Learning and Coordination in Sequential Multiagent Problems

Thomas Kemmerich

Dissertation
in Computer Science

submitted to the

Faculty of Electrical Engineering,
Computer Science and Mathematics

University of Paderborn

in partial fulfillment of the requirements for the degree of

doctor rerum naturalium
(Dr. rer. nat.)

Paderborn, May 2012

To my family.

Acknowledgements

Working on this thesis was a highly interesting experience that taught me a lot. At the final stages of this process, it is time to express my gratitude to all the people who supported me during the last years.

First and foremost, I would like to thank my advisor, Prof. Dr. Hans Kleine Büning, for his constant support and his invaluable advice whenever it was needed. Also, I am grateful to Prof. Dr. Friedhelm Meyer auf der Heide who agreed to review this thesis.

Furthermore, I am indebted to all members of the Knowledge-Based Systems Group who created an inspiring and amicable working atmosphere. In particular, I would like to thank Dr. Natalia Akchurina, Isabela Anciutti, Gerd Brakhane, Dr. Uwe Bubeck, Dr. Theodor Lettmann, Timo Klerx, Felix Mohr, Asmir Vodenčarević, and last but not least Yuhan Yan. In addition, I have really enjoyed the numerous discussions and the time spent with Dr. Markus Eberling. The same applies to Michael Baumann, who also agreed to proofread this thesis and who provided valuable suggestions. Additionally, I am much obliged to Simone Auinger for her support.

I would like to take the opportunity to express my gratitude to the team of the International Graduate School, in particular to Prof. Dr. Eckhard Steffen and Astrid Canisius.

Special thanks I owe to Andreas Kumlehn for proofreading and improving the readability of this thesis. Also, I would like to thank Dirk Meister and Marc-André Preuß for their useful comments and questions. The supervision of many students was an enlightening experience and I am thankful for the various discussions we had.

No words can express my deepest gratitude to my family, in particular to my parents, Angelika and Heinz, and my sister Nicole.

Last but not least, I am very grateful to Alexandra for her lovely support, patience, and understanding.

Thomas Kemmerich
Paderborn, May 2012

Abstract

Computer-based systems change our everyday life. For instance, mobile applications offer location-based services that allow us to find friends or restaurants nearby. Such services are often based on complex distributed systems. The involved devices have to cooperate in order to realize the desired functionality and, thus, they have to coordinate their actions depending on the current situation. From a developer's point of view, it is hardly possible to foresee every scenario and to elaborate appropriate strategies in advance. Hence, one way to reach the desired service quality levels are intelligent systems, whose components are able to learn strategies on their own.

This thesis deals with learning in such large and distributed systems. We focus on settings that show an interesting and frequently observable structure. Namely, devices (agents, hereafter) are often confronted with a sequence of different—probably, but not necessarily comparable—situations. The agents have to solve a common task and, thus, have to learn a good coordinated behavior for each situation. When a new setting occurs, old strategies might either become useless or establish a good basis for further adaption, depending on the similarity of the previous and the new situation.

Models for these problems quickly become complex and introduce research questions on their own. Hence, to focus on the learning process, we will deal with simple sequences of stateless games. Each game is played repeatedly for a certain number of iterations, which the agents do not know in advance, before a new game occurs. We develop a model, called *sequential stage games (SSG)*, that formalizes such problems, and establish some required foundations. Then, we propose *Distributed Stateless Learning (DSL)*, which is a multiagent reinforcement learning approach for cooperative SSGs. To speed up learning in systems with thousands of agents, we also develop several coordination strategies. These strategies coordinate the agents' action choices, e.g., using communication or by storing learned knowledge on so-called storage media in the environment.

We provide a careful theoretical analysis of our approach and prove its convergence to (near-)optimal solutions, if each game is played sufficiently long. Furthermore, we show that DSL enables learning under agent-individual noised reward perceptions. Our theoretical results are supported by empirical analyses, which are mainly conducted in a variant of the *Iterative Agent Partitioning Problem (IAPP)*. The IAPP is a multi-objective optimization problem that requests a repeated partitioning of agents onto targets according to some global performance goals. In addition, we investigate several aspects that are related to the general problem setting.

To summarize, we provided first insights into learning and coordination in sequences of games and developed efficient approaches for the considered scenarios.

Zusammenfassung

Unser Alltag wird immer stärker durch Computer-gestützte Systeme beeinflusst. Hinter vielen mobilen Diensten verbergen sich beispielsweise komplexe verteilte Systeme, die sich aus einer Vielzahl kooperierender Komponenten zusammensetzen. Aus Entwicklersicht ist es kaum möglich, alle Komponenten mit Verhaltensstrategien für alle erdenklichen Situationen auszustatten. Eine Möglichkeit trotzdem ein vorgegebenes Maß an Service Qualität zu erreichen, besteht in der Entwicklung von intelligenten Systemen, die selbstständig und kontinuierlich lernen, sich möglichst gut an die jeweiligen Situationen anzupassen.

In dieser Arbeit beschäftigen wir uns damit, wie Geräte (oder Agenten) in großen verteilen Systemen autonom lernen können, ihre Aktionen so zu koordinieren, dass eine gemeinsame Aufgabe möglichst gut gelöst wird. Insbesondere konzentrieren wir uns auf Probleme, die aus einer Abfolge verschiedener Situationen bestehen. Wir nehmen an, dass eine Problemlösung (Strategie) während des Betriebs gelernt wird und nur solange relevant ist, wie die aktuelle Situation andauert. In der folgenden Situation können gelernte Strategien im einfachsten Fall angepasst und weiter verwendet werden, oder sie müssen wegen zu großen Unterschieden verworfen werden.

Wir stellen ein Modell auf, mit dessen Hilfe solche Probleme formal modelliert werden können und entwickeln darauf aufbauend einen verteilten Lernansatz, der von jedem Agenten im System ausgeführt wird. Dieses sogenannte *Distributed Stateless Learning (DSL)* Verfahren lernt mittels *Reinforcement Learning*, d.h. es lernt welche individuelle Aktion im Zusammenspiel mit dem Verhalten der anderen Agenten aktuell sinnvoll ist, indem Aktionen ausgeführt und numerische Rückmeldungen über die Aktionsgüte in den Lernprozess einbezogen werden. Um das Lernen in sehr großen Systemen zu beschleunigen, präsentieren wir zusätzlich verschiedene Koordinationsstrategien, die beispielsweise Wissen in der Umgebung ablegen oder bei denen Agenten lokal Informationen austauschen.

Der entwickelte Ansatz wird theoretisch und empirisch untersucht, und wir zeigen, dass DSL in der Lage ist, optimale oder beinahe optimale Lösungen zu lernen. Als Testumgebung wird ein generisches Aufteilungsproblem betrachtet, in dem sich Agenten auf eine Menge von Zielen in der Umgebung verteilen müssen, so dass mehrere gegensätzliche Ziele optimiert werden. Neben diesen Ergebnissen stellen wir weitere Verfahren und Konzepte vor und geben Einblicke in relevante Fragestellungen, wie bspw. das Lernen mit verrauschten Wahrnehmungen.

Am Ende bleibt festzuhalten, dass diese Arbeit erste Einsichten in das betrachtete Problemszenario bietet und dass effizientes Lernen in diesem Umfeld mittels DSL möglich ist.

Contents

1 Introduction **1**
1.1 Background . . . 3
1.2 Structure . . . 3

Part I General Background **7**

2 Background and Related Work **9**
2.1 Multiagent Systems . . . 9
2.2 The Online Partitioning Problem . . . 12
2.3 Multirobot Task Allocation . . . 15
2.4 Swarm Intelligence . . . 17
2.5 Transfer Learning . . . 18
2.6 Optimization . . . 21

3 Reinforcement Learning **23**
3.1 Introduction . . . 23
3.2 Single Agent Reinforcement Learning . . . 24
3.3 Multiagent Reinforcement Learning . . . 33
3.4 Reward Shaping . . . 42
3.5 Conclusion . . . 44

Part II Foundations **45**

4 Sequential Stage Games and Engineered Rewards **47**
4.1 Sequential Stage Games . . . 47
4.2 Engineered Reward-Based Models . . . 52

5 The Iterative Agent Partitioning Problem **59**
5.1 Problem Motivation . . . 59
5.2 Formal Definition . . . 61
5.3 Properties . . . 65
5.4 Message Based Optimal Algorithms . . . 69
5.5 Complexity . . . 74
5.6 Summary . . . 75

6 Storage Media **77**
6.1 Concept . . . 77
6.2 Strategic Positioning . . . 83

6.3 Influence of Obstacles 85
6.4 Greedy Placement Strategy 87
6.5 Summary 90

Part III Algorithms **91**

7 Reinforcement Learning with Noisy Rewards **93**
7.1 Basic Definitions 93
7.2 Deterministic Q-Learning with Noisy Rewards 95
7.3 Q-Learning with Noisy Rewards and Optimistic Updates 97
7.4 Conclusion 101

8 Distributed Stateless Learning **103**
8.1 Introduction 104
8.2 Approach 106
8.3 Formal Analysis 107
8.4 Evaluation 113
8.5 Complexity 116
8.6 Distributed Stateless Learning with Noisy Rewards 117
8.7 Discussion 125

9 Coordination in Sequential Stage Games **129**
9.1 Coordination Strategies 130
9.2 Modeling the IAPP as Sequential Stage Game 137
9.3 Summary 142

10 Empirical Analysis **143**
10.1 Settings 144
10.2 Basic Distributed Stateless Learning 145
10.3 Strategy Specific Simulations 151
10.4 Reward Types and Coordination Strategies 160
10.5 Evaluation in Different Scenarios 170
10.6 Discussion 175

11 Storage Medium-Based Distributed Stateless Learning **177**
11.1 Approach 178
11.2 Empirical Analysis 180
11.3 Conclusion 190

12 Region-Based Heuristics **191**
12.1 Target-Regions 192
12.2 Approximation of Target-Regions 194
12.3 Voronoi Regions 201
12.4 Discussion 203

Part IV Conclusion **205**

13 Conclusions and Future Work **207**
13.1 Conclusions 207

13.2 Future Work 209

Part V Appendix **211**

A Medium Coverage **213**
A.1 Computation of Medium Coverage 214
A.2 Optimal Coverage 216

B Simulation Results **221**
B.1 Simulation Details for Global Reward Settings 222
B.2 Simulation Details for Local Reward Settings 224
B.3 Simulation Details for Locally Transformed Global Rewards 226

List of Notations **229**

Abbreviations **231**

References **233**

Chapter 1

Introduction

Computer-based systems become more and more ubiquitous and useful in everyday life. For the acceptance and success of such systems *simplicity* is a key requirement. That is, the end-users do not want to deal with technical details. Instead, they expect their devices to work correctly, out of the box, and with a seamless integration into an already existing technical environment.

This demand for simplicity quickly leads to conflicting situations, in particular, because applications nowadays show a steadily increasing complexity. For instance, complexity can arise if application services involve many interacting devices in loosely coupled networks, or if multiple distributed components have to agree on a coordinated joint behavior. One approach to deal with this complexity, while ensuring simplicity, is to build intelligent systems that autonomously decide how to best behave so as to reduce the need for human intervention.

Therefore, it is a promising approach to equip devices with the *ability to learn*. This enables the systems to find strategies not just for fixed problem settings but also for sudden and unexpected changes in dynamic and complex environments. However, the goal is not to find *some* strategy but an *optimal*—or at least near optimal—one. Accordingly, learning is also closely related to *optimization*. Since a device usually has to operate in an environment that contains other devices and because individual actions might influence the behavior of others, *coordinated* (joint) actions become a particularly important issue. Furthermore, a completely *decentralized* approach is desirable because usually there is no central instance and the devices have to base their decisions solely on local information.

A simple navigation example clarifies these points. Assume that you want to visit some friends in another city. Then you might drive on highways and, all of a sudden, an accident blocks your route. Your in-car navigation system then has to determine the "best" detour. However, all other cars around you are faced with the same situation. If all systems decide to take the same detour via a smaller road, then this road might quickly be congested, resulting in further delay for all cars. Instead, it would be better if the devices *coordinate* their route planning to *optimize* the global traffic flow, e.g. by equally distributing the cars to alternative detours. Since not all possible conditions, including current traffic, construction sites or weather, can be foreseen, the behavior of each system should be subject to constant *learning* so as to enable the devices to adapt to these dynamic settings. Also, there is no central

instance that allows the devices to coordinate their actions. Thus they have to deal with the situation in a *decentralized* way using only local information.

In this thesis, we focus on developing learning approaches for such large and decentralized systems involving hundreds or thousands of devices (also called agents) that have to cooperate in order to reach a common global goal with the best possible result. Since this is a very general setting for which a universal solution is hardly achievable, we will focus on problems that exhibit a special structure. Namely, in our setting, agents will be confronted with a particular situation only for some unknown but limited time before a new situation occurs. Hence, the agents will deal with a sequence of different problems, whereas they have to learn how to best solve each problem during the available unknown timespan.

This sequential structure also occurs in the above car navigation example: after you have arrived at your destination city, you are confronted with the problem of finding a free parking lot close to your final destination. However, other people will search for a free space, too. Hence, your navigation system has to deal with a different and new situation, in which, again, a behavior that is coordinated with other participants is highly desirable, e.g. to equally distribute all cars on the possible parking areas. Such parking problems will frequently reoccur in different cities, whereas in each city only the current scenario is of interest (cf. Fig. 1.1). Accordingly, there is also a sequence of such problems that have to be solved one after the other, probably re-using knowledge from previous situations. Using learning mechanisms to adjust strategies for such kinds of sequential problems under dynamic and unforeseeable conditions again can lead to better solutions.

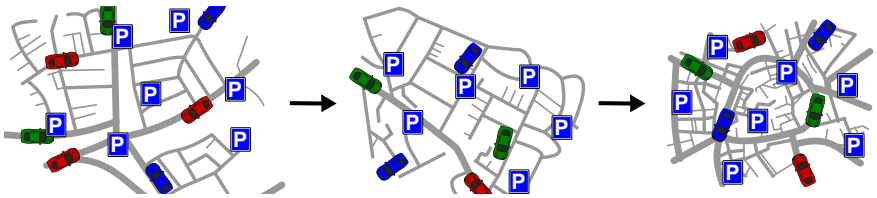


Fig. 1.1 Sequence of similar partitioning problems in the car parking example.

In both situations of the simplified and exemplary car navigation problem, we are able to identify that a distribution or partitioning of some objects (cars) to another set of objects (detours; parking areas) has to be found such that some global criteria are optimized (equal distribution on detours and minimization of the delay; equal distribution to parking areas and minimization of the distance to a final destination). Since we focus on systems with a sequence of problems, a sequence of such distributed and repeated partitioning problems establishes a nice challenge for the developed approaches, in particular if it additionally includes a multi-objective optimization component. One such problem is the *Iterative Agent Partitioning Problem (IAPP)*. It describes a multi-objective optimization problem in the form of a *Multiagent System (MAS)* that demands a repeated partitioning of mobile agents onto a special set of targets located in an environment. Since the IAPP considers only stateless settings, i.e. the solution qualities in a particular situation do only depend on the joint action, the problem is well suited as a first testbed to investigate learning in game sequences.

In summary, this thesis introduces and investigates *sequential stage games (*SSG*)* which are a structured game class for learning in the described settings. The goal is to develop an efficient distributed learning approach for *cooperative* SSG*s*. Given that specific structure, one question is how well agents can reuse learned knowledge such that they are able to benefit from solutions of previous situations. We consider a particular cooperative SSG that is a variant of the general *Iterative Agent Partitioning Problem (IAPP)* as testbed for the presented approaches.

1.1 Background

One seminal contribution in the area of learning is the development of the Q-Learning algorithm [Wat89], which is a *reinforcement learning (RL)* approach. In RL, an agent learns a policy, that returns the best action to execute in every state, by executing actions, observing reactions, and getting numerical feedback in the form of rewards (or punishments). It is proven to converge to optimal policies in single-agent Markov Decision Processes under some assumptions [WD92] [JJS94].

In general, learning in systems composed of many autonomous entities is conducted in multiagent systems (MAS). Such system consists of a set of autonomous units called *agents* that are located in an environment which can also contain further objects. Agents are able to perceive and modify the environment by performing actions. MAS are frequently used by researchers [Wei99] [Fer99] [Woo09] due to their flexibility and generality. According to e.g. [Wei99], they have for instance been applied in areas as diverse as electronic commerce, telecommunication networks, transportation systems, traffic scenarios, or social systems, to name a few.

Based on the idea of reinforcement learning, and often inspired by Q-Learning, many algorithms for *Multiagent Reinforcement Learning (MARL)* have been developed since the late 1990s, e.g. [CB98] [LR00] [Lit01] [KK02] [WS03] [GLP02] [Bow04] [AL08]. A common framework, which is a generalization of single agent Markov Decision Processes, is known as *stochastic games (*SG*)*. Games of this class can have arbitrary probabilistic state transitions and thus generally show no particular structure. Nevertheless, there are algorithms for special subclasses of stochastic games, e.g. also for cooperative stochastic games [LR00] [KK02] [WS03]. Furthermore, there are also algorithms that try to exploit special problem properties, e.g. [GLP02] use problem structures to identify relevant joint actions instead of learning values for all possible joint actions.

However, most of the existing MARL approaches deal with rather small MAS, frequently involving only two agents [CB98] [LR00] [KK02]. Also, algorithms that exploit the aforementioned sequential problem structure, to the best of our knowledge, have not yet been investigated. Hence, efficient learning algorithms that provably converge to optimal joint behaviors in large problems that have the structure highlighted before are a challenging topic that is covered in this work.

1.2 Structure

Figure 1.2 illustrates the overall structure of this thesis. As one can see from the figure, we organized the text into five separate parts:

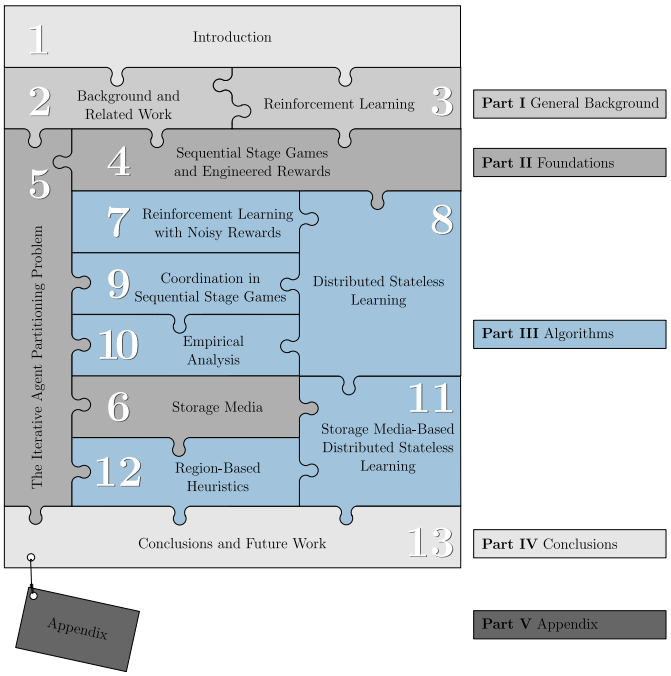


Fig. 1.2 The puzzle form is used to express the relations between the different chapters of this thesis. The coloring represents the used part structure in the text, whereas parts II and III contain our contributions.

Part I General Background: The first part deals with the background of this thesis. Chapter 2 introduces existing general concepts and research areas that are related to our research. Then, a more specific and detailed introduction to reinforcement learning in single and multiagent systems is presented in Chapter 3.

Part II Foundations: Whereas the first part describes existing work, the second part introduces the particular foundations developed for this thesis. Chapter 4 introduces a novel game class, called *sequential stage games (*SSG*)*, and a conceptual model that will be used for learning in the proposed game class. Chapter 5 presents the *Iterative Agent Partitioning Problem (IAPP)*, an exemplary multi-objective optimization problem that can be modeled using the novel game class. We motivate, define, and analyze that problem in detail as it will be used as testbed throughout this work. Since the agents need to coordinate their actions to successfully (and fast) learn in the considered games, we introduce a coordination concept in Chapter 6. The concept defines external storage media as a means for coordination and discusses some relevant aspects. Later, it will be combined with the proposed MARL approach.

Part III Algorithms: The main part of this work develops, describes, and evaluates several (learning) algorithms for the considered problem domain. We begin in Chapter 7, where we investigate learning in deterministic single agent systems with noised rewards. Then, Chapter 8 presents and analyzes the main learning approach, called *Distributed Stateless Learning (*DSL*)*. DSL is made for learning in cooperative SSGs with potentially many agents and will be shown to be able to converge to optimal solutions. Based on the insights of learning under noisy re-

wards from Chapter 7, we also investigate the influence of noise on DSL. In order to speed up learning in large systems composed of hundreds or even thousands of agents, Chapter 9 introduces several coordination strategies that can be used in DSL. Thereafter, Chapter 10 empirically analyses DSL and these strategies in various settings of the IAPP testbed. The last two chapters of this part deal with the concept of storage media as means for coordination. In Chapter 11 we combine it with DSL to allow environment-mediated knowledge transfer during the learning process. Finally, Chapter 12 presents a combination of a state-of-the-art communicative approach for the IAPP with storage media.

Part IV Conclusions: This part contains Chapter 13, which summarizes this thesis and presents an outlook on possible future work.

Part V Appendix: The last part contains a geometric proof for the best placement of storage media with respect to obstacles and communication, as well as details on some simulation results.

Part I

General Background

Chapter 2

Background and Related Work

The purpose of this section is two-fold. First, we introduce some general background knowledge concerning algorithms and concepts that will re-appear later in this thesis. At the same time, we present different research areas and some representative approaches and relate them to the context of this work.

In detail, Sect. 2.1 introduces the reader to agents and multiagent systems. Then, Sect. 2.2 provides a detailed overview on the Online Partitioning Problem which inspired the agents-to-targets assignment problem considered throughout the thesis. We approach this problem from the perspective of multirobot research in Sect. 2.3. Since we deal with large numbers of agents, Sect. 2.4 thereafter introduces the reader to swarm intelligence. Sect. 2.5 presents the idea of learning in complex tasks based on knowledge acquired in related tasks. Finally, Sect. 2.6 shortly discusses optimization from a general perspective.

2.1 Multiagent Systems

Roughly speaking, a multiagent system (MAS) consists of *autonomous agents* that are situated in some *environment*. Situated agents are able to observe and to modify their environment. Accordingly, there is a tight connection between both entities which is often considered as "agent/environment duality" [Fer99], i.e. the environment is everything outside of the agent, everything beyond the control of the agent. One can already conclude from this abstract description that multiagent systems can be used as model in a broad number of applications areas including robotics, software design, simulation of artificial social societies, or distributed problem solving to name a few (see e.g. [Fer99] [Wei99] [Woo09] [RN10]). As a result, also a wide variety of definitions can be found in the literature. We do not intend to discuss all of them in detail. Instead, we refer the reader to [LBEK11], where we provide a survey on the most important models. In that work, we consider existing models and present a universal formal model whose description is beyond the scope of this thesis.

Given this abstract notion of a MAS, we now want to develop an understanding of what is considered to be an agent in this work. Therefore, consider the definitions of Wooldridge [Woo09] and Ferber [Fer99]:

Definition 1 (Wooldridge's Agent). An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its delegated objectives.

Definition 2 (Ferber's Agent). An *agent* is a physical or virtual entity

- which is capable of acting in an environment,
- which can communicate directly with other agents,
- which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize),
- which possesses resources of its own,
- which is capable of perceiving its environment (but to a limited extent),
- which has only a partial representation of this environment (and perhaps none at all),
- which possesses skills and can offer services,
- which may be able to reproduce itself,
- whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and the communications it receives.

While the definition of Wooldridge is still rather vague—but already captures the most relevant properties mentioned above—Ferber's definition describes an agent in much more detail. One remarkable consensus in both definitions is *autonomy*, which is also a general property attributed to agents in literature[1]. Further commonalities include that an agent is situated in an environment, the ability to interact with the environment, and the notion of an agent's objective.

Based on these insights, we provide the following definition:

Definition 3 (Agent). Let $\mathcal{A}$ be a set of agents. Then, an agent $i \in \mathcal{A}$ is an autonomous entity that is located in an environment and which tries to meet some given objectives. Agents possess sensors and actuators to perceive their environment resp. to act in their environment and to possibly modify it. Both, sensors and actuators may have limited ranges. Furthermore, communicative agents can communicate within a certain radius r_i from their position.

This definition should be understood as a minimal agent definition. It is supposed to be a flexible and expandable definition that only captures our understanding of an agent rather than being a strict formal definition. The best way to think of agents in this thesis hence may be to treat them as tiny robots that work in a (physical) environment and which have limited abilities, sensors, actuators, and internal resources. In this way, a concrete "realization" and implementation of a robot finally describes an instance of a particular agent in a given problem context.

Due to the agent/environment duality, an environment hence contains at least one agent in a *single agent system* or more agents in a *multiagent system*. According to Russel and Norvig [RN10], an environment generally can be categorized according to the following dimensions:

[1] However, one might discuss whether agents are really autonomous since they follow some sort of algorithm that "dictates" which actions to execute. From our point of view, autonomy hence is only possible within the boundaries established by the agent's designer. We will not pursue this issue any further as its discussion quickly becomes philosophical.

Fully observable vs. partially observable: If the sensors of an agent are always able to exactly observe all relevant properties of an environment with respect to the intended objectives, then the environment is said to be fully observable. Otherwise, it is only partially observable which includes noised or limited perceptions.

Single agent vs. multiagent: The number of agents basically determines this dimension. However, one could argue what an agent from its own local perspective considers to be an object or to be part of the environment and what it considers to be an agent. For instance, if agents decide to ignore all other agents then their ignored actions could be seen as part of the environment's dynamics. Multiagent environments further can be divided into cooperative, competitive, or mixed environments depending on the behavior and goals of the agents.

Deterministic vs. stochastic: While the outcome of an action in a particular state of a stochastic environment depends on some probability distribution, it is fixed in a deterministic environment. From an agent's perspective, a partially observable and deterministic environment could appear to be stochastic due to limited observations.

Episodic vs. sequential: In an episodic environment, agent decisions depend only on current perceptions and only influence the current episode. Given a sequential environment, agent decisions are likely to have an impact on future behaviors.

Static vs. dynamic: An environment that can change its behavior or properties over time is considered to be dynamic. Static environments do not change in the course of time.

Discrete vs. continuous: This dimension refers to the state of the environment, to the way of dealing with time, as well as to observations and actions of agents. For instance a state can be described with continuous or discrete variables.

Known vs. unknown: From an agent's perspective an environment is said to be known if the agent has knowledge about the way how the environment "works", i.e. how it would respond to certain actions, and unknown otherwise.

Based on these properties, one can easily imagine that the environment has a strong influence on agents, e.g. with respect to what agents are able to learn and to achieve. A multiagent system, however, is more than just a set of agents and an environment. This becomes clear in Ferber's definition (quote from [Fer99], with renamed identifiers to match the notations used in this work):

Definition 4 (Ferber's Multiagent System). The term *multiagent system* (or MAS) is applied to a system comprising the following elements:

1. An environment, $\mathcal{E}$, that is, a space which generally has a volume.
2. A set of objects, O. These objects are situated, that is to say, it is possible at a given moment to associate any object with a position in E. These objects are passive, that is, they can be perceived, created, destroyed and modified by the agents.
3. An assembly of agents, $\mathcal{A}$, which are specific objects ($\mathcal{A} \subseteq O$), representing the active entities of the system.
4. An assembly of relations, Rel, which link objects (and thus agents) to each other.
5. An assembly of operations, A, making it possible for the agents of $\mathcal{A}$ to perceive, produce, consume, transform and manipulate objects from O.
6. Operators with the task of representing the application of these operations and the reaction of the world to this attempt at modification, which we shall call the laws of the universe.

We will basically follow Ferber's understanding of a multiagent system since it fits well to the considered problem context. However, in the context of the MAS treated in this work, we will later incorporate some adjustments and extensions to that definition, including concrete sets of objects, objectives, or models of the environment.

2.2 The Online Partitioning Problem

The Iterative Agent Partitioning Problem (IAPP) that is examined in this work to evaluate the proposed algorithms is based on the *Online Partitioning Problem (OPP)* [GKBPW05] [Goe07]. The basic description of the problem reads as follows: Given a set of agents and targets (e.g. tasks) that are distributed in some Euclidean space. The goal of the agents is to find an assignment to the targets that satisfies the following three objectives:

1. create a uniform distribution, i.e. assign the same number of agents to each target
2. minimize the overall sum of distances between each agent and its assigned target
3. use simple agents with minimal abilities

Accordingly, in [Goe07], Goebels introduces the OPP as

> a multi-objective optimisation problem [...] for the coordination of large groups of agents.

There, he also provides the following formal definition:

Definition 5 (Online Partitioning Problem [Goe07]). An instance of the OPP with $OPP = (\mathcal{A}, \mathcal{T}, \rho)$ consisting of the agent set $\mathcal{A} = \{a_i, \ldots, a_n\}$, the target set $\mathcal{T} = \{T_1, \ldots, T_m\}$, and a position function ρ is an instance to an optimisation problem. The question is whether there exists a partition $\{S_1, S_2, \ldots, S_m\}$ of $\mathcal{A}$ with $S_i \subseteq \mathcal{A}$ $(i \in \{1, \ldots, m\})$ such that $\mathcal{A} = \{S_1 \uplus S_2 \uplus \ldots \uplus S_m\}$, which simultaneously maximises

$$\prod_{i=1}^{m} |S_i|$$

and minimises

$$\sum_{i=1}^{m} \sum_{a \in S_i} \delta(a, T_i)$$

This can be combined in the following equation. There, b_i denotes the number of agents that have chosen the target T_i in the current partitioning decision and o_i the number of agents that would have chosen target T_i in an optimal partitioning. $\tau(a_i)$ denotes the target currently chosen by agent a_i.

$$f = \alpha \cdot \left(\frac{\prod_{i=1}^{m} b_i}{\prod_{i=1}^{m} o_i} \right) + \beta \cdot \left(\frac{\sum_{i=1}^{n} \min_{j=1\ldots m} (\delta(a_i, T_j))}{\sum_{i=1}^{n} \delta(a_i, \tau(a_i))} \right) \tag{2.1}$$

under the constraints $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$ for the weights α and β.
The higher the value f $(f \in [0; 1])$, the better is the solution found.

