning algorithms lies beyond the scope of this work. A second reason is that randomisation in action choice and the close interplay between action and learning imply that only asymptotic bounds for convergence of the learning algorithms can be provided (as is common practice in the field of machine learning). Furthermore, rather restrictive assumptions have to be made to provide precise theoretical results, as will be illustrated in the paragraph on the theoretical derivation of the compromise factor. Empirical testing, on the other hand, shows whether, how and why the algorithms work *in practice*, how fast they converge to effective behaviour patterns, how these relate to optimal solutions and whether they remain stable.

The following sections are organised as follows: First, an overview of the evaluation procedure is provided together with a discussion of complexity issues that impose severe restrictions on the scope of covered problem sizes. The subsequent section gives an account of extensive tests[1] divided into three parts — fine-tuning tests, conducted to optimise the performance of the integrated architecture, "bounded rationality" tests, by which the effects of limited reasoning capabilities and noise on agents' performance are assessed and "scaling tests" that analyse the performance of the architecture for larger problem sizes. Finally, we round up with a critical review of the evaluation and a summary of its main results.

[1] All of the reported simulations were carried out using the gsim software package, a JAVA implementation of the LAYLA system that we have developed. It can be downloaded (together with online documentation) from `http://www.ags.uni-sb.de/~rovatsos/layla`.

6.1 Measuring system performance

Before embarking on the presentation of empirical results, a precise description of how these tests will be conducted and what performance measures will be applied has to be presented, together with an outline of the underlying assumptions.
In the case of testing LAYLA societies, we additionally have to explain why most of the conducted tests were performed for two-player games, and this will be the issue that we look at first.

6.1.1 Complexity issues

It is a natural consequence of excluding the possibility of communication and the availability of prior knowledge that all computational effort is devoted to extracting as much information as possible from very scarce input data. In our architecture, agents use a single percept consisting of the current joint action and the associated private payoff to update the neural network of the Utility Engine, to train the genetic algorithms of the Strategy Engine and to predict future opponent actions accordingly. Moreover, this percept is used to update the Social Behaviour Engine POM models, the agent's Gain Model and to generate socially feasible action choices.
Unfortunately, this involves very costly computations in terms of both time and space.
Consider a game with n players and k resources which is played for d rounds. Firstly, the Utility Engine trains the net with the new sample, which can be done in $O(w)$, where w is the number of network weights. According to our choices concerning network structure (cf. Section 5.2.2), the networks[2] agents employ will have $kn \cdot 2n + 2n \cdot 2n + 2n$ weights, so that update (and predict) operations can be done in $O(n^2k)$.
Assuming that the sample sets of the UE are not constrained in size, and that, in every round, the agent updates the network with *all* stored samples (one update iteration for each sample), the total time complexity of the UE after d rounds is $O(d^2n^2k)$, since a total of

$$\sum_{i=1}^{d} i = \frac{d(d+1)}{2}$$

update operations will be performed during the game.
Under the assumption that the GAs in the Strategy Engine use a number of samples in $O(nk)$ (so that the complexity of the UE is not exceeded), and that this layer uses all past examples in each round in the same fashion as the UE, we obtain an overall asymptotic running time of $O(d^2nk)$ for the Strategy Engine[3].
As concerns the Social Behaviour Engine, we assume that the number of ranks is sufficient to capture binary action matrices, i.e. $|R| = |A|^2 = 2^k \cdot 2^k = 2^{2k}$, and that the neighbourhood size is in $O(n)$ (the larger the society, the more neighbours should be reasoned about).
Further, we will assume that the Gain Model (with respect to one neighbour) can be updated in time $O(2^kn^2k)$: retrieving the $maxgain$ and $mingain$ values involves determining

[2]The input layer of each net has size kn, the two hidden layers have size $2n$ and the output layer consists of a single unit. Each unit of each layer is connected to every unit in the subsequent layer.

[3]The GA update runs in time linear in the population size (i.e. $O(nk)$), assuming that crossover and mutation can be done in constant time. The computation of the fitness function takes time linear in the number of past examples and population sizes, so that, once more, this complexity is quadratic in d. Recalling that in every round only one of the 2^k GA populations is updated and used for nearest-neighbour-prediction (which, again, can be achieved in time linear in the number of nearest neighbours) this yields a total complexity of $O(d^2nk + nk + nk) = O(d^2nk)$.

the minimal payoff under the peer's actions, so that actually 2^k payoff values have to be predicted using the UE, resulting in a total complexity of $O(2^k n^2 k)$ (payoff prediction, just like ANN update takes $O(w) = O(n^2 k)$).
Also, at least one add-operation has to be performed in each round and such an operation (cf. Definition 5.3) takes $O(|R|^2) = O(2^{4k})$ (as do the retrieve-operations that are necessary to determine the $P(s(k', l') > s(k, l))$). In the worst case, UPDATE_POMPROFILED will have to be performed in every round which involves $O(|A|) = O(2^k)$ such update and retrieve operations, yielding a total of $O(2^{4k} \cdot 2^{2k}) = O(2^{6k})$ for one round and one neighbour.
Flattening each POM is linear in the number of the model's entries, i.e. $O(2^{2k})$; the same holds for determining the expected gain $g_i(a_i)$, so that for a total of 2^k actions a_i the construction of $\mathcal{L}_j$ takes $O(2^k \cdot 2^{2k}) = O(2^{4k})$. Thus, the total running time of SOCDECIDENEXT is $O(2^{6k}) + O(2^{4k}) = O(2^{6k})$ for one neighbour, so that $O(d \cdot |\mathcal{N}_i| \cdot 2^{6k}) = O(dn2^{6k})$ is the running time of the SBE in d rounds (since $|\mathcal{N}_i| = O(n)$).
Thus, a LAYLA system consisting of n agents has an asymptotic time complexity of

$$O(n \cdot d^2 n^2 k) + O(n \cdot d^2 n k) + O(n \cdot dn2^{6k})$$

if we take the running times of the UE, the SE and the SBE together, which is exponential in the number k of resources, cubic in the number n of players and quadratic in the game duration d.

It is therefore not surprising that a dramatic increase in simulation times can be observed for different problem sizes. A JAVA implementation of the LAYLA prototype took typically around 20 minutes for a 1000-round long two-player simulation, while over twenty hours were necessary for a ten-player five-resource game (benchmarks measured on a Sun Sparc Ultra-2 workstation).
Even worse, we can expect larger games to be much more difficult to solve, so that, in practice, a greater number of rounds will have to be played until they can be expected to converge. Since our framework involves a large amount of randomisation in action choice and since the learning processes very much depend on the actual history of the game so far, multiple testing of identical settings would be necessary to obtain reliable values for learning and action performance. This makes a thorough testing of large-scale interactions even more intractable.

We confront this problem by proceeding as follows: extensive testing for a multitude of parameter choices is conducted for a two-player two-resource game. To ensure that our results are reliable, identical simulations are conducted repeatedly (typically, each test will be repeated 100 times). Convergence of the algorithms is validated by using game durations much longer than the "complexity" of the problem: such a two-player game has only 16 joint action combinations ($\mathcal{A} = |A|^2 = 2^k \cdot 2^k = 2^2 \cdot 2^2 = 16$) and payoff functions and best-response predictions can be learned with very high accuracy within a small number of rounds (cf. the analyses in Section 5.2.3 and Section 5.3.3). Therefore, by running simulations of 1000 rounds we can rule out that agent behaviour might converge after more rounds (even if this were the case, it would point at very low performance). On the other hand, if stable situations are established much earlier, such long simulation times are useful to ensure that equilibria are not left again.
After determining optimal configurations for these games, we will present the results of singular simulations for larger five-player, ten-player and fifty-player examples. In contrast to the two-player games within which agent performance is verified rigorously,

these scaling experiments can only serve as illustrations of how effective the framework *can* be. Most of the parameter choices for these larger problem sizes will be derived (very vaguely) from the two-player example and can by no means be claimed to be optimal. Due to the complexity problems just mentioned, we will be forced to assume that – apart from the compromise factor γ, for the choice of which we will use a theoretical argument – most of the parameter choices simply carry over to larger problems, which is, admittedly, rather undesirable.
However, the surprisingly good performance of the algorithms as we use them is a remedy, because even if parameter choices were sub-optimal, this would prove that there is room for even better performance than what was observed.

6.1.2 Assumptions

In this section, we present the assumptions that will hold throughout the experiments to follow. An assumption that has already been made in the introductory chapter is that we only account for repeated multiple-access resource-balancing games, as introduced in Chapter 3. Once more the fact should be underlined that this specific class of games are *cooperative* in the sense that there exist pareto-optimal joint action combinations which yield higher payoffs than the Nash equilibrium (the situation in which all agents access all resources). In strictly non-cooperative games, quite naturally, we cannot expect cooperation to emerge under any algorithm.
Secondly, the game parameters (n, k, v, T and c) will be fixed during the simulations, i.e. we do not analyse adaptivity in the sense of environmental changes. However, it can be expected that the algorithms should be able to adapt to new situations, at least if a meta-reasoning component tracked current performance (Section 7.2 discusses the necessity of such a component in more detail).
Noise functions will not be applied by the Simulation Engine during the first series of experiments, so that the the best-case performance can be determined independent of perturbations in action execution, payoff perception and action perception. In Section 6.2.2 we will lift this assumption and analyse the effects of noise.
Also, maximal computational effort will be devoted by every agent to learning in each round. This means that all past action-payoff samples will be stored (i.e. agents use unconstrained sample sets for the UE and SE), and that all models of the SBE will be updated after every round and used for social decision-making. This assumption will be lifted in the section on bounded rationality (Section 6.2.2), so that the effects of constraining reasoning capabilities can be examined.
Finally, it should be pointed out that we will exclusively use *homogeneous* populations. This means that all agents taking part in a particular simulation will be identical in their internal structure, that they will use precisely the same reasoning and learning mechanisms. This assumption serves as a means to discover the basic properties of LAYLA reasoning, which is obviously simpler if all agents employ the same parameters.

Next, the goals of conducting the tests are outlined and how the achievement of these will be measured.

6.1.3 Performance measures

What do we expect of a LAYLA system? The ultimate reason for having such an agent architecture is, recalling the discussion of Chapter 1, to prove that effective, coordinated

behaviour in repeated games can be learned without prior knowledge and explicit communication. The top-level learning task **T** of the society specified this goal operationally in Section 3.2.1 and this definition was followed by introducing the inverse of the distance between the optimal-coalition payoff vector and the current payoff vector as a performance measure **P**. We will adhere to **P** as the central performance measure for LAYLA societies, but we also introduce some additional measures that provide information about certain interesting aspects of agents' behaviour.

The *minimal* performance that we require agents to exhibit is that of the equilibrium state, i.e. the situation in which every agent accesses all k resources. The payoff that is distributed to agents under this action combination is

$$u_{greedy} = k \cdot \left(\frac{v}{nT} - c \right)$$

and is, of course, equal for every agent (we refer to this quantity as the *greedy payoff* henceforth). If this payoff is achieved by all agents, this implies that the best-response strategy has been successfully learned. Thus, after each round, the cumulative reward obtained by an agent until round t can be compared to the cumulative greedy payoff $t \cdot u_{greedy}$ to check whether the agent fulfills this criterion.

Although learning the equilibrium strategy is non-trivial by itself, it can only be seen as a first step towards coordinated behaviour – truly intelligent agents should do better than that. In the optimal case, they should form a maximal coalition in which each agent plays the *fair* strategy by which it accesses $\frac{k}{n}$ resources (cf. Section 3.2.2). Then, the respective (expected) *fair payoff* each of the agents would receive per round computes as

$$u_{fair} = \frac{k}{n} \cdot \left(\frac{v}{T} - c \right) \quad .$$

Thus, the cumulative fair payoff $t \cdot u_{fair}$ provides an upper benchmark for system performance in the same way as the greedy cumulative payoff function represents a lower bound for desirable performance.

Using the greedy, fair and actually achieved cumulative payoff functions, simple quantities can be defined that measure the *optimality* of system behaviour, the *evolution* of cooperation within the society, the *convergence* and *stability* of behaviour patterns and *equity* among agents.

Optimality, the criterion that is fulfilled to the same degree as the optimal-coalition payoffs are realised, can be measured by using **P**, or, alternatively, by comparing the gradient of cumulative agent payoffs to that of the fair cumulative function.

Cooperation among agents is evolving whenever the gradient of cumulative agent rewards is higher than that of the greedy function and *rising* so that this measure reflects whether global performance is improving beyond the level of mutual exploitation.

Agent behaviour is said to converge to a certain value if the gradient of some agent's cumulative payoffs converges to some (upper or lower) bound. It is said to be stable if the gradient of the agent's cumulative reward function is some constant quantity (within a certain percentage ϵ of its previous value) after a certain game iteration (in that case we call this round the *convergence-yielding round*).

Equity, finally, reflects the ability of an agent to "avoid doing much worse" than its adversaries: whatever degree of cooperation or blind egoism is established, no agent should allow others to exploit it. Showing that agents have the capability of defending themselves against opponents' attempts to exploit them adds a further dimension to the adequacy of the approach, namely that LAYLA agents exhibit *competitive performance* in iterated

games and is also in concordance with the intuition of individual rationality: only agents that can make at least as much of their situation as other agents can be deemed veritable utility maximisers.
To measure this aspect of agent performance it suffices to compare the difference between the cumulative payoffs of individuals participating in the same simulation. To this end, we will usually compute the mean cumulative payoff $\mu(agents)$, i.e. the average cumulative reward of all players, and the standard deviation between cumulative rewards $\sigma(agents)$ that provides a measure for how much each player's performance deviates from $\mu(agents)$ on average.
All of the above measurements will quite often be conducted for a certain number s of repeated, identical simulations, and in that case we use $\mu(q, s)$ and $\sigma(q, s)$ to denote the mean value and standard deviation of the respective quantity q.
It will often be useful, for example to measure the average convergence-yielding round number $\mu(conv, s)$ between s simulations and the standard deviation $\sigma(conv, s)$ thereof.

This overview of applied benchmarks and performance measures completes the introduction to our testing procedure. The following paragraphs give a detailed description of the actual tests and an analysis of the main results.

6.2 Experimental results

The extensive testing of the system falls into three phases: first, fine-tuning choices of the system parameters are made and the performance of agents under the optimal choices are analysed. Then, issues of robustness, bounded rationality and reduced computational capacities are pondered on. Finally, we test the architecture for larger games.

6.2.1 Fine-tuning LAYLA parameters

Difficulties in testing the performance of the Social Behaviour Engine lead to testing it only for a very stylised IPD example in Section 5.4.3 and to postpone the choice appropriate values for the optimism parameter λ, the POM update factor δ and the compromise factor γ. Furthermore, parameters that determine the integrated algorithm's mode of operation, such as the exploration margin ζ and the variance threshold ϑ have still to be determined. These two issues are dealt with in this section and system performance is evaluated for the resulting choices.

We start by describing the initial system parameters, and subsequently discuss the effects of choosing alternatives for these.

Initial simulation settings

All following experiments are performed on the basis of a two-player two-resource game, where $v = 20$, $T = 4$ and $c = 1$ and $d = 1000$ is the total number of rounds played in each simulation. These settings imply that $u_{greedy} = 3$ (which occurs if both agents chose action 3 ($\beta(3) = (1, 1)$) and $u_{fair} = 4$. The optimal payoff is achieved whenever the two agents play one of the joint actions $(1, 2)$ or $(2, 1)$, to that they access disjoint, non-empty subsets of $\mathcal{R} = \{r_1, r_2\}$.
To speed up learning, unconstrained sample sets are chosen both for the UE and the SE. The optimal settings for the UE and SE are adapted from the fine-tuning choices of G_I

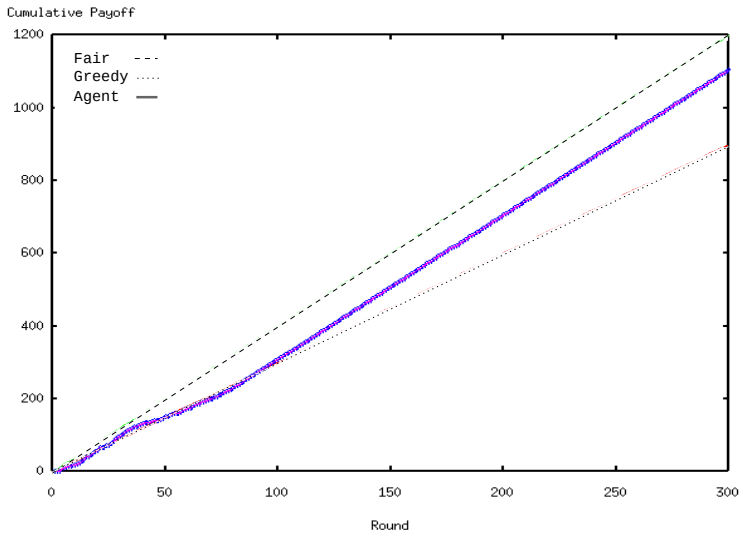


Figure 6.1: Plot of a two-player simulation (the first 300 rounds are shown). The curves between the greedy and fair cumulative payoff lines denote the (almost identical) cumulative agent payoffs.

in Chapter 5, since this game is identical to the one used here. Guided by the optimal UE performance (cf. Figure 5.5), the exploration margin ζ is set to 0.03, so that sufficient exploration of the payoff function can be completed in less than 100 rounds.
As concerns the SBE settings, initial values are chosen rather arbitrarily: the variance threshold is set to $\vartheta = 0.01$ to ensure that peers consider their behaviour to be profiled quite early and we choose $\gamma = 0.5$, $\delta = 0.5$ and $\lambda = 0.5$. Quite naturally, each agent has one neighbour (its opponent) and POMs consist of $4^2 = 16$ ranks to cover all possible matrix orderings in the joint action space. Finally, the employed rank-flattening vectors have linearly decreasing entries starting from one, i.e.

$$\vec{c} = \left(1, 1 - \frac{1}{16}, 1 - \frac{2}{16}, \ldots, 1 - \frac{15}{16}\right).$$

These choices already yield astonishing results for a sample simulation run, in which agents converge[4] to the *optimal* behaviour after 110 rounds (within a tolerance interval of $\epsilon = 5\%$), thus realising as much as 92.9% of the maximally possible fair cumulative payoff after 300 rounds. The low standard deviation of $\sigma(agents) = 10.0$ also shows that the two agents achieve comparable overall performance, so that none of them is really exploited. Plots of the cumulative payoff functions compared to fair and greedy payoffs are shown in Figure 6.1. Starting from these initial settings, we present fine-tuning choices and their effects on system performance in the following paragraphs.

Determining the compromise factor

The compromise factor γ determines which actions will be included in the construction of $\mathcal{L}_j$-sets (the larger γ is chosen, the more alternatives to the best-response strategy will be considered socially feasible). If we look at the way in which γ is used in practice, it becomes obvious that in the $\mathcal{L}_j$-construction criterion (cf. Section 5.4.1)

$$a_i \in \mathcal{L}_j \iff m_i(a_i) \geq \max_{a_i' \in A_i} m_i(a_i') - \gamma \cdot g_i(a_i)$$

[4]Whenever measuring convergence, we refer to the agent that converged *last* – in this example, the two agents actually converged in rounds 80/110.

there is no direct relationship between the quantities of the individual action-value function m_i and the expected gain $g_i(a_i)$, since they are derived from different models (the SE best-response GAs and the SBE opponent models). In an attempt to get a complete picture of how the choice of γ affects the SBE decision-making function, we will first present a theoretical argument to derive appropriate γ-values, which we then validate by experimental results.

In the resource-load balancing game, gain models and action-values can be determined in terms of the payoff received for each resource and are independent of the opponent that is being reasoned about (under the assumption of homogeneous populations).
In order to obtain the precise value of the gain of i with respect to j for one resource $r \in \mathcal{R}$, we firstly observe that all individual actions of i and j fall into two disjoint categories: actions by which r is accessed (A^+) and actions by which the agent refrains from access to r (A^-).
Secondly, the maximal and minimal gain values of i for any joint action tuple (a_i, a_j) are realised when *none* of the remaining agents $\mathcal{P} - \{i, j\}$ accesses r or *all* remaining agents access r, respectively. To see this, we compare the quantity

$$D[r] = u_i(a_i, a_j, a_{-\{i,j\}})[r] - \min_{a'_j \in A_j} u_i(a_i, a'_j, a_{-\{i,j\}})[r]$$

for all a_i, a_j and $a_{-\{i,j\}}$ ($u_i(\ldots)[r]$ is i's payoff for resource r). Clearly this quantity will always yields zero if $a_i \in A^-$ (if the agent does not access r it obtains no payoff for it). Furthermore, it is only positive if $a_j \in A^-$ because the minimal payoff value for resource r is always realised when $a_j \in A^+$. Finally, u_i depends on the number p^+ of agents in $\mathcal{P} - \{i, j\}$ who access r. As a simplified form of D, we thus obtain

$$D[r] = \begin{cases} 0 & \text{if } a_i \in A^- \text{ or } a_j \in A^+ \\ \left(\frac{v}{(p^+ + 1)T} - c\right) - \left(\frac{v}{(p^+ + 2)T} - c\right) & \text{else} \end{cases}$$

where the denominators in the non-zero case depend on whether $a_j \in A^+$ or not.
Simple calculus suffices to validate that $D[r]$ is maximal for $p^+ = 0$ and minimal for $p^+ = |\mathcal{P}| - 2$, so that we can define maximal and minimal gains in terms of r as follows for $a_i \in A^+$ and $a_j \in A^-$ (in all other cases, they will be zero):

$$\begin{array}{rcll} maxgain_{a_i}(a_j)[r] & = & \left(\frac{v}{T} - c\right) - \left(\frac{v}{2T} - c\right) & = \frac{v}{2T} \\ mingain_{a_i}(a_j)[r] & = & \left(\frac{v}{(n-1)T} - c\right) - \left(\frac{v}{nT} - c\right) & = \frac{v}{Tn(n-1)} \end{array}$$

Now if i accesses k_i resources ($k_i \leq k$) by playing a_i, and q of these resources are *not* accessed by j, this yields

$$gain_{k_i}(q) = \frac{q \cdot \left(\lambda \cdot \frac{v}{2T} + (1 - \lambda)\frac{v}{Tn(n-1)}\right)}{k_i \cdot \left(\frac{v}{T} - c\right) - k_i \cdot \left(\frac{v}{nT} - c\right)} \tag{6.1}$$

as the overall gain value of any such binary action tuple for i by applying the constructions of Definition 5.1.
Having determined precise gain values, we still need to analyse the quantities of m_i to simulate the construction of $\mathcal{L}_j$-sets.

As before, we can describe m_i in terms of how many resources are accessed, if we make some important assumption: we consider that the best-response strategy has been

learned *perfectly* by the Strategy Engine and that it has a precise picture of the payoff function, i.e. that $\pi_i = u_i$. We also assume that all agents are currently reacting according to the best-response strategy, meaning that whatever i plays, all agents choose the greedy action choice of accessing all resources simultaneously[5].
The action value $m_i(k_i)$ for any action of i by which k_i resources are accessed can then be computed as

$$m_i(k_i) = k_i \cdot M \quad , \quad M = \frac{\left(\frac{v}{nT} - c\right)}{\sum_{0 \leq k_i \leq k} \binom{k}{k_i} \cdot k \cdot \left(\frac{v}{nT} - c\right)} = \frac{1}{\sum_{0 \leq k_i \leq k} \binom{k}{k_i}} \tag{6.2}$$

given that it is based on the above worst-case argument (M is simply a normalizing constant, i.e. the sum of all $m_i(k_i)$).

Now we are in the state where we can derive upper and lower bounds for γ to ensure that resources are distributed in an optimal way, i.e. that every agent is willing to access $\lceil \frac{k}{n} \rceil$ resources. This can be done by requiring that

$$\begin{array}{rcl} m_i\left(\left\lceil \frac{k}{n} \right\rceil\right) & \geq & m_i(k) - \gamma \cdot g_i\left(\left\lceil \frac{k}{n} \right\rceil\right) \\ m_i\left(\left\lceil \frac{k}{n} \right\rceil - 1\right) & < & m_i(k) - \gamma \cdot g_i\left(\left\lceil \frac{k}{n} \right\rceil - 1\right) \end{array} \tag{6.3}$$

hold, so that i considers precisely those actions as candidates for inclusion into L_j which implement an optimal resource distribution (and does not "put up with" any less exploitative actions).
To solve these inequalities for γ, we need to make one more assumption to obtain the expected gain $g_i(k_i)$ of accessing k_i resources, namely that any opponent j is *exactly as cooperative as* i itself (which can safely be assumed, since we are talking about homogeneous populations).
This means that

$$\begin{array}{rcl} g_i(k_i) & = & \| \sum_{k \in A_j} gain_{k_i}(q) \cdot P(j \text{ does not access } q \text{ of the } k_i \text{ resources accessed by } i) \| \\ & = & gain_{k_i}(k - k_i) \end{array}$$

because j accesses (with probability 1) exactly as many resources as j does, so that $k - k_i$ resources are accessed by j alone.
Replacing the g_i-values accordingly (which involves using Equations (6.l) and (6.2)), the inequalities in (6.3) can finally be solved for γ to yield

$$\frac{2(n-1)^3}{\lambda n^2(n-1) + (1-\lambda)2n} \cdot M \leq \gamma < \frac{2(kn + k - 1)(n-1)^2}{\lambda n^2(n-1) + (1-\lambda)2n} \cdot M \tag{6.4}$$

so that the suggested appropriate values for γ only depend on the problem sizes k and n and on the optimism parameter λ.
For our two-player two-resource example, the corresponding values are 0.25 and 0.625, so that γ should lie somewhere in between these values if optimal cooperation is to emerge.

We verify the adequacy of this lengthy mathematical derivation of the compromise factor by conducting a series of tests for $\gamma \in \{0.1, 0.2, \ldots, 1.0\}$, the results of which are quite surprising.
It turns out (cf. the plots in Figure 6.2) that for compromise factors lower than 0.5 the

[5]This assumption serves to simulate the situation "just before" social reasoning can begin.

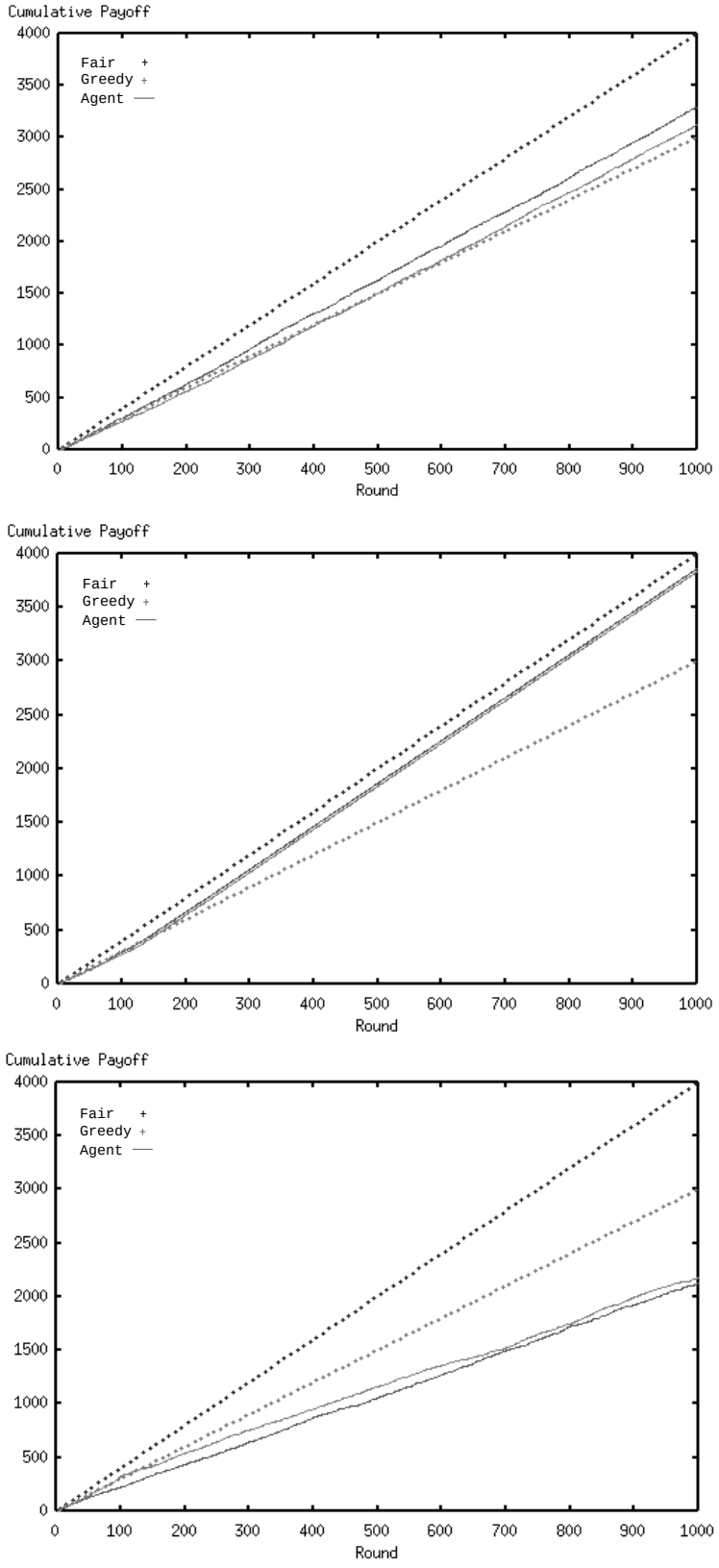


Figure 6.2: Plots for two-agent simulations and different γ-values. For $\gamma = 0.4$ (topmost plot), agents achieve an average payoff just above the greedy (dotted) curve (80.125% of the fair cumulative payoff). The graph in the middle shows agent behaviour for $\gamma = 0.6$ (agents converge to optimum and achieve a final mean payoff of 96.13% of the fair cumulative payoff), and in the third graph we can see that "overtly compromiseful attitudes" ($\gamma = 0.9$) lead to highly sub-optimal performance (only 71.65% of the greedy payoff are realised on average).

previously shown convergence never occurs, while agents do "better than greedy" with increasing performance for increasing $\gamma \in [0; 0.5]$. This already provides us with cumulative payoffs that are much higher than what agents devoid of social reasoning would achieve. For compromise factors of 0.5 and 0.6 agent performance converges to the optimal, fair payoff vector in around 80% of the simulations. This can only mean that since the assumptions made in our theoretical argument always implied that all reasoning components were functioning *perfectly*, compromise factors should in practice be close to the upper bound of the γ range, so that minor errors (e.g. mixed strategies instead of the pure ones assumed before) can be overlooked.
The bounds determined for γ prove more important still, when we consider the results of those simulations in which γ exceeded these. In that case, agent performance was very poor, because agents they would either settle on highly sub-optimal action combinations (e.g. a mixture of $(1,1)$, $(1,0)$ and $(0,1)$) or – even worse with respect to equity – not coordinate their activities at all, so that one of them gets exploited by the other invariably.

This smooth transition from "individually rational" via "socially rational" to "overtly compromiseful" behaviour effected by various choices of γ explains why this parameter was called "compromise factor" in the first place. It illustrates that the choice of this factor is crucial to the system behaviour, and this is also why this lengthy discussion has been devoted to it.

Further SBE parameters: POM update factors and optimism parameters

The update factor δ determines to which degree recursive belief models are altered on new, incoming information about action choices. Guided by the intuition that the way in which δ-choices influence SBE decisions is closely related to the choice of γ (since the belief models are used to compute the $P(a_j = k)$-probabilities in the "expected gain"-computation, cf. Equation (5.7)), we first test the whole range of $\gamma \in \{0.1, 0.2, \ldots, 1.0\}$ against values of δ chosen from $\{0.1, 0.25, 0.75, 1.0\}$, but, quite surprisingly, almost no difference can be observed between these four δ values. For each of them, the effects of γ choice are identical to those described in the previous paragraph, i.e. convergence to optimal payoffs can only be achieved by choosing $\gamma \in \{0.5, 0.6\}$.
To verify this observation, we conduct 100 simulations with $\gamma = 0.6$ for the extreme δ-values of 1.0 and 0.1, and observe that for $\delta = 1.0$ 80% of the runs show convergence ($\mu(conv, 80) = 286.7$ and a $\sigma(conv, 80) = 272.4$) while agents converge to optimal behaviour 79% of the time for $\delta = 0.1$ ($\mu(conv, 79) = 318$ and $\sigma(conv, 79) = 249.5$). Final cumulative payoff as well as its standard deviation (between individual runs) are almost identical ($\delta = 1.0$: $\mu(agents, 100) = 3631.67/\sigma(\mu(agents), 100) = 322.2$, $\delta = 0.1$: $\mu(agents, 100) = 3567.53/\sigma(\mu(agents), 100) = 304.7$).
Since no relation between agent performance and δ-choice can be discerned, $\delta = 1.0$ will be applied henceforth.

The previous paragraph showed that the choice of the compromise factors depends on the optimism parameter λ, but the issue of which γ/λ-pairs result in optimal performance has not been resolved yet. We therefore conduct tests for different value pairs ($\lambda \in \{0.0, 0.1, 0.2, \ldots, 1.0\}, \gamma = 0.6$[6]) and examine average optimality-yielding rounds as well as standard deviation between them.

[6]For the special case of the two-player two-resource game, $\gamma = 0.6$ can be chosen for any λ, because the bounds for γ are independent of the λ choice.

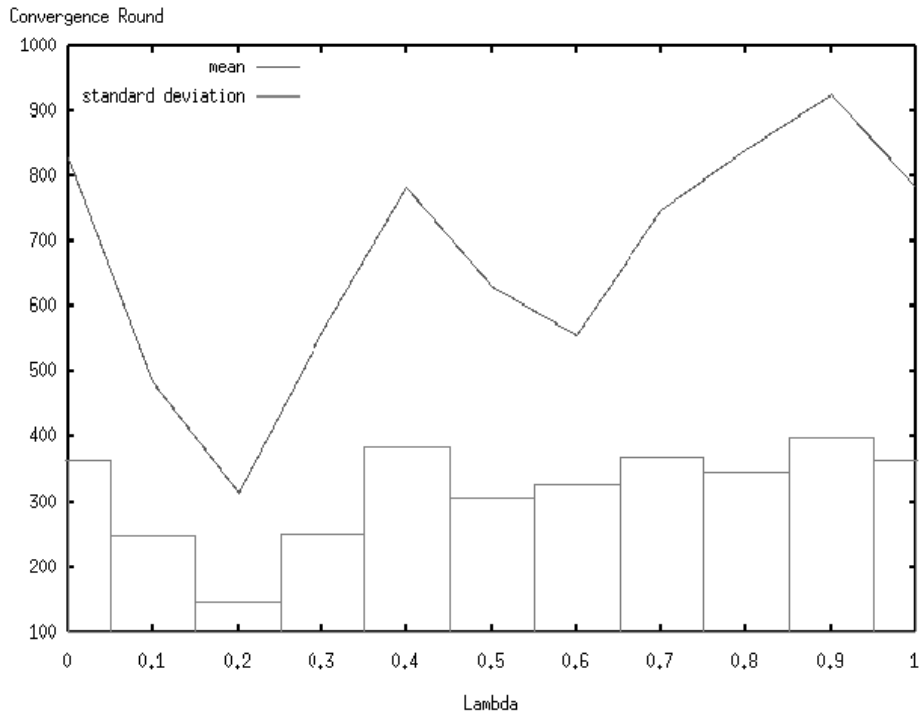


Figure 6.3: Average convergence round (line) and standard deviation (boxes) for various λ choices. For the simulations that did not converge, the "convergence-yielding" round was taken to be 1000.

As can be seen from Figure 6.3, it seems most reasonable to be "rather pessimistic" in determining gain values ($\lambda = 0.2$ is optimal both in average convergence time ($\mu(conv, 100) = 315.0$) as in standard deviation ($\sigma(conv, 100) = 145.8$).
However, we refrain from attempting to provide a general explanation for this observation, because it obviously depends on specific characteristics of the game in question and shall simply use $\lambda = 0.2$ for the remaining experiments.

Global parameters: exploration threshold, "profiled" threshold, randomisation

We now turn to an analysis of the effects of various choices for the exploration threshold ζ (which determines the degree of accuracy that agents expect of UE payoff prediction before moving on to purely strategic/social action choice) and variance threshold ϑ that is used to determine whether opponents are behaving in a "profiled" way or not (cf. Section 5.5).
Furthermore, we explore the possibility of using "purified" versions of m_i and l_i instead of mixed strategies; such "purified" probability vectors are simply obtained by always choosing the action with maximal m_i/l_i with probability 1 and all other actions with probability 0.

For ζ and ϑ, the following sets of parameter values are cross-tested:

$$\zeta \in \{1.0, 0.5, 0.25, 0.1, 0.05, 0.03, 0.01, 0.001, 0.0001, 0.0\}$$

and

$$\vartheta \in \{1.0, 0.1, 0.01, 0.001, 0.0\} \quad .$$

Each of the resulting parameter 50 pairs is applied in 10 repeated simulations. The following observations were made:

1. For $\vartheta \in \{1.0, 0.1\}$, no convergence to optimal behaviour can be achieved. Expecting the opponent to exhibit a very strong preference for certain actions to be considered "rational" is obviously harmful since no "risk" is taken to initiate cooperation.

2. We obtain almost[7] the same results for $\zeta \in \{0.001, 0.0001, 0.0\}$. In this case the UE finishes exploration to late (in the case of $\zeta = 0.0$ exploration actually carries on forever) so that social reasoning cannot begin within less than 1000 rounds.

3. Performance of convergent agents increases with decreasing ϑ and increasing ζ (the absolute optimum is reached for $\vartheta = 0.0/\zeta = 0.25$ with $\mu(conv, 10) = 184.7$ and $\sigma(conv, 10) = 39.1$).