For this mathematical description we use the fact that a product of m numbers is maximal if the numbers are identical.

By examination of the evaluation function (2.1), it is obvious that the third objective concerning the agent's abilities is not formally captured. Instead Goebels usually considers it only indirectly while comparing different algorithms, i.e. agents with different abilities. One exception is a setting in which communication costs are taken into account [Goe07].

In later works, the OPP was extended to include a simple obstacle [Weh08], or a set of different obstacle types that influence e.g. distance calculations or communications [Kem10]. The latter work also contains an attempt to formally capture the abilities of agents. Therefore, agents are considered to be robots such that each action, including calculations, movement, sensing or the like, is assumed to consume energy. Hence, the more complex the agents' abilities, the more energy is required.

A large number of different approaches for the OPP [Goe07] and for OPP-based problems [Weh08] [Kem08] [Kem10] [Kal10] [Swa10] have been presented and discussed. In [Kem08], a detailed description of known approaches at that time is given. Next, we abstract this overview and extend it to include recent developments. The following list indicates that one can mainly distinguish between six different categories (note that, if not stated otherwise, detailed descriptions of the algorithms can be found in [Goe07]):

1. *Central instance* approaches, as the name already suggests, use a central instance that has complete knowledge about the system. Goebels [Goe07] describes three such algorithms. The first is an exact algorithm that finds optimal solutions by examining all m^n possible assignments, where m is the number of targets and n the number of agents. The second is an exact approach for settings with just two targets that requires time in $\mathcal{O}(n^2)$. We will present some more details later in this work. A third algorithm uses a heuristic to quickly generate good solutions for arbitrary many targets by sorting agents in lists for each target and then stepwise iterates over these lists and assigns the closest unassigned agent to the corresponding target.
2. *Non-communicative* methods use either only agent internal properties or local information that can be obtained without communication. The simplest one selects targets randomly; another approach assumes consecutively numbered agents and selects a target by a modulo-operation on the agents' identification numbers. In a third approach, each agent just selects the nearest target. Clearly, one can easily generate settings in which these methods fail.
3. *Communicative* approaches are most common and many different algorithms exist. They range from very simple ones with only local information exchange and simple assignment operations to more complex approaches that collect information about the current system state and apply sophisticated solution methods (e.g. [Weh08]).
   One remarkably powerful communicative strategy is the Exchange Target Strategy (ETS). In fact, this strategy can be considered as the state-of-the-art approach for the OPP. We will come back to ETS in a moment.
4. The *organization-based* methods create groups of agents (organizations) either by learning which groups are beneficial with respect to partitioning qualities and communication costs, or by using an election message protocol. The groups or group heads are assumed to be able to calculate an optimal partitioning for each group.

5. *Biologically inspired* approaches use techniques like cellular automatons and evolutionary algorithms [GWP05], or rules that are evolved using genetic algorithms [Kal10], and techniques that are inspired from insect-behaviors, like ants or wasps [Kem08].
6. *Learning*: A first step towards the application of multiagent reinforcement learning in the context of the OPP has been made in a master thesis under our supervision [Swa10]. The OPP is modeled with the help of a decentralized partially observable model and a rather small state-space abstraction is presented. The main idea of the learning approach is to adjust the number of possible agent actions dynamically during runtime. The global reward function that gives feedback to the learners is based on some assumptions, e.g. it allows only one agent to change its action at a time in order to rate the influence of a single agent's action.

Since two approaches will reoccur during this thesis, we cover them in more detail in the following paragraphs.

The first approach is the aforementioned state-of-the-art *Exchange Target Strategy (ETS)* [GKBPW05] [Goe07]. In ETS, initially all agents are randomly distributed to targets and swap assignments if the local distance sum would improve. Agents only exchange targets under mutual agreement. Accordingly, the initial distribution objective value is maintained during execution. By the law of large numbers and given many agents that chose an initial target uniformly at random, one can further expect that the initial assignment leads to a roughly equal number of agents per target and thus also to a nearly optimal distribution objective value. Other initialization methods, e.g. depending on agent IDs, are possible under assumptions like consecutively numbered agents. Given that one part of the objective function is already optimized in the beginning, the agents then have to improve the distance part. Therefore, each agent communicates with its neighbors as visualized by the edges in Fig. 2.1, where the two agents i and j talk to their neighbors. If they find another agent that is assigned to a different target, they check whether a target exchange would improve the distance sum. For i and j in the given example, such an exchange clearly results in a locally smaller distance sum. Hence, both agents will change their targets. This process is repeated in every time step such that the distance sum is improved over time. Give sparse communication graphs, small neighborhoods, or settings with obstacles, deadlocks may occur in the sense that useful target exchanges are not carried out because they first would lead to a worse distance value before they finally improve it. In order to avoid such deadlocks, agents may decide to swap targets with some "flip" probability $\mathrm{p_{flip}}$ even if the exchange would not improve the distance value.

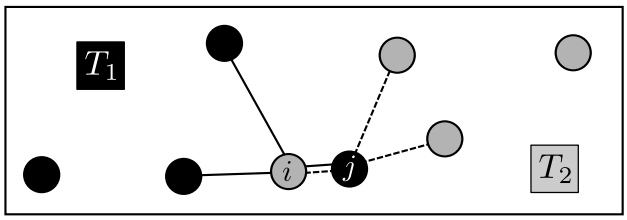


Fig. 2.1 Example for a target exchange that improves the solution quality.

In general, ETS is not guaranteed to find optimal solutions, e.g. due to random initialization. Furthermore, several worst case scenarios can be created. The communicative overhead should also not be neglected, as each agent in every time step communicates with each of its neighbors. In [Kem08], we propose to ignore requests, if two interacting agents are assigned to the same target. By doing so, the communicative overhead is decreased without obtaining worse solution qualities. Besides these issues, ETS has been proven to be able to find (very) high quality solutions on average and in rather short time.

Goebels [Goe07] investigates ETS not only in the static OPP setting but also considers agents that move towards their currently selected target. The results have been positive in the sense that movement is equal to changing neighborhoods. In particular, the agent is able to interact with more agents, which leads to "larger" neighborhoods. Accordingly, new exchange opportunities arise that have a positive effect on the solution quality.

The second approach that will be used in this work is the exact algorithm for two targets as described in [Goe07, Sect. 3.2.2]. We will refer to this algorithm as the *Two Target Optimal (*TTO*)* approach. The basic idea of this central instance algorithm is to sort all agents ascending in a list according to the distance difference with respect to the two targets. In detail, the distance difference for two targets T_1, T_2 of an agent i is defined by $\Delta(i) = \boldsymbol{\delta}(i, T_1) - \boldsymbol{\delta}(i, T_2)$, where $\boldsymbol{\delta}$ returns the distance between an agent and a target. Initially, all agents are assigned to their nearest target, before an iterative process is started. Let us assume that more agents have selected target T_1. Then, the algorithms selects the agent from the list which has the smallest positive distance difference, i.e. an agent that is assigned to T_2 because it is closer to it than to T_1. At the same time, that agent is also the nearest to target T_1 compared to any other agent that is assigned to T_2. This agent then is reassigned to T_1. Accordingly, the distribution objective is improved but the distance objective becomes (slightly) worse. If the resulting solution quality is better than before, then this reassignment step is repeated until the solution became worse for the first time. Given the latter case, the algorithm stops and returns the previous assignment as optimal solution. In [Goe07], Goebels proves that this approach calculates an optimal partitioning.

In the area of the OPP, mainly static scenarios with fixed agent positions have been considered. In contrast, this work will focus on scenarios with (frequently) changing agent positions. Also, we will briefly investigate one setting with changing target positions. Accordingly, our agents have to deal with a sequence of different OPP instances rather than with a single instance. Concerning the algorithms, this work will extend the Exchange Target Strategy by incorporating additional coordination media located in the environment in order to improve efficiency in our dynamic settings. Furthermore, we will dive deep into the area of multiagent reinforcement learning and thus explore a class of approaches that has not been considered in this context until now.

2.3 Multirobot Task Allocation

The considered iterative partitioning problem is also related to the area of *multirobot task allocation (MRTA)*. MRTA deals with the question of how to distribute a set of robots to a set of task, i.e. "which robot should execute which task" [GM04]. Gerkey

and Matarić [GM04] present a taxonomy of MRTA problems with independent tasks. The taxonomy categorizes MRTA problems along three dimensions:

1. Robots are able to either perform one task at a time or multiple tasks at once.
2. Tasks may either require one or more robots to be accomplished.
3. The amount of information available, e.g. about tasks: is there knowledge about future tasks or the entire task set such that planning is possible, or are robots forced to decide based on instantaneous information.

In the context of this work, approaches for robots that perform one task at a time and tasks that typically require more than one agent are of interest. This follows directly from the definition of our (iterative) partitioning problem, where agents can only select one target at a time.

Two such approaches are presented by Ducatelle et al. [DFDCG09]. They consider a problem that is closely related to the (non-iterative) Online Partitioning Problem with obstacles. However, their problem contains tasks that require different numbers of robots instead of a equally many. In detail, they consider a scenario that contains two different types of robots: flying robots and so-called footbots that move on the floor. The flying robots have to identify possible tasks in the environment and announce these tasks to the footbots. Ducatelle et al. present a simple reactive approach that uses light signals to attract robots. In a second approach, they implement a gossip based protocol (see e.g. [JVG$^+$07]) that disseminates knowledge about the tasks via local communications. Simulations of these approaches show that the gossip-based approach works well in complex scenarios with obstacles but has problems concerning scalability and robustness. In simple environments, the light-based approach also performs well. In contrast to our problem domain, the work of Ducatelle et al. only considers a one-time distribution of agents to targets instead of repeated partitionings. Also, they deal with tasks that require different numbers of robots instead of a uniform distribution.

Recently, Elango et al. [ENT11] formally defined the so-called *balanced multi-robot task allocation problem (BMRTA)* and proposed a three step approach. The BMRTA is specified by a set of robots and tasks that are located in a Euclidean environment. In addition, a cost function is given that returns the costs for moving between two positions. The goal is to find an allocation of robots to tasks such that a) the idle times of the robots are minimized and b) the costs for traveling between assigned tasks for each agent are minimal. In the first step of the proposed approach, the task set is clustered based on a modified K-means approach. The clusters are created such that distances between tasks in a cluster are minimized and such that path lengths are balanced over all clusters. The second step calculates some costs for each robot with respect to visiting one or more clusters. Finally, using an auction-based approach, the third step creates the actual task allocation. Elango et al. evaluate their approach in a simple setting with two simulated robots and conclude that there is a necessity for a trade-off between minimizing the total costs and the workload of the robots. Due to the second step, where exponentially many combinations have to be evaluated, the proposed approach does not scale well to large problem instances. In comparison to this setting, we require agents not to "visit" multiple tasks. Also our approaches do not need to enumerate and evaluate all possible assignments (although this corresponds to our search space, too). Instead, the agents construct a solution and try to improve it by efficient and local search methods.

The general idea of adapting market-based approaches towards multirobot coordination has attracted many researches. Dias et al. [DZKS06] survey such approaches.

Another related research area considers the problem of *coalition formation* [SK98] [SA11]. There, the question is which agents should form groups, so-called coalitions, such that a given set of tasks is completed most efficiently by the different groups.

2.4 Swarm Intelligence

Inspired by biological swarms, including social insects like ants, termites, bees, wasps, or social animals like fish schools or flocks of birds, *Swarm Intelligence (SI)* in its current form is investigated since the late 1980s, whereas a large number of fundamental work was accomplished during the 90s [BDT99]. According to Engelbrecht [Eng05, p.2], a group of mobile agents (or robots) that work together in a local environment and which interact directly or indirectly can be considered as a swarm. A swarm usually has many members which only possess limited abilities. Due to the large number of agents and their interaction patterns, complex (global) behaviors can emerge from simple local actions that lead to surprisingly robust distributed problem solving behaviors.

Besides other directions, particle swarm optimization and ant algorithms may be the most prominent models used in the area of swarm intelligence [Eng05]. Ant algorithms, or to be more precise the *ant colony optimization (ACO)* metaheuristic has successfully been applied to a large amount of problems as diverse as routing in networks, vehicle scheduling, the well-known Traveling Salesmen Problem, graph coloring, scheduling, or classification (see [DS04, Table 2.1] for further examples and references to relevant literature). The basic idea of ACO is related to a concept called *stigmergy* that was originally described by Grassé [Gra59]. He investigated the nest building behavior of termites and stated that

> "L'ouvrier ne dirige pas son travail, il est guidé par lui."
> (engl.: "The worker does not direct his work, he is guided by it")

Accordingly, stigmergy describes how behaviors of swarm members are influenced or stimulated by the results of labor, i.e. by the modifications of the environment through the work carried out by the individuals. To put it differently, stigmergy can also be understood as some kind of indirect communication by modifications in the environment and reactions to those changes (cf. also [HM99],[BDT99]).

In ant algorithms, so-called (artificial) *pheromones* realize the concept of stigmergy. In natural ant societies, pheromones are a chemical fluid that is deposited and smelled by ants. For instance, in foraging, agents place pheromones along their path between the nest and a food source. In a probabilistic decision based on the pheromone concentration, ants either follow an existing pheromone trace or explore a new path. The higher the pheromone concentration on a path, the more likely this path is chosen. During the course of time, pheromones evaporate if they are not reinforced. Since more ants can travel a short path than a long path in the same time, the pheromone concentration on a shorter path will likely be higher after some time. This in turn stimulates more and more ants to follow the shorter path. Due to the probabilistic decision making and the explorative behavior, ants are able to find short paths in a complex network of possible paths. Since this is also a common problem in computer science, e.g. in routing packets through a network, the idea of pheromones builds the basis for the Ant Colony Optimization metaheuristic.

Given an appropriate model that decomposes the solution space of an optimization problem into linked components, one could build a so-called construction graph. A construction graph $G_C = (C, L)$ uses the set of components C as nodes which are fully connected by a set of edges L. The graph then is used in the ACO metaheuristic to find (hopefully near optimal) solutions. Dorigo and Stützle [DS04] describe the general approach using the following three steps, that could be realized in parallel or any other ordering, depending on the actual problem:

- The first step is about using artificial ants, or agents, to construct solutions. Ants randomly walk around in G_C and, by doing so, they incrementally build solutions.
- In the second step, components and/or edges are furnished with artificial pheromones. That is, either new ones are added or old ones are updated.
- The third step is optional and deals with centralized actions that cannot be accomplished in a decentralized way. For instance, evaluating a current solution in order to decide whether or not and how pheromones are changed.

Since ACO is a metaheuristic, it generally cannot guarantee to find optimal solutions.

The approaches presented in this work will mainly focus on reinforcement learning in order to solve a multi-objective optimization problem. In contrast to ACO, our main learning approach is guaranteed to converge to optimal solutions under some assumptions. Nevertheless, there are also similarities to ACO since agents and (artificial) ants work together in order to construct a common solution. Though the underlying techniques generally differ from each other, we will also investigate one reinforcement learning approach that is combined with a pheromone-based strategy.

The second important model in swarm intelligence is *particle swarm optimization (PSO)* [KE95]. The basic idea of PSO is to use a set of particles (the swarm), each representing a solution candidate. Particles are distributed in the environment and update their positions based on their own knowledge and on the knowledge resp. positions of either the entire swarm or of particles in a specified neighborhood. Various aspects and variants of PSO are described in detail in [Eng05, Part III].

Somehow similar to PSO, some of our approaches also use knowledge from neighbored agents. However, this is the only slight similarity to PSO. Conceptually, our methods differ in the way that only a single solution is maintained rather than a set of solution candidates.

2.5 Transfer Learning

In this section, we introduce *Transfer Learning (TL)*, which is a recent topic in the AI community [TS09] [TS10] and up to now is mainly applied in single agent systems. The basic idea of TL is to learn a complex task not from scratch but instead to first learn to solve a simplified yet related task. The acquired knowledge is used as basis for learning in the complex task. The simplified task is usually referred to as *source task*, the complex one as *target task*. By reusing the knowledge of a source task, the goal of TL mainly is to improve the speed of learning in the target task. As simulation results show, besides an improved learning speed, TL also leads to higher solution levels compared to learning from scratch in the goal task (e.g. [BTS10] [TSWM10]). Note that in general there could also be more than one source or target task.

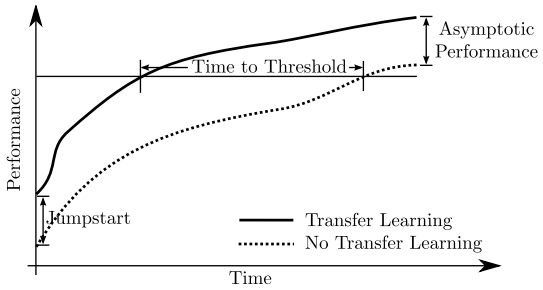


Fig. 2.2 Metrics for measuring the success of Transfer Learning (reproduced from [TS09])

In order to measure the success of TL, different metrics can be used as visualized in Fig. 2.2. In detail, Taylor and Stone [TS09] and Torrey and Shavlik [TS10] identified and agreed in the following:

Jumpstart/initial performance: This metric describes the initial performance increase that results from using the knowledge of a source task compared to settings without TL.

Asymptotic performance/final performance: The difference between the final convergence levels in settings with and without TL.

Time to threshold/time to fully learn the target task: The time that is required to reach a certain performance level.

Furthermore, Taylor and Stone also consider the *total reward* as metric, since faster learning and/or a higher performance level can also lead to a larger total reward compared to not using TL. According to them, another metric is the *transfer ratio* which refers to the ratio between the total reward accumulated by a TL-based agent and the total reward of an ordinary agent. Clearly, these five metrics are not exhaustive and many additional metrics can be considered, too.

Another important factor for evaluating the success of TL can be found in the time spent for learning in the source task. Depending on the goals of an approach, one could either include that time or neglect it. Then, one can conclude how much "faster" a TL-method is compared to a method without knowledge transfer.

Since TL deals with different yet related source and target tasks, one important question is how to transfer the knowledge between different tasks. In the simplest case, state and action spaces are equal or the source task's spaces represent subsets of the target task's spaces. If this is not the case, then additional mappings have to be provided to the agent, e.g. by human experts as in [BTS10] or [TSWM10]. Another possibility is to learn such inter-task mappings, which in itself constitutes a research area [ATTW11]

The idea of transfer learning is not restricted to particular learning techniques. Hence, it can be found in a variety of different research areas, including inductive learning, classification, or reinforcement learning. Torrey and Shavlik [TS10] provide a detailed survey on TL in these areas. Due to the focus of this thesis, the remainder will only consider TL in the area of reinforcement learning.

Torrey and Shavlik [TS10] distinguish between five different types of how transfer can be realized in reinforcement learning: *Starting-point methods* basically use the final value function of a source task to initialize a value function in the target task. In *imitation-based methods*, the idea is to use the learned source task policy from time to time during exploration in the target task. By doing so, the exploration might be

guided towards better solutions more quickly compared to random and undirected exploration. *Hierarchical methods* split a target task into smaller tasks. Afterwards, these are considered as source tasks and their solutions help to solve the actual target task. Another class of methods is known as *alternation methods*. These methods use source task knowledge to modify the state or action space or the reward function of the target task. For instance, by abstracting states in the target task and mapping those states to source task states. Finally, the last class consists of approaches that directly incorporate source task knowledge as an inherent part.

As one can see from the above classifications, several approaches have been followed in the literature. For instance, in [BTS10], transfer learning is applied on a physical humanoid robot that should learn to hit a ball as far as possible. The source task is less complex than the target task as it contains only five instead of nine actions and only four instead of eight possible observations. It was shown that transfer learning is not just feasible in simulations but can also significantly improve the speed of learning and lead to better behaviors in a source task if learning is carried out on a real robot. Another TL approach that is related to some methods considered in this work is presented by Torrey et al. [TSWM10]. Their TL approach is based on advices. Therefore, an agent learns skills in a source task. These skills then are mapped, by human intervention, to rules that can be applied in the target task. A rule then basically advices the agent to take a certain action in the target task if its conditions are met. Torrey et al. investigated their approach in a RoboCup scenario and showed that it leads to significant speed up in learning.

Due to its success in single agent systems, TL more recently is also investigated in multiagent domains. For instance in [PT09], value function knowledge is transferred between tasks that involve different numbers of agents. Also, in [BPV12] and [VDHN11], source and target tasks differ not only in the task's complexity but also in the number of agents involved. Accordingly, the consideration of multiple agents comes with additional challenges. These include, e.g., the question of how to transfer source task knowledge if the target tasks contains more agents, or how to preserve usually distributed problem solving knowledge during the transfer (individual agent strategies define joint strategies which constitute a solution for a cooperative problem).

Later in this thesis, we will present different approaches that are to some extent related to TL. These approaches will use external storage media or pheromones in order to transfer learned knowledge between different agents in a dynamic environment. By doing so, mobile agents can benefit from knowledge of other agents if they have entered new regions in the environment. Since changed agent positions in our formal model lead to new games, one may consider each game as a different task. Since these games in the context of our problem domain are closely related, the knowledge gained in a previous game basically plays the same role as source task knowledge in TL. Related to [TSWM10], some of our region-based heuristics (in which agents select actions based on the region they are located in) can also be considered as approaches that give advices to agents. In contrast to general TL, our work does not just deal with one source and one target task but with a sequence of tasks. In that sequence, each target task becomes a source task for the next target task. Hence, by exploiting the knowledge of the related tasks, our agents should learn to solve an abstracted problem that is described by the task sequence over the course of time. Another difference to existing works can be found in the number of agents involved, as we consider several thousand agents instead of just a small number of agents.

2.6 Optimization

Based on Chapter 2 in Engelbrecht's book [Eng05], we next briefly discuss some general aspects of optimization.

An optimization problem can be defined with the help of a tuple $\langle S, f, \Omega \rangle$ [DS04] [Eng05]. Here, S is a set of solution candidates, $f(s,t)$ a time-dependent objective function (indicated by time parameter t) that evaluates a solution $s \in S$, and Ω is a set of constraints, that could change over time. Note that constraints usually divide the set of all solutions into feasible and infeasible ones. Depending on the nature of the problem, the goal is to find a feasible solution that maximizes or minimizes f under the given constraints.

According to Engelbrecht [Eng05], optimization problems can be classified along several dimensions:

Number of variables: An optimization problem is *univariate* (*multivariate*), if its objective function is influenced by a single (by multiple) variable(s).

Type of variables: The domains of the considered variables determine this dimension. For instance, problems where all variables are real numbers are called *continuous*. Problems with integers are called *discrete* (or *integer*) problems. Those problems with continuous and discrete variables are known as *mixed integer* problems. Finally, *combinatorial optimization problems* deal with permutations of integer variables.

Degree of nonlinearity: The degree of nonlinearity is defined by the corresponding property of the objective function, e.g. linear problems consider linear objective functions.

Constraints: A problem can be *constrained*, i.e. subject to one or more constraints, or *unconstrained* otherwise.

Number of optima: A problem with multiple optima is called *multimodal*, a problem with a single optimum is said to be *unimodal*.

Number of optimization criteria: Here, one usually differentiates between *single-* and *multi-objective* problems.

On the other hand, optimization methods can mainly be divided along three dimensions. The first distinguishes between *local* approaches that try to improve a solution by searching its environment, and *global* approaches that use global knowledge and explore the complete set of candidate solutions. The second dimension classifies approaches into *deterministic* and *stochastic* methods, whereas the latter contain some random element that determines their behavior. Finally, the third dimension considers the problem to be solved. Hence, a method could be an (un-)constrained, a multi-objective optimization, a multi-solution, or a dynamic method. The properties of approaches for the first three classes can be directly inferred from the names. Multi-solution methods search for and provide more than one solution, whereas dynamic methods are able to deal with changing optima in the course of time [Eng05].

The agent-based learning method that we will develop in this work to solve a multi-objective optimization problem can hence be classified as a stochastic, multi-objective, constrained, and dynamic method that uses either global or semi-local knowledge to search for good solutions.

Chapter 3

Reinforcement Learning

In this chapter we present some basics about reinforcement learning in single and multiagent systems which are required as background knowledge in later chapters. By doing so, we also implicitly introduce the notations and terms that are used within the remainder of this work.

The chapter is structured as follows. In Sect. 3.1, we briefly classify reinforcement learning and present its idea. Then, in Sect. 3.2, we formally describe necessary details, models, and approaches for single agent settings. Thereafter, we consider multiagent reinforcement learning in Sect. 3.3 and a technique known as reward shaping in Sect. 3.4. Finally, we conclude this chapter in Sect. 3.5.

3.1 Introduction

According to Russell and Norvig [RN10]

> an agent is *learning* if it improves its performance on future tasks after making observations about the world.

Russell and Norvig point out that there are different *types of feedback* that can be used to let an agent learn from observations. They mainly classify learning into three categories. First, in *unsupervised learning* an agent does not get any feedback but learns patterns contained in the input data. Second, if the agent learns to map inputs to outputs from a set of given input-output pairs, then they speak of *supervised learning*. The third class is established by *reinforcement learning (RL)*. In such settings, the agent learns how to behave solely based on rewards or punishments, without hints on the best action. Accordingly, the agent obtains less information than in supervised learning, where it is told which action would have been best, but more than in the unsupervised case, where it obtains no feedback at all.

Figure 3.1 visualizes the main idea of RL. The basic framework is kept simple: at time t, the agent perceives the state of its environment s_t, selects and executes an action a_t which in turn modifies the environment. Then, the agent observes the resulting environment state s_{t+1} and a numerical reward ρ_{t+1}, which in a sense reflects how "good" it was to execute action a_t in s_t. Throughout this work, and

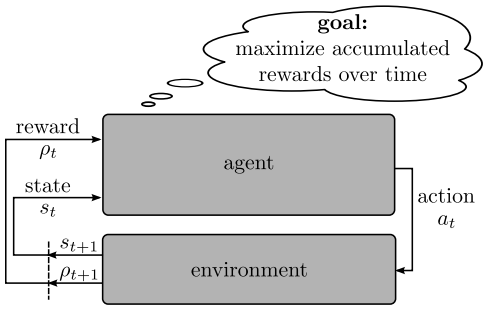


Fig. 3.1 Idea of reinforcement learning (based on [SB98]).

as common in literature, we assume finite state and actions sets and consider only bounded rewards.

The goal of the agent is to maximize not just the immediate reward obtained for executing an action but the long term accumulated reward that results from a sequence of actions. Hence, it has to learn a mapping which determines the action that should be executed in every particular state. Obviously, there is a close relation between the behavior of the agent and the reward function. Thus, the reward function shall return rewards such that maximizing them also leads to the desired goal behavior. This means that a reward function rather determines *what* is to be achieved than stating *how* to achieve it [SB98]. Since the reward is the only available feedback for learning, the agent has to collect information about which state action pairs lead to which rewards. This is done by following a *trial-and-error* approach, i.e. the agent has to *explore* its environment. However, too much exploration might lead to low accumulated rewards, which contradicts the agent's goal of maximizing that value. Accordingly, a good tradeoff between exploration and *exploitation* of learned knowledge is required. A simple yet powerful and frequently used approach is the ϵ-greedy action selection. It chooses the best action with probability $(1 - \epsilon)$ and a random action otherwise.

In the remainder of this section, we present a formal view on this general idea and describe different approaches from literature, for both, single and multiagent reinforcement learning.

3.2 Single Agent Reinforcement Learning

In the previous section, we introduced the basic idea of reinforcement learning (RL). In this section, we describe that idea formally. Based on [SB98], Sect. 3.2.1 provides a basic definition of the reinforcement learning problem, introduces Markov Decision Processes, and discusses optimal solutions. Then, Sect. 3.2.2 describes basic solution methods. With Q-Learning, Sect. 3.2.3 introduces the most influential RL algorithm of the past for deterministic and non-deterministic settings. Section 3.2.4 briefly introduces a model for partially observable environments and finally, Sect. 3.2.5 concludes this section with a short discussion.

3.2.1 The Learning Problem

In this section, we formally state the reinforcement learning problem and define all required terms. Therefore, the section is mainly based on the book of Sutton and Barto [SB98]. Further surveys on RL include, e.g. [KLM96] or [Sze10].

As stated earlier, the goal of an agent is to maximize its (expected) return over time. Assuming that an agents learns for some fixed number of time steps T, the overall return starting form time step t becomes

$$R_t = \rho_{t+1} + \rho_{t+2} + \rho_{t+3} + \ldots + \rho_T$$

However, if the learning task is continuous, i.e. $T = \infty$, the above sum clearly can become ∞ as well. In order to overcome this problem, the concept of *discounting* is required. A discount rate (or factor) $\gamma \in [0, 1]$ is introduced which determines the importance of future rewards. The discounted return then is defined as

$$R_t = \gamma^0 \rho_{t+1} + \gamma^1 \rho_{t+2} + \gamma^2 \rho_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k \rho_{t+k+1}$$

Accordingly, a reward ρ_k that is observed "k steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately" [SB98]. Given a discount factor of $\gamma = 0$, we thus obtain an agent that only maximizes the immediate reward and discards future rewards. This might be desirable but generally can also obstruct higher returns. Note that $R_t = \sum_{k=0}^{T} \gamma^k \rho_{t+k+1}$ can be used to express the overall (discounted) return for a finite horizon and for a continuous problem, if we assume that in the former case an absorbing state is introduced for which always a reward of 0 is obtained. Then, either $T = \infty$ or $\gamma = 1$ has to be allowed such that the sum remains defined. We will use this notation later in this work.