4. The number of non-convergent simulations increases with decreasing ϑ: for $\vartheta = 0.01$, 7% of the agents do not converge to optimal behaviour, while this percentage is 25% for $\vartheta = 0.001$ and 28.6% for $\vartheta = 0.0$.

How can we interpret observations 3. and 4.? As concerns ζ, it is quite obvious that it pays to start considering strategic/social arguments in action choice as soon as possible, even if the picture one has of the payoff function is inaccurate.
The effects of various ϑ-choices seem somewhat more complex, in that there appears to be a trade-off between *ensuring* the emergence of optimal behaviour and achieving it *efficiently*, i.e. after a minimal number of rounds. In fact, though, this is only the effect of inaccurate UE payoff prediction – repeating the tests *without* making use of the UE (so that the real u_i is used instead of π_i) yields a rather astonishing result, namely that in 100 out of 100 simulations (!) convergence to optimal behaviour occurs ($\mu(conv, 100) = 157.8$, $\sigma(conv, 100) = 203.1$ for $\vartheta = 0.001$).
The interplay between UE performance and ϑ can thus be explained as follows: there is always some probability that the neural nets are learning "too slowly". If, in that case, ϑ is too small, the modelling agent will discern some preference structure in the opponent's actions quite early, but, since its own payoff model is inaccurate, it will draw wrong conclusions from the opponent's actions (e.g. construct wrong gain values) and be unable to converge to truly optimal behaviour.
We can draw a further conclusion from the fact that 100% of the simulations converged while the UE was not used, namely the agents' success hinges on the accuracy of the payoff function estimate – without it, none of the other components can ensure good results.

As regards the use of "purified" m_i/l_i vectors, we observe that agents fail to produce as good results as they did with mixed strategies – only 18 out of 100 simulations converge to optimal behaviour, and they converge more slowly and less reliably ($\mu(conv, 18) = 344.5$, $\sigma(conv, 18) = 546.1$). This only confirms the observation we made during preliminary experiments with the SBE (cf. Section 5.4.3): when agents use purified strategy vectors, there is zero possibility of choosing a non-greedy action, so that no agent "takes the risk" of initiating cooperative behaviour. The observation that there has to be some agent that starts deviating from greedy behaviour patterns should be emphasized as one of the central conclusions we draw from the empirical evaluation.

[7] For $\zeta = 0.001$ two of the 50 simulations converge, but only after 803/708 rounds.

6.2.2 Bounded rationality issues

Noisy data

Robustness is an issue that is very important in most real-world environments that provide noisy perception, sensory failure and action execution failure. Even if it is acceptable for system performance to deteriorate under increasing amounts of noise, we expect an "intelligent" system to *degrade gracefully* when faced with such difficulties.
To investigate the impact of noise on agents' performance in the LAYLA system, Gaussian noise functions are used as n^{ae}, n^{pp} and n^{ap} functions (cf. Section 4.4.4) with standard deviations taken from $\{0.0, 0.1, 0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 5.0, 7.5, 10.0\}$. For the discrete-valued functions n^{ae} and n^{ap}, perturbations are computed in a "toroidal" manner, such that, e.g. if in a game with $|A| = 16$ action $a_i = 13$ is perturbed by a quantity of 5.4 when executed, then we obtain[8] $n^{ae}(13) = 13 + \lfloor 5.4 \rfloor \bmod 16 = 2$.
First we test n^{ae}, n^{ap} and n^{pp} separately (10 simulations for each of the above noise levels, i.e. a total of 270 simulations) to determine the maximal noise levels, at which convergence to optimal behaviour can occur at all. The results are that convergence occurs up to $\sigma(n^{ae}) \leq 0.5$, $\sigma(n^{ap}) \leq 0.25$ and $\sigma(n^{pp}) \leq 1.0$. For all noise levels for which convergence occurs, we run another test suite in which each of the individual configurations is repeated 100 times, the results of which are shown in Table 6.1.

There are several conclusions we can draw from the experimental results. The first of these is that there are natural limitations to the robustness of the system towards noise, which seems quite natural. However, there are differences between the system's susceptibility to different kinds of noise: n^{ap} seems to have the severest effects on agent performance, n^{ae} is slightly easier to cope with, and both are considerably more hazardous than payoff noise n^{pp}.
We can explain this effect as follows: Action perception noise n^{ap} implies that both the UE and the SE obtain incorrect samples. Additionally, the UE samples are "more" erroneous than in the case of action execution noise, because inputs to the neural net may be altered in *every* position (*every* resource access of *every* agent).
Action execution noise n^{ae}, on the other hand, does not affect the *learning* mechanisms (except for exploration strategies). All it does is to turn action decisions into different results, so that *acting* is affected but not learning. So as long as there is not too much noise (that either makes it impossible to pursue some deliberation or confuses the opponent completely) agents may very well *learn* the right thing, even if they don't *do* the right thing.
Still, it is much more dangerous than payoff perception noise, to which the system is surprisingly tolerant. The experiments prove that agents may still converge to optimal behaviour even for $\sigma(n^{pp}) = 1.0$. In our game with $u_{fair} = 4.0$ and $u_{greedy} = 3.0$ this actually means that each of these values might be perceived as the other one *on the average*, so that convergence under these conditions is rather impressive. It turns out that this performance is mainly due to the robustness of the neural networks to noise. As is shown in Figure 6.4, their performance decay is rather small compared to the amount of noise in the learning data.

But the experimental results in fact hint at much more unexpected properties of per-

[8]Note that this is a very rigorous form of noise, because the most "extreme" strategies (those that are close to all_i and $none_i$ (cf. Section 3.2.2)) can easily be confused when executed or perceived.

$\sigma(n^{ae})$	0.0	0.1	0.25	0.5	0.75	1.0	2.0
$\mu(agents, 100)$	3521.6	3580.9	**3582.2**	3407.2	3089.7	n/a	n/a
$\sigma(\mu(agents), 100)$	336.2	337.7	290.9	283.5	**161.2**	n/a	n/a
$\#conv$	74	78	**86**	41	0	n/a	n/a
$\mu(conv, \#conv)$	290.9	**251.1**	313.3	545.8	-	n/a	n/a
$\sigma(conv, \#conv)$	208.0	**170.1**	205.5	245.9	-	n/a	n/a
$\mu(\sigma(agents), 100)$	29.3	**30.1**	30.3	66.2	166.9	n/a	n/a
$\sigma(n^{ap})$	0.0	0.1	0.25	0.5	0.75	1.0	2.0
$\mu(agents, 100)$	3521.6	**3618.6**	3547.2	2790.7	n/a	n/a	n/a
$\sigma(\mu(agents), 100)$	336.2	312.2	322.8	**129.9**	n/a	n/a	n/a
$\#conv$	74	**84**	72	0	n/a	n/a	n/a
$\mu(conv, \#conv)$	290.9	**252.0**	262.1	-	n/a	n/a	n/a
$\sigma(conv, \#conv)$	208.0	**176.7**	196.1	-	n/a	n/a	n/a
$\mu(\sigma(agents), 100)$	29.3	**24.2**	27.0	97.2	n/a	n/a	n/a
$\sigma(n^{pp})$	0.0	0.1	0.25	0.5	0.75	1.0	2.0
$\mu(agents, 100)$	3521.6	3462.6	**3590.9**	3489.1	3435.3	3487.1	3016.2
$\sigma(\mu(agents), 100)$	336.2	368.0	393.6	433.2	546.9	238.6	**146.5**
$\#conv$	74	68	**79**	76	75	60	0
$\mu(conv, \#conv)$	290.9	274.2	**227.0**	314.7	414.7	524.7	-
$\sigma(conv, \#conv)$	208.0	219.5	**196.9**	227.3	262.5	225.1	-
$\mu(\sigma(agents), 100)$	29.3	**20.9**	41.6	95.8	56.0	40.3	92.6

Table 6.1: Convergence analyses for (separately tested) kinds of noise. The average convergence round $\mu(conv, \#conv)$ and its standard deviation $\sigma(conv, \#conv)$ are computed for the number of convergent simulations $\#conv$. The other quantities always refer to a total of 100 identical simulations. Optimal values are shown in bold characters, "n/a" stands for "not assessed".

formance under noisy data. Apparently (cf. Table 6.1), the optimal values of all[9] measured quantities are achieved in simulations *with* positive noise, or, more specifically, for small noise levels of 0.1 and 0.25. For simulations with such noise functions (i) highest system performance (in average final payoff and reliability but also in proportion of convergent simulations) is achieved, (ii) equity between agents, as reflected by $\mu(\sigma(agents), 100)$ is maximal and (iii) convergence occurs earlier than in the other cases.
Maybe this is an effect similar to that reported by Thomas and Sycara (1998), namely that *heterogeneity* can add to the stability of a system. Here, this heterogeneity is induced, so to speak, by noise: because learning data is slightly distorted when received by the agents, they all have slightly different knowledge about payoff functions, opponent actions and opponent decisions. Thus, they are maybe less prone to oscillate around non-coordinated patterns of behaviour (just think of the Coordination Game) and converge to optimal behaviour because they make mistakes in acting and sensing. Although this is only one of many possible explanations for the observed effects, we believe that this issue deserves to be examined more thoroughly.

To investigate this issue further, we present one more series of experiments on noisy data, where all noise functions were applied *simultaneously* to the system. These exper-

[9]Except for $\sigma(\mu(agents), 100)$ which is minimal for high noise levels under which all agents behave greedily (in every simulation).

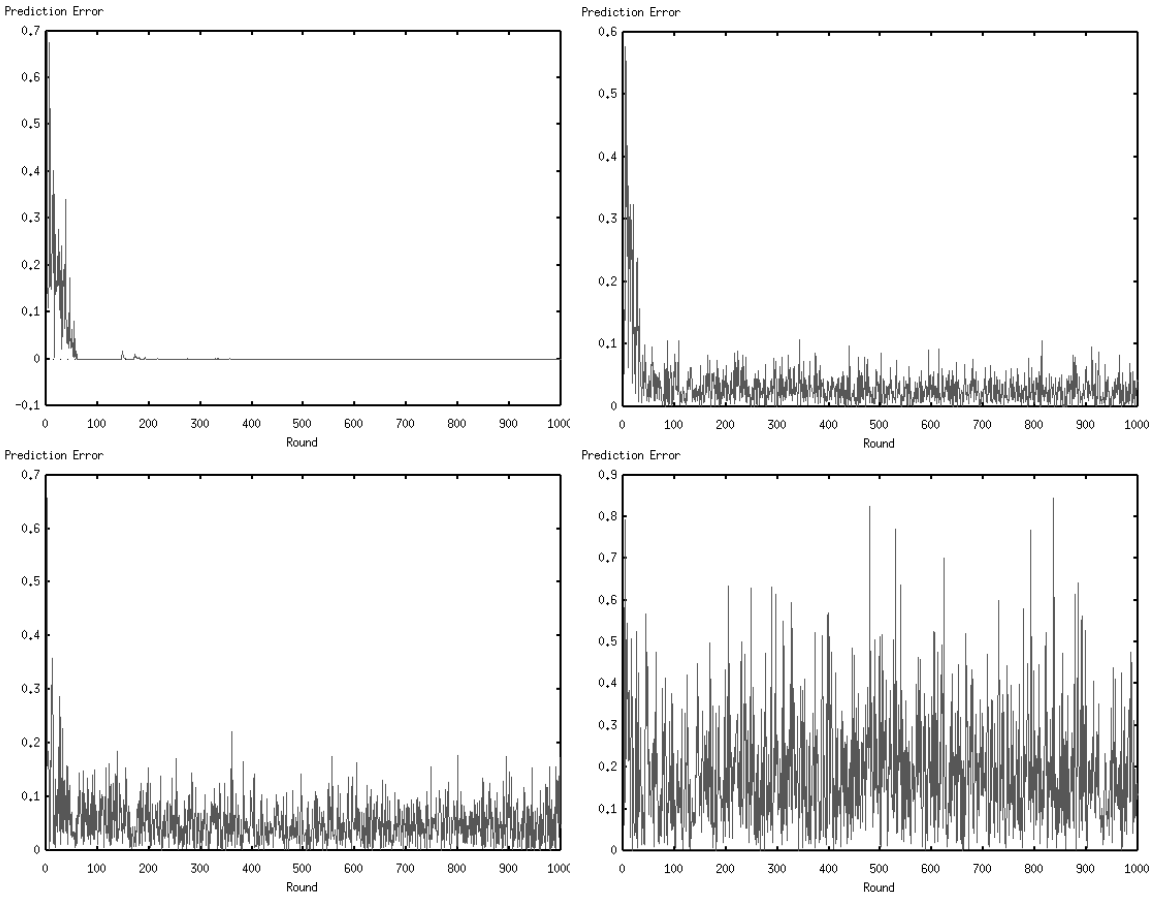


Figure 6.4: Neural network errors for various noise levels. The top left plot shows the performance with no payoff noise, the other three illustrate performance decay for noise with $\sigma(n^{pp}) = 0.25$ (top right), $\sigma(n^{pp}) = 1.0$ (bottom left) and $\sigma(n^{pp}) = 2.0$ (bottom right).

iments (cf. Table 6.2) show that the previously obtained results carry over to the new situation. Except for equity ($\mu(\sigma(agents), 100)$ is the only seriously sub-optimal value for $\sigma(noise) = 0.25$), performance is roughly comparable to that of the previous series of simulations. This shows that there is almost no interference between the different types of noise functions apart from the fact that action perception noise makes it impossible for agents to converge to u_{fair} for noise levels above 0.25.
We infer from this observation that the system does not get confused when exposed to more than one type of noisy data, i.e. that the "joint damage" done by the noise functions is not substantially greater than that of individual types of noise. This is another interesting property in our system that proves its suitability for worlds that provide imperfect information and sensory failure.

In the next paragraph, we focus on another kind of bounded rationality: constraints in reasoning capabilities.

Limited reasoning capabilities

Until now it has been assumed that agents have unlimited computational resources for learning and decision-making. In this section we briefly discuss the effects of lifting this assumption by conducting some experiments with agent groups that have a limited storage space for learning samples and probabilistic ordering models. We analyse these two

$\sigma(noise)$	0.0	0.05	0.1	0.25	0.5
$\mu(agents, 100)$	3521.6	3489.2	3477.6	**3590.4**	2816.7
$\sigma(\mu(agents), 100)$	**336.2**	406.1	428.11	342.5	137.9
$\#conv$	74	80	71	**83**	0
$\mu(conv, \#conv)$	290.9	327.8	255.9	**281.6**	-
$\sigma(conv, \#conv)$	208.0	237.4	218.5	**196.1**	-
$\mu(\sigma(agents), 100)$	**29.3**	30.1	34.2	52.8	121.4

Table 6.2: Convergence analyses for simulations with *simultaneous* action perception, action execution noise and payoff perception noise ($\sigma(noise) = \sigma(n^{ae}) = \sigma(n^{ap}) = \sigma(n^{pp})$). As before, we conduct tests only for noise levels below which at least one of 100 simulations converges.

issues as *examples* of where bounded rationality might come in. Others would include limited training time per round, limited GA population sizes and/or neural network dimensions (these have been implicitly tested in the previous chapter) and the use of more shallow recursive belief models.

We first analyse the effects of restricting the number of POM ranks $|R|$. We test rank numbers of 2, 4, 8, 16, 32 and 64, bearing in mind that 16 ranks are sufficient to represent the preference structures of peers in this game. As can be seen from Figure 6.5, four ranks suffice to capture the orderings of peer gain models and, moreover, this number of ranks is optimal with respect to mean convergence-yielding round ($\mu(conv, 83) = 273.3$) and a standard deviation of $\sigma(conv, 83) = 221.2$. Looking at the *real* gain values of the underlying game, it can clearly be seen (cf. the gain matrix on p. 93) that they consist of four different entries, so that this result confirms our intuitive ideas of SBE learning. The fact that the learners get confused by having more than 16 ranks (which are obviously redundant to model a 4×4 gain matrix) seems even more convincing with this respect.

Finally, tests with various sample set sizes were conducted to examine the effects of constraining learning data storage capabilities on learning performance. To this end, *both* the UE and the SE sample memory spaces were restricted to a size of 10, 25, 50, 75, 100, 250, 500 samples (as opposed to the 1000 that would be necessary to store all samples that appear in the simulation runs). Twenty simulations were run for each of these sample set sizes, but the results are hardly elucidating; agents that stored only 10 samples converged 17 out of 20 times, all others converged in all but one simulation. This might be a hint that there is a minimal number (25) samples that should be stored, but still the differences are not too dramatic.
Figure 6.6 shows mean convergence rounds and their standard deviations for all sample set sizes. While optimal performance was achieved for a sample set size of 75 ($\mu(conv, 20) = 240.2$, $\sigma(conv, 20) = 138.3$), values for the other sample sizes do not allow for a straightforward interpretation since there is very little variation between them. It seems as if either *all* samples should be stored to achieve good results, or some small "memory window" should be applied to keep reasoning at a reasonable time-scale.
This ends our analysis of bounded rationality issues. They have basically shown that LAYLA agents are reasonably robust to noise (and what the limitations of this robustness are) and that, at least in simple two-player games, reducing the computational abilities of agents does not have too severe consequences on their performance. Moreover, there are hints to *positive* effects of bounded rationality, such as the performance increase for

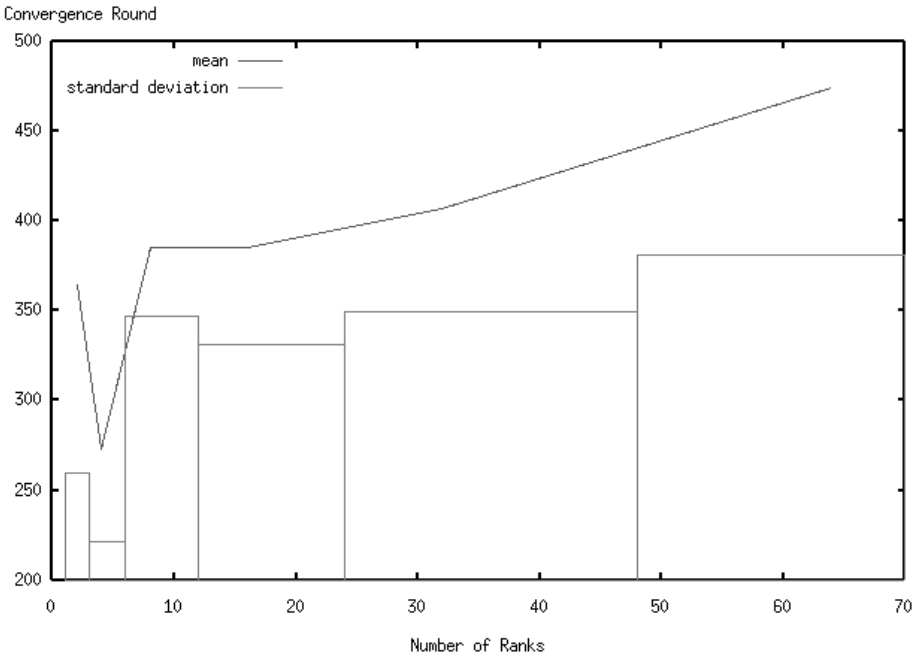


Figure 6.5: Mean convergence-yielding rounds (line) and standard deviations thereof (boxes) for $|R| \in \{2, 4, 8, 16, 32, 64\}$.

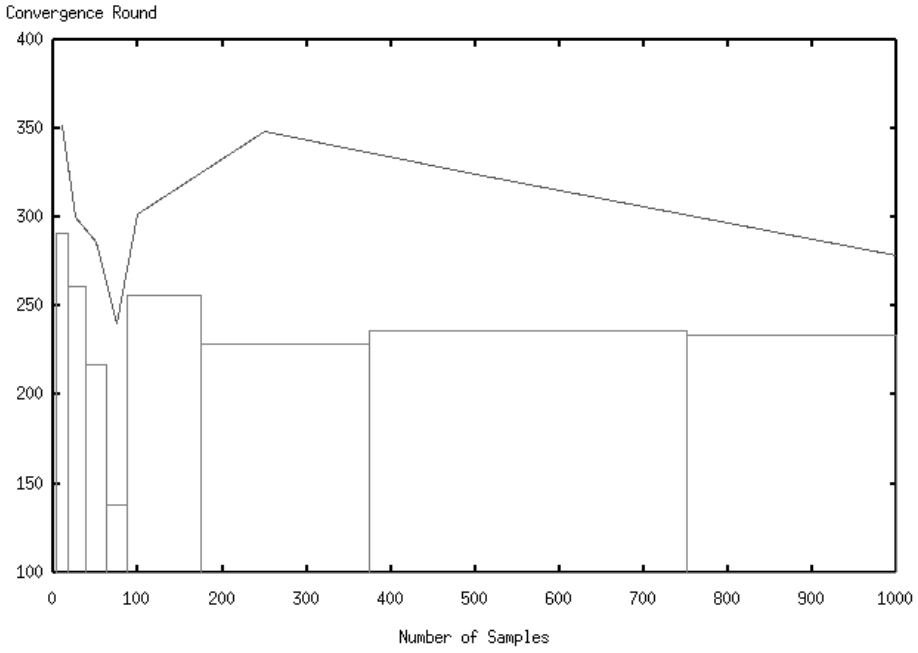


Figure 6.6: Mean convergence-yielding rounds (line) and standard deviations thereof (boxes) for UE and SE sample set sizes of $\{10, 25, 50, 75, 100, 250, 500, 1000\}$.

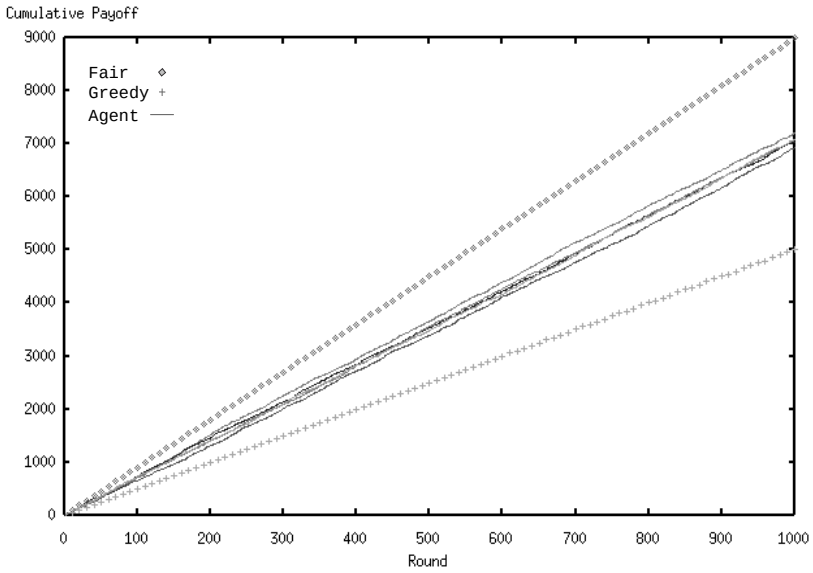
small amounts of noise and the superiority of agents that do not store too much learning data and do not attempt to be overtly precise in modelling gains (this is an effect similar to the phenomenon of overfitting). We now turn to the interesting issue of examining whether and how our observations for two-player games scale.

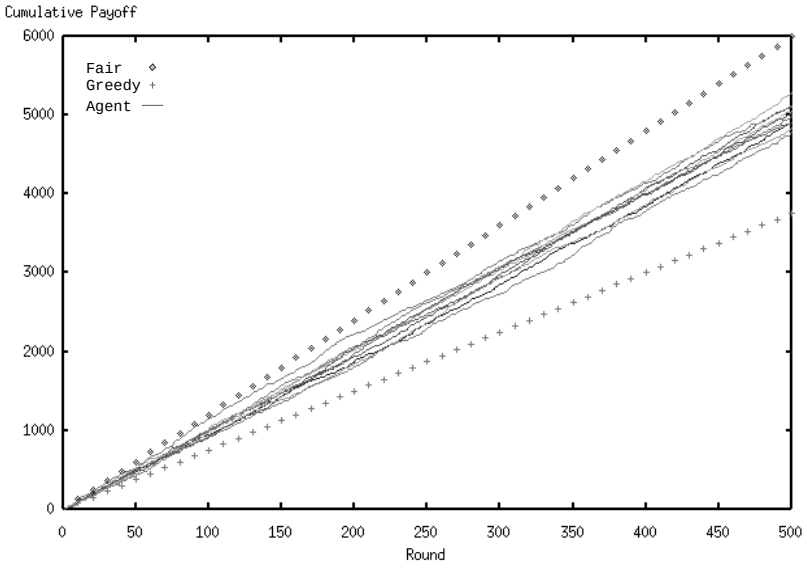
6.2.3 Scaling tests

In this final evaluation step, we show examples for LAYLA system performance in much harder five-player, ten-player and fifty-player games. In all of these games, agents have five resources at their disposal, though, of course, with increasing population size, it will be harder to distribute these efficiently among agents. It should be remarked that the three games we used have joint strategy spaces of (roughly) size 10^7, 10^{15} and 10^{75} and are thus indeed almost insoluble from a mathematical perspective (e.g. with dynamic programming methods).
The employed compromise factors (for $\lambda = 0.5$) were determined using the theoretical derivation of Section 6.2.1, which yielded $0.00306 \leq \gamma \leq 0.3469$ for the five-player game, $0.042 \leq \gamma \leq 0.25023$ for the ten-player game and $0.05051 \leq \gamma \leq 0.26181$ for the fifty-player game. To adhere to the rule of choosing γ close to its upper bound, we applied compromise factors of 0.34, 0.25 and 0.26 respectively. SBE neighbourhoods always consisted of 20% of the population, so that each agent reasoned about 1, 2 and 10 neighbours in the three games.

The results of the scaling tests are shown in Figure 6.7. As before, the plots show cumulative agent payoffs compared to the fair cumulative payoff and greedy cumulative payoff curves. It can directly be seen that the exhibited agent behaviour is far from being optimal for those larger games (it lies somewhere around 80% of u_{fair}) but also that it is well above the greedy payoff curve, which illustrates that LAYLA agents can cope quite well with the problems of large-scale interactions.
There is another, more subtle (but equally important) property of the simulations that cannot directly be deduced from looking at the payoff curves, namely that agent performance only *seems* to converge to some payoff-per-round between u_{greedy} and u_{fair} – in fact, agents continue to improve their performance, even though the progress in learning is hardly measurable: for the three games, the average final payoff rises by about 0.5%, 0.2% and 0.05% every 100 rounds (in terms of u_{fair}). This indicates that the system, even though it is primarily focusing on ensuring a comparatively high performance, still continues to *learn*. Unfortunately, complexity problems (600 rounds of the fifty-player simulation, for example, take more than one week) make it impossible for us to validate whether this learning will continue or whether there is some upper performance bound that cannot be exceeded.

It can safely be said that the performance agents exhibited for these very hard problems is highly reassuring. It shows that LAYLA agents are both capable of achieving high exploitation levels very quickly and yet may continue to learn and not converge to local maxima too soon. However, the right balance between exploration and exploitation still has to be found and we see these results only as indications of the *possibility* to further improve the architecture with this respect.





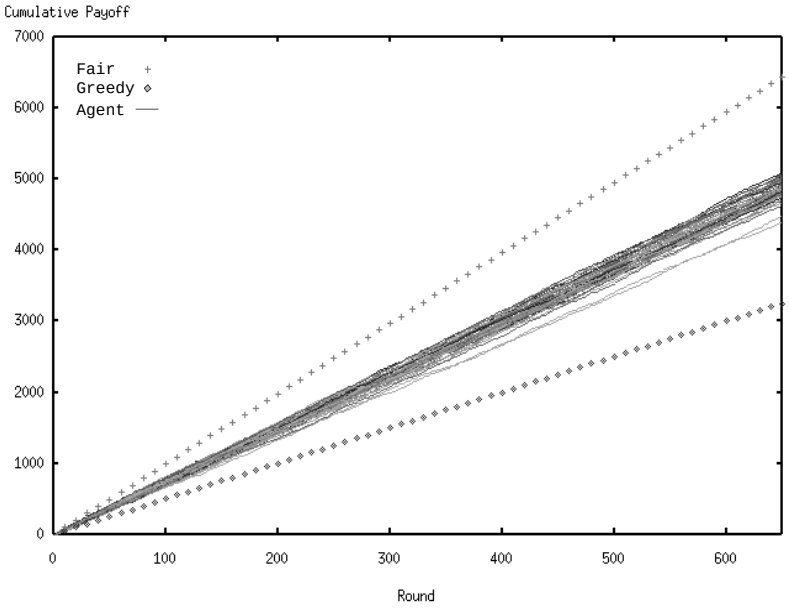


Figure 6.7: Performance plots for a five-player MARLOG simulation (topmost graph), a ten-player game (middle) and a fifty-player society.

6.3 Summary

This section provided a wide range of empirical results which proved the adequacy of our approach and offered valuable insights to properties of the LAYLA system.
The central result is that layered learning agents using our prototypical design did much better than greedy "cooperation-blind" individuals even in very large, inherently complex games without using communication and without any prior knowledge of the interaction situation. Quite naturally, optimality was only attained for simple games but performance is highly satisfactory even for the harder problems.
A second result is that we determined a set of parameters which are crucial to the performance of the system, especially compromise factors, exploration margins and "profiled" thresholds. Most importantly, the success of LAYLA agents obviously always hinges on the performance of the payoff learning component. This is an effect of the interdependence caused by sharing knowledge among different learning components, and such phenomena must not be overlooked in designing hierarchical/distributed learning systems.
On the other hand, we found out that certain other parameters have little or no influence on the system, such as the optimism parameter, the POM update factor, POM dimensions and sample set sizes. Unfortunately, we have not been able to identify general (meta-)rules for determining these parameters yet, so that there clearly is a need for devising some additional meta-reasoning component to cope with this problem (cf. the discussion in Section 7.2).
Surprisingly, limited amounts of noise were found to be helpful, since they apparently increase diversity and randomisation in learning and decision-making. For greater noise levels, agents proved to degrade gracefully, even under combined action execution, action perception and payoff perception noise. This provides us with insights to limitations of the effectiveness of learning algorithms under imperfect information and action.

Ultimately, all of these results can only be seen as an initial assessment of the architecture. Due to space limitations, we have not given an analysis of system performance in heterogenous populations, of the effects of dynamic changes in the environment or of the implications of having various types of neighbourhoods (with different sizes and/or structures).
We still believe that sufficient evidence for the usefulness of the architecture has been provided and this fact hints at possible conclusions that we can draw from our efforts. These will be further discussed in the closing chapter to follow.

Chapter 7

Conclusion

They constantly try to escape
From the darkness outside and within
By dreaming of systems so perfect
that no one will need to be good.
— T. S. Eliot, *Choruses from 'The Rock'*

To evaluate the contribution of a novel approach within the context of a broader field of research it does not suffice to analyse its performance as was done in the previous chapter. If we are to gain insights from the architecture we proposed, we need to question its adequacy on a more abstract level, to describe its advantages and drawbacks, to look at questions that it leaves unanswered and – if possible – to draw some more general conclusions concerning the analysed problems. These issues will be covered in the following sections. Before that, however, we briefly summarise the contents of the previous chapters to provide a concise overview of the main goals of this work, the proposed methods and the results obtained.

7.1 Summary

We have presented a layered learning architecture that allows autonomous agents to learn how to coordinate effectively in multi-agent environments. As a starting point for our considerations, we questioned some of the central assumptions made by many other works from the field – we claimed that neither prior knowledge nor communication is necessary for cooperation to evolve, and that suitable social reasoning mechanisms can be devised to achieve effectively coordinated behaviour even among non-benevolent, self-interested individuals.
Within a game-theoretic framework, we presented an application scenario which captures many hard problems that may occur in interaction situations and determined optimal solutions for the underlying games to ensure a rigorous assessment of the performance of our system.
The learning architecture that we devised subsequently was based on two basic concepts. It merged the idea of a general hierarchical, distributed learning methodology for complex learning problems with the concrete decomposition of the "coordination task" into three learning sub-tasks, each related to one of the so-called *essential determinants* of interaction. It is only this combination of a methodology for layered adaptation and of a learning problem that can be decomposed appropriately that enabled us to extend InteRRaP to a

learning architecture in a straightforward manner.
At the core of the architecture, of course, lay the individual learning algorithms that were supposed to manage the learning tasks of each layer. Much emphasis was put on providing rigorous definitions of the learning problems and on identifying practicable "approximators" for these sub-tasks (which are almost insoluble in their most general form).
At some points off-the-shelf learning techniques (adapted to the underlying learning problems) were employed, but we also explored new, "customised" learning methods that proved to be particularly effective for the special class of learning problems we analysed.
Before going into an overall analysis of the integrated system, preliminary tests with each of the layers were conducted to ensure that each layer was capable of managing its own learning task.
Finally, an extensive account of the empirical evaluation of the system was given, whose focus was on assessing the performance of the proposed system and its limitations. Essentially, it proved that our method *is* appropriate to solve coordination problems in settings where only very little information is available. Yet there is a much wider range of conclusions that can be drawn from the individual results, and these will be critically reviewed in the following.

7.2 Achievements and Shortcomings

In Chapter 1 we argued for a decomposition of the "coordination" task into individual learning goals that would facilitate the emergence of effective interaction patterns in agent societies of purely *selfish* non-communicating agents that that have *no* prior knowledge of the interaction setting. We claimed that the identification of *essential determinants* of interaction was the key to interaction learning; and that a layered learning architecture suited for combining learning components that focus on these determinants would enable agents to coordinate their actions effectively with other individuals.

There are several arguments in support of the fact that these goals were achieved.
Firstly, our agents always did substantially better than in the "greedy" equilibrium states. Thereby, they proved capable of (i) defending themselves against others' attempts to exploit them and (ii) identifying inefficient greed and behaving cooperatively whenever it seemed beneficial to them.
For small problem sizes, they almost invariably converged to optimal behaviour, while their performance was still very good for larger problem sizes (with vast joint strategy spaces and highly complex payoff functions). Given the fact that they start off without *any* information about payoff function properties, opponents' decision-making processes and the utilities other agents obtain, this is a rather impressive result. It proves[1] that coordination can be learned in principle with very little information and without any domain-dependent knowledge concerning resource conflicts or the nature of physical action and perception.
Also, agents need not count on their peers' benevolence and they need not be benevolent themselves, unless it appears to be "a good move". As a consequence, there need not exist some external authority to enforce mutually beneficial agreements, even though its existence would probably make them more reliable and easier to achieve.

[1] Genesereth et al. (1986) have shown that communication is not necessary to achieve cooperation. However, they assumed a great deal of knowledge about opponent reasoning mechanisms; the strength of our approach lies in the fact that agents deduce *all* knowledge about others from action observation.

More generally, we can say that it has been shown that cooperation is very likely to emerge even amongst self-interested individual utility maximisers provided that they be "socially rational" (in the sense that they base their decisions on their dependence on others) if they are given enough time to gather information about the nature of the interaction they are involved in.

A second positive result we can report is that breaking down the task of "learning how to coordinate" into the three learning problems concerned with interdependence modalities, opponent behaviour and cooperation potential is adequate for our purposes.
Tests conducted in Chapter 5 proved that these sub-tasks are learnable and that the algorithms we constructed for them are suitable to learn payoff functions, to predict future opponent behaviour and to approximate the decision mechanisms of peers. Then, by proving that these algorithms can be integrated to work together for solving the top-level task in Chapter 6, it was shown that they can be combined appropriately to yield a self-contained hierarchical learning architecture.
Even though the components interact in a pre-designed way that seemed intuitively adequate, it is not a trivial matter that an effective agent-level (and, perhaps even more surprising, a society-level) learning process emerged.

There are several further aspects of the observed system performance that underline the effectiveness of our architecture. The resource-load balancing games we used for validating agent performance are very hard games. Their equilibrium states tempt agents to act in a greedy, un-coordinated fashion, so that we would probably assume humans to submit to this temptation unless they are forced to behave cooperatively (by some external authority) or have reached enforceable agreements with their opponents (which they can obviously only achieve if they are able to communicate). But even cooperative agents would most likely get confused by the multiple fair allocation states the games offer, so that convergence to optimal behaviour would be very hard to attain. In the light of these properties of the games we used it seems quite reassuring that LAYLA agents do that well. Moreover, the system was reasonably robust to noise and agents converged to effective coordination patterns within an acceptable amount of time, considering that they enter the interaction without any prior knowledge.

All this strongly supports our claim that the main problems our work was concerned with were adequately solved. However, there are also some points of criticism that should be addressed.

One major problem of our architecture is complexity. The overall computational complexity of the system effectively restricts it to use in small agent groups only. Unless immense computational resources can be provided, it is unlikely that it can be used in large-scale simulations.
However, there are some possible remedies to this problem (other than supplying more powerful hardware). One of them is to increase the amount of information that is made available to the agents by communication or by prior knowledge. Quite obviously "knowing more" implies "having to learn less". The modularity of a layered architecture such as LAYLA allows for an identification of what kinds of information would be useful for the agents and of "where to put it". A (possibly tentative) explicit representation of the payoff function might be made available to the UE if, for example, the agent has some domain knowledge with which it can model its own utility function. OBP learning could

be replaced by actual opponent behaviour commitments that are communicated to the agent. In a similar way, the revelation of preference structures by peers could speed up the learning process of the SBE[2].
Another possibility is to manage the time and space resources spent on learning dynamically (cf. Gerber and Jung (1998), Horvitz, Cooper, and Heckerman (1989)) so that whenever a stable state is reached the agent can suspend the learning process and resume it as soon as it notices some changes in the environment. In the simulations of Chapter 6, for example, it seems a sheer waste of computational resources to conduct any further learning as soon as the greedy or fair convergence-points are reached, but still the agents actually conducted exactly the same amount of learning after the emergence of these stable states as they had done before.