The RL task is usually described as a *Markov Decision Process (MDP)*. Before we provide a formal definition for MDPs, we first have to specify the Markov property. In general, the probability of transitioning to state s_{t+1} and observing reward ρ_{t+1} after action a_t was executed in state s_t can depend on the entire history of past states, actions, and rewards. Formally, this probability is given by

$$\Pr(s_{t+1}, \rho_{t+1} \mid s_t, a_t, \rho_t, s_{t-1}, a_{t-1}, \rho_{t-1}, \ldots, s_0, a_0, \rho_0) \tag{3.1}$$

The same probability may also depend only on the last state and action, i.e.

$$\Pr(s_{t+1}, \rho_{t+1} \mid s_t, a_t) \tag{3.2}$$

According to Sutton and Barto [SB98], an environment is *Markovian* or said to have the *Markov property*, if both equalities are equal for all states, actions, and rewards. Russell and Norvig [RN10] provide a more general definition of this Markov assumption by demanding that the second probability depends on a finite fixed number of previous observations instead of just the last state and action. Based on their understanding, Russell and Norvig would consider Sutton and Barto's definition as the simplest case and refer to an environment that fulfills the corresponding condition as *first-order* Markov Decision Process. Based on the Markov property, we now define a MDP as follows:

Definition 6 (Markov Decision Process). A *Markov Decision Process* (MDP) is described by a tuple $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$, where the finite set $\mathcal{S}$ contains all environment states and the finite set A denotes all agent actions. A state-transition function $\delta : \mathcal{S} \times A \to \Pi(\mathcal{S})$ returns a probability distribution of transitioning to state $s' \in \mathcal{S}$ after action $a \in A$ in state $s \in \mathcal{S}$ was executed. The actual state s' is derived according to that probability distribution. The reward function $\rho : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$ gives an immediate payoff for action $a \in A$ in state $s \in \mathcal{S}$ and transitioning to $s' \in \mathcal{S}$. The factor $\gamma \in [0, 1)$ is used for discounting the cumulative reward.

For simplicity, we may write $\delta(s, a, s')$ to denote the probability for reaching state s' after a was executed in s. Note that in a *deterministic* MDP, the state transition function reduces to $\delta : \mathcal{S} \times A \to \mathcal{S}$, and hence the reward function can also be simplified to $\rho : \mathcal{S} \times A \to \mathbb{R}$.

Although MDPs can be defined over infinite, e.g. continuous, state and action sets as well (see e.g. [GW99] or [Buş08]), we will only consider finite sets in this work. Furthermore, we assume bounded rewards. These common assumptions allow us to concentrate our investigations on the learning itself instead of dealing, e.g., with issues that follow from using infinite sets. Note also, that we included the discount factor into the formal definition of the MDP, although it determines how much future rewards are appreciated by a *particular* agent. By doing so, we have clearly defined the values that can be achieved by a learning agent. This approach does not limit the general model. Note that this kind of definition is found less frequently in the literature (e.g. in [NHR99] [Lit01]) than definitions which exclude the discount factor but consider it when it comes to concrete calculations of values (cf. [KLM96] [SB98] [Wei99] [HMLIR07] [PTT10] [RN10] [BBDS10]).

The goal of the agent is to learn a behavior that maximizes the return over time. Therefore, most methods consider a so-called *state-value function* which determines the "attractiveness" for an agent to be in particular state in terms of the (expected) return that can be achieved from this state. Clearly, this value depends on the agents behavior which is expressed by a (stochastic) policy $\pi : \mathcal{S} \times A \to [0, 1]$, where $\pi(s, a)$ denotes a probability for executing action a in state s. On the other hand, a *deterministic* policy $\pi : \mathcal{S} \to A$ returns a particular action for each state. Since the deterministic case is a special case of the stochastic one, we will continue our description with the more general case. The value $V^\pi(s)$ of a state $s \in \mathcal{S}$ under a policy π then is defined by the *expected return* the agent will achieve if it starts in state s and follows policy π in each time step t [SB98], i.e.

$$V^\pi(s) = E_\pi[R_t \mid s_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \rho_{t+k+1} \mid s_t = s \right], \tag{3.3}$$

where $E_\pi[\cdot]$ denotes the expected value under policy π. Similar to (3.3), we can define a *state-action value function* $Q^\pi(s, a)$ for executing action a in state s under policy π by

$$Q^\pi(s, a) = E_\pi[R_t \mid s_t = s, a_t = a] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \rho_{t+k+1} \mid s_t = s, a_t = a \right]. \tag{3.4}$$

Note that the value function (3.3) can also be expressed recursively, whereas (3.5) is known as the *Bellman equation for* V^π:

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}[R_t \mid s_t = s] \\
&= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \rho_{t+k+1} \mid s_t = s\right] \\
&= E_{\pi}\left[\rho_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k \rho_{t+k+2} \mid s_t = s\right] \\
&= \sum_{a \in A} \pi(s,a) + \sum_{s' \in \mathcal{S}} \delta(s,a,s') \\
&\quad \cdot \left[\rho(s,a,s') + \gamma E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \rho_{t+k+2} \mid s_{t+1} = s'\right]\right] \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in \mathcal{S}} \delta(s,a,s') \left[\rho(s,a,s') + \gamma V^{\pi}(s')\right] \qquad (3.5)
\end{aligned}
$$

Basically, the Bellman equation states that the value for a state is determined by the expected immediate reward plus the expected discounted value of the successor states weighted by their probability. Note that there is a set of Bellman equations, i.e. one for each state. Also note that the value function V^{π} is the unique solution to this set of equations.

Based on the above definitions, the optimal state-value function is given by

$$V^{\star}(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S}$$

and since a policy π is better than or equal to a policy π' if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$, an optimal policy $\pi^{\star}$ is formally defined as

$$\pi^{\star} = \arg\max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S}$$

Accordingly, the optimal action-value function for all $s \in \mathcal{S}$ and $a \in A$ is given by

$$Q^{\star}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

which can be expressed depending on $V^{\star}$ as shown in (3.6) since its value denotes the expected immediate reward plus the discounted expected return for following an optimal policy afterwards:

$$Q^{\star}(s,a) = E[\rho_{t+1} + \gamma V^{\star}(s_{t+1}) \mid s_t = s, a_t = a] \qquad (3.6)$$

Since the optimal value $V^{\star}(s)$ of a state s is determined by executing the action which results in the best expected immediate reward plus the best expected value that can be obtained afterwards, it holds that [SB98]:

$$
\begin{aligned}
V^{\star}(s) &= \max_{a \in A} Q^{\pi^{\star}}(s,a) \\
&= \max_{a \in A} E_{\pi^{\star}}\left[R_t \mid s_t = s, a_t = a\right] \\
&= \max_{a \in A} E_{\pi^{\star}}\left[\sum_{k=0}^{\infty} \gamma^k \rho_{t+k+1} \mid s_t = s, a_t = a\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \max_{a \in A} E_{\pi^\star} \left[\rho_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k \rho_{t+k+2} \mid s_t = s, a_t = a \right] \\
&= \max_{a \in A} E \left[\rho_{t+1} + \gamma V^\star(s_{t+1}) \mid s_t = s, a_t = a \right] \\
&= \max_{a \in A} \sum_{s' \in \mathcal{S}} \delta(s, a, s') \left[\rho(s, a, s') + \gamma V^\star(s') \right] \qquad (3.7)
\end{aligned}
$$

Equation (3.7) is known as *Bellman optimality equation for* $V^\star$. Transfered to the state-action function, the Bellman optimality equation for $Q^\star$ is given by [SB98]:

$$
\begin{aligned}
Q^\star(s, a) &= E \left[\rho_{t+1} + \gamma \max_{a' \in A} Q^\star(s_{t+1}, a') \mid s_t = s, a_t = a \right] \\
&= \sum_{s' \in \mathcal{S}} \delta(s, a, s') \left[\rho(s, a, s') + \gamma \max_{a' \in A} Q^\star(s', a') \right] \qquad (3.8)
\end{aligned}
$$

Finally returning to the learning task of an agent, we can state that the agent's goal is to learn an optimal policy $\pi^\star$ that satisfies the Bellman optimality equations. Having learned the optimal $Q^\star$-values, for instance, an optimal policy can easily be obtained by choosing the action with the maximum Q-value in each state.

Given complete model knowledge on transition probabilities and rewards, one may solve the set of Bellman equations in order to solve the reinforcement learning problem. However, this quickly becomes infeasible, for instance due to large state and action spaces or due to the lack of model knowledge. In the following subsection, we will describe different techniques for finding (near-)optimal solutions.

3.2.2 Basic Methods

As of the previous section, an optimal policy $\pi^\star$ describes a solution to a Markov Decision Process. Finding such an optimal solution can be accomplished in various ways. Reinforcement Learning algorithms can mainly be classified in *model-free* and *model-based*. In a model-based approach, the dynamics of the MDP, i.e. state-transition and reward function, are known and used in Dynamic Programming algorithms. Model-free approaches initially have no knowledge about the MDPs dynamics but gather information from interacting with the environment. Model-free approaches include Monte Carlo and temporal difference methods. Based on [SB98] and [OW12], the core ideas of these three kinds of algorithms can briefly be summarized as in the following list. Additional introductory literature includes [KLM96] and [RN10].

Dynamic Programming: Dynamic programming (DP) based algorithms require a perfect model of the considered MDP, i.e. state transition function δ and reward function ρ. Given such a model, the idea is to calculate the value function in order to construct an optimal policy. Having this model knowledge, the Bellman equations describe a set of $|\mathcal{S}|$ equations with $|\mathcal{S}|$ unknowns. Although one might solve these equations with different mathematical techniques, fundamental algorithms iteratively construct a solution. Therefore, a *policy evaluation* step iteratively approximates the state-value function V^π for a policy π by turning the Bellman optimality equation into an update rule. Given a specified approximation quality, the policy evaluation step stops and returns the (approximated) value

function. Knowing this value function, one can try to improve the underlying policy. Therefore, the *policy improvement* theorem is used. It roughly states that for a given policy π, if deviating from that policy in any state s by choosing an action $a \neq \pi(s)$ and then following π results in a higher value than $V^{\pi}(s)$ then this new policy is better than π. In that case, one evaluates the new policy using another policy evaluation step. Again, the resulting value function can be used for a policy improvement step and, thus, in the long run, one iteratively improves the policy until an optimal policy is found. The latter happens, if deviating from a policy does never lead to better values. Basically, this approach is known as policy iteration. Another approach that avoids unnecessary updates of the value function is known as value iteration. It mainly differs from policy iteration in the way how and when policy evaluation and policy improvement steps are interleaved and carried out.

DP methods like policy or value iteration can quickly become inefficient as either the policy evaluation process becomes expensive or the number of required iterations increases. Further details on the complexity of these approaches can be found e.g. in [OW12] and the references therein. More details on the aforementioned DP methods and even more approaches that fall into this category, including formal descriptions and results, can be found in [SB98].

Monte Carlo: Different from DP, Monte Carlo (MC) methods do not require a full model of the MDP. Instead, these approaches learn from interaction with an environment. The key idea is to consider the expected return as a random variable and to learn its expectation value from average returns that are obtained from samples [OW12]. By the law of large numbers, and given enough samples, it can be expected that the average return obtained for starting in a state and then following a policy will converge to the true value of that state for the given policy. Since the policy is fixed, maybe not all state-action transitions will be visited. Thus, exploration is an important issue in MC methods. One approach to overcome this problem can be found in *exploring starts* [SB98], which demands that each state-action pair becomes the starting point of a return-generating episode with a non-zero probability. In particular, MC methods are considered for episodic tasks since they require reaching a terminal state in order to calculate proper returns. Accordingly, this step corresponds to the policy evaluation process found in DP methods. The difference is that MC methods estimate V^{π} rather from "experiments" than from model knowledge. Given the state-value or state-action value function of π, this policy can again be improved like in DP algorithms. Repeating both steps finally allows MC methods to converge to optimal policies. Note that this general schema of interacting policy evaluation and improvement steps is known as *generalized policy iteration* (cf. [SB98, Ch. 4]).

Although Monte Carlo methods do not require model knowledge, one drawback of these methods is that they potentially may require many episodes for convergence to optimal solutions. In contrast to DP approaches, MC-based algorithms do not rely on updating estimated values based on other (older) estimates.

Temporal Difference Learning: Methods from this category try to combine the advantages of Monte Carlo and Dynamic Programming methods. In detail, they try to combine learning from small experiences with updating values based on previous estimates [OW12] [SB98], i.e. they use state, action, and reward tuples $\langle s_t, a_t, \rho_{t+1} \rangle$ instead of learning from the experience (return) obtained after a whole episode.

Accordingly, and different from MC-based approaches, Temporal Difference (TD) methods can deal with continuous tasks and are able to learn online, i.e. while interaction with the environment takes place. A further advantage of TD methods compared to DP is that they do not require a model of the considered MDP. Like in MC methods, exploration is of key importance in TD. Since Temporal Difference learning was one of the main novel ideas in reinforcement learning, many different approaches exist that we do not intend to discuss in detail. Instead, we refer to [SB98] for a comprehensive and exhaustive overview.

From these basic approaches it is not surprising that the most influential reinforcement learning algorithm of the past decades is a temporal difference method. It is called Q-Learning and will be described in the next section.

3.2.3 Q-Learning

Q-Learning [Wat89] is an off-policy approach[1] which iteratively updates estimation $\widehat{Q}$ of the state-action value function Q for each state-action pair. In [WD92], Q-Learning is shown to converge to optimal $Q^\star$ values under some conditions that we will present shortly. We will consider two variants of Q-Learning, one for deterministic MDPs and one for general (non-deterministic) MDPs.

We begin with the deterministic version and follow the description given by Mitchell [Mit97]. In a deterministic MDP, state transitions and rewards are deterministic. Hence, the Bellman optimality equation (3.8) for $Q^\star$ is reduced to

$$Q^\star(s,a) = \rho(s,a) + \gamma \max_{a' \in A} Q^\star(s',a'), \tag{3.9}$$

where $s' = \delta(s,a)$ is the state resulting from executing action a in state s. The idea of Q-Learning now is to iteratively approximate $Q^\star(s,a)$ for all state-action pairs by turning (3.9) into an update rule. The approximation is denoted by $\widehat{Q}$ and its value at iteration t is $\widehat{Q}_t(s,a)$. Although arbitrary values may be used for initialization, the initial values usually are set to $\widehat{Q}_0(s,a) = 0, \forall s \in \mathcal{S}, a \in A$. Now, in any iteration t the agent is in a state s_t and executes a (random) action a_t according to some *behavior policy* [SB98]. In the next iteration, the agent then obtains a reward $\rho_{t+1}(s_t,a_t)$ for its previous behavior. The $Q^\star$-value approximations then are updated according to the following equation

$$\widehat{Q}_{t+1}(s,a) = \begin{cases} \widehat{Q}_t(s,a) & \text{, if } s \neq s_t \text{ or } a \neq a_t \\ \rho_{t+1}(s_t,a_t) + \gamma \max\limits_{a' \in A} \widehat{Q}_t(\delta(s_t,a_t),a') & \text{, otherwise.} \end{cases} \tag{3.10}$$

[1] An *off-policy* approach learns an *estimation* policy without executing it. The term estimation policy reflects that the agent tries to estimate an optimal but unknown policy. While learning that estimation policy, the agent follows a *behavior* generating policy which usually takes care of executing each possible action in each state with a certain non-zero probability. In contrast, an *on-policy* approach requires to follow the currently learned policy during the learning process. See [SB98] for further details.

For $t \to \infty$, this update rule leads to convergence of $\widehat{Q}_t(s,a) \to Q^\star(s,a), \forall s \in \mathcal{S}, a \in A$, i.e. the agent will learn an optimal policy. This is proven, for instance, in Theorem 13.1 in Mitchell's book [Mit97]. Since we will later refer to this theorem, we quote it using our notations:

Theorem 1 (Convergence of Q-Learning for deterministic Markov Decision Processes). *Consider a Q-Learning agent in a deterministic MDP with bounded rewards $(\forall s,a)|\rho(s,a)| \leq c$. The Q-Learning agent uses the training rule (3.10), initializes its table $\widehat{Q}(s,a)$ to arbitrary finite values, and uses a discount factor γ such that $0 \leq \gamma < 1$. Let $\widehat{Q}_t(s,a)$ denote the agent's hypothesis $\widehat{Q}(s,a)$ following the t-th update. If each state-action pair is visited infinitely often, then $\widehat{Q}_t(s,a)$ converges to $Q^\star(s,a)$ as $t \to \infty$, for all s, a.*

The idea of the corresponding proof is the fact that the maximum error of the estimations compared to the optimal values is reduced after every interval in which each state-action pair was visited. Note that the convergence requirements hence demand an infinite number of iterations. In practice, however, Q-Learning converges much faster to optimal policies. In large state-action spaces, e.g. in continuous state spaces, additional approximation techniques are required, which we will not consider here but refer to surveys like [BEDSB11] or [vH12].

In a non-deterministic MDP both, state transition function δ and reward function ρ, might have non-deterministic outcomes. Hence, given a (more or less) random reward for the same state-action pair, the deterministic update rule will constantly change $\widehat{Q}$. This prevents convergence of deterministic Q-Learning to an optimal policy in general non-deterministic MDPs [Mit97]. Accordingly, another update rule for such settings is required. Watkins and Dayan [WD92] update the estimates according to (3.11), where α_t denotes the learning rate at iteration t. The rule's idea is to calculate a "decaying weighted average of the current $\widehat{Q}$ value and the revised estimate" [Mit97].

$$\widehat{Q}_{t+1}(s,a) = \begin{cases} \widehat{Q}_t(s,a) & \text{, if } s \neq s_t \text{ or } a \neq a_t \\ (1-\alpha_{t+1})\widehat{Q}_t(s,a) + \alpha_{t+1} & \\ \quad \cdot \left[\rho_{t+1}(s_t,a_t) + \gamma \max_{a' \in A} \widehat{Q}_t(\delta(s_t,a_t),a')\right] & \text{, otherwise} \end{cases} \tag{3.11}$$

The correct choice of learning rates is important with respect to convergence issues. In [WD92], Watkins and Dayan proved Theorem 2 that states conditions on learning rates such that Q-Learning with the update rule (3.11) converges to an optimal policy:

Theorem 2 (Convergence of Q-Learning). *Let $n(j,s,a)$ be the index of the j-th time that action a is tried in state s. Given bounded rewards $|\rho_t| \leq c$, learning rates $0 \leq \alpha_n < 1$, and*

$$\sum_{j=1}^{\infty} \alpha_{n(j,s,a)} = \infty, \sum_{j=1}^{\infty} \left[\alpha_{n(j,s,a)}\right]^2 < \infty, \forall s \in \mathcal{S}, a \in A, \tag{3.12}$$

then $\widehat{Q}(s,a) \to Q^\star(s,a)$ as $t \to \infty$, $\forall s, a$ with probability 1.

A simple way to calculate such learning rates can be found in [Mit97]. He proposes to use $\alpha_t = \frac{1}{1+\text{visits}_t(s,a)}$, where $\text{visits}_t(s,a)$ is a counter that indicates how often the corresponding state-action pair has been visited until the t-th iteration.

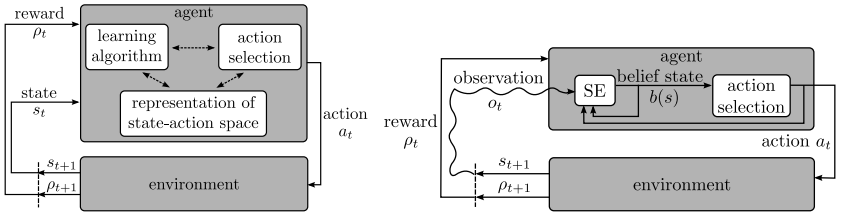
3.2.4 Partially Observable Markov Decision Processes

Up to now, we only considered approaches which require full observability, i.e. an agent has to be able to observe the full system state. Clearly, this may not be practical in every scenario, for instance, if a robot can only observe parts of its environment due to limited sensor ranges. In that case, other models and techniques have to be considered. One of these models is the *Partially Observable Markov Decision Process (POMDP)*. We quote its definition from Kaelbling et al. [KLC98] and adjust the identifiers to match our notations:

Definition 7 (Partially Observable Markov Decision Process). A partially observable Markov Decision Process can be described as a tuple $\langle \mathcal{S}, A, \delta, \rho, \Omega, O \rangle$, where

- $\mathcal{S}, A, \delta$, and ρ describe a Markov Decision Process;
- Ω is a finite set of observations the agent can experience of its world; and
- $O : \mathcal{S} \times A \to \Pi(\Omega)$ is the observation function, which gives, for each action and resulting state, a probability distribution over possible observations (we write $O(s', a, o)$ for the probability of making observation o given that the agent took action a and landed in state s').

Accordingly, a POMDP basically describes a MDP in which the agent only makes observations of the environment state and tries to maximize the expected return over the long run [KLC98]. Figure 3.2 opposes MDP and POMDP. Due to partial observations, an agent in a POMDP needs to work with *belief states* obtained from a state estimator component (SE) instead of (environment) states. In order to solve a POMDP, different techniques can be applied. Since we will mainly deal with Q-Learning in fully observable settings, we will not present those details here. Instead, we refer the reader to surveys like [Mon82], [Lov91], [Abe03], or the more recent [Spa12].



(a) Markov Decision Process. (b) Partially Observable Markov Decision Process.

Fig. 3.2 Two state-of-the-art models that are used for single agent RL [KKB11c].

3.2.5 Discussion

In this section, we have defined the reinforcement learning problem in a formal way and briefly introduced ideas of fundamental techniques. In particular, Q-Learning was presented in detail as it builds the core of the multiagent reinforcement learning approach proposed in this work.

Although Q-Learning is proven to converge to optimal strategies, it has—like most approaches—some issues which might also be traced back to general aspects found in RL. Since we just slightly touched some of them in the above descriptions, let us now identify the main problems in the following list:

Credit Assignment: Since the agent only gets feedback in the form of an immediate reward, it is hard to decide which action or (sub-)sequence of actions from a longer interaction sequence was responsible for reaching a goal. That is, there exists a *delay* between obtaining a reward and observing its effect. Hence, assigning credit to actions may not be easy and is known as *temporal credit assignment problem*.

Exploration-Exploitation Tradeoff: Clearly, an agent is interested in maximizing its return. Accordingly, it should on the one hand *exploit* actions that have been found to result in good rewards. Sticking only to such actions, on the other hand, may not be advisable since there might be other actions that could potentially lead to even better rewards. Since the agent is usually not aware of the dynamics of its environment, it is forced to interact with the environment in a *trial-and-error* manner so as to *explore* its world and to obtain new knowledge. Too much exploration is clearly also not advisable as this may result in a low return. Hence, the agent has to find a good tradeoff between exploring its environment and exploiting learned knowledge. Also note that the agent's action choices, particularly in Monte Carlo and Temporal Difference methods, influence the generated samples that are used for learning.

Curse of Dimensionality: The curse of dimensionality refers to the *exponential growth* of state-action spaces with an increasing number of variables. In particular, since most approaches maintain a value for each state-action pair in a tabular form, learning and storing all these values quickly becomes computationally expensive.

Partial Observability: The methods presented above rely on perfect observation of states. In realistic problems, however, this assumption does often not hold. Hence, a good policy might include actions that do not directly lead to the desired goal behavior but are used to gather information on the state. In such cases, it is hardly possible to reach the (theoretically) maximal returns.

We should also mention here that a large number of different approaches to deal with these challenges exist. Since it is natural that one approach performs well in some settings but (slightly) worse in another, it is hence left to conclude that single agent reinforcement learning is powerful tool. In particular, its extension to multiagent settings is interesting and will be presented in the next section.

3.3 Multiagent Reinforcement Learning

In this section, we describe the extension of single agent reinforcement learning to settings with multiple agents, mainly based on [BBDS10]. In particular, we first define the formal framework and the learning problem. Then, we provide a short overview on algorithms for cooperative settings and give an in-depth presentation of an algorithm called Distributed Q-Learning that is fundamental to our work. Finally, we close this section with a short discussion.

Since the remainder cannot describe all works in detail, please consult recent surveys on *Multiagent Reinforcement Learning (MARL)* for further information. In

particular, have a look at [YG04], [HTP+06], and [BBDS08] resp. its extended and revised version [BBDS10]. Another survey focuses on learning in cooperative MAS from a wider perspective [PL05].

3.3.1 Stochastic Games

Single agent settings are usually modeled as Markov Decision Process. A natural generalization of this model is a stochastic game [Sha53] (or Markov game), which constitutes the framework for most MARL work (cf. [BBDS10]). In [KKB12], we defined a stochastic game as follows:

Definition 8 (Stochastic Game). A *stochastic game (*SG*)* is defined by a tuple $\Gamma = \langle s_0, \mathcal{S}, \mathcal{A}, U, \delta, \{\rho_i\}_{i \in \mathcal{A}} \rangle$, where $s_0 \in \mathcal{S}$ is a start state from the set of states $\mathcal{S}$. $\mathcal{A}$ is a set of agents playing the game and $U = \times_{i \in \mathcal{A}} A_i$ is the set of joint actions, where A_i denotes the actions available to agent $i \in \mathcal{A}$. The state-transition function $\delta : \mathcal{S} \times U \times \mathcal{S} \to [0, 1]$ returns the probability of transitioning to state $s' \in \mathcal{S}$ after joint action $u \in U$ in state $s \in \mathcal{S}$ was executed. The set of all reward functions is given as $\{\rho_i\}_{i \in \mathcal{A}}$ having $\rho_i : \mathcal{S} \times U \to \mathbb{R}$ for all agents $i \in \mathcal{A}$.

Before we continue with a classification of stochastic games, we will first provide an alternative perspective for their definition. Therefore, we consider the static game (or stage game) that belongs to a particular state of a SG (cf. [SLB09]). Such a *static (stateless) game* can be defined as follows:

Definition 9 (Static (Stateless) Game). A static (stateless) game is described by a tuple $G = \langle \mathcal{A}, U, \{\rho_i\}_{i \in \mathcal{A}} \rangle$, where $\mathcal{A}$ denotes the set of agents, $U = \times_{i \in \mathcal{A}} A_i$ the set of joint actions composed of the agent individual action sets A_i, and $\rho_i : U \to \mathbb{R}$ the reward function of agent i.

From that perspective, a stochastic game hence consists of a set of states and a state transition function such that in each state a certain static game is played and afterwards a state transition occurs.

To be more precise, a static game that is played in a state of a SG is usually referred to as *stage game*. Since states in stochastic games tend to be visited more than once, the same stage game is played repeatedly, which means that one may also refer to a stage game as *repeated* game [BBDS10]. In general, stochastic games are a unifying and powerful framework as they include MDPs, which are stochastic games with a single agent, as well as *repeated games*, which are stochastic games with a single state [SLB09].

To clearly distinguish between settings with states, i.e. general MDPs or stochastic games, and stateless settings, i.e. static games or stage games, we will from now on use the term *strategy* if we talk about a policy in a static game. An agent's strategy $\sigma_i : A_i \to [0, 1]$ returns the probability of executing an action in a stateless setting. In contrast, a *policy* $\pi_i : \mathcal{S} \times A \to [0, 1]$ always refers to settings with states. If not mentioned otherwise, we will concentrate in this work on settings with deterministic policies and strategies. A deterministic policy can be defined as $\pi_i : \mathcal{S} \to A$, and a deterministic strategy reduces to executing a certain action, i.e. $\sigma_i : \varnothing \to A$.

Stochastic games can be classified into three categories depending on the reward functions: fully competitive, fully cooperative (or collaborative), and mixed [BV00], [BBDS10]. To explain the first class, consider a SG with only two agents $i_1, i_2 \in \mathcal{A}$

(more agents are possible in general) and having $\rho_1 = -\rho_2$ as reward functions. Then, this zero-sum game is (fully) competitive as the win of agent i_1 is the loss of agent i_2 and vice versa. Given that all agents share the same reward function, i.e. $\rho_i = \rho_j, \forall i, j \in \mathcal{A}$, the stochastic game is called (fully) cooperative. In particular, all agents obtain the same reward and thus try to optimize the same return over time, i.e. they intend to reach the same goal. The third class, *mixed* stochastic games, does not fall in any of the two aforementioned classes.

Different approaches for all categories exist. However, many algorithms try to solve a SG by finding strategies for each stage game while their combination forms a solution to the SG [BBDS10]. In the next section, we will provide some basic approaches for cooperative games. Further details on other algorithms can be found in relevant surveys like those listed in the introduction to this section.

Next, let us briefly discuss the learning goal in stochastic games. In competitive games, it is often hard to state which policy is optimal since the agents' returns strongly depend on the behavior of all agents. Thus, multiagent reinforcement learning algorithms often search for equilibria, in particular, *Nash equilibria*. In a static game, a Nash equilibrium [Nas51] basically describes a set of strategies such that no agent can benefit by deviating from its strategy if all other agents stick to their current strategy. To formally define such an equilibrium based on [BBDS10], let Σ_i denote the set of strategies of agent i and let $\sigma_{-i} = \langle \sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots \sigma_n \rangle$ describe a strategy profile of all agents except i. A *best response* $\sigma_i^\star$ of agent i to a strategy profile σ_{-i} leads to the maximum expected reward ρ_i that can be achieved if all other agents follow the given strategy profile. Formally, for a best response $\sigma_i^\star$ it holds that:

$$E\left[\rho_i \mid \sigma_{-i} \cup \sigma_i\right] \leq E\left[\rho_i \mid \sigma_{-i} \cup \sigma_i^\star\right], \forall \sigma_i \in \Sigma_i. \tag{3.13}$$

A joint strategy $\boldsymbol{\sigma}^\star = \langle \sigma_1^\star, \sigma_2^\star, \ldots, \sigma_n^\star \rangle$, which solely consists of best responses, is called *Nash equilibrium (NE)*. In particular, a Nash equilibrium in a static game reduces to a joint action that leads to the highest value [CB98].

Although there is an on-going discussion on the usefulness of NE as solution concept in the context of MARL (cf. [SPG07]), many algorithms seek for Nash equilibria. In particular, if multiple NE exists, convergence to an "arbitrary" NE can lead to suboptimal performance. Also, aiming for NE assumes rational agents with a utilitarian behavior. In practice, however, agents (or humans) often do not act rational. Furthermore, it is not yet clear how combined strategies for general stochastic games which are composed of strategies that lead to NE in the stage games relate to each other [BBDS10]. Besides convergence to equilibria, which basically expresses the desire to find a *stable* solution, also the ability of an agent to *adapt* to other learning agents is usually considered to be a goal in MARL [BBDS10]. A further discussion on this issue is left out in this work since we focus on cooperative settings.