Conceptually, i.e. as regards the scope of modelled interaction situations, the most severe problem is that we do not account for *sequential interactions*. LAYLA agents engage in repeated interactions, but these are identical one-shot games with no intermediate steps (or, if these exist, they are encapsulated in the black-box procedures represented by abstract strategy sets). And while we have tried to do away with most of the other problems of game-theoretic models (especially the epistemic aspects discussed in Section 3.1.3) there does not seem to be a reasonably simple way to get around this issue.
Imagine we were to extend our framework to cope with state-oriented world models, in which certain states are goal states (possibly those states in which agents consider their tasks completed). Regardless of whether agents receive intermediate rewards or if they construct these themselves to identify optimal policies (as is common in reinforcement learning methods), the rewards will depend on the actions of other agents. Hence, we would obtain one interaction situation for *each* state the agent might find itself in. Even under the assumption that the state spaces which are normally huge in multi-agent domains can be clustered to yield a limited number of normal-form games, agents would have to learn several games. We can only guess what implications this would bear on complexity and training times. Thus, although in theory our framework can be extended to accommodate the needs of agents interacting in stateful environments it would probably produce intractable learning problems.

Furthermore, some minor points concerning specific design decisions and the employed testing policy should be mentioned. The architecture was tested only for a single, very stylised scenario – many more would be necessary before we could safely speak about LAYLA as a "generic" paradigm. A great deal of internal agent parameters and learning algorithm configurations were determined through experiments with this particular class of games. Choices concerning UE neural network dimensions, SE population sizes and nearest-neighbour numbers were made in a clearly domain-dependent way, as was the case for most of the SBE parameter choices (some of them were even set to arbitrary, fixed values, as e.g. rank-flattening vectors and the depth of nesting in recursive belief modelling).
The compromise factor γ constitutes the most crucial system parameter when it comes to changing from exploitative to cooperative actions. The fact that it was determined according to the theoretical derivation of Section 6.2.1 is therefore perhaps the most severe shortcoming with this respect. However, we have not been able to identify a generic

[2]In the case of communication, one would have to compare the benefits of additional information to the computational overhead produced by allowing for explicit communication.

method of computing appropriate values for this parameter yet (ideally, it should be *learned* as well).
Finally, our framework does not allow for major dynamic changes in the environment, such as the entrance of new agents into the society or changes in the number of available resources. But, quite obviously, this problem could easily be done away with: UE neural network dimensions could be changed during the learning processes, SE populations and their bit-string individuals could be replaced by larger/smaller hypotheses online and, since the SBE is based on many binary dependence analyses, its adaptation to new group sizes is straightforward.

Despite all these drawbacks, we still believe that our approach is a useful step towards more generic concepts of *coordination learning* in multi-agent environments. It has been shown that the layered learning methodology works in principle, that it can be useful for boundedly rational agents to coordinate their actions in open systems in which they may encounter malevolent, cooperative or even irrational peers.
On the other hand, the LAYLA architecture suggested a way for extending InteRRaP to a hybrid *learning* architecture by showing that a hierarchical layering can be used in learning in the same way as it is useful for reasoning, planning and decision-making.
In our view, one of the most important achievements lies in the fact that we confronted some of the strongest assumptions of game-theoretical models, especially that we replaced perfect rationality, i.e. the ability of agents to analyse the mathematical structure of payoff functions exhaustively and to make decisions accordingly, by adaptation.

In the next section, we discuss some issues that should be looked at in the future.

7.3 Open questions

LAYLA agents are designed for *communication-less* repeated *stateless* interactions amongst *egoist* utility maximisers. They use *fixed* learning and *myopic* decision-making mechanisms and internal parameters, and their learning focuses on gathering information *about* their adversaries. Quite naturally, most of the possible extensions to the system that can be thought of relate to each of these assumptions.

The most important of these extensions would certainly be the development of a *meta-reasoning* component for the LAYLA architecture. Ideally, such a component would be capable of

- flexible resource management that allows agents to adapt their learning and reasoning efforts to time and space constraints dynamically,
- monitoring the learning layers' performance, adapting their parameters whenever necessary, managing stored training sets and discarding current hypotheses upon unexpected changes to the environment and
- controlling the interaction between learning and decision-making layers in a more flexible way than is possible now.

The benefits of having such a component would be that we can tackle some of the complexity problems previously discussed, and that the development of short-sighted decision-making strategies (as it is currently reflected in the computation of best-response strategies

and expected gains) would be controlled by a planning component to yield more reliable long-term meta-strategies: only if the agent can reason about its learning and decision-making components can it improve its long-term abilities beyond the myopic perspective of the current SE and SBE implementations.
The central advantage of re-configuring internal parameters dynamically would be that the compromise factors could be determined by the agents themselves. The performance of agents in *heterogenous* societies or in societies in which opponents' social stance *changes* over time probably hinges on the capability of adapting "cooperativeness" to the current situation.
At a more conceptual level, it appears most natural to include a means of reflection about "coordination learning" once we set out to develop such a learning methodology – if there exists a generic theory of learning how to coordinate, individuals should also be able to reason about it.

Then, of course, the "no communication" and "no sequentiality" assumptions should be reconsidered. In Chapter 1 we gave several reasons for not including the possibility of explicit communication in our system but, obviously, it does constitute a central part of interaction in real-world problems. The importance of communication for our architecture would probably lie in the fact that it can be used to speed up the learning process: communication-less environments provide a very limited amount of information so that the information agents have at their disposal would be substantially enriched if they were exchanging messages.
In a more general sense, communication should be analysed with respect to its effects on the evolution of interaction. An interesting question would be, for example, whether communication-less environments provide more or less possibilities of deception than settings in which agents can communicate or vice versa. Another interesting issue to look at is *concept formation* (cf., e.g., de Jong (1999)) among coordination-learning agents, i.e. to ask what kind of concepts would be formed by agents if, in an evolutionary way, they began by exchanging random strings and assigned meaning to them over time. This would maybe be an alternative way of determining what essential concepts govern the process of interaction.
Including the possibility of sequential interactions with intermediate steps was already discussed in the previous section. The main argument that speaks for such an extension is that natural worlds require the explicit planning of future activities, sequences of strategies that will lead to desirable states. As we have mentioned, our methodology cannot be extended to accommodate these requirements in a straightforward manner, but they certainly are an interesting extension to it.

Another interesting issue is the inclusion of alternative *forms* of learning. In our architecture, agents build models of others, i.e. they learn something *about* their opponents, but what if they learned *from* or *with* others?
Bandura (1977), for example, argued that a great deal of social learning is not based on reinforcement but on imitation, and there is a growing body of research in DAI that follows this intuition (cf., e.g., Demiris and Hayes (1996), Kuniyoshi et al. (1994), Price and Boutilier (1999) or Bakker and Kuniyoshi (1996) for an overview). This is because the risks connected with the exploration-exploitation problem of reinforcement learning usually make humans rather imitate some behaviour of successful individuals than to try random behaviours until an optimal strategy is found. In the same way, interacting agents might be able to learn something from each other, even if their observations are usually

limited/unreliable and if, in many cases, there is some heterogeneity between the imitator and the imitated (so that supposedly effective, observed behaviours are not necessarily useful for the imitator).
Alternatively (or additionally), agents could engage in collaborative learning to speed up the learning process. They might, for example, make agreements about a certain method of exploring the joint strategy space so that no action has to be repeated more often than is absolutely necessary. Or compare their gain models to find out to which degree they are interdependent. Note that such collaborative learning does not require that the agents themselves be collaborative – they could make deals about who is going to learn what and only participate in the societal learning efforts if it pays for themselves.

Until now we have only contributed to something like "small group research" for multi-agent systems. It has not been analysed whether the architecture is suitable for *large-scale* societies of possibly non-recognisable agents or for societies that evolve over time (e.g. in generations) and these are also questions that deserve our attention.
While complexity reasons were the main reason for restricting ourselves to relatively small societies, it should be made clear that "scaling" does not imply that we should look at games with, say, a million players; even in such large populations, it is very unlikely that the particular interactions comprise more than a few dozens[3] of agents – larger games are simply unsolvable.

7.4 Theoretical value vs. practical relevance

Quite often – especially in such novel fields as multi-agent learning – there is some discrepancy between the theoretical value of developing an architecture (and testing it within some abstract, mathematically modelled scenario) and the practical use of the suggested methodology. Admittedly, this work has been mostly concerned with providing a theoretical, hierarchical approach to the problem of "learning how to coordinate effectively" and there are only limited possibilities to use it directly in some concrete application.

The resource-sharing scenarios mentioned in Section 1.3.2 give examples for the kind of real-world applications for which the approach is suitable. Typically, it can be used for the long-term optimisation of systems in which agents represent human users and suggest/decide which of the available resources to use at some given point in time. There, it is particularly effective and useful whenever the global system behaviour is not known in advance and users can only learn from their past observations, i.e. from occasional system feedback (e.g. response times). In that case, having LAYLA agents monitor and manage the interactions probably yields much more reliable strategies than the subjective heuristics that humans usually develop over time. This is because the latter might be biased by personal feelings ("Oh my God, the guy next door is using that server *again*!"), obliviousness ("Wasn't Altavista always fast during early evening hours? Or was it that other search engine?") etc.

[3] According to Rosaria Conte (personal communication), "fifty" seems to be a magical number in current research on social simulation/artificial societies – systems either consist of up to fifty deliberative agents with elaborate reasoning/communicative/adaptive capabilities *or* they comprise thousands (or even millions) of simple, reactive agents (e.g. cellular automata). However, it is common practice to have two-player encounters or interactions between a manageable number of groups (clusters of individuals) in large societies.

Another type of systems that are possible candidates for the use of our framework are applications in which a centralised resource allocation is either impracticable or intractable. Consider, for example, external modem access to some local area network (LAN). Obviously, the LAN is not able to control which external users will clog the telephone connections at what time even if it may very well control access to the network itself. Also, given the fact that in having multiple external users with different needs at different times it is very difficult to schedule efficient modem-caller allocations in a centralised manner. If, on the other hand, resource allocation was managed distributedly by the callers, they might learn that it does not pay to dial-in all the time, because if everyone does so, they will never access the network themselves.

Still there is no doubt that the benefits of the suggested framework lie clearly on the theoretical side. We contributed to the comparatively unexplored fields of multi-agent learning and hierarchical learning and we presented an adaptive architecture for effective behaviour in repeated games, thereby lifting some of the critical assumptions made by other works from the field.
Many theoretical problems remain unanswered. Most importantly, it would be desirable to have some insights into the theoretical properties of layered and distributed learning as we employ it. In the experiments of Chapter 6 it was very often the case that we had only very vague, intuitive explanations for *why* the system worked instead of a well-founded theory that would provide more reliable arguments. Given, though, that no *multi-agent learning theory* has been developed so far in the way it exists for single-agent learning (the work of Vidal and Durfee (1998b) is a first step in that direction), it is very hard to formalise general learning properties of the LAYLA system, especially properties that are independent of the underlying games.
Furthermore, it seems that many of the observed behaviours should in some way be linked to the terminology of the social sciences via *social metaphors*, because these fields have a long tradition in studying social interaction that AI can benefit from. Concepts like social norms, trust, deception, group formation, conformity, reputation and isolation have not been explicitly connected to the system behaviour that was observed, but they are certainly useful (if not essential) abstractions when talking about social dynamics in co-inhabited worlds.

We believe that what is still missing in DAI research is a *social level characterisation* (Jennings and Campos, 1997) of intelligent systems that describes global system behaviour on the grounds of local agent interactions – maybe social metaphors are one of keys to the explanation and prediction of emergent behaviours in multi-agent societies. Since the micro-macro effect is one of the most exciting features of distributed intelligent systems it surely deserves further analysis – our work can be seen as a contribution to these efforts.

Appendix A

Notation

A.1 Mathematical symbols

a_i, a_Q, a	A pure strategy in the MARLOG game (of player i, of a subset of players $Q \in \mathcal{P}$ and of the entire player set).
a_{-i}, a_{-Q}	A pure strategy of all players *but* i and a joint strategy of all players but $\mathcal{P} - Q$.
α_i, α_Q, α	A mixed strategy in the MARLOG game (same indexing as with pure strategies).
$\alpha_i^{(t)}$, $\alpha_Q^{(t)}$, $\alpha^{(t)}$	Mixed strategies in round t of a repeated MARLOG game.
α^{UE} , α^{SE}	(Intermediate) strategies output by the UE and SE in the SOCCER algorithm.
A, $A^{\|Q\|}$, $\mathcal{A} = A^{\|\mathcal{P}\|}$	The strategy set of one player, of a subset of players $Q \in \mathcal{P}$ and the joint strategy space of all players in the MARLOG game.
$\mathsf{add}(s, (l', k'), (l, k), \delta)$	Add-operation for POMs: adds probability δ to the statement $P((l', k') > (l, k))$ in a POM s (Social Behaviour Engine).
all_i, all_Q, $all_{\mathcal{P}}$	Strategy $2^k - 1$ in a MARLOG with k resources; the strategy by which a player (set) accesses (jointly) all available resources.
β	A function that transforms decimal integers to binary vectors (usually employed to transform actions to k-long resource access bit vectors).
c	Access costs of a resource (in the general MARLOG, a function of player i and resource r; in the simple version of the game it degenerates to a function).
$\vec{c}$	Rank-flattening vector for Probabilistic Ordering Models. Converts a POM to a real-valued vector (Social Behaviour Engine).
γ	Compromise factor (Social Behaviour Engine).
$\Gamma = \langle N, S, u \rangle$	One-shot n-person game in normal form.
Γ^t	The t-repeated version of a normal-form game Γ, i.e. a sequence of t joint strategies.
d	A function that randomly picks a value according to a given probability distribution.
D	Set of training examples (Utility Engine, Strategy Engine).

δ	POM update factor (some probability $\in [0;1]$) in the Social Behaviour Engine.
δ_i, δ	Transition function in a game for a player i and for the whole player set; computes the next action as a function of the history of the game (in Markov Decision Processes (Section 5.3.1), state transition function).
Δ	Transition probabilities for opponent behaviour (Strategy Engine).
$distance$	Distance function for nearest-neighbour prediction (Strategy Engine).
e_i^k	The k-th pure strategy of player i written as a mixed strategy.
e_i	Utility Engine exploration bias function of agent i.
$E(V)$	Neural network performance measure (mean error on a set V of generated samples).
$E(t)$	Strategy Engine prediction error in round t.
ϵ	Some percentage of tolerance in determining whether a simulation has converged (Chapter 6).
ε	The accessor-resource delay factor of the MARLOG game (in the general version, a function of the player; a constant in the simple MARLOG).
$\varepsilon_j(k)$	Probability that j will play k deduced from the recursive belief models (Social Behaviour Engine).
$\mathbf{E}$	Learning experience for a given machine learning algorithm.
$\mathbf{E_i}$	Learning experience for an agent learner in a repeated game scenario. Consists of a series of joint actions and the respective payoffs for i (equivalent to H_i in the LAYLA system).
f_i	The learning hypothesis by which agent i predicts its opponents' actions.
$fitness$	Strategy Engine GA fitness function.
φ	Policy in a Markov Decision Process.
$g_i(a_i)$	The expected gain of playing a_i for agent i (Social Behaviour Engine).
$\text{gain}_{a_i}(a_j)$	Gain value of j's action a_j for i's action a_i (Social Behaviour Engine).
$G_i(0), \ldots G_i(2^{k-1})$	Genetic algorithm populations of a Strategy Engine in a MARLOG game of k resources.
$G(L)$, G_i, G_j	Learning goals of learners L, L_i and L_j.
h_i, $length(h_i)$	The learning hypothesis by which agent i computes a future action sequence that will lead to a globally optimal behaviour and the length of the predicted sequence.
$\vec{h}$	Hidden layer dimension vector of Utility Engine neural networks.
h, h_l	Hypothesis (bit-string) in a Strategy Engine GA population and its l-th bit.
H_i^t	The history of a repeated game as player i perceives it in round t (a sequence of past joint actions and the corresponding payoffs for i).
$\mathcal{H}$	Hypotheses (individuals) of a genetic algorithm population.
i, j	Indices for players.
η	Neural network learning rate.
k	Number of available resources in a MARLOG game.

$\mathcal{K}$	Number of nearest neighbours used for opponent behaviour prediction (Strategy Engine).
l_i	Social Behaviour Engine action-value function of player i (social action-value function).
l, k	Indices for actions in Probabilistic Ordering Models.
λ	Optimism parameter for the computation of gain values (Social Behaviour Engine).
L, L_i, L_j	Learning components in layered learning architectures.
$\mathcal{L}_j$	Set of socially feasible actions with respect to neighbour j.
L^{IM}, L^{OBP}, L^{CP}	Interdependence modalities (IM), opponent behaviour prediction (OBP) and cooperation potential (CP) learners.
m_i	Strategy Engine (greedy) action-value function of player i.
μ_i	Average Strategy Engine prediction error.
$\mu(agents)$	Mean final payoff of all agents in a single simulation.
$\mu(q, s)$	Mean of quantity q in s identical simulations.
$\mu(conv, s)$	Average convergence-yielding round number in s simulations.
M	Arbitrary real-valued matrix that is modelled by a POM (Social Behaviour Engine).
mr	Mutation rate of a genetic algorithm (Strategy Engine).
mv	Match value function for GA hypotheses (Strategy Engine).
n	The number of players in a game in normal form.
n_i^{ae}	Simulation Engine action execution noise function for player i.
n_i^{pp}	Simulation Engine payoff noise function for player i.
n_i^{ap}	Simulation Engine action perception noise function for player i.
$N = \{1, \ldots n\}$	The set of players of a game in normal form.
$N(\mathcal{K}, a_i, a_{-i})$	Set of $\mathcal{K}$ nearest neighbours for a given action a_i of player i and a given previous opponent action a_{-i} (Strategy Engine).
$\mathbf{N}$	The set of natural numbers (including zero).
$\mathcal{N}_i$	Neighbourhood of agent i (Social Behaviour Engine).
$none_i$, $none_Q$, $none_\mathcal{P}$	Strategy 0 in any MARLOG game. Opposite to "*all*", it is the strategy by which none of the available resources is accessed.
OPT/opt	Set of individually and collectively rational joint strategy combinations/one such strategy.
p	Rank probability function that assigns a probability to each rank statement (Social Behaviour Engine).
π_i	Payoff estimate of an agent that tries to learn its true payoff function u_i.
$P(s)$	The (real) probability of statement s.
$\mathcal{P}$	The set of players in a MARLOG game.
$\mathbf{P}$	Learning performance measure (in general, and as a function of the LAYLA hypotheses π, f, and h).
Pr(s)	The posterior probability with which s was found to be true (the frequency with which it occured in prior observations).
$\mathrm{Pr}_j(k)$	The posterior probability with which agent j has played action l so far (Social Behaviour Engine).
$predict$	Prediction function that outputs the next opponent action on the basis of nearest-neighbour outputs (Strategy Engine).
$profiled(j)$	Boolean function by which the behaviour of agent j is judged to be "profiled" or not (Social Behaviour Engine).

r	Variable: a resource in the MARLOG game/a rank in a Probabilistic Ordering Model. Function: reward function in an MDP.
ρ	Ranking function in Probabilistic Ordering Models. Assigns a rank to each matrix position (Social Behaviour Engine).
$\mathcal{R}$	Set of resources in a MARLOG.
R	Set of ranks in Probabilistic Ordering Models (Social Behaviour Engine).
$\mathbf{R}$	The set of real numbers.
$RL = \langle n, k, v, c, T, \varepsilon \rangle$	A general multiple-access resource-load balancing game.
rr	Replacement ratio of a genetic algorithm (Strategy Engine).
$s_i^t,\ s_Q^t,\ s^t$	t-long sequences of strategy choices of player i, the subset Q of players or the whole player set.
$s_i^{(t)},\ s_Q^{(t)},\ s^{(t)}$	The t-th strategy choice of player i, the subset Q of players or the whole player set in a sequence of actions.
$s^0,\ s^1,\ s^2,\ s^3$	Recursive belief models of an agent (Social Behaviour Engine).
$\mathcal{S}^0,\ \mathcal{S}^1,\ \mathcal{S}^2,\ \mathcal{S}^3$	Collections of recursive belief models for all neighbours (SOCCER algorithm).
$s_\emptyset$	The "empty" POM, has a rank probability function that assigns each matrix position to each rank with uniform distribution.
$\sigma(agents)$	Standard deviation between final agent payoffs in a single simulation.
$\sigma(q, s)$	Standard deviation of quantity q in s identical simulations.
$\sigma(conv, s)$	Standard deviation between convergence-yielding round numbers for different simulations.
s_i	Pure strategy of player i in a normal-form game.
$S_i,\ S$	The strategy space of player i in a game in normal form, the cross-product of these for all players.
$S_i^t,\ S_Q^t,\ S^t$	Sets of t-long sequences of strategies.
$\mathcal{S}$	State space in a Markov Decision Process.
$\sigma_i,\ \sigma_Q,\ \sigma$	Mixed strategies for general normal form games (cf. α for indexing conventions).
$\Sigma_i,\ \Sigma_Q,\ \Sigma$	Mixed strategy spaces for games in normal form.
sim	A function that measures the similarity of opponent strategy sequences.
T	Access time factor for a resource in the MARLOG (function/constant, cf. also c, v).
$t,\ t_1,\ t_2$	Numbers of played one-shot games in a repeated game.
ϑ	Variance threshold for determining whether a peer is profiled or not (Social Behaviour Engine).
$\mathbf{T}$	The task of a learning algorithm.
$\Theta(RL)$	Set of Nash equilibria of a simple MARLOG game RL.
$u_i,\ u_Q,\ u$	The utility function of a game Γ for one player, a subset of players and the whole set of players.
$u_{max}, u_{greedy}, u_{fair}$	The maximally possible/greedy/fair single-agent payoff in a MARLOG game.
$\lvert\vec{u}_1 - \vec{u}_2\rvert$	(Geometrical) distance between two payoff vectors.
v	Value assigned to a resource in the MARLOG game (function/constant, cf. also c, T).
$V_\varphi / \tilde{V}_\varphi$	Value/expected value of a policy φ (MDPs, Section 5.3.1).

$\vec{w}$	Vector of all weights in a neural network (Utility Engine).
ζ	Exploration threshold of the Utility Engine (SOCCER algorithm).
#	Wildcard symbol for bit-strings (Strategy Engine).
$\|\| \cdot \|\|$	Normalising function.

A.2 Abbreviations

ANN	Artificial Neural Network
BBL	Behaviour-Based Layer
CG	Coordination Game
CP	Cooperation Potential
DAI	Distributed Artifical Intelligence
DE	Decision Making and Execution Module
DFA	Deterministic Finite Automaton
DPS	Distributed Problem-Solving
IM	Interdependence Modalities
InteRRaP	Integration of Reactive Behaviour and Rational Planning
GA	Genetic Algorithm
GM	Gain Model
LAN	Local Area Network
LAYLA	Layered Learning Agent (architecture)
LPL	Local Planning Layer
MAL	Multi-Agent Learning
MARLOG	Multiple-access resource-load balancing game
MDP	Markov Decision Process
MAS	Multi-Agent System
OBP	Opponent Behaviour Prediction
PD	Prisoners' Dilemma
PFA	Probabilistic Finite Automaton
POM	Probabilistic Ordering Model
RL	Reinforcement Learning
SBE	Social Behaviour Engine
SE	Strategy Engine
SG	Situation Recognition and Goal Activation Module
SOCCER	Social Cooperation-Enabling Reinforcement
SPL	Social Planning Layer
UE	Utility Engine

Appendix B

Solution concepts for games

We provide an introduction to some solution methods of classical game theory, which we use to determine optimal solutions for the resource-load balancing game (cf. App. C). Section B.1 introduces several concepts that constitute formalisations of *individual rationality*, i.e. that are solely based on the principle that players will use those strategies that promise maximal payoffs. These can be thought of as *agent-level* solution concepts, since they make predictions about how utility-maximising individuals will behave.
Then, we look at the *system-level* view of games by presenting the *kernel* solution method for n-person cooperative games in Section B.2. Although this concept still obeys the principle of *individual rationality*, it also analyses the optimal behaviours that collections of cooperating players should exhibit[1].

B.1 Dominance relations, equilibria and pareto-optimality

As Fischer et al. (1998, p. 29) have pointed out, defining what should be thought of as a "solution" of a game is not a trivial issue. From some individual player's standpoint, solutions might be defined as those behaviours which yield high worst-case payoffs for that player, but from the system designer's viewpoint, this might be inadequate, because such "solutions" fail to get the most out of the game for all players involved (from a global perspective, the four "wasted" units of payoff in the PD when playing (D,D) instead of (C,C) are an inacceptable outcome).
In this section, we will present some very basic solution concepts which are based on simple mathematical properties of the payoff matrices and the assumption that players are individual utility maximisers.

B.1.1 Dominance, best replies

The notion of *dominance* is essential to the analysis of solution concepts, because it provides a simple criterion to rule out strategies that are worse than some other dominant strategy for at least one joint strategy combination, while they never outperform the dominant strategy. Under the assumption that agents act rationally, dominated strategies

[1] This section closely follows the lines of the introduction given in (Fischer et al., 1998) and the introductory chapter in Weibull's book (Weibull, 1995). As in Section 3.1.2, we will use the PD and CG games to illustrate the concepts we formally introduce in the following sections.

can be neglected, and some games can in fact be completely solved by applying this principle.

Definition B.1 (Strict and Weak Domination)

- *A strategy $s_i \in S_i$ is said to* **weakly dominate** *some other strategy $t_i \in S_i$, iff*

$$\forall s_{-i} \in S_{-i}.u_i(s_i, s_{-i}) \geq u_i(t_i, s_{-i})$$

(for any opponent behaviour, s_i yields at least as high a payoff as t_i), and

$$\exists s_{-i} \in S_{-i}.u_i(s_i, s_{-i}) > u_i(t_i, s_{-i})$$

holds (s_i is more successful for at least one joint opponent strategy).

- *The strategy s_i is said to* **strictly dominate** *t_i, if the second condition holds for any s_{-i}, i.e. iff*

$$\forall s_{-i} \in S_{-i}.u_i(s_i, s_{-i}) > u_i(t_i, s_{-i}).$$

In the PD, the "defect" strategy strictly dominates the "cooperate" strategy. If the iterative elimination of strictly dominated strategies reduces all players' strategy sets to singletons $\{s_i^*\}_{i \in N}$, then the game is said to be *strictly dominance solvable* and the resulting joint strategy s^* is the only essentially rational choice for the players[2]. While this provides an intuitive and simple solution method for the PD, it does not apply to the CG, since in that game no strategy dominates the other.
A concept somewhat orthogonal to that of dominance is the notion of *best replies*. Instead of ruling out the dominated strategies that are less profitable under *all* joint opponent strategies, best replies are maximally utile strategies with respect to some *fixed* opponent strategy. They are the answer to the question "what is the rational action choice for a player if she knows the actions others are going to perform in advance?" (In that sense, dominant strategies denote "rational choices if nothing is known about the behaviour of the remaining players".)

Definition B.2 (Best Replies)

- *The* **pure-strategy best reply correspondence** *BR_i for player i is a function*

$$BR_i : \Sigma_{-i} \rightarrow 2^{S_i}$$

which maps each mixed-strategy opponent profile to the non-empty, finite set

$$BR_i(\sigma_{-i}) = \{h \in S_i : u_i(e_i^h, \sigma_{-i}) \geq u_i(e_i^k, \sigma_{-i})\, \forall k \in S_i\}$$

of so-called **best replies** *to σ_{-i}.*

- *Similarly, the* **mixed-strategy best reply correspondence** *$\tilde{BR}_i$ is a function*

$$\tilde{BR}_i : \Sigma_{-i} \rightarrow \Sigma_i$$

defined by

$$\tilde{BR}_i(\sigma_{-i}) = \{h \in \Sigma_i : u_i(h, \sigma_{-i}) \geq u_i(k, \sigma_{-i})\, \forall k \in \Sigma_i\}.$$

[2]Of course, it will only be chosen if all players are rational in the sense that they seek to maximise their own profits, if they know all opponents' payoff functions and if all agents know of that knowledge.

- *As before, we will write* $BR = \times_{i \in N} BR_i$ *for the* **combined pure-strategy best-reply correspondence** *and* $\tilde{BR} = \times_{i \in N} \tilde{BR}_i$ *for the* **combined mixed-strategy best-reply correspondence**.

The formalisms just introduced may seem somewhat complex, but including mixed strategy combinations in these definitions will be of great advantage when we get to the theorem of the existence of equilibria soon.
Basically, all that best-reply correspondences do is to map some (mixed or pure) joint opponent strategy to those strategies in a player's strategy set that ensure the highest payoff. Since the payoffs of mixed strategies are convex combinations of those of pure strategies, no mixed best reply can yield a higher payoff than any of the pure best replies. However, there can be mixed best replies that are much more important than pure best replies as we shall shortly see.

Let us return to the two example games. In the PD, D is the single (pure) best reply to any (mixed or pure) opponent strategy. This supports the argument for (D,D) as the only stable solution, because it gives players another "reason" to pick D. In the CG, on the contrary, the best reply to some opponent strategy is to play A with probability 1 if the opponent plays A with probability greater than 0.5, and to play B with probability 1 if the opponent plays A with probability less than 0.5 (if the opponent chooses A with probability 0.5, any strategy is a best reply)[3]. So for the CG the concept of best replies provides us with a new result, namely that we can restrict the analysis of rational strategies to the two pure strategies, and that with only very little information about the opponent's preferences we can infer how the player should behave. Still, the problem which remains is that unless one of the players knows the precise strategy of her adversary, there is no way to respond optimally.
The notion of best replies would be only half as valuable, if it didn't serve as the starting point for the definition of equilibria, which we turn to in the following paragraph.

B.1.2 Nash equilibrium vs. pareto-optimality

The notion of Nash Equilibrium, conceived by John Nash as early as 1951 is a very simple yet very interesting solution concept for games, based on defining joint strategy combinations that are best replies to themselves, i.e. states in which none of the players has an incentive to deviate from her current strategy.

Definition B.3 (Nash Equilibrium)
Let $\Gamma = \langle N, S, u \rangle$ *be a game in normal form, and* $\tilde{BR}$ *be a combined mixed-strategy best-*

[3]To prove this, we consider some arbitrary mixed strategy of player 1 σ_1 given by the probability $p \in [0;1]$ with which player 1 will choose action A (this implies that $\sigma_1(B) = 1 - p$). If player 2 plays A with probability q using a mixed strategy σ_2, then her expected payoff will be

$$u_2(\sigma_2) = u_2(q) = pq - p(1-q) + (1-p)(1-q) - (1-p)q = 4pq - 2p - 2q + 1$$

This linear function in q has positive gradient if $p > 0.5$ (case (a)), negative gradient for $p < 0.5$ (case (b)) and is constantly 0 for $p = 0.5$ (case (c)).
If (a) is the case, then obviously the maximum of $u_2(q)$ is reached in $q = 1$. Conversely, in the case of (b), the function is maximal for $q = 0$ (since it decreases monotonically). For (c), $\forall q \in [0;1].u_2(q) = 0$ holds, so any q will be a best reply (yielding zero payoff).
Since the payoff function is *symmetric*, i.e. $u_1(\sigma) = u_2(\sigma)$, the same holds *mutatis mutandis* for the best replies of player 1.

reply correspondence for Γ. *The set*

$$\Theta = \{\sigma \in \Sigma \quad | \quad \sigma \in \tilde{BR}(\sigma)\}$$

is called the set of **Nash equilibria** *of* Γ.

Thus, if an equilibrium strategy $\theta \in \Theta$ is played, either θ itself is a best reply to it, or some equilibrium other than θ. Nash himself found out (Nash, 1951), that $\Theta \neq \emptyset$ for any finite game[4], and this is a very strong result, since it proves the existence of stable strategies, yet only stable in the sense of "once Θ is reached, it won't be left again"; it does *not* mean that some *particular* equilibrium strategy will be a stable solution, if several exist.
This problem of *equilibrium selection* occurs in many games. In fact, it is the whole point of analysing such games as the CG in which the equilibria (A, A) and (B, B) can be easily determined, but it is impossible to predict to which of these (if any) the players' strategy choices will converge.
In the PD, though, the (only) NE is to be found in (D, D), so from an equilibrium-analysis standpoint, players will fail to recognise that (C, C) would have been a more preferable choice.
Thus, the PD is one example of a game in which equilibria yield sub-optimal results for both players, which shows that this cannot be the ultimate solution for any game.

With this respect, the criterion of pareto-optimality often proves to be more appropriate than equilibria for defining rational choices, and we therefore introduce it next.

Definition B.4 (Pareto-Optimality)
Let $\Gamma = \langle N, S, u\rangle$ *be any finite game in normal form. A joint strategy combination* $\sigma \in \Sigma$ *is called* **pareto-optimal**, *if it meets the condition*

$$\forall \sigma' \in \Sigma . \sigma' \neq \sigma \Rightarrow (\exists i \in N . u_i(\sigma') > u_i(\sigma)) \Rightarrow (\exists j \in N . u_j(\sigma') < u_j(\sigma)) \,.$$

This means that a conjoint strategy is pareto-optimal, if and only if any alternative joint strategy that is better for at least one player is also worse for at least one of the other players. Hence, pareto-optimal strategies are exactly those strategies that are profitable for at least one player, while none of the other players has to sacrifice the possible payoffs she might have obtained from choosing some other strategy.
The importance of this concept becomes clear if we consider the PD game once more: changing from the safe (D, D) strategy combination to (C, C) – the pareto-optimal solution – is profitable for both players at the same time, but unfortunately (C, C) does not constitute an equilibrium, so neither of the players can be sure that her opponent will actually choose to play D, and since this "epistemic argument" is of recursive nature, neither of the players would actually be acting rationally if they chose C.
In the CG both equilibria are pareto-optimal, but even though both solution concepts coincide, the initial problem of coordinated action selection remains unsolved.

With the above definitions and observations we have given a concise overview of certain simple game-theoretic solution concepts. What we still lack, though, is the precise formulation of a solution method that can be applied to the application scenario to find optimal solutions. This will be presented in the following section.

[4] A game with a finite number of players and finite strategy spaces, that is.

B.2 A collectively and individually rational solution concept

The first solution concept for n-person cooperative games was proposed by von Neumann and Morgenstern. It is based on the assumptions that

1. players are able to communicate with each other and thus able to make agreements and to form coalitions;
2. that the distribution of coalition payoffs is "decoupled" from the obtainment of the profit, i.e. payoffs can be transferred between players.

In the settings we consider, neither of these two assumptions hold. However, it is important to analyse the existence of beneficial coalition-forming, because it might occur in an emergent way *as if* agents had agreed on them, and in fact it is our very goal to prove that these can emerge without explicit communication, at least in our application scenario (which exhibits certain properties that allow for such cooperation).

B.2.1 The characteristic form

To analyse the profit structure of games, they first need to be transformed into the so-called *characteristic form*, in which the usual payoff function is replaced by a characteristic function that reflects for each possible coalition the collective payoff that it can ensure, no matter what the remaining players do.

Definition B.5 (Characteristic Form of a Game)
A n-person game $\Gamma = \langle N, v\rangle$ in **characteristic form** *is defined by*

- *the* **set of players** $N = \{1, \ldots, n\}$ *and*
- *the* **characteristic function** $v : 2^N \to \mathbf{R}$ *which assigns a real value (the* **coalition payoff***) $v(K)$ to each $K \subseteq N$ (it is required that $v(\emptyset) = 0$).*
- *The game is called* **superadditive***, if for any two disjoint $K_1, K_2 \in 2^N$, v satisfies the inequality*
$$v(K_1 \cup K_2) \geq v(K_1) + v(K_2).$$
- *It is called* **essential***, if '$\leq$' can be replaced by strict inequality for least one K_1 and K_2 in the above condition*[5].

One possibility to convert a game in normal form into characteristic form is to define

$$v(K) = \max_{\sigma \in \Sigma_K} \min_{\tau \in \Sigma_{N-K}} \sum_{k \in K} u_k(\sigma, \tau), \tag{B.1}$$

i.e. to assign to each K the "maximin" sum of the coalition players' payoffs under the least profitable action combination of the non-coalition players[6].

[5] If this does not hold for any two K_1 and K_2, then

$$\sum_{i \in N} v(\{i\}) = v(N)$$

follows directly, which means that even the maximal coalition N cannot add to the player's individual profit, thus an analysis of possible coalitions in such games is not interesting.

[6] This calculation is not possible if forming the coalition itself produces some extra cost. Since we do not account for games with communication here, though, it can be safely applied

B.2.2 von Neumann and Morgenstern's solution: the kernel

The *kernel*, the solution method we present here, is a classical solution concept for n-person cooperative games and is applicable to the resource load game we use (hence we shall neglect alternative methods).
The kernel method is based on ruling out certain payoff distributions within coalitions if these contradict two basic assumptions, *individual rationality* and the already mentioned *pareto-optimality*. Individual rationality requires that the player obtains at least as much payoff as she could have gained by her own strength. It is most natural that this can be seen as a requirement for any payoff distribution to be called rational, because otherwise the player has no incentive to participate in the respective coalition. The notion of pareto-optimality has been formally introduced before, but seen in the light of coalitions, it denotes the property of payoff *distributions* to be optimal (in the sense that there are no alternative distributions which assign a larger payoff to some player without decreasing some other player's payoff) rather than strategy combinations. Based on these two concepts, von Neumann and Morgenstern defined the set of *imputations*, i.e. "allowed" payoff vectors. We now introduce these formally.

Definition B.6 (Individual Rationality, Pareto-Optimality (II) and Imputations)
Let $\Gamma = \langle N, v \rangle$ *be any finite game in characteristic form, and* $u = (u_1, \ldots, u_n) \in \mathbf{R}^n$ *a* **payoff vector**.