Other than in competitive games, the learning goal in cooperative stochastic games is simpler. Here, the goal of all agents is to maximize the same expected (discounted) return over time. In a cooperative static game with deterministic strategies, the agents implicitly determine a Nash equilibrium while they try to find a joint action that leads to the highest reward. Games in which agents try to reach a common goal are often called *common payoff* or *team games*. In the next section, we will present some basic approaches towards reinforcement learning in cooperative stochastic games.

3.3.2 Algorithms for Cooperative Stochastic Games

As mentioned above, the goal of all agents in a cooperative SG is to maximize a common expected (and discounted) return over time. Note that joint actions are defined through all individual agent actions and thus are responsible for the immediate reward that is given to all agents. Based on this observation a simple learning algorithm can be constructed to solve the problem by basically reducing the cooperative stochastic game to a central instance Markov Decision Process. In detail, given a cooperative stochastic game $\Gamma = \langle s_0, \mathcal{S}, \mathcal{A}, U, \delta, \{\rho_i\}_{i \in \mathcal{A}} \rangle$, a discount factor γ, and a central instance, transform Γ into a Markov Decision Process $\mathrm{M} = \langle \mathcal{S}', A, \delta', \rho', \gamma' \rangle$ such that:

- $\mathcal{S}' = \mathcal{S}$, i.e. the MDP and the SG share the same state set.
- The action set A is constructed by considering each joint action $u \in U$ to be an atomic action $a \in A$ of a central agent.
- The dynamics of the state transition function δ are preserved in δ' by defining the latter such that performing an action $a \in A$ in a state $s \in \mathcal{S}'$ leads to the same dynamics as performing the underlying joint action in the SG.
- Since all reward functions $\{\rho_i\}_{i \in \mathcal{A}}$ in the cooperative SG are equal, we can use one of these functions to construct ρ' such that it, depending on the states, returns the same reward for an action $a \in A$ as for the corresponding joint action in the stochastic game.
- $\gamma' = \gamma$, i.e. the MDP and the SG share the same discount factor.

Given such a transformed MDP, we can use any algorithm for single agent reinforcement learning, e.g. Q-Learning, to solve it. The resulting policy then can be decomposed into agent specific policies and transferred to the agents in the stochastic game. We will refer to this approach as *centralized (Q-Learning) approach*. Note that this procedure is not new and is already described e.g. in [LR00] and [BBDS10].

Demanding a central instance is a strong assumption as agents are autonomous in their decisions and, in particular, may not even be able to communicate with each other. Also from a computational point of view, this approach suffers from huge state-action spaces which grow exponentially in the number of agent actions and agents. Hence, maintaining a central Q-table that holds values for all state and joint action pairs soon becomes impractical. Accordingly, other approaches than the one described above have been developed, and some of them will be presented in the next paragraphs.

Before we do so, let us briefly classify MARL algorithms according to Claus and Boutilier [CB98]. According to their work, one can consider an agent to be an *independent learner (IL)* or a *joint action learner (JAL)*. An independent learner ignores the presence of other agents, for instance because it is simply not aware of them or cannot observe the other agents' actions. The agent then learns Q-values for its own actions. Due to the influence of the other agents, a deterministic environment hence can become non-deterministic from the individual agent's point of view. A joint action learner accordingly learns values for the joint actions, that is it has to be able to observe the other agents. This is a strong assumption, especially in large systems. Finally, there are also learners in between, i.e. those who consider only actions of some neighbors or those who consider other actions only in states where coordination is required.

In [GLP02], Guestrin et al. present an approach that belongs to the latter class. It decomposes the joint value function into independent local value functions such that

they can only be influenced by a smaller number of agents in the system. That is, a larger learning problem is split into several smaller independent problems which can be solved more efficiently. The local results are used to derive an (optimal) joint policy for the global problem. Therefore, the algorithm relies either on communication or on a common observation. However, creating such a factorization of the problem is not always possible.

Wang and Sandholm [WS03] have proposed the first algorithm that is—under some assumptions on e.g. convergence rates—guaranteed to converge to optimal solutions in general fully cooperative stochastic games. The core idea of the approach is to eliminate suboptimal equilibria in each stage game by creating and playing a virtual game that reflects the properties of the underlying stage game. Therefore, those joint actions that lead to the highest reward in a stage game return a reward of 1 in the corresponding virtual game, and all suboptimal joint actions return 0. The approach, called *Optimal Adaptive Learning (OAL)*, therefore does not even need to know the game structure in advance. Instead, these virtual games are learned simultaneously while they are used to determine an optimal policy of the actual game. Though this approach is able to solve cooperative stochastic games, it produces high costs and storage requirements due to the virtual games.

Kapetanakis and Kudenko [KK02] present an independent learner approach based on Q-Learning that is called *Frequency Maximum Q Value (FMQ)*. It uses experiences in order to guide the agents towards choosing those actions that led to high rewards. In detail, given a cooperative repeated game, FMQ uses Q-Learning with the Boltzmann action selection that calculates a probability for executing a certain action using

$$\Pr(a) = \frac{e^{\frac{\mathrm{EV}(a)}{T}}}{\sum_{a' \in A_i} e^{\frac{\mathrm{EV}(a')}{T}}},$$

where T denotes a temperature that controls how random actions are selected and $\mathrm{EV}(a)$ denotes the estimated value of an action. The heuristic itself defines how the estimated value of an action is calculated. Kapetanakis and Kudenko use the following equation

$$\mathrm{EV}(a) = Q(a) + c \cdot \mathrm{freq}(\mathrm{maxR}(a)) \cdot \mathrm{maxR}(a) \tag{3.14}$$

where $\mathrm{maxR}(a)$ reflects the highest reward obtained for executing action a and $\mathrm{freq}(\mathrm{maxR}(a))$ gives the probability for observing the maximum reward whenever action a was executed. The game-depending factor c weights the influence of the FMQ heuristic. Note that the Q-value is calculated only for actions since the approach deals with repeated games. It was shown experimentally that FMQ is able to converge with almost 100% probability to an optimal joint action. FMQ is also able to find optimal joint actions in games that contain some slightly random rewards. However, this only works in restricted settings under specific assumptions. In games with general stochastic rewards, FMQ performs poorly. Nevertheless, this approach has shown that proper coordination techniques can help to improve the results of Q-Learning in multiagent settings compared to basic settings like those considered in [CB98]. In [MLFP08], some experiments show that an extended FMQ heuristic is able to slightly better deal with stochastic reward settings than the original heuristic.

In the next subsection, we will present another independent learner method called Distributed Q-Learning. A variant of it will build the core of our later presented learning approach.

3.3.2.1 Distributed Q-Learning

In [LR00], Lauer and Riedmiller proposed *Distributed Q-Learning (*DQ*)*, an algorithm for fully cooperative deterministic stochastic games that provably converges to optimal joint policies. The idea of this independent learner approach is based on the so-called *optimistic assumption*, which means that each agent assumes that the others play optimally. Instead of learning a global Q-table that contains values for state and joint action pairs, like in the centralized Q-Learning approach described above, DQ uses local tables. These tables are denoted by a lowercase q^i, whereas the superscript refers to an agent $i \in \mathcal{A}$. Each agent learns local q-values for each state and agent individual action, i.e. for all $q^i(s,a), s \in \mathcal{S}, a \in A_i$, such that the local values for an action always capture the maximum value that would occur in a central Q-table for that action. The update rule used by the agents contains a maximum argument which ensures that the q-values are only updated on improvements, i.e. if the new q-value estimation is higher than the previous one. In order to ensure coordination towards the same joint action, agents change their policy only if a real improvement is observed. Given bounded positive rewards, it can be proven that Distributed Q-Learning converges to optimal joint policies. Based on [LR00], we will present the formal details of this approach in the remainder of this section.

First, each agent initializes its local q^i-table for all state-action pairs to zero, i.e. $q_0^i(s,a) = 0, \forall s \in \mathcal{S}, a \in A_i$. After performing an action in a state at time step t, the agent receives the common reward in the next time step and updates its table according to (3.15):

$$q_{t+1}^i(s,a) = \begin{cases} q_t^i(s,a) & \text{, if } s \neq s_t \text{ or } a \neq a_t^i \\ \max\{q_t^i(s,a), & \\ \quad \rho_{t+1}(s_t,u_t) + \gamma \max\limits_{a' \in A_i} q_t^i(\delta(s_t,u_t),a')\} & \text{, otherwise} \end{cases} \tag{3.15}$$

where $a_t^i \in A_i$ denotes the agent's action and s_t the state at time step t. Furthermore, $u_t = \langle a_t^0, \ldots, a_t^i, \ldots a_t^{n-1} \rangle$ describes the joint action of all n agents at that time step. By using the maximum operator, updates are only performed if the approximated value for an action is improved. Given this update rule, Lauer and Riedmiller [LR00] proved Prop. 1, which is reformulated to match the terms and notations used in this work:

Proposition 1. *Let $\Gamma = \langle s_0, \mathcal{S}, \mathcal{A}, U, \delta, \{\rho_i\}_{i \in \mathcal{A}} \rangle$ be a deterministic cooperative stochastic game with $n = |\mathcal{A}|$ agents and positive bounded rewards $\rho(s,u) \geq 0, \forall s \in \mathcal{S}, u \in U$, and let $\gamma \in [0,1)$ be the discount factor. Then for every time step t, agent i, state s, and elementary action a the following equation holds:*

$$q_t^i(s,a) = \max_{\substack{u = \langle a^0, \ldots, a^{n-1} \rangle \\ a^i = a}} Q_t(s,u)$$

whereby Q_t-values are computed by the centralized Q-Learning algorithm and the q_t^i-tables are initialized to zero and updated according (3.15) with the same sequence of current states s_t and joint actions u_t.

Given that each state action pair is visited infinitely often (like in single agent Q-Learning) and by the above proposition, it follows that the local q^i-values capture the maximum value achievable for an action a in a state s.

As Lauer and Riedmiller point out, this result is not sufficient to learn optimal joint policies in any cooperative game. In particular, if multiple joint actions lead to the same highest return, then the agents might choose actions that belong to different optimal joint actions. Hence, their combination could result in a suboptimal joint action. Accordingly, a tie-breaking rule has to be established to ensure mutual agreement on a particular (optimal) joint action. This is done by initializing the policy $\pi_0^i(s)$ of an agent i for each state $s \in \mathcal{S}$ with an arbitrary action from the agent's action set and then updating it according to (3.16):

$$\pi_{t+1}^i(s) = \begin{cases} \pi_t^i(s) & \text{, if } s \neq s_t \text{ or } \max\limits_{a' \in A_i} q_t^i(s,a) = \max\limits_{a' \in A_i} q_{t+1}^i(s,a) \\ a_t^i & \text{, otherwise} \end{cases} \tag{3.16}$$

where a_t^i is agent i's action at time step t that is contained in joint action $u_t = \langle a_t^0, \ldots, a_t^i, \ldots a_t^{n-1} \rangle$. Clearly, the policy is only updated if a (real) improvement of the local q-values is observed. It follows, that agents will stick to the first observed optimal joint action, since all other optimal joint actions cannot result in a higher value and, thus, never fulfill the required conditions for changing the policy.

Given this policy update rule, Lauer and Riedmiller prove the following, again reformulated, proposition:

Proposition 2. *Let $\Gamma = \langle s_0, \mathcal{S}, \mathcal{A}, U, \delta, \{\rho_i\}_{i \in \mathcal{A}} \rangle$ be a deterministic cooperative stochastic game with $n = |\mathcal{A}|$ agents and positive bounded rewards $\rho(s,u) \geq 0, \forall s \in \mathcal{S}, u \in U$, and let $\gamma \in [0,1)$ be the discount factor. Then for every time step t and state s it is:*

$$Q_t\left(s, (\pi_t^0(s), \ldots, \pi_t^{n-1}(s))\right) = \max_{u \in U} Q_t(s,u)$$

This means that the combination of the current policies $(\pi_t^0(s), \ldots, \pi_t^{n-1}(s))$ computed by the policy update rule (3.16) is greedy with respect to the non-distributed Q-table created with the centralized Q-Learning algorithm with the same sequence of current states s_t and joint actions u_t.

Finally, they conclude that Distributed Q-Learning converges to optimal joint policies due to Prop. 2 and the convergence of single agent Q-Learning.

Lauer and Riedmiller also investigate their approach in a stochastic environment. In this case, Distributed Q-Learning, even for a q-value update rule that reflects stochastic environments, is unable to converge to optimal joint policies. This follows because the agents cannot distinguish the influence of stochastic rewards and that of the other agents' action choices. In a later work [LR04], Lauer and Riedmiller provided another approach that can learn optimal joint policies in stochastic environments. That approach uses a special list data structure to enable agents to mutually agree on joint actions without communication or observations on the other agents.

3.3.3 Decentralized Partially Observable Markov Decision Processes

While previously presented approaches and models dealt only with fully observable environments, i.e. agents are able to observe the complete environment state, this section briefly presents a model for partially observable environments.

In single agent settings, Partially Observable Markov Decision Processes (POMDP) can be used to model partial observability. The generalization of this model in multi-agent settings are cooperative partially observable stochastic games which generally are known as *decentralized partially-observable Markov Decision Processes (Dec-POMDP)*. Based on [BGIZ02] (and influenced by [BHZ04] and [KKB11c]), we define a Dec-POMDP as follows:

Definition 10 (Decentralized Partially Observable Markov Decision Process). A *decentralized partially-observable Markov Decision Process (Dec-POMDP)* is described by a tuple $\langle \mathcal{S}, \mathcal{A}, U, \delta, \rho, \mathbf{\Omega}, O \rangle$, where

- $\mathcal{S}$ is a finite set of states with an initial state s_0.
- $\mathcal{A}$ is the set of agents.
- $U = \times_{i \in \mathcal{A}} A_i$ is a joint action set, where A_i denotes actions of agent i.
- $\delta : \mathcal{S} \times U \times \mathcal{S} \to [0, 1]$ is a probabilistic state transition function, where $\delta(s, u, s')$ denotes the probability of transitioning from state s to state s' when joint action $u \in U$ is executed.
- $\rho : \mathcal{S} \times U \times \mathcal{S} \to \mathbb{R}$ is a reward function, where $\rho(s, u, s')$ denotes the reward for executing joint action u in state s and transitioning to state s'.
- $\mathbf{\Omega} = \times_{i \in \mathcal{A}} \Omega_i$ is a finite set of observations, where Ω_i denotes the set of observations agent i can make.
- O is a table of observation probabilities. $O(s, u, s', \boldsymbol{o})$ is the probability of observing joint-observation $\boldsymbol{o} = \langle o_1, \ldots, o_n \rangle \in \mathbf{\Omega}$ when executing joint action u in state s and transitioning to state s'. Each agent i only perceives the element o_i of the joint-observation $\boldsymbol{o}$.

An interesting result for the complexity of solving Dec-POMDPs, i.e. finding joint policies that maximize the expected cumulative return over time, is presented in [BGIZ02]. Namely, for already two agents Dec-POMDPs with a finite horizon are NEXP-complete. Algorithms for Dec-POMDPs can be found in the literature, e.g. in [BAHZ09] or [YT07].

3.3.4 Discussion

Despite lots of progress in (multiagent) reinforcement learning research in recent years, several yet unsolved open issues and challenges exist. Some of them are inherent to RL in general while others emerge in MARL only. According to Panait and Luke [PL05], Buşoniu et al. [BBDS08] [BBDS10], and 't Hoen et al. [HTP+06] the list includes the following:

Scalability: The problem of scalability can be found in many multiagent learning algorithms and is not specific for MARL. The basic problem is that search spaces grow too quickly. This follows from the increased number of agents since each

agent adds its own actions to the joint strategy space that has to be explored. Also, each agent, resp. its "state", becomes part of the environment which leads to larger state spaces, too. Since many algorithms focus on finite state and action sets and use tabular representations for Q-values, large (or continuous) state-action spaces quickly become hard to handle. This holds in particular, because most Q-Learning based algorithms demand infinitely many visits of each state-action pair. To produce relief, approximative techniques can be used (see e.g. [BEDSB11] for a survey on approximate reinforcement learning). Another approach is to exploit problem structures, e.g. by decomposition of larger problems [GLP02], or by masking states as we will propose later in this work.

Exploration-exploitation tradeoff: Like in single agent reinforcement learning, the question of exploring new knowledge or exploiting learned knowledge also exists in MARL. One approach towards finding a balance in this tradeoff can be found in Bayesian MARL. There, the expected gain of exploration is compared to the costs by reasoning about other agents [CB03].

Coordination: Coordination in cooperative MAS can be defined as "the ability of two or more agents to jointly reach a consensus over which actions to perform in an environment" [KK02]. As we have seen, there are algorithms like Distributed Q-Learning which can reach this goal under some assumptions in restricted settings without communication. However, in general or in possibly partially observable domains, mutual agreement can be hard to establish because an agent's action choice might also depend on the actions of other agents.

Dynamics of environment: Because multiple agents learn and act simultaneously in the same environment, the learning environment becomes non-stationary. Thus, the agents need to be able to adapt to a constantly changing goal. This is also known as *moving target problem*.

Credit assignment: The credit assignment problem known from single agent RL, can also be found in MARL. However, in MAS, the question goes even further as it sometimes might be unclear which agent was responsible for reaching a particularly high or low reward. This question is hard to answer in general and many different approaches for giving credit have been investigated (see e.g. [PL05, Sect. 3.1] for a detailed overview).

Emergent behavior: Given complex problems and agents with many actions, the outcome of certain joint actions could become unpredictable and potentially lead to emergent behavior. Hence, the influence of such emergent behaviors to learning becomes interesting and should be investigated.

Rationality: As we have discussed in the context of algorithms that search for equilibrium solutions, e.g. Nash equilibria, the best "team" solution does not necessarily correlate to such equilibria. Hence, rationality in cooperative systems might be considered as being less important than achieving a good result for the entire agent society.

Specification of MARL goal: Also, the specification of a good MARL goal is not easy. As we stated earlier, the two most important aspects include the stability of the learning dynamics, e.g. by converging to stationary policies, and the ability to maintain performance by adaptation to changing behavior of other agents. As pointed out in [BBDS10], several different properties can be considered in order to fulfill these aspects.

Convergence: Besides defining proper learning goals in MARL, convergence to exact optimal solutions often is hardly possible, e.g. due to scalability issues. Hence, several MARL approaches try to converge to near optimal solutions.

Partial observability: In partially observable settings the agents are unable to observe the full state and/or the actions of all other agents. Since the approaches presented above rely on either of them, they cannot be applied and different techniques and models are required. While solving POMDPs in single agent settings is PSPACE-complete [PT87], the problem becomes NEXP-complete for Dec-POMDPs, even in the two agent case [BGIZ02].

Heterogeneity: Many algorithms are evaluated in self-play and consider only homogeneous agent sets. In more realistic settings, however, these assumptions might not hold, e.g. due to specialized agents (robots with different abilities), or because different manufactures are involved. These kinds of heterogeneity do influence the learning behavior and hence should be investigated. One work that deals with different learning algorithms is [KK04].

Design of reward functions: Since most research deals with small or well-known problems, the above cited works do not consider the aspect of designing reward functions. Given a new problem, the design of proper reward functions is challenging and should be accomplished with care. Otherwise, agents might learn a (near) optimal strategy, but the observed behavior does not match with expectations. In particular, this becomes an issue if techniques like reward shaping (see Sect. 3.4) are used (improperly).

As we can see from this list, there is still room for improvement in the research area of multiagent reinforcement learning. This is also underlined by Shoham et al. [SPG07], who pointed out that "Learning in MAS is conceptually, not only technically, challenging".

3.4 Reward Shaping

Reward shaping is a technique which is intended to improve the learning performance in single and multiagent reinforcement learning [NHR99] [DK11]. The basic idea is to guide the learner by providing an additional shaped reward that comes in combination with the true reward function such that a good—ideally optimal—policy is learned faster.

Formally, the idea of reward shaping can be described as follows [NHR99]: Assume a single agent MDP $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$ that should be solved. Instead of dealing with M the agent is confronted with a different MDP $\mathrm{M}' = \langle \mathcal{S}, A, \delta, \rho', \gamma \rangle$ that corresponds to M but uses a new reward function ρ' such that $\rho'(s, a, s') = \rho(s, a, s') + F(s, a, s')$. Here, $F : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$ denotes the additional *shaped reward function*. This shaped reward should guide the agent in M' towards states with higher rewards more quickly. The learned behavior for M' finally should correspond to the desired (optimal) behavior for the original MDP M.

Although the influence of shaped rewards to learning has been unclear for some time, reward shaping was successfully applied in different domains. However, there also have been problems. One remarkable example mentioned in [NHR99] is the work of Randløv and Alstrøm [RA98], in which an agent was supposed to learn to ride a bicycle from a starting position to a goal position. The authors state that in an early stage of their research, they used a reward function that rewards the agent for approaching the goal but does not include any punishment when the agent drove away from the goal. In this setting, the agent learned to drive in circles of 20-50 meters

around the starting position instead of driving towards the goal. Besides the reward that can be gathered by this behavior, the agent was also performing good in terms of learning to ride a bike, because circles build a physically stable formation. Hence, the agent performed well with respect to the provided (shaped) reward function but bad with respect to the original problem setting.

The question of how reward shaping influences learning an optimal policy for the original MDP was investigated by Ng et al. [NHR99]. In particular, they identified that if there are state sequences $s_1 \rightarrow s_2 \rightarrow \ldots s_n \rightarrow s_1 \rightarrow \ldots$ that can occur repeatedly and which result in a positive overall shaped reward, i.e. $F(s_1, a_1, s_2) + \ldots + F(s_{n-1}, a_{n-1}, s_n) + F(s_n, a_n, s_1) > 0$, then the agent might refrain from learning the optimal policy for the original MDP and instead learns to run into this circle. In order to avoid such settings, Ng et al. proposed to use *potential-based* reward shaping functions and proved the following theorem:

Theorem 3 ([NHR99]). *Let any $\mathcal{S}, A, \gamma$, and any shaping reward function $F : \mathcal{S} \times A \times \mathcal{S} \rightarrow \mathbb{R}$ be given. We say F is a* potential-based *shaping function if there exists a real-valued function $\Phi : \mathcal{S} \rightarrow \mathbb{R}$ such that for all $s \in \mathcal{S} \setminus \{s_0\}$[2], $a \in A$, $s' \in \mathcal{S}$,*

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s), \tag{3.17}$$

(where $\mathcal{S} \setminus \{s_0\} = \mathcal{S}$ if $\gamma < 1$). Then, that F is a potential-based shaping function is a necessary and sufficient condition for it to guarantee consistency with the optimal policy (when learning from $\mathrm{M}' = \langle \mathcal{S}, A, \delta, \rho + F, \gamma \rangle$ rather than from $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$), in the following sense:

- *(Sufficiency) If F is a potential-based shaping function, then every optimal policy in M' will also be an optimal policy in M (and vice versa).*
- *(Necessity) If F is not a potential-based shaping function (e.g. no such Φ exists satisfying (3.17)), then there exist (proper) transition functions δ and a reward function $\rho : \mathcal{S} \times A \rightarrow \mathbb{R}$, such that no optimal policy in M' is optimal in M.*

Accordingly, if a potential-based shaping function is used, then the agent will be able to learn the optimal policy for MDP M from the "modified" MDP M' with shaped rewards. Some additional experiments conducted by the authors of [NHR99] also support this theoretical result and showed improved convergence speed with reward shaping compared to settings without this technique.

Which shaping function is used in a particular context usually depends on the considered problem and can be a problem on its own, particularly in large and complex domains. Mainly, heuristics that incorporate some domain knowledge are used. A more recent approach applies planning techniques to automatically generate potential-based shaping functions [GK08].

Another interesting result in this context was presented by Wiewiora [Wie03]. He proved that potential-based reward shaping is equivalent to initializing the Q-table with the potential function Φ, e.g. $Q_0(s, a) = \Phi(s)$.

Reward shaping was also applied in multiagent settings (e.g. in [BdCL08]) although no formal proofs concerning the invariance of the learned policy with respect to shaped rewards were known. Recently, Devlin and Kudenko [DK11] closed this gap. They proved that the results of Wiewiora [Wie03] concerning the equivalence of Q-table initialization and using potential-based reward shaping, can be extended

[2] s_0 denotes the starting start of the MDP.

to the multiagent case, too. The authors also proved that the set of Nash equilibria in a stochastic game remains stable if (some or all) agents use potential-based reward shaping. However, potential-based reward shaping influences the behavior of the agents with respect to exploration and, thus, convergence to different (possible bad) Nash equilibria may occur as demonstrated empirically in [DK11].

To conclude, it should be stated that reward shaping is a powerful technique to improve convergence speed in single and multiagent RL on the one hand, but should be used with care to avoid misguided coordination on the other hand.

Later in this work, we will consider learning under noised rewards. Comparable to reward shaping, our noisy rewards are also a sum of the original reward and an additional function. Hence, the result of Ng et al. will become useful in our setting, too. Contrary to reward shaping, which tries to guide a learning agent, noised rewards distract agents from learning correct behaviors.

3.5 Conclusion

In this section, we provided the necessary background for the learning parts of this thesis. We established the notations and terms used throughout this work. In addition, we presented both, a broad overview on the (multiagent) reinforcement learning research area, its algorithms, and challenges, as well as detailed descriptions of the most important algorithms and models used later on.

Finally, it should not remain unmentioned that reinforcement learning (RL) has been applied successfully in a wide variety of real world problems. A well-known example is that of Tesauro's world-class backgammon program that is on par with the best human players in the world [Tes95]. Another example is presented in [KS04], where the well-known robot dog Aibo uses RL to learn a gait to move faster than with any previously known hand-coded gait. Ng et al. [NCD+04] have used RL to control a small usually radio-controlled helicopter. In particular, the helicopter has learned a policy that allows it to perform an autonomous inverted hovering flight maneuver. In the area of operations research, Abe et al. [AVAS04] applied reinforcement learning in a customer relationship management tool in order to optimize mail campaigns, which resulted in increased profits.

Clearly, this list of successful reinforcement learning applications is not exhaustive. However, it provides reasonable evidence that RL is a particularly useful technique that can be applied to different problem areas due its low requirements, e.g. concerning learning feedback and the use of very general models.

Part II

Foundations

Chapter 4

Sequential Stage Games and Engineered Rewards

This chapter presents two own contributions. The first result, described in Sect. 4.1, is the definition of a special type of games for learning in multiagent settings. We motivate this class and show that it constitutes an interesting subclass of stochastic games which we will pursue in the remainder of this thesis.

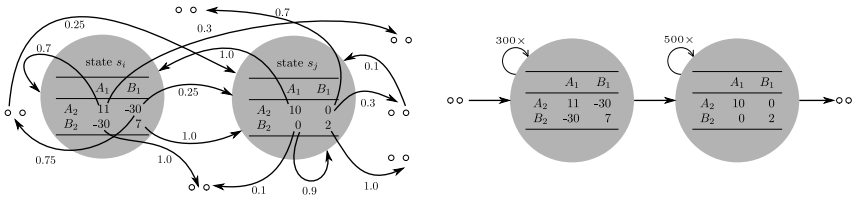
The second contribution, presented in Sect. 4.2, is a conceptual model for reinforcement learning that we will use to develop an approach for learning in the aforementioned novel game class. In this concept, only an engineered reward signal is provided to a learning agent instead of a state *and* a reward signal. We prove that it is equivalent to standard models with full observability in terms of expressiveness and computational effort. However, we also show that the concept, compared to standard models, adds more expressiveness in partially observable domains.

Both sections describe foundations for the main learning algorithms that we will propose later in this work. The contents of this chapter have already (partly) been published in [KKB11c], [KKB11a], and [KKB12].

4.1 Sequential Stage Games

In this section, we will introduce a novel subclass of stochastic games called *sequential stage games (*SSG*)*. As the name already indicates, a sequential stage game consists of a set of stage games that are played in a row one after the other. Stochastic games, in contrast, consist of static games for each state and allow arbitrary probabilistic transitions between these games, resp. states. Hence, the assumption on the transition function in SSGs mainly distinguishes our class from stochastic games. Figure 4.1 visualizes this difference.

Next, we motivate sequential stage games in Sect. 4.1.1 and provide a formal definition in Sect. 4.1.2. Then, we relate the novel class to existing game classes in Sect. 4.1.3. In Sect. 4.1.4, we conclude this section with a short discussion.



(a) Stochastic game with arbitrary probabilistic state transitions.

(b) Sequential stage game with clear transition between the stateless games.

Fig. 4.1 Structural difference between stochastic games and sequential stage games.

4.1.1 Motivation

Although there are many algorithms for learning in general stochastic games (see surveys like [HTP$^+$06] or [BBDS10]), convergence towards optimal joint policies is still a challenge under general settings. Given that a subclass of stochastic games is considered, e.g. cooperative stochastic games, one can construct algorithms that try to exploit the special structures of the considered class.

In this work, we are interested in such a special game class. As shown in Fig. 4.1, our class makes an assumption on state transitions. In detail, we assume that agents play one particular stateless game for a certain number of iterations which is unknown to the agents. When a game is "over", the agents are confronted with a new game, which again is played repeatedly for some given timespan.

From our point of view, this special structure is highly interesting since it can be found in various problem domains. For instance, in economic problem settings where the market follows some defined rules for a given amount of time. The players have to adjust to this setting, i.e. they have to learn how to best deal with the current situation. From time to time, market rules change, e.g. if new players enter a market or if new laws are established. In that case, the agents face a new situation to which they have to adapt again.

Similar structures can be found in other dynamic systems like, e.g., distributed control problems in production systems. During a work shift, a production plant might only produce a single product A. Hence, the machines have to be adjusted to this setting. Later—planed or unexpected—another product B has to be manufactured and the machines have to be adjusted again. Such changes might be unpredictable, e.g. due to changed customer demands or urgent orders. After a change, however, the system will remain in a stable state for some unknown timespan before the next change occurs.