- u *is called* **individually rational**, *if and only if every player gets at least as much payoff as she could have obtained on her own, i.e. iff*

$$\forall i \in N . u_i \geq v(\{i\}).$$

- u *is called* **pareto-optimal**, *if and only if it meets the condition*

$$\sum_{i \in N} u_i = v(N)^7.$$

- *the set* $I(v)$ *of* **imputations** *for the game* Γ *is defined as the set of individually rational and pareto-optimal payoff vectors.*

Thus, imputations define the set of possible payoff distributions that are both acceptable for each individual, because they ensure *at least as much* payoff as they would have obtained "outside" the coalition and *fair*, because all of the possibly obtainable collective coalition profit is distributed among the participating players.
In order to define the kernel of the game, the concept of individual rationality is extended to that of *collective rationality*. Collective rationality means that a coalition K will only accept imputations in which it obtains at least as much payoff from the maximal coalition N that it could gain as an independent coalition by its own strength. So only imputations that respect this condition will be accepted by all possible alternative coalitions, and will hence eventually be formed.

[7] To see that this definition is equivalent to Definition B.4, note that $\sum_{i \in N} u_i \leq v(N)$ must hold for any payoff vector (since the coalition cannot distribute more than its collective payoff) and that if u were not pareto-optimal, then some other u' would exist, by which a higher profit could be distributed to one player without affecting other players' payoffs. This would imply that $\sum_{i \in N} u'_i \geq v(N)$, which cannot be the case, and hence such a u' cannot exist, so u was pareto-optimal.

Definition B.7 (The Kernel of a Game)
The **kernel** *of a game* $\Gamma = \langle N, v \rangle$ *is the set of all imputations* $u \in I(v)$ *for which the condition of collective rationality holds:*

$$\forall K \subseteq N. \sum_{i \in K} u_i \geq v(K)$$

If a payoff vector from the kernel is used to distribute $v(N)$ the game is called *stable* – no individual or group of individuals has an incentive to form a coalition other than N. Therefore the kernel of a game can be seen as a solution which respects the criteria of individual and collective rationality.

Appendix C

Proofs of theorems in Chapter 3

This chapter provides the full details of the mathematical proofs of Lemma 3.2, Corollary 3.2 and Theorem 3.2 that determine the equilibrium strategy for the simple MARLOG and a class of kernel payoff vectors. We start by introducing some auxiliary definitions that will be needed for the main proofs.

C.1 Auxiliary definitions

Definition C.1 (Resource Load)
The **load** *of some resource $r \in \mathcal{R}$ induced on it by a joint action combination $a \in \mathcal{A}$ of a set of players $\mathcal{P}$ equals the number of agents from $\mathcal{P}$ that access r by playing a, i.e.*

$$\forall r \in \mathcal{R}. \forall a \in \mathcal{A}.\ load(a, r) = \sum_{j=1}^{n} \beta(a_j)[r] \quad .$$

Definition C.2 (Resource Payoff) *The* **resource payoff** *any agent can receive for any resource $r \in \mathcal{R}$ under some fixed resource load $1 \leq l \leq n$ computes as*

$$rp(l) = \frac{v}{l \cdot T} - c$$

Using this definition, we can rewrite $u_i(a)$ as

$$u_i(a) = \sum_{r=1}^{k} \beta(a_i)[r] \cdot rp(load(a, r))$$

for any $a \in \mathcal{A}$.

Definition C.3 (Expected resource payoff)

- *For any $i \in \mathcal{P}$ with mixed strategy α_i and any $r \in \mathcal{R}$, the* **resource access probability** *$P(\alpha_i, r)$ can be computed as the sum of the probabilities of those actions in A whose binary encoding is 1 in the r-th component:*

$$P(\alpha_i, r) = \sum_{a_i \in A, \beta(a_i)[r]=1} \alpha_i(a_i) \tag{C.1}$$

- *With this definition we are able to define for any joint mixed strategy α_Q of some subset of players $Q \subseteq \mathcal{P}$ the probability with which r will be accessed by exactly l agents ($0 \leq l \leq |Q|$):*

$$P(\alpha_Q, r, l) = P(l \text{ agents access } r) \tag{C.2}$$

This probability can be computed by defining

$$X(l) = \{x \in \{0;1\}^n | x \text{ is 1 in exactly } l \text{ positions } \}$$

and taking

$$P(\alpha_Q, r, l) = \sum_{x \in X(l)} I(1) \cdot P(\alpha_1, r)^{x_1} \cdot \ldots, I(n) \cdot P(\alpha_n, r)^{x_n}, \quad I(i) = \begin{cases} 1 & \text{if } i \in Q \\ 0 & \text{else} \end{cases} .$$

- *Thus, the* **expected payoff for resource** *r $u_i(\alpha, r)$ that player i can expect to receive from r can be expressed as*

$$u_i(\alpha, r) = \sum_{l=0}^{n-1} P(\alpha_{-i}, r, l) \cdot rp(l+1), \tag{C.3}$$

i.e. the weighted sum of the probabilities that the resource (if i itself accesses it) will have **opponent load** *0,1,..., n-1 (depending on how many other agents access the resource at the same time) and the respective resource payoffs for i.*

- *It follows from this construction that the (overall) expected payoff for i can be rewritten as*

$$u_i(\alpha) = \sum_{r=1}^{k} P(\alpha_i, r) \cdot u_i(\alpha_i, r) \tag{C.4}$$

by weighing the expected resource payoffs by the probabilities with which i will access them.

C.2 Globally dominant/dominated strategies and equilibrium strategy

Lemma 3.2
For any player p_i,
(1) the pure strategy $all_i = e_i^{2^k-1}$ strictly dominates all other strategies and
(2) the strategy $none_i = e_i^0$ is strictly dominated by all other strategies.

Proof:
(1) By contradiction:
Let α_{-i} be any mixed opponent strategy. Using equation (C.2) in Definition C.3 it can be seen that for any resource r some probability distribution $P(\alpha_{-i}, r, l)$ over opponent loads $0 \leq l \leq n-1$ is induced by α_{-i}.
Now assume that there is some strategy $\alpha_i \neq all_i$ which is not strictly dominated by all_i, i.e.

$$\exists \alpha_{-i} \in \Sigma_{-i}. u_i(\alpha_i, \alpha_{-i}) \geq u_i(all_i, \alpha_{-i}).$$

The expected payoff $u_i(\alpha)$ is (by Equation (C.4)) linear in r, and the summands are non-negative (by construction, $\frac{v}{n \cdot T} - c > 0$ and $l + 1 \leq (n-1) + 1$, and $P(\alpha_{-i}, r, l) \in [0;1]$). This sum is maximal, if for all $r \in \mathcal{R}$, $P(\alpha_i, r) = 1$.
Since $\alpha_i \neq all_i$, $P(\alpha_i, r) < 1$ for at least one r (because all_i is the only strategy that accesses all resources with probability 1) there can be no such α_{-i}, hence *no* α_i is *not* strictly dominated by all_i, which proves our claim.

■

(2) As in the previous argument, assume that there exists an $\alpha_i \neq none_i$ such that

$$\exists \alpha_{-i} \in \Sigma_{-i}.u_i(none_i, \alpha_{-i}) \geq u_i(\alpha_i, \alpha_{-i}),$$

i.e. that for at least one α_i and one α_{-i}, $none_i$ will yield a payoff at least as high as α_i. Taking into account that $P(none_i, r) = 0$ for all $r \in \mathcal{R}$ (it is in fact the only strategy with this property), equation (C.4) yields $u_i(none_i, \alpha_{-i}) = 0$, so if $\alpha_i \neq none_i$, $u_i(\alpha_i, \alpha_{-i}) > 0$, for which reason no α_i and α_{-i} exist that fulfill the above condition. This proves our argument.

■

Corollary 3.2
The set of Nash equilibria of the simple multiple-access resource load game $RL = \langle n, k, v, c, T \rangle$ is given by

$$\Theta(RL) = \{all_{\mathcal{P}}\}, \tag{C.5}$$

that is the singleton set containing the joint combination of the '$2^k - 1$'-th pure strategy of all players (the joint strategy in which all players access all resources).

Proof: (trivial)
Since for every $i \in \mathcal{P}$ the strategy all_i strictly dominates *any* other strategy for *any* joint opponent strategy, it is in particular the single best reply to any joint opponent strategy (cf. App. B, Section B.1.1). Extending this argument to all agents yields for the joint mixed-strategy best reply correspondence

$$\forall \alpha \in \Sigma.\tilde{BR}(\alpha) = \{all_{\mathcal{P}}\}$$

which proves the proposition.

■

C.3 Kernel analysis

C.3.1 Characteristic form of the simple MARLOG

We first present a proof of the following lemma.
Lemma C.3.1
Let $Q \subseteq \mathcal{P}$ be an arbitrary non-empty coalition in a simple MARLOG $RL = \langle n, k, v, c, T \rangle$ with $|Q| = q$. Further, let $a_{all}^{\mathcal{P}-Q}$ be the joint strategy of all non-coalition members by which each of them accesses all resources (cf. Lemma 3.2). Then

$$\forall \alpha_Q \in \Sigma_K.all_{\mathcal{P}-Q} = \arg\min_{\alpha_{\mathcal{P}-Q}} \sum_{i \in Q} u_i(\sigma_Q, \sigma_{\mathcal{P}-Q}) \quad .$$

Proof:
First, we extend the mapping $u_i(\alpha, r)$ that returns the expected value of each resource $r \in \mathcal{R}$ for one potential accessor given some opponent behaviour to arbitrary non-empty sets of potential accessors $Q \subseteq \mathcal{P}$ in a straightforward way by defining

$$u_Q(\alpha, r) := \sum_{i \in Q} P(\alpha_i, r) \cdot u_i(\alpha, r), \tag{C.6}$$

so that the coalition would potentially receive the sum of its participant payoffs if it accessed r.
Now for any two l_1, l_2 with $n \geq l_1 > l_2 > 0$, obviously

$$\frac{v}{l_1 \cdot T} - c < \frac{v}{l_2 \cdot T} - c$$

holds, so that

$$rp(n - q + 1) \leq \sum_{l=0}^{n-q} P(\alpha_{\mathcal{P}-Q}, r, l) \cdot rp(l + 1) \tag{C.7}$$

because a convex combination linear in l can never yield a smaller value than the minimum of the summed entries.
Hence, non-coalition members $\mathcal{P} - Q$ will reduce the payoff for Q maximally if the probability $P(\alpha_{\mathcal{P}-Q}, r, l)$ is one for $l = n - q$ and zero for all $l < n - q$ for all $r \in R$, i.e. if they all play $all_{\mathcal{P}-Q}$ which proves the above proposition.
∎

So to define the characteristic function, we are left with the question "how can $u_Q(\alpha_Q, \alpha_{\mathcal{P}-Q}, r)$ be maximised for every resource if the opponent coalition plays $all_{\mathcal{P}-Q}$?" The following auxiliary definitions are necessary to resolve this issue.

Definition C.4 (Coalition access payoff for a resource)
For any non-empty coalition $Q \subseteq \mathcal{P}$, we define the **coalition access payoff** *in terms of the load induced by the joint Q-strategy α_Q additionally to any resource $r \in \mathcal{R}$, i.e.*

$$CAP(x, r) = x \cdot rp(n - q + x)$$

for $x \in [0; q]$ ($q = |Q|$), and x can be deduced from any α_Q by taking

$$x = \sum_{i \in Q} P(\alpha_i, r)$$

for any $r \in \mathcal{R}$.

The intuition behind this definition is that every Q-action defines an additional load x (that may be different for every resource). $CAP(x, r)$ (whose value does actually not depend on r in terms of x, for which reason we will often write $CAP(x)$) computes the sum of all payoffs the x accessors of r receive. All this is done *only* for the case in which the opponent coalition plays in the most unfavourable way.
We now define the maximum of this quantity for integer values of x as a function of the opponent load $l = n - q$.

Definition C.5 (Optimal resource access for a coalition)
Let $x_{opt}(l) := \sqrt{\frac{vl}{cT}} - l$ *for any non-empty coalition* Q *of size* $q = |Q|$.
The **optimal resource access** $opt(q)$ *is defined as*

$$opt(l) := \arg\max_{l' \in \{l_1, l_2\}} CAP(l'), \qquad l_1 = \lfloor x_{opt}(l) \rfloor, \quad l_2 = \lceil x_{opt}(l) \rceil \quad .$$

Simple calculus suffices to show that the CAP-function reaches its (only) maximum at $\sqrt{\frac{vl}{cT}} - l$. All the optimal resource access does is to derive the maximal integer-valued additional load that Q can induce on the resource from the (possibly real-valued) maximum x_{opt} (opt and x_{opt} of course depend on the size of the coalition q).
Let $M_Q \subseteq \Sigma_Q$ be the set $M_K := \{\alpha_Q \in \Sigma_K \quad | \quad \forall r \in \mathcal{R}.P(\alpha_Q, r, opt(n-q)) = 1\}$. M_Q is the set of all joint strategies of players in Q in which exactly $opt(n-q)$ of these agents accesses every resource. We present the following theorem.

Theorem C.3.1
Let $RL = \langle n, k, v, c, T \rangle$ and $Q \subseteq \mathcal{P}$ be an arbitrary non-empty subset of $\mathcal{P}$. Then M_Q defines the maximin joint strategy set of Q, i.e.

$$M_Q = \arg\max_{\alpha_Q \in \Sigma_Q} \min_{\alpha_{\mathcal{P}-Q}} \sum_{i \in Q} u_i(\alpha_Q, \alpha_{\mathcal{P}-Q}).$$

Proof: By contradiction:
Assume the existence of some $\alpha_Q \notin M_Q$ with

$$\sum_{i \in Q} u_i(\alpha_Q, all_{\mathcal{P}-Q}) > \sum_{i \in Q} u_i(\mu_Q, all_{\mathcal{P}-Q})$$

for any $\mu_Q \in M_Q$. (All expected payoffs of strategies in μ_Q are equal, so it suffices to pick an arbitrary μ_Q.)
Then by the definitions of μ_Q and $all_{\mathcal{P}-Q}$, and because of $\alpha_Q \neq \mu_Q$, at least one $r \in R$ has load greater than $opt(n-q)+n-q$ or smaller than $opt(n-q)+n-q$ with non-zero probability. We have shown that the expected resource payoff $u_Q(\alpha_Q, a_{all}^{\mathcal{P}-Q}, r)$ is lower than if r had load $opt(n-q)+n-q$ in both cases, and any joint strategy of Q other than μ_Q assigns non-zero probability to the sub-maximal components of the expected resource payoff in a convex combination (cf. Equation (C.6)) and is thus sub-optimal. This holds in the same way for the overall expected payoff by summing over all resources (which are independent of each other).

■

This provides a maximin strategy for any coalition Q, i.e. a coalition payoff that can be ensured irrespective of the behaviour of non-coalition players, and this enables us to define the game in characteristic form.

Definition C.6 (Characteristic form of the simple MARLOG)
For any $RL = \langle n, k, v, c, T \rangle$, *the characteristic form* $CF(RL) = \langle \mathcal{P}, v \rangle$ *is defined by*

- *the set of players* $\mathcal{P}$ *and*
- *the characteristic function* $v : 2^{\mathcal{P}} \to \mathbf{R}$, *which maps every non-empty subset players* $Q \subseteq \mathcal{P}$ *to the coalition profit* $v(Q)$ *defined by*

$$v(Q) = \sum_{i \in Q} u_i(\mu_Q, all_{\mathcal{P}-Q}) = k \cdot opt(n-q) \cdot rp(opt(n-q)+n-q)$$

for an arbitrary $\mu_Q \in M_Q$.

C.3.2 Kernel payoff distributions

To prove that the kernel of the game is non-empty (in fact, we define a payoff vector that is in the kernel of the game), we first need to prove some properties of the characteristic function.

Lemma C.3.2
The MARLOG in its characteristic form $CF(RL)$ is superadditive.

Proof:
As in the definition of super-additivity (cf. Definition B.5, App. B), we have to show that for any two disjoint $Q_1, Q_2 \in 2^{\mathcal{P}}$, v satisfies the inequality

$$v(Q_1 \cup Q_2) \geq v(Q_1) + v(Q_2) \quad . \tag{C.8}$$

We first prove that $CAP(x_{opt}(l))$ is a function that decreases in l.
Simplifying $CAP(\sqrt{\frac{vl}{cT}} - l)$ yields the function

$$CAP(x_{opt}(l)) = c \cdot l - 2\sqrt{\frac{v}{cT}} \cdot \sqrt{l} + \frac{v}{T},$$

which can be converted into a quadratic function if we substitute d for $\sqrt{l}$, i.e.

$$CAP(x_{opt}(d)) = c \cdot d^2 - 2\sqrt{\frac{v}{cT}} \cdot d + \frac{v}{T}.$$

The first derivative of this function $CAP'(x_{opt}(d)) = 2c \cdot d - 2\sqrt{\frac{v}{cT}}$ is negative for $l < \frac{v}{cT}$, but since we require that $n < \frac{v}{cT}$ and $l \leq n$ by definition of Q this holds for all l, hence $CAP(x_{opt}(l))$ decreases with strict monotonicity.
Since $CAP(x_{opt}(l))$ is a function that decreases in l, it obviously increases with decreasing $l = n - q$, i.e. the resource a coalition can ensure for its members by accessing a resource increases with the size of the coalition.
Next, we prove that

$$CAP(x_{opt}(|Q_1 \cup Q_2|)) > CAP(x_{opt}(|Q_1|)) + CAP(x_{opt}(|Q_2|)) \quad . \tag{C.9}$$

Using the same explicit function representation $CAP(x_{opt}(l))$ as in the previous proof, the statement becomes (after sufficient simplification) equivalent to

$$\begin{aligned} CAP(x_{opt}(n - q_1 - q_2)) &> CAP(x_{opt}(n - q_1)) + CAP(x_{opt}(n - q_2)) \\ &\Leftrightarrow \sqrt{1 - \frac{q_1}{n}} + \sqrt{1 - \frac{q_1}{n}} - \sqrt{1 - \frac{q_1}{n} - \frac{q_2}{n}} > \frac{1}{2}, \end{aligned}$$

if $q_1 = |Q_1|$ and $q_2 = |Q_2|$. Since $\frac{q_1}{n} < 1$, $\frac{q_2}{n} < 1$ and $q_1 + q_2 < n$ must hold, the proof of this statement can be reduced to proving

$$\forall 0 \leq a, b \leq 1, a + b \leq 1. \sqrt{1 - a} + \sqrt{1 - b} - \sqrt{1 - a - b} > \frac{1}{2}.$$

It turns out that the left-hand side of the inequality is even greater than one, which yields as a final condition

$$ab \geq 0,$$

and since q_1 and q_2 are non-empty, this condition holds invariably. Finally, we can use the transformation

$$v(Q) = k \cdot CAP(opt(n-q))$$

and inequality (C.9) (which we just proved) to conclude that

$$v(Q_1 \cup Q_2) \geq v(Q_1) + v(Q_2).$$

taking into account that the constant k has no effect on the inequality, and that $x_{opt}(l)$ can safely be replaced by $opt(l)$ without affecting the result.

■

It follows from the above considerations that $v(Q)$ is maximal for $Q = \mathcal{P}$. For the case of $Q = \mathcal{P}$ no opponent coalition exists, so $l = 0$; furthermore $CAP(x)$ is maximal for $opt(0) = 1$, because it decreases for all $x > \sqrt{\frac{vl}{cT}} - l = \sqrt{\frac{v \cdot 0}{cT}} - 0 = 0$, and it is undefined for $x = 0$ (hence 1 is the integer that yields the highest value for $CAP(x)$).
Thus,

$$v(\mathcal{P}) = k \cdot rp(1)$$

and this is the maximally possible value of $v(Q)$. We define a payoff vector $u^* = (u_1^* \ldots, u_n^*) \in \mathbf{R}^n$ by letting

$$\forall i \in \mathcal{P}.u_i^* = \frac{v(\mathcal{P})}{n} = \frac{1}{n} \cdot k \cdot rp(1), \tag{C.10}$$

and claim that this vector is in the kernel of $CF(RL)$.

Theorem 3.2
The vector u^* is in the kernel of the game $CF(RL)$ where RL is an arbitrary simple MARLOG in normal form.

Proof:
Recalling the definition of the kernel of a game (cf. App. B, Definitions B.6 and B.7), we have to show that

1. $\forall i \in \mathcal{P}.u_i^* \geq v(\{i\})$
2. $\sum_{i \in \mathcal{P}} .u_i^* = v(\mathcal{P})$
3. $\forall Q \subseteq \mathcal{P}. \sum_{i \in Q} u_i^* \geq v(Q)$

To see that 1. is the case it suffices to compare $v(\{i\})$ to u_i^*, i.e. to check whether

$$v(\{i\}) = k \cdot rp(n) \leq \frac{1}{n} \cdot k \cdot rp(1) = u_i^*$$

hold. Simple algebraic transformation shows that this is true for any $n \geq 1$ which is given by the game definition.
Claim 2. is true by definition of u^* and 3. can be proven by verifying that

$$\sum_{i \in Q} u_i^* = q \cdot \frac{k}{n} \cdot rp(1) \geq k \cdot CAP(opt(n-q)) = v(Q)$$

which, given that $opt(n-q)$ is some $1 \leq q' \leq Q$ (and by leaving out k) can be transformed into the inequality

$$\frac{q}{n} \cdot rp(1) \geq q' \cdot rp(n - q + q')$$

Simple algebra reduces the statement $\frac{q'}{n-q+q'} \leq \frac{q}{n}$ to $q' \leq q$ which is always true, so multiplying by $\frac{v}{T}$ yields

$$\frac{q'}{n - q + k'} \cdot \frac{v}{T} \leq \frac{q}{n} \cdot \frac{v}{T}.$$

Thus it remains to prove that $-cq' \leq -c\frac{q}{n}$ which is trivial because it holds for all $q' \geq \frac{q}{n}$, and $\frac{q}{n} \leq 1$ while $q' \geq 1$ by definition. This completes the proof.

■

References

Agre, P. E., and Chapman, D. (1987). Pengi: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 268–272). San Mateo, CA: Morgan Kaufmann.

Aumann, R., and Maschler, M. B. (1996). *Repeated Games with Incomplete Information.* Cambridge, MA: The MIT Press.

Axelrod, R. (1984). *The Evolution of Cooperation.* New York, NY: Basic Books.

Bachem, A., Hochstättler, W., and Malich, M. (1996). The Simulated Trading Heuristic for Solving Vehicle Routing Problems. *Discrete Applied Mathematics, 65*(1-3), 47–72.

Bakker, P., and Kuniyoshi, Y. (1996). Robot See, Robot Do: An Overview of Robot Imitation. *AISB Workshop on Learning in Robots and Animals, Brighton, UK.*

Bandura, A. (1977). *Social Learning Theory.* Englewood Cliffs, NJ: Prentice-Hall.

Bicchieri, C., Pollack, M., and Rovelli, C. (1996). The potential for evolution of cooperation among Web agents. In S. Sen (Ed.), *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems, Stanford, CA* (pp. 6–11).

Brooks, R. A. (1991). Intelligence Without Representation. *Artificial Intelligence, January 1991(1-3), 47,* 139–159.

Bouron, T., and Collinot, A. (1992). SAM: A Model to Design Complex Computational Social Agents. In B. Neumann (Ed.), *Proceedings of the 10th European Conference on Artificial Intelligence, Vienna, Austria* (pp. 239–243).

Carmel, D., and Markovitch, S. (1996). Opponent Modeling in Multi–Agent Systems. In G. Weiß and Sandip Sen (Eds.), (Vol. 1042, pp. 40–52). Berlin: Springer.

Claus, C., and Boutilier, C. (1998). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)* (pp. 746–752). Menlo Park, CA: AAAI Press.

Corkill, D. (1991). Blackboard Systems. *AI Expert, 6*(9), 40–47.

Cybenko, G. (1988). *Continuous valued neural networks with two hidden layers are sufficient* (Technical Report). Medford, MA: Department of Computer Science, Tufts University.

Dagaeff, T., Chantemargue, F., and Hirsbrunner, B. (1997). Emergence-based Cooperation in a Multi-agent System. In *Proceedings of the Second European Conference on Cognitive Sciences (ECCS'97), Manchester, UK.*

Davis, R., and Smith, R. G. (1981). *Negotiation as a Metaphor for Distributed Problem Solving* (AI Memo No. 624). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology. (Also published in *Readings in Distributed Artificial Intelligence*, A. H. Bond and L. Gasser (Eds.), (pp. 333–356), San Mateo, CA: Morgan Kaufmann, 1988.)

Dawkins, R. (1976). *The Selfish Gene.* Oxford: Oxford University Press.

Demiris, J., and Hayes, G. (1996). Imitiative Learning Mechanisms in Robots and Humans. In V. Klingspor (Ed.), *Proceedings of the Fifth European Workshop on Learning Robots*, Bari, Italy.

Doran, J. E. (1998). Social Simulation, Agents and Artificial Societies (Extended Abstract of Invited Address). In *Proceedings of Third International Conference on Multi-Agent Systems, Paris, France* (pp. 4–5).

Drogoul, A., and Ferber, J. (1994). Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies. In C. Castefranchi and E. Werner (Eds.), *Proceedings of the 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Artificial Social Systems* (Vol. 830, pp. 3–23). Berlin: Springer.

Durfee, E. H. (1991). Special Issue on DAI: Ten years later. *IEEE Transactions on Systems, Man, and Cybernetics*, *21* (6).

Durfee, E. H., and Lesser, V. R. (1988). Using Partial Global Plans to Coordinate Distributed Problem Solvers. In A. H. Bond and Les Gasser (Eds.), *Readings in Distributed Artificial Intelligence* (pp. 285–293). San Mateo, CA: Morgan Kaufmann.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML – A Language and protocol for Knowledge and Information Exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence, Seattle, WA* (pp. 126–136).

Fischer, K., Ruß, C., and Vierke, G. (1998). *Decision Theory and Coordination in Multi-agent Systems* (Research Report No. RR-98-02). Saarbrücken, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz.

Freund, Y., Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., and Schapire, R. E. (1995). Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries. In *36th Annual Symposium on Foundations of Computer Science, Los Alamitos (FOCS'95)* (pp. 332–343). IEEE Computer Society Press.

Fudenberg, D., and Tirole J. (1991). *Game Theory.* Cambridge, MA: The MIT Press.

Genesereth, M. R., Ginsberg, M. L., and Rosenschein, J. S. (1986). Cooperation without Communication. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-86)* (pp. 561–567). Menlo Park, CA: AAAI Press.

Georgeff, M. P. (1984). A Theory of Action for Multi-Agent Planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence, Austin, TX (AAAI-84).* Menlo Park, CA: AAAI Press.

Gerber, C., and Jung, C. G. (1998). Resource Management for Boundedly Optimal Agent Societies. In *Proceedings of the Workshop on Monitoring and Control of Real-Time Intelligent Systems (ECAI-98)*, Brighton, UK.

Gmytrasiewicz, P. J. (1996). An Approach to User Modeling in Decision Support Systems. In *Proceedings of the Fifth Conference on User Modeling (UM96)* (pp. 121–128). User Modeling.

Gmytrasiewicz, P. J., and Durfee, E. H. (1995). A Rigorous, Operational Formalization of Recursive Modeling. In V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA* (pp. 125–132). Cambridge, MA: The MIT Press.

Gmytrasiewicz, P. J., Durfee, E. H., and Wehe, D. K. (1991). The Utility of Communication in Coordinating Intelligent Agents. In T. Dean and K. McKeown (Eds.), *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 166–172). Menlo Park, CA: AAAI Press.

Gmytrasiewicz, P. J., Noh, S., and Kellogg, T. (1998). Bayesian Update of Recursive Agent Models. *User Modeling and User-Adapted Interaction*, *8*(1-2), 49–69.

Grosz, B. J., and Snider, C. L. (1988). *Distributed Know-how and Acting: Research on Collaborative Planning.*

Hewitt, C. (1977). Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, *8*(3), 323–364.

Hewitt, C. (1991). Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial Intelligence*, *47*(1-3), 79–106.

Hogg, L. M., and Jennings, N. R. (1997). Socially Rational Agents. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents, Boston, MA* (pp. 61–63).

Horvitz, E. J., Cooper, G. F., and Heckerman, D. E. (1989). Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study. In N. S. Sridharan (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, MI* (pp. 1121–1127). San Mateo, CA: Morgan Kaufmann.

Jennings, N. (1996). Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings (Eds.), *Foundations of Distributed Artificial Intelligence* (pp. 187–210). New York, NY: John Wiley & Sons.

Jennings, N. R., and Campos, J. R. (1997). Towards a Social Level Characterisation of Socially Responsible Agents. *IEE Proceedings on Software Engineering*, *144*(1), 11–25.

de Jong, E. D. (1999). Autonomous Concept Formation. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm, Sweden (IJCAI-99).* San Mateo, CA: Morgan Kaufmann.

Jung, C. G. (1998). Experimenting with Layered, Resource-Adapting Agents in the Robocup Simulation. In *Proceedings of the 1998 Robocup Workshop, Paris, France.*

Jung, C. G., and Fischer, K. (1997). A Layered Agent Calculus with Concurrent, Continuous Processes. In M. P. Singh, A. S. Rao, and M. J. Wooldridge (Eds.), *Preproceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages.* (Also published as *Intelligent Agents IV*, Lecture Notes in Artificial Intelligence, Springer (Vol. 1365), Berlin: Springer, 1998.)

Kalenka, S., and Jennings, N. R. (1995). On Social Attitudes: A Preliminary Report. In *Proceedings of the First International Workshop on Decentralised Intelligent Multi-Agent Systems, Krakov, Poland* (pp. 233–240).

Kennedy, J. (1997). Minds and cultures: Particle swarm implications. In *Socially Intelligent Agents: Papers from the 1997 AAAI Fall Symposium* (pp. 67–72). Menlo Park, CA: AAAI Press. (published as AAAI Technical Report FS-97-02)

Kuniyoshi, Y., Rougeaux, S., Ishii, M., Kita, N., Sakane, S., and Kakikura, M. (1994). Cooperation by Observation – The Framework and Basic Task Patterns. In *Proceedings of the International Conference on Robots and Automation.*

Lansky, A. L. (1989). *A Perspective on Multi-Agent Planning* (Tech. Rep. No. Technical Note 474). SRI International.

Lomborg, B. (1994). Game Theory Versus Multiple Agents: The Iterated Prisoner's Dilemma. In *Proceedings of the 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Artificial Social Systems* (Vol. 830, pp. 69–93). Berlin: Springer.

Luce, R. D., and Raiffa, H. (1958). *Games and Decisions: Introduction and Critical Survey.* New York, NY: John Wiley & Sons.

Malone, T. W., and Crowston, K. (1991). *Toward an Interdisciplinary Theory of Coordination* (Center for Coordination Science Technical Report No. CCS-TR-120). MIT Sloan School of Management.

von Martial, F. (1993). Planen in Multi-Agenten Systemen. In H. J. Müller (Ed.), (pp. 92–132). Mannheim, Germany: BI-Wissenschaftsverlag.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Müller, H. J. (Ed.). (1993). *Verteilte Künstliche Intelligenz: Methoden und Anwendungen.* Mannheim, Germany: BI-Wissenschaftsverlag.

Müller, J. P. (1996). *The design of intelligent agents: A Layered Approach* (Vol. 1177). New York, NY: Springer-Verlag Inc.

Müller, J. P., and Pischel, M. (1993). InteRRaP: Eine Architektur zur Modellierung flexibler Agenten. In H. J. Müller (Ed.), (pp. 45–54). Mannheim, Germany: BI-Wissenschaftsverlag.

Myerson, R. B. (1991). *Game Theory: Analysis of Conflict.* Harvard University Press.

Nwana, H. S. (1995). Software Agents: An Overview. *Knowledge Engineering Review, 11*(2), 205–244.

Pfeffer, A. J., and Koller, D. (1997). Representations and Solutions for Game-theoretic Problems. *Artificial Intelligence, 94*(1), 167–215.

Price, B., and Boutiliter, C. (1999). Implicit Imitation in Multiagent Reinforcement Learning. In J. M. Vidal (Ed.), *Proceedings of the Workshop on Agents learning about, from and with other Agents (IJCAI-99), Stockholm, Sweden.*

Rao, A. S., Georgeff, M. P., and Sonenberg, E. A. (1992). Social Plans: A Preliminary Report. In E. Werner and Y. Demazeau (Eds.), *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)* (pp. 57–76). Amsterdam: Elsevier Science Publishers.

Rosin, C. D., and Belew, R. K. (1996). A competitive approach to game learning. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory* (pp. 292–302). New York, NY: ACM Press.

Rovatsos, M., and Lind, J. (1999). Learning Cooperation in Repeated Games. In J. M. Vidal (Ed.), *Proceedings of the Workshop on Agents learning about, from and with other Agents (IJCAI-99), Stockholm, Sweden.*

Rubenstein, A. (1985). *Finite Automata Play the Repeated Prisoner's Dilemma* (Tech. Rep.). London: London School of Economics. (ST/ICERD Discussion Paper 85/109.)

Russell, S. J., and Norvig, P. (1995). *Artificial Intelligence. A Modern Approach.* Englewood Cliffs, NJ: Prentice-Hall.

Schaerf, A., Shoham, Y., and Tennenholtz, M. (1995). Adaptive Load Balancing: a study in co-learning. In S. Sen (Ed.), *IJCAI-95 Workshop on Adaptation and Learning in Multi-Agent Systems* (pp. 78–83).

Schillo, M., Funk, P., and Rovatsos, M. (1999). Who can you Trust: Dealing with Deception. In R. Falcone (Ed.), *Proceedings of the Workshop "Deception, Fraud and Trust" of the Autonomous Agents Conference, Seattle, WA.*

Seel, N. R. (1991). Perspectives for Distributed Artificial Intelligence. *AISB Quarterly*(76).

Sen, S., and Sekaran, M. (1996). Multiagent Coordination with Learning Classifier Systems. *Lecture Notes in Computer Science, 1042.* Berlin: Springer.

Sen, S., Sekaran, M., and Hale, J. (1994). Learning to Coordinate without Sharing Information. In *Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1* (pp. 426–431). Menlo Park, CA: AAAI Press.

Shoham, Y. (1996). The Open Scientific Borders of AI, and the Case for Economics. *ACM Computing Surveys, 28*(4es), 11.

Shoham, Y., and Tennenholtz, M. (1997). On the emergence of social conventions: Modeling, analysis and simulations. *Artificial Intelligence, 94*(1), 139–166.

Simon, H. A. (1982). Models of bounded rationality. In *Behavioural economics and business organization* (Vol. II). Cambridge, MA: The MIT Press.

Smith, J. M. (1982). *Evolution and the Theory of Games.* Cambridge, England: Cambridge University Press.

Stirling, W. (1994). Multi-Agent Coordinated Decision-making Using Epistemic Utility Theory. In *Proceedings of the 4th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Artificial Social Systems* (Vol. 830). Springer, Berlin.

Stone, P. (1998). *Layered Learning in Multi-Agent Systems.* Unpublished doctoral dissertation, School of Computer Science, Carnegie Mellon University.

Stone, P., and Veloso, M. (1999). Layered Learning. In J. M. Vidal (Ed.), *Proceedings of the Workshop on Agents learning about, from and with other Agents (IJCAI-99).* Stockholm, Sweden.

Sundermayer, K. (1993). Modellierung von Agentensystemen. In H. J. Müller (Ed.), (pp. 22–54). Mannheim, Germany: BI-Wissenschaftsverlag.

Tesfatsion, L. (to appear). Agent-based Computational Economics: A Brief Guide to the Literature. In *Reader's Guide to the Social Sciences.* London, UK: Fitzroy-Dearborn.

Thomas, J. D., and Sycara, K. (1998). Heterogeneity, Stability, and Efficiency in Distributed Systems. In *Proceedings of the Third International Conference on Multi-Agent Systems, paris, france.*

Vidal, J. M., and Durfee, E. H. (1998a). Learning Nested Models in an Information Economy. *Journal of Experimental and Theoretical Artificial Intelligence, 10*(3), 291–308.

Vidal, J. M., and Durfee, E. H. (1998b). The Moving Target Function Problem in Multi-Agent Learning. In *Proceedings of the Third International Conference on Multi-Agent Systems.* Paris.

Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Machine Learning Journal, 8.* (Special Issue on Reinforcement Learning)

Weibull, J. (1995). *Evolutionary Game Theory.* Cambridge, MA: The MIT Press.

Weiß, G. (1996). Adaptation and learning in multi-agent systems: Some remarks and a bibliography. In G. Weiß and Sandip Sen (Eds.), (Vol. 1042, pp. 1–21). Berlin: Springer.

Weiß, G., and Dillenbourg, P. (1999). What is 'multi' in multiagent learning? In P. Dillenbourg (Ed.), *Collaborative learning. Cognitive and computational approaches* (pp. 64–80). Pergamon Press. (pre-version)

Weiß, G., and Sen S (Eds.). (1996). *Adaption and learning in multi-agent systems* (Vol. 1042). Berlin: Springer.

Wooldridge, M., and Jennings, N. R. (1995). Agent Theories, Architectures, and Languages: A Survey. In M. J. Wooldridge and Nicholas R. Jennings (Eds.), *Proceedings of the ECAI-94 Workshop on Agent Theories, Architectures and Languages: Intelligent Agents I* (Vol. 890, pp. 1–39). Springer, Berlin.

Zlotkin, G., and Rosenschein, J. S. (1989). Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 912–917).

Zlotkin, G., and Rosenschein, J. S. (1993). Negotiation with Incomplete Information about Worth: Strict versus Tolerant Mechanisms. In G. Schlageter, M. Huhns, and M. P. Papazoglou (Eds.), *Proceedings of the International Conference on Intelligent and Cooperative Information Systems, Rotterdam, the Netherlands* (pp. 175–184).