According to these two (simplified) examples, it becomes clear that proper models and well-tailored general methods for such sequential problem structures are an interesting area in both, research and practice. Hence, the remainder of this section will define a special class of games called *sequential stage games* that formalizes the corresponding properties for simplified stateless games.

4.1.2 Definition

As mentioned above, the core idea of sequential stage games is that agents consecutively play a sequence of stage games. In particular, we assume that the set of agents as well as the set of actions available to each agent in every stage game is equal. This leads to a first definition of the set of *common static games*:

Definition 11 (Common Static Games [KKB12]). Let $\mathrm{CSG}(\mathcal{A}, U)$ denote the set of static games that have the same agents $\mathcal{A}$ and the same joint action set U. We refer to this class as *common static games*.

With the help of this set, we can now state the definition of sequential stage games:

Definition 12 (Sequential Stage Game [KKB12]). A *sequential stage game* (SSG) is a game defined by $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$, where

- $\mathcal{A}$ is the set of agents
- $U = \times_{i \in \mathcal{A}} A_i$ is the set of joint actions, where A_i denotes the actions available to agent $i \in \mathcal{A}$
- $\mathcal{G} = \{\langle G_0, n_0 \rangle, \langle G_1, n_1 \rangle, \ldots, \langle G_m, n_m \rangle\}$ is a set of $m+1$ pairs $\langle G_j, n_j \rangle$, and $G_j = \langle \mathcal{A}, U, \{\rho_i^j\}_{i \in \mathcal{A}} \rangle \in \mathrm{CSG}(\mathcal{A}, U)$ is a common static game that is played repeatedly for $n_j \geq 1$ times
- $\langle G_0, n_0 \rangle \in \mathcal{G}$ describes the initial common static game G_0 that is played $n_0 \geq 1$ times
- $g : \mathcal{G} \to \mathcal{G}$ is the game transition function that transitions from game G_j to game G_{j+1} after G_j was played n_j times

Note that the common static games contained in the tuples of the game set $\mathcal{G}$ usually are played repeatedly and thus can be referred to as stage games, too. Since a stage game can be either cooperative, competitive, or a mixture of both, the same also applies for sequential stage games. To be clear about terminology, a SSG is said to be cooperative, resp. competitive, if all contained stage games are cooperative, resp. competitive. It is a mixture of both, if it either contains more than one type of stage games or if all stage games are a mixture. Since we will concentrate on cooperative SSGs in this work, we introduce the abbreviation CSSG to refer to such cooperative games.

Given this definition, we now describe what is considered to be an optimal joint strategy in a cooperative SSG. In such a setting, a sequence of different cooperative common stage games is played. It is easy to define an optimal joint strategy for each of these stage games: An optimal joint strategy for a stage game G_j leads to the highest reward. Let us for a moment pretend that each stage game of a CSSG describes a state. Under this assumption, an optimal joint *policy* for a CSSG can be defined by using the optimal joint strategies for each state's stage games. In terms of the cumulative discounted reward, this definition describes the highest value achievable for playing the cooperative SSG.

However, there is no formal state in a SSG. Thus, agents at any point in time only have one *strategy* that they follow. This strategy has to be adjusted dynamically over time to reflect the changing demands of the sequence of stage games. Accordingly, it is hard to describe an optimal joint strategy formally, at least without adding an artificial state (as above) or another form of time dependency. As a consequence, when we will talk about an optimal joint strategy for a cooperative sequential stage

game, we mean the (artificial) optimal joint policy described above. If we consider the behavior of a learning approach in a particular stage game G_j, then we will directly refer to the optimal joint strategy for G_j.

4.1.3 Relationship to other Models

In this section, we relate sequential stage games to other existing models. The first result provided in Prop. 3 has been published in [KKB12]. It shows the relation of SSGs to stochastic games:

Proposition 3 ([KKB12]). *Sequential stage games are a proper subclass of stochastic games, i.e.* $\mathrm{SSG} \subset \mathrm{SG}$.

Proof. In order to prove this proposition, consider the following construction. It allows us to reformulate any sequential stage game as a stochastic game. Let $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$ be an arbitrary sequential stage game. Then a corresponding stochastic game $\Gamma' = \langle s_0, \mathcal{S}, \mathcal{A}', U', \delta, \{\rho'_i\}_{i \in \mathcal{A}'} \rangle$ is constructed by:

- $\mathcal{A}' = \mathcal{A}$ and $U' = U$.
- Recall the definition of the set of games $\mathcal{G}$. Then let $\mathcal{S}$ be a set of $2 + \sum_{j=0}^{m} n_j$ states, i.e. $\mathcal{S} = \left\{ s_\emptyset, s_0^1, \ldots, s_0^{n_0}, s_1^1, \ldots, s_1^{n_1}, \ldots, s_m^1, \ldots, s_m^{n_m}, s_\infty \right\}$. Here, s_j^v denotes the state that is obtained when game G_j was played for the v-th iteration.
- The initial state s_0 corresponds to state $s_\emptyset$, which is the state before the first game is played. In addition, s_∞ defines an absorbing state that will be entered when the game sequence is completed.
- The state transition function δ for any joint action $u \in U$ is constructed such that it stays in stage game G_j until it is played n_j times and then transitions to the next game G_{j+1}. Therefore, the transition function has to ensure to iterate over all states in the imposed ordering as indicated in the definition of $\mathcal{S}$ above. The state transition function δ thus realizes the game transition function g of the SSG. Formally, δ is given by:
  - Start:
    $\delta(s_\emptyset, u, s_0^1) = 1 \wedge \delta(s_\emptyset, u, s) = 0, s \neq s_0^1, \forall u \in U, s_\emptyset, s_0^1, s \in \mathcal{S}$
  - Play G_j for n_j times:
    $\delta(s_j^v, u, s_j^{v+1}) = 1 \wedge \delta(s_j^v, u, s) = 0, s \neq s_j^{v+1}, \forall u \in U, s, s_j^{v+1} \in \mathcal{S}, s_j^v \in \mathcal{S} \setminus \{s_j^{n_j}\}$
  - Transition from G_j to G_{j+1}:
    $\delta(s_j^{n_j}, u, s_{j+1}^1) = 1 \wedge \delta(s_j^{n_j}, u, s) = 0, s \neq s_{j+1}^1, \forall u \in U, s_j^{n_j}, s, s_{j+1}^1 \in \mathcal{S}$
  - Enter absorbing state s_∞ if all games are played:
    $\delta(s_m^{n_m}, u, s_\infty) = 1 \wedge \delta(s_m^{n_m}, u, s) = 0, s \neq s_\infty, \forall u \in U, s_m^{n_m}, s, s_\infty \in \mathcal{S}$
  - Stay in absorbing state s_∞:
    $\delta(s_\infty, u, s_\infty) = 1 \wedge \delta(s_\infty, u, s) = 0, s \neq s_\infty, \forall u \in U, s, s_\infty \in \mathcal{S}$
- The reward function ρ'_i of an agent i is given by $\rho'_i : S \times U \to \mathbb{R}$, where the rewards for all joint-actions $u \in U$ in state $s_v^w \in \mathcal{S}$ correspond to the reward obtained when playing u in stage game G_v. In addition let $\rho'_i(s_\emptyset, u) = \rho'_i(s_\infty, u) = 0, \forall u \in U$[1].

[1] Contrary to this work, we wrongly defined the agents' reward functions for the start and the absorbing state in [KKB12] to return the rewards of the first, resp. last game in the sequence.

By this construction, we have shown that each sequential stage game can be transformed into a stochastic game. To prove that sequential stage games are a proper subclass of stochastic games consider two distinct stochastic games Γ_A and Γ_B with disjoint state sets. Then let Γ_C be a SG that combines both games Γ_A and Γ_B such that no connection from Γ_A to Γ_B exists in Γ_C. Now choose the initial state s_0 of Γ_C from say Γ_A's states. Accordingly, not all stage games in Γ_C can be played. Since by definition of SSGs, each stage game must be played at least once, not all stochastic games can be translated into a sequential stage game. Together with the construction above, Prop. 3 follows. □

This result is not surprising, since SSGs by definition are composed of the same kind of games, namely static (or stage) games, as ordinary stochastic games. For cooperative sequential stage games, we can show the following:

Proposition 4. *Cooperative sequential stage games are a proper subclass of decentralized partially observable Markov Decision Processes, i.e.* CSSG $\subset$ *Dec-POMDP.*

Proof. Let Γ be an arbitrary cooperative sequential stage game. Then, we can easily reformulate Γ as Dec-POMDP $\Delta = \langle \mathcal{S}, U, \delta, \rho, \mathbf{\Omega}, O \rangle$ by using:

- The same state set $\mathcal{S}$ and the same transition function δ as in the proof for Prop. 3.
- The same set of joint actions U.
- A reward function $\rho : \mathcal{S} \times U \times \mathcal{S} \to \mathbb{R}$ that returns for $\rho(s, u, s')$ the reward which corresponds to the reward obtained when executing joint action u in game G_j of the CSSG corresponding to state s. Note that this step is only valid because all agents share the same reward function in a CSSG.
- An empty set of observations $\mathbf{\Omega} = \emptyset$ and, thus, an observation function O that maps to an empty observation for all inputs, as agents in the original CSSG can neither observe each other nor the environment.

From an analog construction as for stochastic games, it immediately follows that not every Dec-POMDP can be formulated as CSSG, which finally proves the proposition. □

According to Hansen et al. [HBZ04], cooperative partially observable stochastic games (coop. POSG) are known as Dec-POMDPs, which have been defined by Bernstein et al. [BGIZ02].

In the end, Fig. 4.2 illustrates the relations of (cooperative) SSGs to other models.

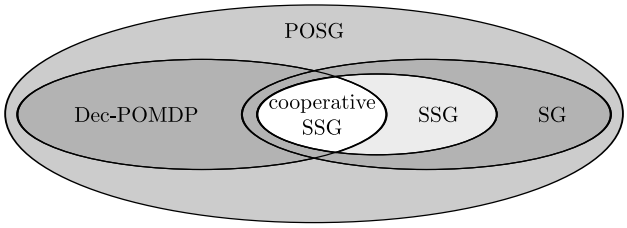


Fig. 4.2 The relationships of (cooperative) sequential stage games.

4.1.4 Discussion

Given the model of sequential stage games, we now can develop multiagent reinforcement learning algorithms that can benefit from the aforementioned structural properties of this class. The relation of SSGs with respect to existing well-known game classes is helpful in order to identify algorithms that might establish a basement for novel approaches.

4.2 Engineered Reward-Based Models

As we have seen in Sect. 3.2, most models used for RL assume an agent to be able to perceive at least two signals via its sensors: the state and the reward signal. In this section, we present a different approach that combines both signals into an engineered artificial reward. By doing so, an agent in our model does not need to observe two, but only one signal. We analyze the properties of this concept and show that it is equivalent to ordinary models in terms of expressiveness and computational effort. We also point out that it has to be used carefully in order to avoid undesired knowledge transfer from the central environment to the agent(s). The results presented in the following subsections have already been published in [KKB11c].

4.2.1 Concept

The idea of the engineered reward-based model is to encode original state and reward information of an underlying MDP into a single numerical value—the *engineered reward*. This value is the only signal observed by an agent. Thus, the environment does not need to offer directly visible state information, or to put it the other way around, the agent can be "blind" and only needs to be able to observe the reward signal. When the agent receives the engineered reward signal, it can decode the signal and extract both, original reward and state information. Clearly, the time required by the de- and encoding functions has to be polynomial.

Figure 4.3 visualizes the concept of engineered reward-based Markov Decision Processes. In addition, it shows important components of a learning agent which usually depend on the availability of a state and a reward signal. Given appropriate de- and encoding functions, the approach maintains state and reward information and, thus, all algorithms for ordinary MDPs can be applied in this model, too.

Next, we quote a slightly reformulated definition for engineered reward-based Markov Decision Processes from [KKB11c]:

Definition 13 (Engineered Reward-Based MDP). Consider the Markov Decision Process $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$. An *engineered reward-based Markov Decision Process (ERBMDP)* is defined by a tuple $E = \langle \mathrm{M}, \textsc{enc}, \textsc{dec} \rangle$. Here, $\textsc{enc} : \mathbb{R} \times \mathcal{S} \to \mathbb{R}$ is a polynomial time function that calculates an engineered reward which encodes original rewards and states into a single real value. Agents can only observe the engineered reward, and are able to decode these rewards using a polynomial time decoding function $\textsc{dec} : \mathbb{R} \to \mathbb{R} \times \mathcal{S}$. It has to hold that $\textsc{dec}(\textsc{enc}(\rho(s,a),s)) = (\rho(s,a),s)$.

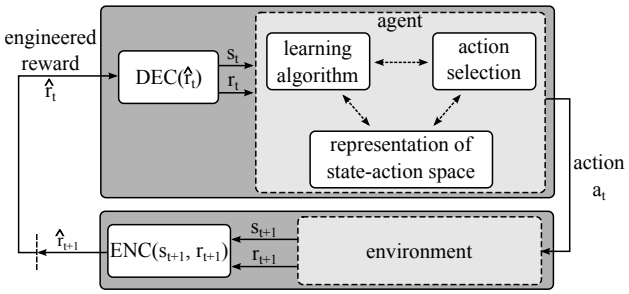


Fig. 4.3 Engineered reward-based model ([KKB11c]).

In order to realize an engineered reward-based Markov Decision Process, one needs to specify a proper de- and encoding function. In [KKB11c], we presented such a simple polynomial time approach that basically combines reward and state information into a single string. Therefore, it uses a specific marker and prepends the state information and appends the reward value to it. The resulting string is converted to a real number, e.g. using the ASCII encoding. This number can be used as engineered reward. The decoding function simply reverses this process. We skip further details on these two functions here and refer the reader to [KKB11c] for a detailed description and pseudocode, as that general coding approach should only be a proof-of-concept for the realization of an ERBMDP.

Having proposed a simple coding approach, the next question deals with the calculation of the engineered reward. Therefore note that there are basically two different approaches. The approach that is usually followed, e.g. in robotic applications, lets a robot itself calculate a reward based on its observations. For instance, in a simple navigation problem, it may punish itself whenever it collides with an obstacle and rewards itself whenever it reached an identifiable goal position. The second approach is to consider everything outside of the agent to be part of the environment. Thus, the reward has to be calculated within the environment. Therefore, one can either think of the environment as an external teacher that gives rewards as stated in [Mit97], or as a central computational instance that has full information. From this point of view, it becomes clear that the engineered reward could either be calculated "online", i.e. while the agent interacts with the (computational) environment, or "offline" during the design of a system. Which approach should be used depends on different factors like the complexity of the problem or the storage required to pre-calculate all possible engineered rewards.

Later, when we will consider the proposed learning approach, we will investigate both, a centrally calculated and engineered reward that ensures convergence to (near) optimal solutions as well as a locally (i.e. on agent-level) calculated and transformed reward.

4.2.2 Equivalence Results

In this section, we show that the concept of using engineered rewards has the same expressiveness and complexity as state-of-the-art Markov Decision Processes. Therefore, we follow the approach of Seuken and Zilberstein who proved several equiv-

alence results for models used in MARL [SZ08]. Based on their work, we define model equivalence as follows:

Definition 14 ([KKB11c]). Two models M_1 and M_2 are said to be *polynomial time equivalent* if and only if $M_1 \leq_p M_2$ and $M_2 \leq_p M_1$.

In order to prove the equivalence of MDPs and engineered reward-based Markov Decision Processes, we need to recall that reinforcement learning basically is an optimization problem seeking for an optimal policy that maximizes the cumulative discounted reward. Tools for analyzing complexity, however, usually deal with decision problems (cf. [GJ79] or [CLRS01]). Thus, we have to construct decision variants of our optimization problems. This can be done by adding a constant B to the problem and by asking whether the optimized value exceeds bound B. Based on this, we next quote our equivalence result from [KKB11c], which is slightly reformulated to match the notations of this work:

Theorem 4 ([KKB11c]). *Engineered reward-based Markov Decision Processes (ERBMDP) and Markov Decision Processes (MDP) are polynomial time equivalent.*

Proof. We have to show that the decision problem variants are mutually reducible in polynomial time, i.e. ERBMDP $\leq_p$ MDP and MDP $\leq_p$ ERBMDP. To construct a decision problem, we add the bounds B and B' to the respective tuples.

1. ERBMDP $\leq_p$ MDP: An arbitrary ERBMDP $E = \langle\langle \mathcal{S}, A, \delta, \rho, \gamma\rangle, \text{ENC}, \text{DEC}, B\rangle$ can be transformed into an MDP $M' = \langle \mathcal{S}', A', \delta', \rho', \gamma', B'\rangle$ in polynomial time by setting $\mathcal{S}' = \mathcal{S}, A' = A, \delta' = \delta, \rho' = \rho, \gamma' = \gamma$, and $B' = B$. The agents also have to be allowed to observe the state signal. Since the encoding and decoding function, ENC and DEC, are only required to construct the engineered reward and because $\text{DEC}(\text{ENC}(\rho(s,a),s)) = (\rho(s,a),s)$ holds by definition, these functions have no influence on the expected return. Thus, whenever a policy in E results in a return of at least B, this also holds for the constructed MDP M'.
2. MDP $\leq_p$ ERBMDP: Let $M = \langle \mathcal{S}, A, \delta, \rho, \gamma, B\rangle$ be an arbitrary MDP. Then, an ERBMDP $E = \langle\langle \mathcal{S}', A', \delta', \rho', \gamma'\rangle, \text{ENC}, \text{DEC}, B'\rangle$ can be obtained by forbidding the agent to observe the state signal and by setting $\mathcal{S}' = \mathcal{S}, A' = A, \delta' = \delta, \rho' = \rho, \gamma' = \gamma$, and $B' = B$. It remains to show that there are polynomial time algorithms for encoding and decoding such that $\text{DEC}(\text{ENC}(\rho(s,a),s)) = (\rho(s,a),s)$ holds—this was discussed in Sect. 4.2.1. Finally, as before, a policy for M that results in a return of at least B will result in the same return in E.

□

In [KKB11c], we argued that this equivalence result can easily be extended to multiagent settings with full observability. Here, we formally state this result:

Theorem 5. *Engineered reward-based stochastic games (ERBSG) and stochastic games (SG) are equivalent.*

Proof. Again, we have to show that both decision problem variants are mutually reducible in polynomial time, i.e. ERBSG $\leq_p$ SG and SG $\leq_p$ ERBSG. Therefore, note that an ERBSG basically is a SG with additional de- and encoding functions, which do not influence the solution. Since state transitions and rewards in stochastic games only depend on the state and the combined actions of all agents, the aforementioned general coding approach can also be used to deal with engineered rewards in stochastic games in polynomial time. The theorem then trivially follows by the same arguments as in the proof for Theorem 4. □

4.2.3 Partial Observability

Having shown the equivalence of engineered reward-based models and their standard counterparts with full observability, we now concentrate on settings with partial observability. In Sect. 3.2.1, we gave a definition for partially observable MDPs (POMDP) and visualized it in Fig. 3.2(b). Like in MDPs, an agent in a POMDP also observes a reward signal. Contrary to MDPs, the agent however does not perceive full state information but only makes (partial) observations. The reward is calculated through the environment which possesses full state information. Thus, using the coding approach from above, the environment can construct engineered rewards that may contain state information which an agent is not able to obtain via its observations. Accordingly, engineered reward-based POMDPs and ordinary POMDPs are not equivalent and the former establishes a richer concept.

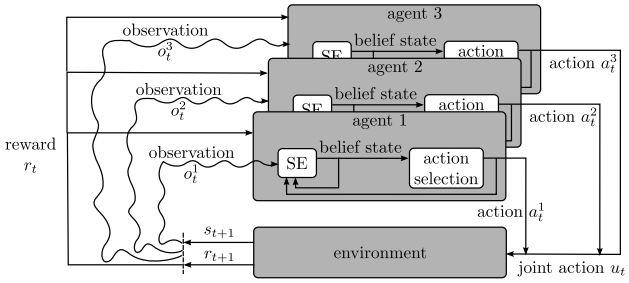


Fig. 4.4 Decentralized POMDP with three agents.

In the context of multiagent systems, a decentralized POMDP (Dec-POMDP) as illustrated in Fig. 4.4 (cf. also Def. 10) is one model that realizes partial observability in cooperative games. In particular, each agent might make a different observation in the same state, but all agents observe the same global reward signal. Since this reward is calculated in the environment, which therefore formally has to know the joint action and full state information, the concept of engineered rewards can be used to exploit the system. For instance, one might include information on a joint action as part of the state and encode it together with full state knowledge in the engineered reward. With the help of such a rich reward, the partial observability assumption is basically circumvented and the Dec-POMDP is reduced to a cooperative stochastic game with full observability.

In order to avoid such undesired exploitation, we finally argue that the following property should be mandatory for rewards in ordinary models with partial observability:

Property 1 ([KKB11c]). *Besides the true reward, reward signals in partially observable domains are not allowed to carry additional polynomial time decodable information.*

4.2.4 Discussion

The equivalence of engineered reward-based models and their standard counterparts with full observability is not surprising as no additional information can be supplied to the agents by using an engineered reward.

In partially observable domains, however, the concept has to be applied with care. Let us now, based on [KKB11c], further investigate this issue. In order to discuss the risks and benefits of engineered rewards in such settings, let us consider the Iterative Agent Partitioning Problem (IAPP, see Ch. 5) modeled as decentralized POMDP. In detail, the corresponding Dec-POMDP is given by $D = \langle \mathcal{S}, \mathcal{A}, U, \delta, \rho, \mathbf{\Omega}, O \rangle$, where the elements are defined as follows:

- A state $s \in \mathcal{S}$ describes the positions of all targets $T \in \mathcal{T}$ and agents $i \in \mathcal{A}$ as well as their current assignments.
- Each agent $i \in \mathcal{A}$ has one action $a \in A_i$ for selecting each target; the joint action set is given by $U = \times_{i \in \mathcal{A}} A_i$.
- The state transition function δ is deterministic. It depends on the previous state, the movement of agents, and on their joint action which, together, determine the next state.
- The reward function ρ returns the partitioning quality according to (5.2).
- An agent's observation is determined by its local perception of its environment. It contains its neighbors and at least its currently selected target. The combination of all agent observation sets then defines the joint observation set $\mathbf{\Omega}$.
- The observation function O is designed such that each agent correctly observes its local environment.

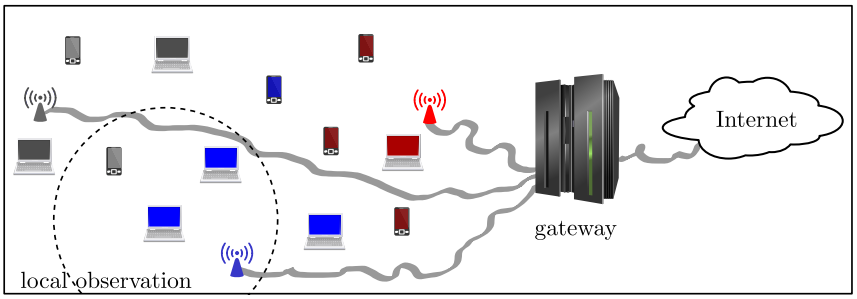


Fig. 4.5 The Iterative Agent Partitioning Problem as Dec-POMDP.

Figure 4.5 visualizes an exemplary IAPP setting with access points that are connected to a gateway. In this scenario, access points (targets) may gather information about positions of mobile devices (agents) and current assignments. These information then are processed by a (powerful) gateway machine to calculate the current partitioning quality according to (5.2), i.e. to calculate the reward. The resulting value then is distributed to the agents as learning feedback via the access points.

Now, let us consider how the gateway can help to circumvent the partial observability of the agents. Therefore note that the gateway has *full* information about the entire system state, in particular about all agent and target positions. Based on that, it can construct an engineered reward which encodes all these information. Hence, each agent can also obtain full information by decoding the engineered reward. With the

help of these information, it then can construct its own projection of the cooperative multiagent problem. By considering joint actions as elementary actions, knowing positions of all elements and evaluation function (5.2), it can simulate the cooperative problem as MDP which can be solved using single agent learning techniques (e.g. policy iteration or Q-Learning). Given that each agent calculates the same optimal (joint) strategy, the cooperative problem is solved, too. This is a major departure from the original Dec-POMDP which is known to be NEXP-hard even for two agents [SZ08].

Clearly, such an approach demands high computational power and storage space for each agent. However, this example impressively shows that engineered rewards should be used carefully and thus strongly supports our proposed extension concerning the properties of reward values in standard models (cf. Prop. 1).

Besides these risks, the concept also offers potential for the development of efficient algorithms for learning in cooperative SSGs. A corresponding learning algorithm will be presented in Ch. 8. Briefly, the advantages from using engineered rewards in that algorithm are:

- Without encoding full state information, engineered rewards indirectly communicate global state changes, which enable the agents to continuously adjust their current strategies to new settings.
- Since it is sufficient to know that a state transition at a global level has occurred, there is no need to store any state information, and thus also no values for each state-action pair have to be stored, which leads to a space efficient approach.

In the end, the proposed concept thus is a good means for our problem domain, but it should be used carefully.

Chapter 5

The Iterative Agent Partitioning Problem

With the definition of sequential stage games in the previous chapter, we introduced an interesting class for learning in multiagent systems. This thesis will mainly deal with the mentioned game class. In particular, we will propose several approaches and strategies to solve cooperative sequential stage games.

In order to better evaluate these algorithms, we consider a distributed and dynamic multi-objective optimization problem that can be modeled as a cooperative sequential stage game. This chapter motivates and formally introduces this problem, the so-called *Iterative Agent Partitioning Problem (IAPP)*. Besides the necessary background, we present a concrete implementation of the IAPP that is similar to the access point selection problem found in large wireless networks. We formally analyze the problem's properties, present a message based optimal algorithm for a special case, and discuss its complexity.

5.1 Problem Motivation

The inherent structure of sequential stage games establishes an interesting area for research on multiagent reinforcement learning. In detail, the key structure is that different stage games are played for a fixed, but to the agents unknown number of iterations, such that the games build a sequence. This structure can be found, for instance, in dynamic systems where each situation can be considered as a special game that models the current situation. When some basic conditions change, a new situation will arise and thus also a new game will be played for some unknown timespan until the situation changes again.

The *Iterative Agent Partitioning Problem (IAPP)* describes such a dynamic system. In this problem, a set of mobile agents has to be repeatedly partitioned onto a set of special target objects such that a global performance measure is optimized. Since the situations, e.g. the agents' positions, can change over time, the problem also contains the structure found in sequential stage games. Besides this, the problem itself also

describes an interesting domain. Consider, for example, the problems illustrated in Fig. 5.1 and the subsequent descriptions:

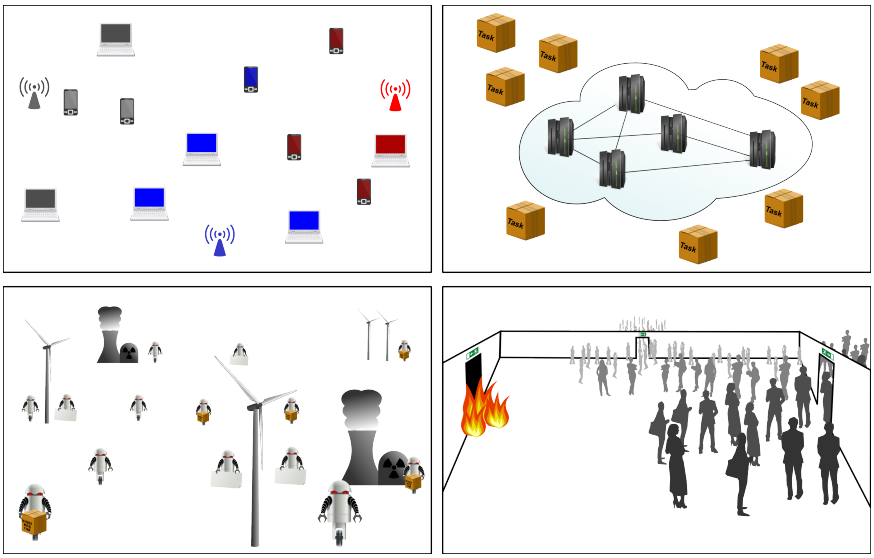


Fig. 5.1 Problems: a) access point selection b) task allocation c) robotics d) crowd evacuation.

Access Point Selection: Consider a set of static access points that are distributed in an environment. Each access point connects its registered clients to a network and owns a certain amount of bandwidth that it can distribute among its clients. In addition, consider a (large) set of mobile devices that are also located in this environment. Each device intends to select one access point in order to access the network. From a device's perspective on the one hand, it would be best to select the nearest access point as this would decrease the amount of power for radio transmission. On the other hand, a device is also interested in a high speed connection, which means that it wants to obtain a maximum part of the bandwidth resources. Obviously, these requirements are contradictory in general and, thus, a good trade-off must be found. Since devices are autonomous and no central instance is available, they must find a good assignment on their own using only local information. The problem is known as *Access Point Selection Problem* in literature [JS02] [KKN06] [NCC+06] [BHJ+07] and clearly is much more complex in reality than described above as, e.g., traffic patterns, physical effects, and the distributions and movements of users must be taken into account.

Load Balancing: The *load balancing problem* is a basic problem in computer science. It deals with the question of how to distribute workload to available resources, for instance, the distribution of a set of computational jobs onto a set of computing nodes in a datacenter. Such a distribution usually has to obey different objectives which often are contradictory. The scheduling of tasks in a multi-processor setting is another related problem.

Robotics: Several scenarios, including search and rescue settings [SMY+04], exploration tasks on distant planets (cf. [Fer99, Ch. 1]), simple box pushing tasks [WdS06], or mining resources on distant planets require the use of many robots

to ensure robustness or to achieve a common goal that a single robot cannot accomplish. These settings may include situations where many robots have to access a set of resources that are distributed in the environment at the same time. A simple example is the distribution of robots to charging stations, where different constraints have to be considered and optimized. Such constraints may include the minimization of the distance each robot has to travel to reach its charging station as well as finding a uniform distribution to the charging stations.

Crowd control: Nowadays, there are several events like concerts or sport games where many people meet in a rather small space. In case of an evacuation, those people must quickly be guided towards emergency exits. It is, however, not only important to guide them, but also to ensure that the available exits are not overloaded due to a too large number of fugitives. In addition, pre-calculated evacuation plans might not be applicable due to blocked or destroyed emergency exists. Accordingly, an optimal assignment has to take all actually available emergency exists and different, probably contradicting objectives into account. Some works that deal with such crowd evacuation settings include [HW05], [PHDL07], or [ZZL09].

Looking at these examples, we can observe that they all can be reduced to the Iterative Agent Partitioning Problem (IAPP). This follows since solutions for all these problems can be reduced to repeatedly finding a mapping from one set of objects to another set of objects that optimizes some (possibly contradicting) objectives. In addition, such a mapping should be determined in a distributed way using only local information, either because no central instance and global information are available, or because obtaining such information is too expensive in terms of time or computing power. Another important aspect of these kinds of distributed multi-objective optimization problems is the possibility that settings change, e.g. by movement of people, mobile devices, or robots.

Accordingly, the general question behind the Iterative Agent Partitioning Problem and its match to the considered class of cooperative sequential stage games makes this problem a good testbed for the approaches developed in this wok.

5.2 Formal Definition

The exemplary problems provided in Fig. 5.1 are related to the *Iterative Agent Partitioning Problem (IAPP)*, which basically is a (distributed) multi-objective optimization problem modeled as a multiagent system. In the IAPP, a set of mobile agents has to be repeatedly partitioned onto a set of target objects. Agents and targets are randomly positioned in an environment. Each agent works on a job at its position for a certain number of time steps. Upon completion, agents are assigned to new jobs at a random position, which leads to dynamically changing scenarios. The goal of the agents is to find an assignment at any time step such that the considered objectives are optimized over a finite but usually unknown number of iterations. In the context of this work, we restrict the environment to a two-dimensional grid, although other environments can generally be considered.

To formally define the problem, we first have to define its elements:

Definition 15 (Environment). An environment $\mathcal{E}$ is a rectangular two-dimensional grid that contains a set of agents, targets, jobs, and other objects. A position in the environment is defined by a tuple $(x, y) \in \mathbb{N} \times \mathbb{N}$.

Definition 16 (Target). A target $T \in \mathcal{T}$ is a special object located in an environment.

Definition 17 (Job). A job $j = \langle p, d \rangle$ is defined by a position $p \in \mathbb{N} \times \mathbb{N}$ in an environment and a duration $d \in \{d_{\min}, \ldots, d_{\max}\}$, where $d_{\min}, d_{\max} \in \mathbb{N}$ denote the minimum/maximum duration.

In Sect. 2.1, we already defined our general understanding of an agent. This definition also holds in this problem context, but it is extended to include a specific position of the agent in the environment. In detail, an agent's position is determined by the job that is assigned to it. The agent's (internal) resources, its (internal) representation of the environment, as well as its sensors and actuators are specified implicitly by the algorithm that determines the agent's behavior.

Definition 18 (Neighborhood). The neighborhood of an agent i with communication radius r_i is defined by the set of agents $\mathcal{N}_i$ that are located within a radius r_i from the position $p(i)$ of agent i according to some distance measure (e.g. Euclidean or Manhattan distance). The k-neighborhood contains the k nearest agents to agent i in the communication radius. If only $\mathsf{k}' < \mathsf{k}$ agents are reachable, then these k' agents are considered as neighbors. An agent is not part of its own neighborhood.

An assignment or *partitioning* of agents to target objects is defined as follows:

Definition 19 (Partitioning, Partition). Let $\boldsymbol{p} : \mathcal{A} \to \mathcal{T}$ be a total function that maps elements from $\mathcal{A}$ to elements from $\mathcal{T}$. Then a *partitioning* of two non-empty sets $\mathcal{A}$ and $\mathcal{T}$ is defined as a multi-set $\mathcal{S}_{\mathcal{A},\mathcal{T}} = \{S_1, S_2, \ldots, S_{|\mathcal{T}|}\}$ having $S_i = \{i \in \mathcal{A} \mid \boldsymbol{p}(i) = T_i\}$, i.e. each *partition* S_i contains those elements from $\mathcal{A}$ that are mapped to element $T_i \in \mathcal{T}$.

The quality of a partitioning $\mathcal{S}_{\mathcal{A},\mathcal{T}}$ is calculated via an *evaluation function* $f : \boldsymbol{S_{A,T}} \to [0, 1]$, where $\boldsymbol{S_{A,T}}$ denotes the set of possible partitionings for the respective agent and target sets. Since we do not want to deal with Pareto optimal solutions, the evaluation function combines different objectives into a single value.

Given these definitions, we can now formally define the IAPP:

Definition 20. The *Iterative Agent Partitioning Problem (IAPP)* is a multi-objective optimization problem modeled as a multiagent system. The problem is defined through the following elements:

- $\mathcal{E}$ is an environment
- $\mathcal{A}$ denotes the set of $n = |\mathcal{A}|$ agents
- $\mathcal{T}$ is a set of $m = |\mathcal{T}|$ target objects
- $\mathcal{J}$ is a set of jobs
- k is the maximum number of considered iterations (time steps)
- $\textsc{job} : \{0, \ldots, k-1\} \times \mathcal{A} \to \mathcal{J}$ is a function that returns a job for every agent in each iteration
- $\mathcal{O}$ is a non-empty set of partitioning objectives
- $f_{\mathcal{O}} : \boldsymbol{S_{A,T}} \to [0, 1]$ is a function that evaluates a partitioning according to the objectives $\mathcal{O}$

The agents' goal is to find a partitioning $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ in each iteration $t \in \{0, \ldots, k-1\}$ such that the *average partitioning quality*

$$\mathsf{APQ} = \frac{1}{k} \sum_{t=0}^{k-1} f(\mathcal{S}^t_{\mathcal{A},\mathcal{T}}) \tag{5.1}$$

is maximized.

Note that we have defined the IAPP in previously published works (e.g. [KKB10a], [KKB11d], [KKB11b]) with two fixed objectives only. Although the above definition generally allows arbitrary objectives, we also want to consider these two fixed objectives in this work. In detail, we investigate an instance of the IAPP that is related to a simplified access point selection problem. In particular, the objectives are the same as in the (static) Online Partitioning Problem [Goe07]:

1. create a *uniform distribution*, i.e. assign approximately $\frac{|\mathcal{A}|}{|\mathcal{T}|}$ agents to each target,
2. minimize the *sum of distances* between agents and selected targets, and
3. minimize the *costs* that are produced according to a cost model.

For simplicity we will, hereafter, talk about the "IAPP" or "our instance of the IAPP" to denote this restricted problem instance. We will explicitly refer to the general problem whenever necessary. Figure 5.2 shows an exemplary setting of our IAPP variant which uses a simple grid environment containing all objects.

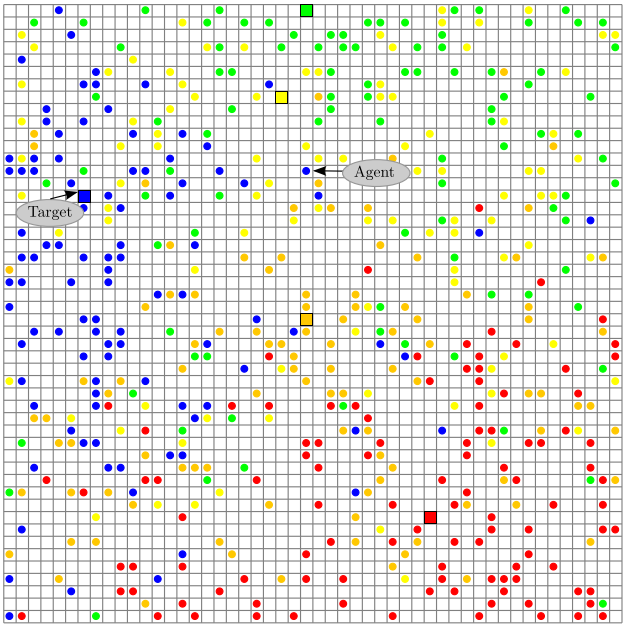


Fig. 5.2 Exemplary scenario of our Iterative Agent Partitioning Problem variant in a grid environment with $n = 500$ agents (circles) and $m = 5$ targets (boxes).

The quality of a partitioning $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ at iteration t with respect to the first two aforementioned objectives can be calculated in the same way as in the Online Partitioning Problem (OPP) [Goe07], i.e. according to

$$f(\mathcal{S}^t_{\mathcal{A},\mathcal{T}}) = \alpha \cdot \left(\frac{\prod_{S_j \in \mathcal{S}^t_{\mathcal{A},\mathcal{T}}} |S_j|}{\left(\frac{|\mathcal{A}|}{|\mathcal{T}|}\right)^{|\mathcal{T}|}} \right) + \beta \cdot \left(\frac{\sum_{i \in \mathcal{A}} \boldsymbol{\delta}(i, \tau(i))}{\sum_{i \in \mathcal{A}} \boldsymbol{\delta}(i, \boldsymbol{p}(i))} \right) \tag{5.2}$$

In (5.2), $\boldsymbol{\delta} : \mathcal{A} \times \mathcal{T} \rightarrow \mathbb{R}^+$ returns the Euclidean distance between an agent and a target, $\tau : \mathcal{A} \rightarrow \mathcal{T}$ returns the nearest target to an agent, and $\boldsymbol{p}$ is the current agent to target mapping described by $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$. Note that the idea of the formula is to normalize the first two objectives against theoretical optimal values such that, together with the weights $\alpha + \beta = 1, \alpha, \beta \geq 0$, the overall partitioning quality will be in $[0, 1]$. If not stated otherwise, we will use $\alpha = \beta = \frac{1}{2}$ in this work. Note that a higher value of $f(\mathcal{S}^t_{\mathcal{A},\mathcal{T}})$ thus denotes a better solution, however, an optimal solution must not necessarily have a value of 1.0.

The optimal value for the DISTRIBUTION[1] should be achieved if the same number of agents is assigned to each target. This is modeled mathematically in (5.2) based on the following proposition:

Proposition 5. *If the sum of n positive numbers is C, i.e. $\sum_{i=1}^{n} x_i = C$ and $\forall i : x_i > 0$, then a function $f(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} x_i$ is maximized if $x_1 = x_2 = \cdots = x_n = \frac{C}{n}$.*

Proof. The well-known inequality of arithmetic and geometric means as shown in (5.3) holds for any n positive real numbers. One way to proof this inequality can be found e.g. in [See73, p.223].

$$\frac{x_1 + x_2 + \cdots + x_n}{n} \geq \sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n} \tag{5.3}$$

Using $\sum_{i=1}^{n} x_i = C$ it directly follows:

$$\left(\frac{x_1 + x_2 + \cdots + x_n}{n}\right)^n \geq x_1 \cdot x_2 \cdot \ldots \cdot x_n \tag{5.4}$$

$$\left(\frac{C}{n}\right)^n \geq x_1 \cdot x_2 \cdot \ldots \cdot x_n \tag{5.5}$$

Obviously, the product becomes maximal if and only if $x_1 = x_2 = \cdots = x_n = \frac{C}{n}$. □

Regarding the third objective, it is worth mentioning that it is not formally captured in (5.2). Given two equally valued solutions, we will in general consider the solution that produces less costs according to a specific cost model to be the "better" one. We will come back to the costs later in this work whenever it is reasonable.

Note that the IAPP, and in particular also the evaluation function (5.2), are based on the Online Partitioning Problem (OPP) defined by Goebels et al. [GKBPW05] [Goe07]. In contrast to our instance of the IAPP, the OPP only considers static scenarios. Another related problem is the General Online Partitioning Problem as defined in [Kem10], which allows obstacles and special regions in the environment. Chapter 2 presents more details on these related problems.

[1] Throughout the thesis, we refer to the first objective as the "distribution" objective and to the second as the "distance" objective. For better readability, we will write DISTRIBUTION resp. DISTANCE to refer to either the objective itself or to the value of the objective.

5.3 Properties

Concerning the general characteristics of optimization problems as presented in Sect. 2.6, our variant of the Iterative Agent Partitioning Problem is an optimization problem that is

- multivariate, as agents and targets influence the objective function,
- discrete, since we basically search for a mapping of agents to targets,
- linear, as the objective function is a linear combination of a set of objective functions,
- constrained, because there is a non-empty set of partitioning objectives,
- multimodal, because there generally is more than one optimal solution, and
- a multi-objective optimization problem, as there is a set of objectives to be optimized at the same time.

In the remainder of this section, we will investigate some formal properties of the used evaluation function and of partitionings.

5.3.1 Evaluation Function

The evaluation function (5.2) combines two different objective values which we will analyze separately in this section. We begin with the DISTANCE objective that evaluates m^n different assignments of n distinguishable agents to m targets. Its value is hardly analyzable for general settings, as it strongly depends on the positions of agents and targets, on the spatial distribution of both, and the partitioning itself. Since the position related dependencies are obvious, we provide an example that underlines the influence of the partitioning on the DISTANCE value. Therefore, consider the exemplary setting sketched in Fig. 5.3. Given that all agents are assigned to target T_1, which is the closest target for all agents, the DISTANCE value will be 1.0. Since the DISTRIBUTION value in this case is 0.0, we obtain an overall rating of 0.5 (using $\alpha = \beta = 0.5$ as weights). On the other extreme, assume an optimal distribution value such that each agent selects the target on the outer circle that lies on a line connecting agent and target T_1 as indicated by the dotted lines. In this case, the DISTANCE value becomes $\frac{n}{1+z(n-1)}$ since T_1 is only 1 unit away from any agent, but the selected targets of all agents (except agent i) are z units away for all agents. By assumption $z > 1$ and thus the DISTANCE value will become 0 in the limit for $z \to \infty$.

In contrast to the DISTANCE value, the DISTRIBUTION objective value does not distinguish which agents actually selected which target, but only rates the number of agents per target. As agents in this case cannot be distinguished, the number $N_{\text{all}}(n, m)$ of all possible assignments is given by (5.6).

$$N_{\text{all}}(n, m) = \binom{m + n - 1}{n} \tag{5.6}$$

Since the DISTRIBUTION value evaluates to zero if at least one partition is empty, i.e. at least one target is not selected by any agent, the likelihood of obtaining a DISTRIBUTION of zero becomes interesting. With the help of the inclusion-exclusion principle (cf. e.g. [Doh03, Ch. 1]), we can use (5.7) to calculate the number $N_{\emptyset}(n, m)$ of all possible assignments where at least one partition is empty.

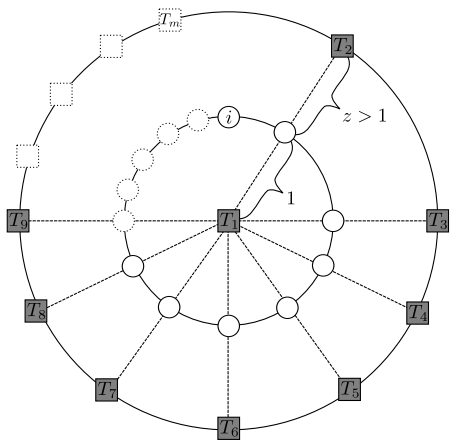


Fig. 5.3 Influence of the partitioning on the DISTANCE value.

$$N_{\emptyset}(n,m) = \sum_{i=1}^{m} (-1)^{i+1} \binom{m}{i} \binom{m-i+n-1}{n} \tag{5.7}$$

The likelihood $L_0(n,m)$ to obtain a DISTRIBUTION of zero then is given by (5.8).

$$L_0(n,m) = \frac{N_{\emptyset}(n,m)}{N_{\text{all}}(n,m)} \tag{5.8}$$

Figure 5.4 illustrates that likelihood for $n \in [64, 1000]$ agents and $m \in [2, 64]$ targets. Figure 5.5 shows the agents per target ratio for the same number of agents and targets as in Fig. 5.4. Matching both figures reveals that the larger the ratio between agents and targets, the lower the probability of obtaining a DISTRIBUTION value of zero.

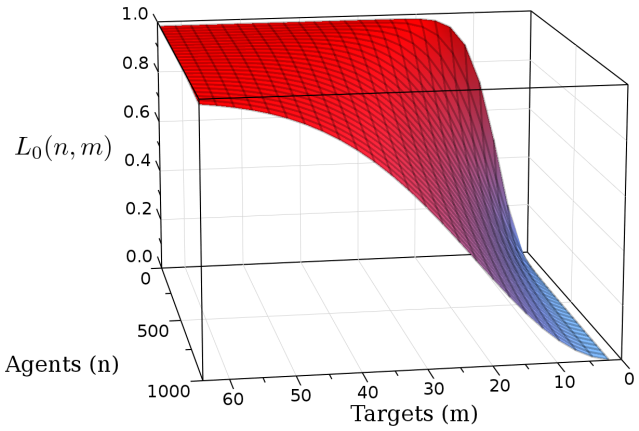


Fig. 5.4 Likelihood of a zero-valued DISTRIBUTION.

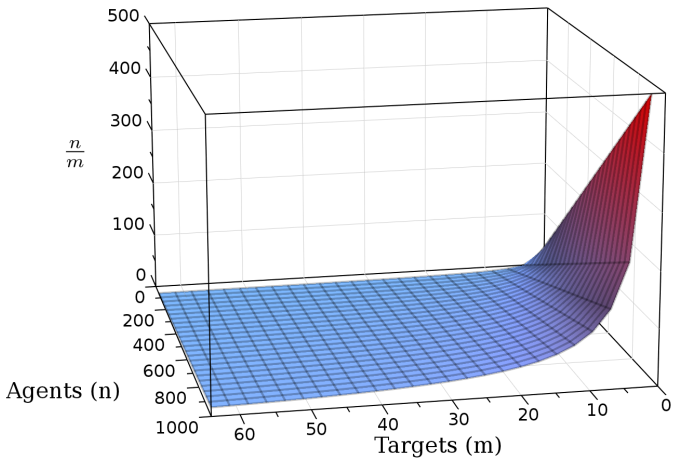


Fig. 5.5 Number of agents per target in an optimal distribution.

Next, we investigate the behavior of the DISTRIBUTION value if the partitionings deviate from the optimum. Therefore, consider Fig. 5.6, which exemplary shows the objective value for settings with $n = 333$ agents and different numbers of m targets. For the graphs, we assumed that $m - 2$ targets are assigned the optimal number of agents ($\frac{n}{m}$). The two remaining targets, say T_1 and T_2, in sum are assigned $2\frac{n}{m}$ agents. However, the number of agents i that are assigned too less to T_1 are assigned to T_2. Formally, the figure thus visualizes the following function (5.9).

$$v_{n,m}(i) = \frac{\left(\frac{n}{m} - i\right)\left(\frac{n}{m} + i\right)\left(\frac{n}{m}\right)^{m-2}}{\left(\frac{n}{m}\right)^{m}} = \frac{\left(\frac{n}{m} - i\right)\left(\frac{n}{m} + i\right)}{\left(\frac{n}{m}\right)^{2}} \tag{5.9}$$

Consider the scenario with $m = 111$ targets where the optimal number of agents are assigned to 109 targets (three agents per target) and only two targets are not optimally selected, i.e. selected by 1 resp. 5 agents. Intuitively, one might consider such a solution as "good" since it is close to the optimal solution as just two agents have to be reassigned. However, this is not reflected by the DISTRIBUTION objective value which is only 0.6 in this case. This effect obviously depends on the ratio between the number of agents and targets as illustrated in Fig. 5.6 and it is less distinct in settings with a higher agent to target ratio.

According to the above observations, it is important to keep in mind how the evaluation function (5.2) is structured when one compares solution qualities, in particular for different settings.

5.3.2 Properties of Partitionings

Next, we present and prove some properties of solutions for the partitioning problem. We will refer to these properties in the remainder of this work. Parts of these results have already been published in a technical report [KKB10b]. We start by considering partitionings at a fixed iteration t.

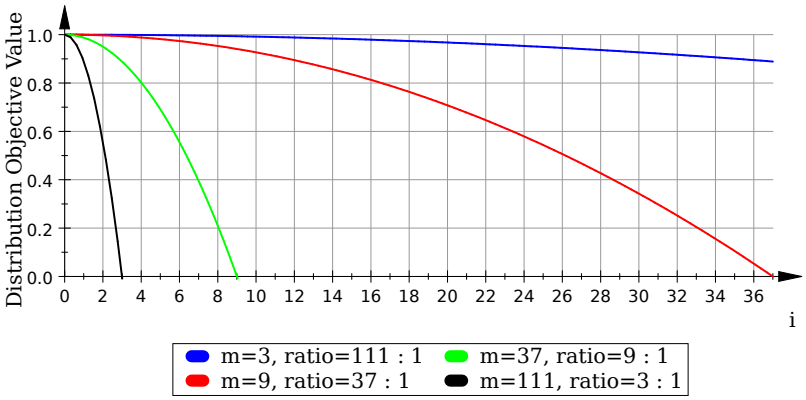


Fig. 5.6 Influence of deviation from optimal assignments.

Property 2 ([KKB10b]). *In general, there can be more than one optimal solution.*

Proof. We assume the opposite, i.e. the optimal solution is always unique. Figure 5.7 shows two different optimal solutions for the same scenario according to (5.2). This is a contradiction to the assumption. □

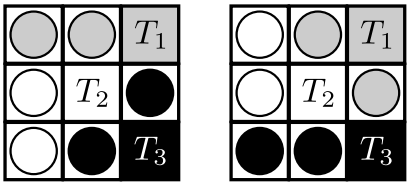


Fig. 5.7 Two optimal solutions for the same scenario with a partitioning quality of ∼0.96.

Property 3 ([KKB10b]). *Consider a setting with $m \geq 2$ targets and $n \geq m$ agents. Then an optimal solution according to (5.2), having $\alpha = \beta = \frac{1}{2}$, does not necessarily imply an equal distribution.*

Proof. We have to distinguish the following two cases:

Case 1: $n \mod m = 0$, i.e. $n = qm$ for some $q \in \mathbb{N}$
Proof by contradiction. Assume an optimal solution under the given preconditions always results in an equal distribution, i.e. q agents are assigned to each target. Figure 5.8 provides a counter example.

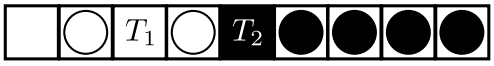


Fig. 5.8 Counter example used in case 1. According to evaluation function (5.2), this setting is an optimal solution which has a distribution value of $0.\bar{8}$, and a distance value of 1.0.

Case 2: $n \mod m \neq 0$.
Since each agent has to select one target, equal distributions cannot occur in these settings.

Finally, the property is proven as it holds in both cases. □

Property 4 ([KKB10b]). *An optimal solution does not necessarily imply a distance function value of* 1.0.

Proof. Figure 5.9 shows an optimal solution according to (5.2) having a distribution value of 1.0 and a distance value of 0.875. □

Fig. 5.9 Non-optimal distance value in an optimal solution.

We next investigate a worst case of how an optimal partitioning for iteration t may perform in iteration $t + 1$.

Property 5. *Let* $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ *be an optimal partitioning for iteration* t*. Assume restricted movement of agents by at most one step per iteration. Then, the same partitioning is not guaranteed to remain optimal in the next iteration* $t + 1$.

Proof. Consider the optimal partitioning shown on the left side in Fig. 5.10. Assume that $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ is also an optimal partitioning at iteration $t + 1$, after both agents have moved by one step. The right half of Fig. 5.10 shows a counter example. □

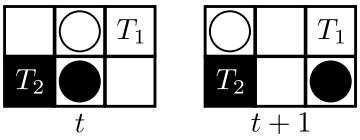


Fig. 5.10 Non-optimality of partitioning $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ in iteration $t + 1$.

From these properties and from the analysis of the evaluation function in the previous subsection, it follows that the value obtained from (5.2) should be carefully interpreted. In particular, if solution values of different settings should be compared.

5.4 Message Based Optimal Algorithms

This section introduces a distributed, communication based approach for calculating optimal partitionings in any iteration of our Iterative Agent Partitioning Problem with two targets. The idea of the proposed algorithm is to realize Goebels' Two Target Optimal approach [Goe07, Sect. 3.2.2] with the help of a communication protocol. Under some assumptions, we prove that our distributed variant maintains the ability of calculating an optimal solution for two target settings. We prove that the approach requires $\mathcal{O}(n)$ messages. Then, we also briefly discuss a (theoretical) distributed approach for settings with arbitrarily many targets.

5.4.1 Distributed Optimal Approach for Two Targets

In this section, we propose a distributed algorithm and prove that it finds optimal partitionings for a fixed iteration of the IAPP in settings with two targets. Therefore, the approach assumes the following:

1. To ensure coordination through a token message, the communication graph has to be connected and communication has to be error-free.
2. Agents can be distinguished to ensure proper routing of messages.
3. Each agent knows the weight factors α and β as required by the evaluation function (5.2).

The proposed distributed algorithm is inspired by the central instance TTO algorithm of Goebels [Goe07, Sect. 3.2.2], as it uses the same idea of sorting agents according to the distance difference to both targets. In contrast to TTO, which initially assigns agents to their nearest target, our approach starts by assigning all agents to one target, as this will simplify the presented proof.

5.4.1.1 Algorithm

Before we present the *Distributed Optimal Approach for Two Targets (DOTT)* in detail, first consider a rough overview on the procedure, which can be divided into five steps:

1. Using a communication-based sorting procedure, sort the agents ascending according to their distance difference between the two targets.
2. Initially assign all agents to target T_1.
3. Use a communication protocol to collect some information required to locally determine the current solution quality.
4. Use a token message to let agents consecutively, i.e. along the sorting criterion, decide to switch to target T_2 if this improves the solution. Information on the solution quality stem from the token.
5. The algorithm stops if the first agent decides not to switch targets or all agents have switched.

The key idea of the algorithm can be found in the sorting of the agents according to the distance difference between the two targets (cf. [Goe07, Sect. 3.2.2]). Formally, the n agents are sorted in a list $\ell = \langle i_0, i_1, \ldots, i_{n-1} \rangle$ such that $\boldsymbol{\delta}(i_j, T_2) - \boldsymbol{\delta}(i_j, T_1) \leq \boldsymbol{\delta}(i_{j+1}, T_2) - \boldsymbol{\delta}(i_{j+1}, T_1), \forall j \in [0, n-2]$. From the distance difference of an agent one can deduce if it could either improve (negative distance difference), retain (no distance difference), or worsen (non-negative distance difference) the DISTANCE objective by switching to target T_2. To stepwise improve the solution quality by assigning agents to T_2, the best way would be to increase the DISTRIBUTION objective value by also improving the DISTANCE value. Or, if this is not possible, to reduce the latter as little as possible until an optimal overall solution is found. Accordingly, the best order of reassigning agents is given by the sorted list ℓ.

Next, we describe the algorithmic details. Based on the list ℓ as introduced above, we define the ID of an agent as its position in ℓ, starting with zero. Please note that this ID can easily be calculated by the agents in a distributed manner using a simple

message protocol. With the help of this ID definition, the basic idea of the algorithm is as follows. First, all agents are assigned to the same target T_1. Each agent knows the total number of agents n in the system, as well as the sum of distances $\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}$ over all agents and their respective nearest targets. Note that these two values can easily be calculated and distributed using an appropriate message protocol. With the help of a token message the agents then iteratively determine the optimal solution. Only the agent that currently holds the token is active. Initially, the agent i with ID 0 obtains the token tok and initializes the token's values, which in detail are:

- tok.id: The ID of the last agent that held the token (initially 0).
- tok.distance: The current sum of distances of all agents that select target T_1.

Note that the sum tok.distance can easily be calculated in the beginning while n and $\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}$ are determined. In the ongoing procedure, it will be updated using only local information of the currently active agent.

The agent i that holds the token calculates the following two hypothetical values, which are equal to the value of evaluation function (5.2), if the agent selects target T_1 (5.10) or T_2 (5.11). Note that all these values are known to the agent either by assumption or from the token message.

$$h(i,T_1) = \alpha \cdot \left(\frac{(n - \mathsf{tok.id}) \cdot \mathsf{tok.id}}{\left(\frac{n}{2}\right)^2} \right) + \beta \cdot \left(\frac{\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}}{\mathsf{tok.distance}} \right) \tag{5.10}$$

$$\begin{aligned} h(i,T_2) = \alpha \cdot & \left(\frac{(n - \mathsf{tok.id} - 1) \cdot (\mathsf{tok.id} + 1)}{\left(\frac{n}{2}\right)^2} \right) \\ + \beta \cdot & \left(\frac{\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}}{\mathsf{tok.distance} - \boldsymbol{\delta}(i,T_1) + \boldsymbol{\delta}(i,T_2)} \right) \end{aligned} \tag{5.11}$$

Next, the agent compares the two values which results in one of the following cases:

1. $h(i,T_2) > h(i,T_1)$: Selecting target T_2 instead of T_1 would improve the overall solution quality. Thus, the agent assigns itself to T_2 and updates the token:
   - $\mathsf{tok.id} = \text{ID}(i)$, ID of this agent i
   - $\mathsf{tok.distance} = \mathsf{tok.distance} - \boldsymbol{\delta}(i,T_1) + \boldsymbol{\delta}(i,T_2)$

   Finally, the agent sends the token message to the agent with the next higher ID.
2. $h(i,T_2) \leq h(i,T_1)$: In this case the agent cannot improve the objective function value and thus sticks to target T_1. Hence, the token is not sent to another agent and the algorithm is terminated.

The algorithm resubmits the token until case two occurs or all agents have been activated by receiving the token.

5.4.1.2 Optimality

The optimality of the approach follows from Theorem 6:

Theorem 6. *Under the given assumptions, the Distributed Optimal Approach for Two Targets terminates with an optimal solution.*

Proof. The algorithm terminates if the token message has been processed by all agents, or if one agent is unable to improve the solution.

In the first case, it is obvious that all agents improved the solution by reassigning from target T_1 to T_2. Then, since no agent can be reassigned, the solution is optimal.

The second case is more interesting. Let the agent with ID i be the first to experience that a target change from target T_1 to T_2 does not improve the solution quality. Thus, all agents with an ID smaller than i clearly improved the overall solution value by changing to T_2. By construction of the algorithm, the agent with ID i sticks to target T_1. Then, it remains to show that there is no other agent with an ID j, $i < j \leq n-1$, which could improve the solution quality.

Therefore, let us assume that there is at least one agent with ID j that improves the solution quality although agent i could not improve the solution. Since the DISTRIBUTION objective part of the solution quality is the same, independent of whether i or j change to target T_2, the improved solution could only be attributed to the DISTANCE objective. Thus we can neglect the DISTRIBUTION objective. Because the weight factor β is equal for all agents, we can transform the following DISTANCE-related inequality to obtain (5.12).

$$
\begin{aligned}
h_{\text{DISTANCE}}(j, T_2) &\geq h_{\text{DISTANCE}}(i, T_2) \\
\frac{\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}}{\mathsf{tok.distance} - \boldsymbol{\delta}(j, T_1) + \boldsymbol{\delta}(j, T_2)} &\geq \frac{\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}}{\mathsf{tok.distance} - \boldsymbol{\delta}(i, T_1) + \boldsymbol{\delta}(i, T_2)} \\
\mathsf{tok.distance} - \boldsymbol{\delta}(i, T_1) + \boldsymbol{\delta}(i, T_2) &\geq \mathsf{tok.distance} - \boldsymbol{\delta}(j, T_1) + \boldsymbol{\delta}(j, T_2) \\
\underbrace{\boldsymbol{\delta}(i, T_2) - \boldsymbol{\delta}(i, T_1)}_{\text{ID}(i)} &\geq \underbrace{\boldsymbol{\delta}(j, T_2) - \boldsymbol{\delta}(j, T_1)}_{\text{ID}(j)}
\end{aligned}
\tag{5.12}
$$

As (5.12) reveals, the distance objective depends on the distance difference between the two targets of both involved agents, and thus also on the ordering of the agent list ℓ as described earlier. Next, consider the following cases for (5.12):

- $\text{ID}(i) = \text{ID}(j)$: In this case, the DISTANCE objective value would be the same for both agents if they switch to target T_2. Since the DISTRIBUTION value is the same, too, and because i by assumption cannot improve the solution, this also holds for agent j.
- $\text{ID}(i) > \text{ID}(j)$: This is a contradiction to the definition of the ID by which $\text{ID}(i) \leq \text{ID}(j)$ holds. Accordingly, $h_{\text{DISTANCE}}(j, T_2) < h_{\text{DISTANCE}}(i, T_2)$ holds, and because agent i cannot improve the solution, the same also follows for agent j.

Since no agent j with $i < j \leq n-1$ can improve the solution in both cases and due to the optimality of the TTO approach [Goe07, Sect. 3.2.2] the theorem follows. □

According to this result the message-based realization of the TTO approach is guaranteed to terminate with an optimal solution.

5.4.1.3 Complexity

In this section, we determine the number of messages of the algorithm. Therefore, we assume that our approach runs at the application layer of the OSI model (see e.g. [Zim80] or [TW11]), i.e. our analysis does not consider routing overhead etc.

The runtime of the proposed approach depends on the implementations of the mechanisms that determine the initial knowledge and the IDs. The number of agents n, the sum of distances $\boldsymbol{\delta}^{\Sigma}_{\text{opt.}}$, and the initial distance sum when all agents are assigned to target T_1, can be determined using $\mathcal{O}(n)$ messages. In detail, such a protocol sends one message from one agent to the next. Each agent updates the three values stored in that message by adding its own information. The final message is forwarded to each agent and the required information are stored locally.

A simple protocol for determining the IDs can also be implemented using $\mathcal{O}(n)$ messages. Again, one message is send from one agent to the next. Each agent calculates its distance difference and appends this information to the message before retransmission. Finally, when all agents appended their information, the list is sorted and the resulting IDs are distributed by sending a message to each agent. Note that this approach leads to message sizes in $\mathcal{O}(n)$. Smaller messages can be achieved if IDs are determined consecutively by first determining an agent with ID n, i.e. the largest distance difference, and then restarting the protocol to find an agent with ID $n-1$ and so on.

The actual process of determining the optimal solution requires at most n messages. In the end, the approach hence requires $\mathcal{O}(n)$ messages in total.

5.4.2 Distributed Approach for Many Targets

The optimal central instance approach for arbitrary numbers of targets m and agents n with runtime $\mathcal{O}(m^n)$ proposed by Goebels [Goe07, Sect. 3.2], can theoretically be implemented by a distributed algorithm. Therefore, such an algorithm would require error-free communication and a connected communication graph. Furthermore, one agent must be able to act as central instance and execute Goebels' algorithm. The required information about partitioning qualities can be obtained using $\mathcal{O}(n)$ messages. Also assignment commands calculated by that super-agent can be submitted to each agent, again involving $\mathcal{O}(n)$ messages. Thus, the number of messages involved in such a distributed version of the algorithm grows exponentially in $\mathcal{O}(m^{n+1})$, as $\mathcal{O}(n)$ messages are required to test each of the m^n possible partitionings.

5.4.3 Discussion

We proposed a distributed, message-based algorithm that provably calculates an optimal solution for a fixed instance of the IAPP with two targets. The algorithm basically realizes Goebels' [Goe07] central instance algorithm for two targets as a distributed variant. We provided detailed descriptions for the approach and the message structures. The ability to find optimal solutions is proven formally. We also showed that the approach requires $\mathcal{O}(n)$ messages in total. In the end, we discussed briefly how Goebels' brute force approach could be implemented using messages.

A connected network is an essential assumption that is required to allow proper routing of messages to any node in the network. Routing itself can be realized using different algorithms, as they are developed for wireless sensor or (mobile) ad hoc networks (e.g. AODV [PR99]). The knowledge of the weight factors could either

be hard-coded into the agents or published to all agents using sophisticated efficient broadcast algorithms in wireless networks (e.g. [LH06]).

Both algorithms intend to find an optimal partitioning for a particular iteration of the IAPP. Thus, their applicability under dynamic settings involving moving agents is restricted due to the time required to realize communication. In particular, the sketched approach for many targets in large systems is impractical even for a single iteration.

5.5 Complexity

This section addresses the complexity of our IAPP variant. Therefore, we concentrate on solving the partitioning problem at a fixed iteration t. We start with the consideration of two extreme cases for the weight factors in evaluation function (5.2):

Case 1 ($\alpha = 0$ and $\beta = 1$): For $\alpha = 0$ and $\beta = 1$ the problem is trivially solvable in polynomial time as it is reduced to assigning each agent to the nearest target. Therefore, each agent simply needs to know the distance to each target, sort the distances and select the nearest target.

Case 2 ($\alpha = 1$ and $\beta = 0$): In this case, the problem is reduced to finding a partitioning with equally sized subsets. This obviously can be accomplished in polynomial time as well, e.g. by assigning $\lfloor \frac{n}{m} \rfloor$ (randomly chosen) agents to each target and, in case of an odd number of agents, assigning the remaining agents to one distinct target each. An example for this approach is the ID-Dependent Strategy proposed by Goebels [Goe07].

Note that approaches for both cases can be realized either using a central instance or in a distributed way if the agents know the distances to all targets (case 1) or have unique, consecutively numbered IDs (case 2).

For general weights and settings with two targets, Goebels [Goe07, Sect. 3.2.2] proposed the central-instance Two Target Optimal algorithm, that is proven to terminate with an optimal solution (see also Sect. 2.2). Hence, the problem can be solved in polynomial time for up to two targets if a central instance is available. In Sect. 5.4.1, we provided a distributed message-based variant of TTO that provably finds an optimal solution under some assumptions.

The partitioning problem with three targets at a fixed iteration is related to several $\mathcal{NP}$-complete problems (e.g. *Graph Partitioning* [GJ79, Problem ND14], *3-Dimensional Matching* [GJ79, Problem SP1], or *Exact Cover by 3-Sets* [GJ79, Problem SP2]). Due to this relatedness, we strongly assume the problem to be at least $\mathcal{NP}$-hard. This conjecture is underlined because all aforementioned problems are solvable in polynomial time for "two"—as it is the case in our problem—but become $\mathcal{NP}$-complete for "three". A proof for this conjecture, however, remains an open issue. The only known central instance algorithm that provably finds an optimal solution in general settings is a brute force approach proposed by Goebels [Goe07, Sect. 3.2], whose runtime is in $\mathcal{O}(m^n)$.

Table 5.1 summarizes the above complexity results for finding a partitioning in a fixed iteration t of our IAPP variant. The complexity of the general IAPP is omitted in this work, as it depends on the used set of objectives.

Table 5.1 Overview on algorithms and complexity results for finding a partitioning in a fixed iteration t of our IAPP variant.

central instance	weight factors	number of targets	approach	complexity
indifferent	$\alpha = 0$, $\beta = 1$	arbitrary	select nearest target	time (per agent for sorting): $\mathcal{O}(m \log m)$
indifferent	$\alpha = 1$, $\beta = 0$	arbitrary	ID-based random target selection	time (per agent): $\mathcal{O}(1)$
yes	arbitrary	2	TTO [Goe07]	time: $\mathcal{O}(n \log n)$
no	arbitrary	2	DOTT (see Sect. 5.4.1)	messages: $\mathcal{O}(n)$
yes	arbitrary	arbitrary	brute-force algorithm [Goe07]	time: $\mathcal{O}(m^n)$
no	arbitrary	arbitrary	message-based distributed brute-force algorithm (cf. Discussion in Sect. 5.4.2)	messages: $\mathcal{O}(m^{n+1})$

5.6 Summary

The presented Iterative Agent Partitioning Problem in its general form constitutes an interesting research area. Due to its open definition with respect to the considered objectives, it can be used to formally model a wide variety of different problems. In addition, the variant that we will consider in this work is easy to understand but not trivially solvable.

Since the IAPP represents a problem that can also be modeled as a cooperative sequential stage game (see Sect. 9.2 for details), the problem is a good means to evaluate the learning approaches developed in this work.

Chapter 6

Storage Media

The previous chapter introduced the Iterative Agent Partitioning Problem (IAPP) which, as discussed, could be used, for instance, to realize the distribution of robots onto a set of charging stations. In such settings, robots might be unable to communicate with other robots due to their spatial distribution and limited communication radii. If a robot (agent, hereafter) has gathered useful knowledge in a particular region but moves to another area in the environment, then its knowledge is lost due to the movement if the agent cannot communicate it.

The concept of storage media that are located in an environment can help to deal with such situations. Agents can access these media within a limited communication radius and write or read information. In general, this concept hence enables agents to coordinate their behavior via indirect interaction through the media. Additionally, the media decouple agent interaction from time and space. Later in this work, we will successfully combine storage media with the proposed distributed multiagent reinforcement learning approach (Chapter 11). Also, we will show how the media can be used in the context of the IAPP with a communication intensive state-of-the-art algorithm (Chapter 12).

Therefore, the purpose of this chapter is to introduce this concept as one foundation of this work and to discuss its benefits and challenges, and to present related work (Sect. 6.1). In addition, the chapter is used to discuss some related aspects. Section 6.2 investigates the strategic positioning of media and proposes an approach which could help to optimize the number of media required in our variant of the Iterative Agent Partitioning Problem. Then, in Sect. 6.3, we theoretically investigate the influence of a single obstacle on the communication coverage of a storage medium. In Sect. 6.4, we present and analyze a greedy storage media placement algorithm. Finally, we summarize the obtained results in Sect. 6.5.

Parts of this chapter have been published in [KKB10a].

6.1 Concept

The idea of externally storing information and knowledge is not new but well established in human societies: For hundreds of years, we use books as external storage.

Basically the emergence of writing systems, like the Egyptian hieroglyphs, enabled the humans to externally store information. In particular, this concept allowed us to transfer knowledge without the need of direct communication and thus, decouples information passing from time and space.

In the context of multiagent systems, we decided to model such external storage as devices that are located in an environment [KKB10a]. Before we discuss the concept in more detail, let us first give a formal definition of a storage medium:

Definition 21 (Storage Medium). A *Storage Medium (SM)* is a passive object located in an environment $\mathcal{E}$ of a multiagent system. It can be described by a tuple $M = \langle p, r_M, \mathsf{cap}, \mathsf{read}, \mathsf{write} \rangle$, where

- p is the position of the medium in the environment,
- $r_M \in \mathbb{R}^+$ is the medium's communication radius,
- cap denotes an (abstract) storage capacity,
- $\mathsf{read}, \mathsf{write}$ are actions for reading resp. writing information to/from the medium.

The set of all storage media is denoted by $\mathcal{M}$.

By this definition, storage media are passive which means that they do not actively issue any actions. Together with the fixed position attributed to a medium, these two facts mainly distinguish media from agents. Since actuators and sensing abilities of agents are also restricted to certain radii (cf. Def. 3), it follows from the definition of a storage medium, that it can be used by an agent if and only if they can mutually reach each other within the respective radii.

Storage media in MAS offer several of opportunities:

Extended memory: In MAS with agents that have a limited internal memory, the storage media can be used as an extended external memory. In [KKB10a], we investigated the DISTANCE objective of the IAPP in capacity-constrained MAS. In that MAS, agents in an extreme scenario were only able to store two information items internally. We showed that the media enabled the mobile agents to solve the problem successfully.

Indirect communication: Agents can indirectly communicate with each other through storage media by reading and writing information. On the one hand, a medium hence can act as mediator by enabling communication between agents that can interact with the same medium, but which are too far away from each other to communicate directly. On the other hand, the indirect communication can also be used to realize the concept of stigmergy [BDT99], i.e. indirect communication by modifying the environment. For instance, the media can store artificial pheromones as they are used in ant colony optimization algorithms [BDT99].

Persistence of data: In general, mobile agents will frequently be confronted with new situations, which require them to rebuild their knowledge-bases. In case of limited storage capacities the agents might be forced to forget, i.e. to override, previously obtained knowledge. By storing data on storage media that—by assumption—have higher capacities than single agents, once gained knowledge can be preserved, e.g. for later use by other agents.

Location-based knowledge: From the previous point, it also follows that location-related knowledge can be preserved in regions where it is applicable, even if the agent that originally gathered the knowledge has left the region. Since knowledge acquisition processes like learning often require a long time, agents thus can significantly profit from such externally available knowledge.

Shared memory: By using storage media as extended and external memory, agents that access the same medium obtain a shared memory. Such shared memory then can be used to cooperatively construct solutions for complex tasks.

The usage of storage media in MAS, and thus also the challenges that have to be addressed, highly depend on the system and problem characteristics. In general, we can identify the following issues:

Strategic positioning: Since agents and media interact via communication in a limited range, the positioning of the media plays an important role. We will come back to this question in Sects. 6.2 and 6.3.

Number of media: This issue is closely related to the previous item. In problems where agents only have to coordinate in so-called *coordination locales*, i.e. in a subset X of all possible states, it might be sufficient to use $\mathcal{O}(|X|)$ many media. Such problems can be modeled as *Distributed POMDPs with Coordination Locales* [VKT+09]. In Sect. 6.2, we will also present an idea of how the number of media in the IAPP can be determined.

Content: A core question deals with the kind of items that should be stored on the media. Possible contents include gathered information, region-specific knowledge or strategies, and learned knowledge, e.g. in the form of Q-tables for Q-Learning based approaches. Clearly, the content depends on the considered approach and settings.

Additional element: Since media introduce an additional type of objects to deal with, one should consider the question of whether or not the usage of such an additional element is required. Whether there are benefits from using the media again depends on the considered settings. As we will see in this work, storage media offer a good means for coordination in highly dynamic settings with mobile agents.

Access: One question in this context is, when should agents interact with a medium? As more than one agent at the same time may try to interact with the medium, techniques that ensure proper read/write access are required to ensure data integrity.

Accordingly, storage media are an interesting concept for multiagent systems as they offer various benefits. However, using such media also comes with new issues that have to be considered carefully. In this work, we intend to provide some first results and propose ideas for applying the concept in our IAPP variant. This chapter, in particular, will focus on the general concept of storage media, and only in some subsections relate to the IAPP. Later, in Ch. 11 we will provide some results for the combination of learning algorithms and storage media. In addition, in Ch. 12, we will provide an approach that uses external storage media in a state-of-the-art algorithm for the IAPP.

6.1.1 Related Work

In this section, we review two concepts from the literature that are related to storage media: blackboard systems (Sect. 6.1.1.1) and the Agents & Artifacts meta-model (Sect. 6.1.1.2).

6.1.1.1 Blackboard Systems

We adopt the terminology of the "Blackboard Systems" book edited by Engelmore and Morgan [EM88]. They define a *blackboard model* as a problem-solving model composed of a blackboard and a set of independent knowledge sources (domain experts). Based on this, a *blackboard framework* specifies or implements the components of a blackboard model. Hence, a *blackboard application*, which usually is constructed based on a framework, is intended to solve a particular problem using domain-specific knowledge sources. The term *blackboard system* is used as a generic term for blackboard frameworks and applications.

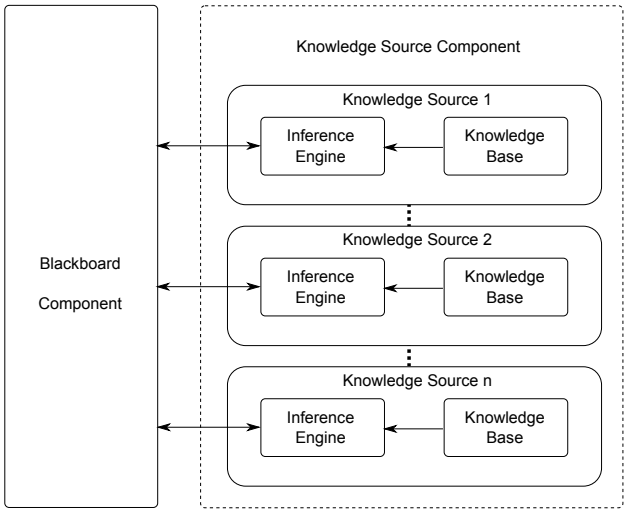


Fig. 6.1 Basic blackboard model (based on [EM88]).

Based on [EM88, Ch. 1], we now introduce the general and basic blackboard model, which consists of two components as shown in Fig. 6.1. The *Blackboard* itself can be regarded as shared working memory, i.e. a global database that holds the current state of the problem solving procedure, input data, partial or final solutions, hypotheses or control information. The knowledge stored in the blackboard is often organized in a hierarchy of different abstractions. A *knowledge source* (KS) usually is an expert for one (or more) abstraction levels. It consists of an inference module and a problem-specific knowledge base. A KS is able to contribute partial solutions to a certain level of the hierarchy. The internal knowledge representations of the knowledge sources may be different, however it is required that they use a common format when reading and writing to the blackboard. Knowledge sources in general are independent from each other. In particular, they only communicate indirectly by reacting to changes on the global blackboard. The domain knowledge needed to solve the considered problem is partitioned among the KSs.

It is particularly noticeable, that the blackboard model—other than the general architecture of blackboard frameworks—does not explicitly define a control module. This is due to the fact, that the blackboard model itself only describes a problem-solving model, i.e. it defines a general scheme for solving problems. However, a control mechanism that selects the next activated KS to be executed is also considered

as an inherent part of each blackboard-based system [DJBS89]. Accordingly, control can be located at one or more different locations, for instance within the KSs, in a separate controller, or on the blackboard itself.

Problem solving in the context of a blackboard model is accomplished in an opportunistic fashion: a knowledge source applies its knowledge as it fits to the current problem-solving state on the blackboard. Thus, it creates a new hypothesis which may trigger other knowledge sources to apply their knowledge and so forth until one or more solutions are incrementally constructed on the blackboard.

In order to realize a blackboard system, the model's components have to be specified in more detail. According to the general *blackboard framework* presented by Engelmore and Morgan [EM88, Ch. 1], a blackboard framework contains knowledge sources, a blackboard, as well as a distinct control component.

Since the blackboard is a component that is accessed by a set of different knowledge sources, that architecture seems to be closely related to the storage media concept in MAS. However, there are several key differences that distinguish the two concepts from each other. Table 6.1 summarizes these differences based on properties that are mainly deduced from [EM88, Ch. 1] and [Cor03].

Property	**Blackboard System**	**Storage Media**
Blackboard	global database	local(ly accessible) database
	active: may provide a KS trigger mechanism	passive: storage object
	Solution constructed *on* blackboard	coordination artifact
Agent/KS	KS provides (expert) knowledge	Comparable to KS, agents provide (local) knowledge, *but* they also extract and store knowledge from the blackboard
	static knowledge in KS	agents may have dynamic, state depended knowledge
Communication	"Blackboard systems prohibit direct interaction among modules, as all communication is done via the blackboard." [Cor03, p. 6] Allowing KS activations to talk directly "is a major departure from the blackboard-system model" [Cor03]	direct communication between agents, or indirect via SM
Control	Control component is independent from KSs	Agents act autonomously (in particular there is no external control unit)
	activated by scheduler/focus-of-control-DB or by control KSs	autonomous agents
	KS may provide separate control knowledge to control shell	autonomous agents
	centralized	de-centralized
Parallelism	traditional blackboard systems execute only one KS activation at a time	fully parallel working agents

Table 6.1 Differences between blackboard systems and MAS with storage media.

6.1.1.2 Agents and Artifacts Meta-Model

The *Agents and Artifacts (A&A)* meta-model [ORV08] provides an approach for agent-based software development. In particular, in open MAS where agents can join or leave a system at any time, some kind of organizational infrastructure is required to keep such a dynamic system up and running. The basic idea of the A&A framework is to divide the system into three conceptually different entities [ORV08]:

1. *Agents*, as usual, are those entities that autonomously work in a given environment (one or more workspaces) so as to reach their design objectives.
2. *Artifacts* are a conceptual means that realize tools, functions, or represent resources which can be used by the agents. In particular, artifacts are not autonomous but might show reactive behavior.
3. A *workspace* basically defines an environment and contains agents and artifacts.

Given proper descriptions and documented interfaces for artifacts, programmers can create agents which interact in an open MAS that is realized based on the A&A meta-model.

With respect to the concept of storage media considered in this work, we are mainly interested in artifacts. Artifacts can be used by agents in various ways. In an early work, Omicini et al. [ORV$^+$04] introduce *coordination artifacts*. This type of artifacts provides an organizational means for indirect communication so as to enable coordination through the environment. *Cognitive artifacts* [ORV08] [PR09] are supposed to be used in the cognitive process of the agents. Thus, they influence the design of agents since they have to be aware off e.g. the existence and functionality of artifacts. In [PR09], Piunti et al. consider cognitive artifacts in the form of *log artifacts* in a simple simulated robotic room cleaning task. The logs contain information about recent cleaning operations. These information can be taken into account by the agents when they have to decide whether they should enter and clean a room.

The A&A meta-model, resp. the artifacts, are also used in and have inspired different models. For instance [KBHR08] uses artifacts to incorporate organizational infrastructures in MAS, e.g. by using artifacts to model functions of organizations on the one hand, and agents as (organizational) decision makers on the other hand. In a recent work, Ricci et al. [RPV11] show how artifacts can be used from a software engineering point of view by introducing the notion of *environment programming*. In their framework, for instance, artifacts might realize shared data objects or resources and communication or coordination services. By doing so, the re-usability of MAS-based software can be increased.

The concept of storage media is closely related to artifacts. In fact, a storage media can be considered to be an implementation of an artifact. In general, however, artifacts can realize arbitrarily complex functionalities. Our storage media instead are supposed to “store” and provide knowledge rather than realizing other complex behaviors. Further differences result from the focus of the A&A meta-model, which is MAS-based software development. Artifacts in the A&A meta-model can be part of various workspaces at the same time and mainly are considered to be “programs”. In contrast, our storage media are supposed to be located in a (simulated) physical environment where they should support robots (agents) by providing the ability to store knowledge outside of the agents.

6.2 Strategic Positioning

A first application of the storage media concept is found in [KKB10a]. There, we used the media in our IAPP variant to optimize the DISTANCE objective, i.e. we want the agents to select the nearest target depending on their position. We briefly investigated the influence of randomly and regularly positioned storage media. In the latter case, media are aligned on a regular grid such that, besides some boundary agents, all agents can interact with at least one medium independent from their current location in the environment. Besides storing "suitable" knowledge on the "right" storage media[1], we identified that media can make a remarkable contribution to the success of the system as a whole, given that all agents are able to interact with the media.

Accordingly, the strategic positioning of storage media in an environment is one important factor for a successful application of the concept. Before we next sketch one approach for the positioning of media in our IAPP variant, note that what is considered to be a good strategic positioning clearly depends on the considered problem domains and settings.

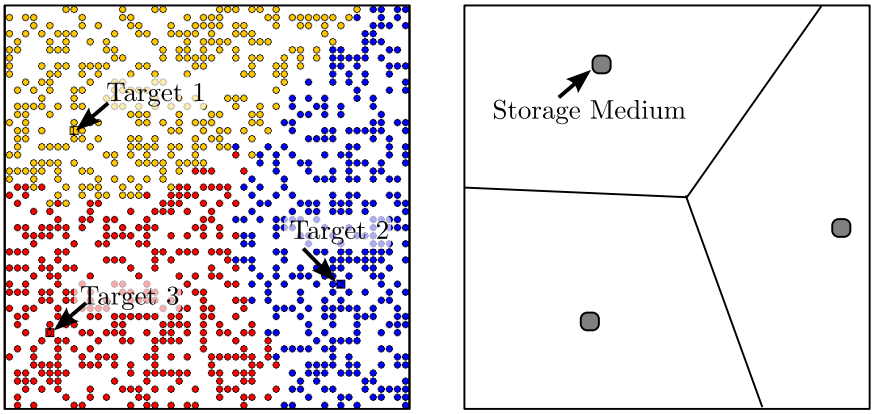


Fig. 6.2 Strategic positioning of storage media to reflect target regions.

In Chapter 12, we will identify that high quality solutions for our IAPP variant usually contain regions. There is one region for each target. If an agent selects the target with respect to the region it is located in, we are likely to observe high quality solutions. Figure 6.2 illustrates these regions in a simple example. With this property in mind, it seems unnecessary to place a large number of media on a regular grid as mentioned above. Instead, it would be sufficient to just place one media per region such that, whenever the agent is in a particular region, the agent will interact with that particular medium. On the right side of Fig. 6.2, we sketched such a beneficial placement. It corresponds to a Voronoi diagram that uses the media as generating sites. Hence, whenever an agent asks its nearest medium for advice, it will obtain the best action with respect to its location. Note that the media positions, which generate the Voronoi diagram, depend on the targets' positions. However, these positions

[1] In the setting investigated in [KKB10a], knowledge about the nearest target to a medium should be stored on every medium. Due to the spacial relation, the "right" medium for a knowledge item on a target T then is located closer to T than to any other target.

usually differ because otherwise we would optimize only the DISTANCE objective while the DISTRIBUTION could become arbitrarily bad.

In general, however, those target regions can have arbitrary shapes or might not even be connected (cf. Prop. 8 on page 192). Also, agents usually have limited communication radii and thus may be unable to reach any medium at all. Accordingly, more sophisticated strategies for placing the media are required. It would be beneficial, if the agents learn where to best place the media while they learn to solve the partitioning problem.

In this work, we do not intend to provide such a placement approach in every detail, as this is an interesting research question on its own. Instead, we just briefly sketch a promising idea based on a work of Baumann and Kleine Büning [BKB11]. Considering an agent position as a state, one can reinterpret these target regions as a set of related states that all require the same action, namely selecting the same target. In [BKB11], the authors propose an approach that identifies such similarities between states and automatically generates a state-space abstraction while learning to solve a simple grid-based shortest path problem. The general idea of that approach, called GNG-Q, is to combine Q-Learning with growing neural gas (GNG, [Fri95]). GNG-Q uses units called neurons as representative for a set of similar states and learns Q-values for these neurons. New neurons are added if the learning process indicates a too coarse abstraction, e.g. when policy changes occur too frequently. Neurons might also be deleted, if they become unnecessary, due to changes in the topology. In the end, the approach can come up with regions like those shown in Fig. 6.2, whereas the learned neurons' positions would be equal to the storage media positions. Since the approach autonomously learns how to aggregate states, the resulting regions can also have arbitrary geometric forms. Hence, placing storage media at the learned neuron positions seems to be a good approach.

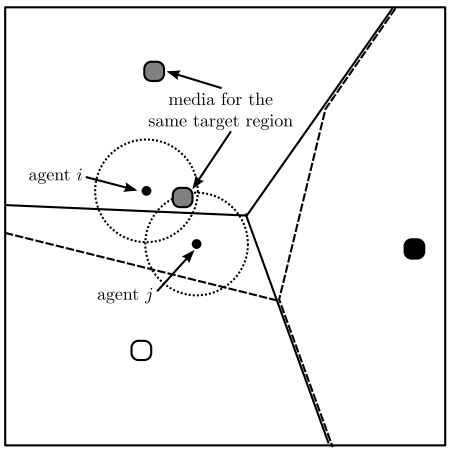


Fig. 6.3 Agent i is unable to communicate with the single media that represents the target region for the agent's location. Adding additional media in the region, however, also influences the boundaries of the other target regions as indicated by the dashed lines. Adding the shown media, for instance, would mislead agent j. Accordingly, it needs to be elaborated how to position media under limited communication radii and under more complex geometric shapes. Furthermore, additional media are required within the regions.

Up to now, GNG-Q is applied only "within" an agent, i.e. to create a representation of the actual environment for the learning algorithm. Hence, it needs to be adapted to support multiple agents which at the same time have to learn and to agree on a common state-space abstraction. Then, external storage media have to be placed or moved according to the learned neuron positions. Also, as indicated in Fig. 6.3 and discussed in the figure's caption, limited agent communication radii have to be taken into account. Besides their positioning, the number of media required becomes an issue, too.

Finally, we believe that a self-adaptive learning approach, which is based on the above sketched idea of using neurons generated by GNG-Q as basis for determining storage media positions, could lead to good results in our IAPP variant. The realization of such an approach, however, is left for future work.

6.3 Influence of Obstacles

In this section, we will briefly discuss the influence of obstacles on the communication of agents. We will show in how far a single storage medium can be used to enable communication between two agents on different sides of an obstacle. Figures 6.4 and 6.5 visualize the considered setting schematically.

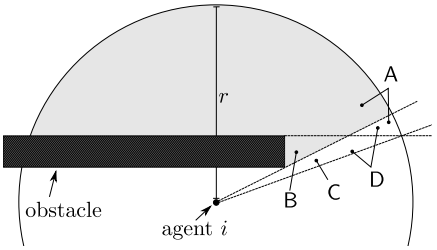


Fig. 6.4 Area that is shadowed by the obstacle.

The gray shade in Fig. 6.4 highlights the area that is said to be *shadowed* by the obstacle for a particular agent i with communication radius r. The term *shadowed* means that all agents which are located in the gray area cannot be reached by agent i due to the obstacle. The area of the shadowed region can be calculated by decomposing it into smaller regions as indicated in the figure. Based on these identifiers, the shadowed area hence is given by $\mathrm{A}\,(\mathsf{A}) + \mathrm{A}\,(\mathsf{B}) - (\mathrm{A}\,(\mathsf{D}) - \mathrm{A}\,(\mathsf{C}))$, where $\mathrm{A}\,(X)$ denotes the area of region X.

Given that an agent is located below such an obstacle, the question is where to place a single storage medium such that it can help to reduce the influence of the obstacle on the agent's communication to a minimum. Therefore, it is assumed that the medium acts as a mediator, i.e. agents indirectly communicate via the media. Due to limited communication radii of the agent and the medium and because the goal is to enable agent i to communicate with as many agents as possible that are located in the shadowed area, only the (light) gray areas (Z and U) in Fig. 6.5 describe possible locations for a medium. In that figure, M denotes the position of the medium and the dashed circle describes the medium's communication range.

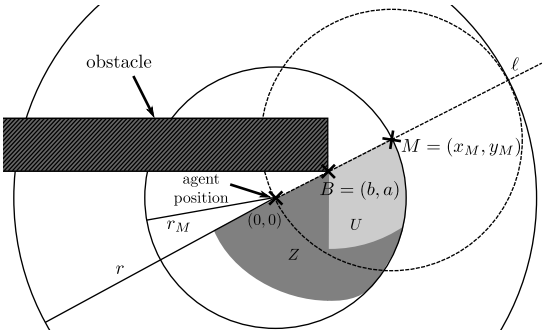


Fig. 6.5 Agent and obstacle alignment setting considered in this section.

Under some assumptions, e.g. on the positions, obstacle dimensions, or communication radii, we were able to prove that the best position (x_M, y_M) for a medium M is given by the intersection of the visibility line ℓ and the circle described by the communication radius of the medium centered at the agents position (cf. Fig. 6.5). Formally, this position can be calculated by the agent using

$$(x_{\text{opt.}}, y_{\text{opt.}}) = \left(\sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}}, \frac{a}{b} \sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}} \right),$$

where the agent is assumed to be placed at position $(0, 0)$ and only needs to know the position (b, a) of the bottom right corner of the obstacle, and the medium's communication radius r_M. Given that medium M is placed at this position, the *coverage* through M is maximized, i.e. the area that is accessible by indirect communication via M compared to the area shadowed by the obstacle is maximized.

Due to the geometric nature of this placement problem, the corresponding proof involves several different cases that can be reduced to basic geometric arguments. We skip the details on this proof here and refer the reader to Appendix A. There, a detailed description on this placement problem can be found.

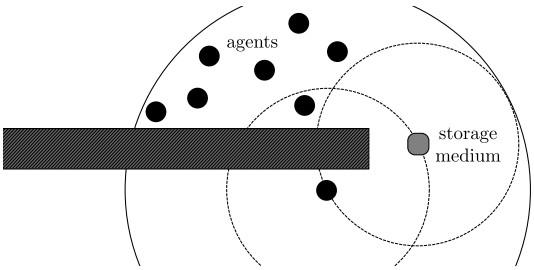


Fig. 6.6 Setting in which no agent benefits from the storage medium.

Given the optimal position for the maximum coverage, it is easy to see that storage media cannot totally undo the shadow effect introduced by obstacles. Figure 6.6 depicts a simple worst case, in which the medium cannot reach any of the shadowed agents although it is placed at the optimal position. Accordingly, the benefits from using storage media as means for indirect communication in settings with obstacles

can be quite small. This, however, does not subvert the general benefits of storage media as listed in Sect. 6.1, but it should be kept in mind when designing storage media-based algorithms in scenarios with obstacles.

6.4 Greedy Placement Strategy

In this section, we will present a simple greedy storage media placement strategy for settings with obstacles that aims at maximizing the number of indirectly reachable agents.

Algorithm 1 presents the *Greedy-Placement-Strategy (GPS)*, which is executed by each agent i. Its idea is to systematically evaluate all possible positions for a new storage medium, i.e. all positions p with $\boldsymbol{\delta}_{\mathsf{so}}(\mathsf{pos}(i), p) \leq \min\{r_i, r_M\}$, where r_i (r_M) is the communication radius of the agent (storage medium), and $\boldsymbol{\delta}_{\mathsf{so}}$ is a function that calculates the sensor distance between two positions w.r.t. obstacles (see Appendix A). Each position is assigned a value that indicates the number of additionally reachable agents by indirect communication through the storage medium. The media is then placed on the position with the highest value. If several positions are rated with the same highest value, the first found position will be selected.

Algorithm 1 Executed by each agent i located at $\mathsf{pos}(i)$.

procedure GREEDY-PLACEMENT-STRATEGY
  $p_{\text{best}} \leftarrow$ null ▷ best position
  $g_{\text{best}} \leftarrow 0$ ▷ best gain
  $\mathbb{P} = \{\}$ ▷ set of possible medium positions
  $D \leftarrow$ set of directly reachable agents in communication radius r_i
  $r \leftarrow \min\{r_i, r_M\}$
  $\mathbb{P}_{\text{tmp}} \leftarrow$ positions in radius r ▷ ordered from top left to bottom right
  for all $i \in \{1, 2, \ldots, |\mathbb{P}_{\text{tmp}}|\}$ **do** ▷ Determine possible medium positions
    if $\boldsymbol{\delta}_{\mathsf{so}}(\mathsf{pos}(i), \mathbb{P}_{\text{tmp}}[i]) \leq r$ **then**
      $\mathbb{P} = \mathbb{P} \cup \{\mathbb{P}_{\text{tmp}}[i]\}$
  for all $i \in \{1, 2, \ldots, |\mathbb{P}|\}$ **do** ▷ Find best placement position
    value $\leftarrow$ EVALUATE($\mathbb{P}[i], D$)
    if value $> g_{best}$ **then**
      $g_{\text{best}} \leftarrow$ value
      $p_{\text{best}} \leftarrow \mathbb{P}[i]$
  if $p_{best} \neq$ null and p_{best} is free **then**
    place storage medium on p_{best}
  else
    refrain from placing storage medium

procedure EVALUATE(position p, set of directly reachable agents D)
  if p is free **then**
    place storage medium M on position p
    $C_M \leftarrow$ set of agents that can interact with M within radius r_M
    $C_i \leftarrow C_M \setminus D$ ▷ indirectly reachable agents
    remove M
    return $|C_i|$
  else
    return $-\infty$

6.4.1 Analysis

For the analysis, we assume time $\mathcal{O}(1)$ for placing a medium and for determining the number of agents reachable within a distinct radius.

First, consider the EVALUATE subroutine. It requires time $\mathcal{O}(|D|)$, because each directly reachable agent in D has to be considered to create the set of indirectly reachable additional agents.

Next, consider the main procedure. Creating the set of possible medium positions $\mathbb{P}$ requires time $\mathcal{O}(|\mathbb{P}_{\text{tmp}}|)$, as the distance to each position must be calculated. (Note that this assumes constant time for determining the distance.) Now, since each position in $\mathbb{P}$ has to be evaluated using EVALUATE, time $\mathcal{O}(|\mathbb{P}_{\text{tmp}}||D|)$ is required to find the best position, as $|\mathbb{P}|$ is bounded from above by $|\mathbb{P}_{\text{tmp}}|$. The remaining steps can be performed in constant time.

Thus, the overall runtime of the Greedy-Placement-Strategy is in $\mathcal{O}(|\mathbb{P}_{\text{tmp}}||D|)$. The cardinality of both sets depends on the radii of the agent and the medium. With $r_{\max} = \max\{r_i, r_M\}$ as maximum radius, a bounding box with $4r_{\max}^2$ positions is an upper bound for the positions/agents in both sets. Thus, GPS requires at most $\mathcal{O}(r_{\max}^4)$ time.

Since the algorithm evaluates all possible placement positions and has to store the set of directly reachable agents D, the space required for this algorithm depends also on the used radii and is in $\mathcal{O}(|D| + |\mathbb{P}_{\text{tmp}}|)$, i.e. in terms of the maximum radius in $\mathcal{O}(r_{\max}^2)$.

6.4.2 Number of Media

Having analyzed the runtime and space requirements, we now consider the number of media placed by the strategy.

In principle, GPS is able to find optimal solutions as shown in Fig. 6.7. There, no agent is able to communicate with any other agent since either their distance is larger than the communication radius (e.g. agents 0 and 2), or the line of sight is intersected by a sensor obstacle (e.g. agents 0 and 1). As the figure shows, indirect communication via a storage medium can be realized. In this particular example, this is the only valid placement position for both agents since it is the only position where a medium can be accessed by both agents. Independent from synchronization issues, both agents in the GP-Strategy will find that position, but only one is able to place a medium. The other agent will observe an occupied position and will thus refrain from placing a medium. The optimality of this solution follows because agents closer to different obstacles can not be connected by a single medium.

Next, we consider a worst case scenario shown in Fig. 6.8. Therein, the number of storage media placed by GPS is n for n agents. The construction of this worst case scenario for an arbitrary number of agents $n \geq 2$ is as follows. To restrict the placement of media to the area below the vertical sensor obstacles, a long horizontal obstacle is placed on top of the regularly located vertical obstacles. Hence, $\lceil \frac{n}{2} \rceil$ vertical obstacles are placed at regular distance to each other. From left to right, each obstacles is one cell longer than the preceding one. The initial length as well as the distance between the vertical obstacles depend on the medium's interaction radius. In the next step, we sort the agents by IDs ascending and place one agent after the

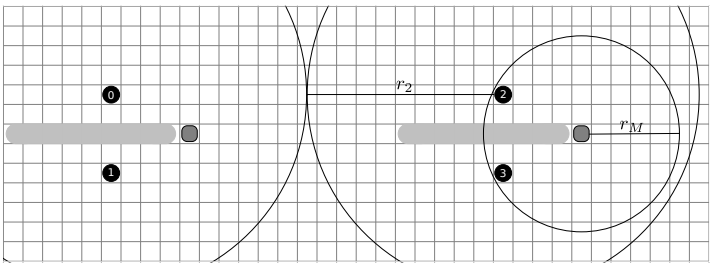


Fig. 6.7 Best case. The large circles indicate the agents' communication radii radii; the small circle illustrates the medium's interaction radius.

other; always two agents per indentation in a row. The first agent with ID 0 is placed such that the first best placement position, when traversing possible positions in GPS from top left to bottom right, is at the end position of the right vertical obstacle. Agent 1 is placed one cell below agent 0. The remaining $\lceil \frac{n}{2} \rceil - 1$ indentations are filled with two agents in the same manner, starting at the y-coordinate of the second agent placed in the previous indentation. Therefore, we obtain the stair-like structure shown in the figure. Note that for an odd number of agents, the last indentation will only contain one agent. However, this still leads to n placed media in the worst case. A similar worst case for $n = 2$ can be created by placing the agents on the positions of agent 1 and 3 in Fig. 6.8.

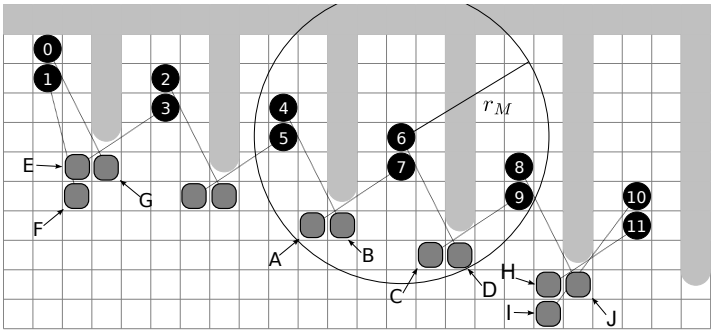


Fig. 6.8 Worst case. The lines connect each medium to its owner. The circle with radius r_m centered at the position of agent 6 visualizes the media interaction radius.

The worst case of placing n media occurs, if the agents place their respective medium one after the other sorted by ascending agent IDs. First, let us consider the inner indentations 2 to $\lceil \frac{n}{2} \rceil - 1$, using agents 6 and 7 as examples. The lines in Fig. 6.8 connect each storage medium with its owner, i.e. the agent that placed it. As indicated by these connections, the upper agent 6 will place its medium at the end of its right sensor obstacle at position D. Position A is not considered, as the agent would be unable to communicate with a medium placed there. With a gain of 2, position B would offer the same as D. However, B is already occupied by agent 4's medium. Finally, position C is not selected due to a lower gain of 1. Agent 7, will place its medium at position A, as it offers a gain of 2, C only a gain of 1, and B and D are already occupied. Thus, both agents will place a medium.

Next, we have to consider the remaining two indentations, i.e. the first and the last one. Agents 0 and 1 have no agents on their left side. Since the gain on position E for agent 0 is 1, but 2 on G, agent 0 will place its medium at position G. Agent 1's gain at position E is 1, and since G is already occupied and F offers a gain of 2, agent 1 will place its medium on F. In the last indentation, agent 10's highest gain is possible at position I as a medium on H would not be reachable and J is already occupied by agent 8's medium. Finally, agent 11's medium is placed on H as this is the first time when the highest possible gain of 2 is reached.

The worst case for two agents as described above is easily comprehensible, and for $n = 1$ agent no medium results in a positive gain. It follows that we can always construct a worst case where n media are placed if n agents are involved.

6.5 Summary

In this chapter, we introduced the concept of storage media in multiagent systems and presented related work. We discussed opportunities and challenges that come along with this concept. In particular, we presented some insights into the strategic positioning of media in the context of the IAPP. Furthermore, we investigated the influence of simple rectangular obstacles to the application of storage media as mediators for indirect communication. We proved an optimal positioning of a medium that ensures the maximum coverage from the perspective of a single agent. Finally, we proposed and analyzed a simple greedy approach that allows each agent to place exactly one storage medium in environments with obstacles. However, this greedy strategy was shown to place up to one medium per agent in the worst case.

Part III

Algorithms

Chapter 7

Reinforcement Learning with Noisy Rewards

In this chapter, we investigate the influence of randomly noised rewards on learning in deterministic environments with one agent. Although we are mainly interested in multiagent reinforcement learning (MARL), the insights provided by this chapter for single agent settings with noise will become helpful when we will consider our novel MARL approach (cf. Ch. 8) under noise.

We define *noise robust rewards* and *noisy Markov Decision Processes* in Sect. 7.1 and prove that an optimal policy obtained under noise is also optimal for the underlying original noise-free process. Section 7.2 shows that deterministic Q-Learning is unable to converge to such an optimal policy under noise—even if rewards are noise robust. Beside these contributions, we give the main results of this chapter in Sect. 7.3. We adopt an optimistic update rule from MARL and prove that it requires one assumption on the noise distribution in order to ensure convergence to an optimal policy under noise. By counterexample, we also show that noise robust rewards cannot replace the assumption on the noise distribution. Finally, in Sect. 7.4, we discuss the obtained results and relate them to learning in non-deterministic environments.

7.1 Basic Definitions

The model that we will use for learning with noisy rewards is presented in Def. 22.

Definition 22 (Noisy Markov Decision Process). Let $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$ be an ordinary MDP. A *noisy Markov Decision Process* is defined by a tuple $\widetilde{\mathrm{M}} = \langle \mathrm{M}, \widetilde{\rho}, \mathrm{ND}, \theta \rangle$. The agent in $\widetilde{\mathrm{M}}$ does not observe the original reward according to $\rho : \mathcal{S} \times A \to \mathbb{R}$, but a noisy reward given by $\widetilde{\rho} : \mathcal{S} \times A \to \mathbb{R}$ having $\widetilde{\rho}(s,a) = \rho(s,a) + \mu_{\mathrm{ND}}^{\theta}(\rho(s,a)), \forall s \in \mathcal{S}, a \in A$. Here, $\mu_{\mathrm{ND}}^{\theta} : \mathbb{R} \to \mathbb{R}$ denotes a noise generating function such that $\mu_{\mathrm{ND}}^{\theta}(v)$ returns a random value from $[-\theta v, \theta v]$ drawn according to a random distribution ND, where $\theta \in [0, 1)$ is a noise rate.

From the above definition, it is clear that a noisy MDP basically is an ordinary MDP which uses rewards that are augmented with an additive random noise. The

interval of the noise is determined by a percentage of the respective clear reward. Due to random noise, a noisy Markov Decision Process is said to be *semi-deterministic*, if the state transition function δ is deterministic; it is *nondeterministic* if δ is probabilistic. Let us concentrate on the *semi-deterministic* case hereafter. Recall, that throughout this work, we assume finite state and actions sets and consider only bounded rewards.

The consideration of randomly noised rewards requires a closer look to what is considered to be an optimal solution. Let all rewards be noised such that the respective maximal positive noise value is added to the clear reward. Given such rewards, we define the optimal Q-values $Q^{\star}_{\widetilde{\mathrm{M}}}$ for $\widetilde{\mathrm{M}}$ according to the Bellman equations for all state action pairs (s, a) as

$$\begin{aligned} Q^{\star}_{\widetilde{\mathrm{M}}}(s,a) &= [\rho(s,a) + \theta\rho(s,a)] + \gamma \max_{a' \in A} Q^{\star}_{\widetilde{\mathrm{M}}}(\delta(s,a), a') \\ &= \widetilde{\rho}^{\max}(s,a) + \gamma \max_{a' \in A} Q^{\star}_{\widetilde{\mathrm{M}}}(s', a') \end{aligned} \tag{7.1}$$

For brevity, we will use $\widetilde{\rho}^{\max}(s,a) = \rho(s,a) + \theta\rho(s,a)$ to denote the maximal positive noised reward value for a state-action pair, and we define $s' = \delta(s,a)$ as the state resulting from executing action a in state s. If the context is clear, we skip the index and write $Q^{\star}$ if we refer to $Q^{\star}_{\widetilde{\mathrm{M}}}$. Based on these equations, an optimal policy corresponds to a greedy policy.

Note that, according to this definition of $Q^{\star}_{\widetilde{\mathrm{M}}}$, an optimal policy for $\widetilde{\mathrm{M}}$ is also optimal for the underlying MDP M and vice versa. This follows from Lemma 1, since $\widetilde{\rho}^{\max}(s,a) = \rho(s,a) + \theta\rho(s,a) = (1+\theta)\rho(s,a)$ and $(1+\theta)$ is a constant factor.

Lemma 1. *An optimal policy $\pi^{\star}$ for a Markov Decision Process $\mathrm{M} = \langle \mathcal{S}, A, \delta, \rho, \gamma \rangle$ is optimal for another MDP $\mathrm{M}' = \langle \mathcal{S}, A, \delta, \rho', \gamma \rangle$, having*

$$\rho'(s,a) = \Phi\rho(s,a)$$

for all state action pairs (s,a) and a constant factor $\Phi > 0$.

Proof. Consider an optimal policy $\pi^{\star}_{\mathrm{M}}$ for MDP M, i.e.

$$\begin{aligned} \pi^{\star}_{\mathrm{M}} &= \arg\max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \\ &= \arg\max_{\pi} E\left[\sum_{k=0}^{\infty} \gamma^k \rho_k \mid \pi\right], \forall s \in \mathcal{S} \end{aligned}$$

Since the policy selected via the $\arg\max$ operator remains the same if the values of all policies are multiplied by the same constant Φ, we get

$$\begin{aligned} \pi^{\star}_{\mathrm{M}} &= \arg\max_{\pi} \Phi E\left[\sum_{k=0}^{\infty} \gamma^k \rho_k \mid \pi\right], \forall s \in \mathcal{S} \\ &= \arg\max_{\pi} E\left[\Phi \sum_{k=0}^{\infty} \gamma^k \rho_k \mid \pi\right], \forall s \in \mathcal{S} \\ &= \arg\max_{\pi} E\left[\sum_{k=0}^{\infty} \gamma^k \Phi\rho_k \mid \pi\right], \forall s \in \mathcal{S} \end{aligned}$$

$$= \arg\max_{\pi} E\left[\sum_{k=0}^{\infty} \gamma^k \rho'_k \mid \pi\right], \forall s \in \mathcal{S}$$
$$= \pi^{\star}_{\mathrm{M}'}$$

Clearly, this result also holds for deterministic MDPs, since the value function for following a policy π starting in s is determined by $V^{\pi}(s) = \sum_{k=0}^{\infty} \gamma^k \rho_k$. □

Accordingly, an algorithm that learns an optimal policy for $\widetilde{\mathrm{M}}$ under noisy rewards also finds an optimal policy for the underlying noise-free MDP M.

An interesting property of rewards concerns their relations with respect to a given noise rate:

Definition 23 (θ-Noise Robust Rewards). Two clear reward values u, v are said to be robust under noise rate $\theta \in [0, 1)$ if

$$[(1-\theta)u, (1+\theta)u] \cap [(1-\theta)v, (1+\theta)v] = \emptyset. \tag{7.2}$$

In this case, we say that u, v are *θ-noise robust*. Given a reward function $\rho : \mathcal{S} \times A \to \mathbb{R}$, the function is said to be *θ-noise robust*, if for each state $s \in \mathcal{S}$ condition (7.2) holds pairwise for all distinct reward values that can result from executing an action in state s.

Note that we will repeatedly return to noise robust rewards in the following two chapters, as they *seem* to establish an interesting property that probably could help to ease learning under noised rewards.

Let us now proceed in the following section by considering the influence of noise to deterministic Q-Learning.

7.2 Deterministic Q-Learning with Noisy Rewards

The way how we create noisy rewards, i.e. by adding a noise value to the true reward, technically corresponds to the approach followed in reward shaping as explained in detail in Sect. 3.4. However, the aim of reward shaping differs from our goal. In reward shaping, the idea is to guide the learning process of an agent by adding additional rewards to the actual reward, i.e. using $\rho' = \rho + F$ as reward function, where F denotes the shaping reward. The shaping reward is supposed to improve the agent's speed of learning and to guide its learning. Contrary to this, our goal is to enable agents to learn from noisy rewards. That is, rewards are observed only with a certain precision through possibly erroneous sensors. In particular, the noisy rewards do neither guide nor speed up the agent's learning but instead exacerbate it.

Ng et al. theoretically analyzed reward shaping ([NHR99], see also Sect. 3.4). They showed that only potential-based reward shaping functions, i.e. functions in the form $F(s, a, s') = \gamma\phi(s') - \phi(s)$, where $\phi : \mathcal{S} \to \mathbb{R}$ is a potential function and γ a discount factor, are guaranteed to preserve optimal policies in general MDPs. Ng et al. also pointed out that additional knowledge on the state transition and/or reward function would be required in order to allow other shaping functions to preserve optimal policies.

Since reward shaping and noisy rewards both modify the original reward in the same way, i.e. by adding a value to the true reward, the results of Ng et al. also hold

in our noisy setting. Accordingly, it follows that noisy rewards prohibit the agent to learn an optimal policy, as the noise generating function μ_{ND}^{θ} (cf. Def. 22) returns a random value, and thus does not satisfy the aforementioned condition of being a potential-based reward shaping function.

The remaining question is, whether noise robust rewards are a game-changer, i.e. if the usage of noise robust rewards in deterministic Q-Learning leads to convergence towards optimal policies. Unfortunately, this is not the case as shown in the following simple example.

Example 1. *Consider a simple noisy MDP as shown in Fig. 7.1 using the discount factor $\gamma = 0.9$ and the noise rate $\theta = 0.3$. At any time, the agent either selects and executes action* left *or* right*. The agent should learn that action* right *is best, as it results in the largest immediate reward of 2 compared to 1 for the* left *action. Note that these rewards are θ-noise robust. Furthermore, assume that the agent learns using deterministic Q-Learning (cf. Sect. 3.2.3), i.e. it updates its Q-values according to $\widehat{Q}(s,a) = \rho(s,a) + \gamma \max_{a' \in A} \widehat{Q}(s',a')$.*

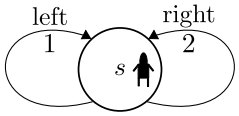


Fig. 7.1 Simple learning task.

Given noise rate $\theta = 0.3$, the maximal positive noised reward for state s and going right *becomes $\widetilde{\rho}^{\max}(s, \text{right}) = \rho(s, \text{right}) + \theta\rho(s, \text{right}) = 2 + 0.3 \cdot 2 = 2.6$. The maximal positive noised reward for going* left *hence is $\widetilde{\rho}^{\max}(s, \text{left}) = 1 + 0.3 \cdot 1 = 1.3$. Based on these values and using techniques like, e.g., policy or value iteration as described in Sect. 3.2.2, one can solve the Bellman optimality equations:*

$$Q(s, \text{right}) = \widetilde{\rho}^{\max}(s, \text{right}) + \gamma \max_{a'} Q(s, a')$$
$$Q(s, \text{left}) = \widetilde{\rho}^{\max}(s, \text{left}) + \gamma \max_{a'} Q(s, a')$$

The resulting optimal Q-values for state s are $Q^{\star}(s, \text{right}) = 26$ and $Q^{\star}(s, \text{left}) = 24.7$, as they form the unique solutions to the Bellman optimality equations:

$$Q(s, \text{right}) = \widetilde{\rho}^{\max}(s, \text{right}) + \gamma \max_{a'} Q(s, a') = 2.6 + 0.9 \max\{26, 24.7\} = 26$$
$$Q(s, \text{left}) = \widetilde{\rho}^{\max}(s, \text{left}) + \gamma \max_{a'} Q(s, a') = 1.3 + 0.9 \max\{26, 24.7\} = 24.7$$

Assume that we interchangeably execute action left *and* right*, and that we obtain the maximal noised rewards 1.3, resp. 2.6. Then, the Q-estimates $\widehat{Q}$ according to the used update rule will have the same values as the optimal ones after 510 iterations*[1]*. Thereafter, assume that the agent goes* right *and observes the minimal possible noised reward of $2 - 0.3 * 2 = 1.4$. Then $\widehat{Q}(s, \text{right}) = 1.4 + 0.9 \max\{26, 24.7\} = 24.8$. Going* right *and observing the reward 1.4 again, we obtain $\widehat{Q}(s, \text{right}) = 1.4 + 0.9 \max\{24.8, 24.7\} = 23.72$. Clearly, at this point in time the greedy action is going* left*, and thus not optimal.*

[1] Feel free to use your favorite spreadsheet program to verify this statement.

Based on this plot, it is easy to see that one can construct alternating $\widehat{Q}$-values where once going right *and once going* left *is optimal. Accordingly, the learning process will not stabilize in the sense that it converges to the optimal action.*

The key to that example is that the $\widehat{Q}$-values lose the property of being monotonically increasing if noisy rewards are used. Since the exemplary rewards are noise robust for the used noise rate $\theta = 0.3$, as $[0.7, 1.3] \cap [1.4, 2.6] = \emptyset$, it also follows that noise robust rewards are not a game-changer with respect to the results of Ng et al. as discussed above.

In the next section, we will slightly adjust the update rule and show under which conditions the resulting algorithm then converges under noisy rewards.

7.3 Q-Learning with Noisy Rewards and Optimistic Updates

The update rule for deterministic Q-Learning is given by $\widehat{Q}(s,a) = \rho(s,a) + \gamma \max_{a' \in A} \widehat{Q}(s',a')$ (cf. Sect. 3.2.3 or [Mit97, (13.7)]). As we have argued in the preceding section, this update rule is unable to deal with noisy rewards. In order to deal with such rewards, we thus propose to adopt the *optimistic assumption* as stated in (7.3).

$$\widehat{Q}(s,a) = \max \left\{ \widehat{Q}(s,a), \rho(s,a) + \gamma \max_{a' \in A} \widehat{Q}(s',a') \right\} \tag{7.3}$$

The optimistic assumption was originally proposed by Lauer and Riedmiller [LR00] in the context of multiagent reinforcement learning in cooperative deterministic games. In this setting, its idea is to take care of capturing the maximum value that is obtained by a single agent for executing its actions as part of a joint action. Note that the outcome of an individual agent's action might be different depending on the actions of the other agents. Thus, from an individual agent's perspective without knowledge about other agents, the obtained rewards seem to be stochastic. Since this basically corresponds to noisy rewards, we will show that the provided update rule (7.3) in a noisy setting can

1. not ensure convergence to optimal policies in noisy MDPs, neither in general (see Example 2) nor in settings with noise robust rewards (see Example 3), but
2. is guaranteed to find an optimal policy for a noisy MDP—and by Lemma 1 also for the underlying original MDP—if the maximum positive noised rewards occur with a non-zero probability (Lemma 2, Theorem 7, Corollary 1).

Before we continue to show these results, we want to be clear about the assumptions made throughout this section. They are summarized as follows:

Assumptions 1 :

- *We consider a semi-deterministic noisy Markov Decision Process.*
- *The original rewards are bounded non-negative values.*
- *All Q-estimates $\widehat{Q}$ are initialized to zero.*
- *Each state-action pair is visited infinitely often with a non-zero probability.*
- *The noise distribution ND is stationary, i.e. the probability of observing a certain value does not change over time.*

To see that the adoption of the optimistic assumption alone cannot ensure convergence to an optimal policy, have a look at the following simple example:

Example 2. *Consider the settings shown on the left hand side of Fig. 7.2, where an agent should learn to move to the* right. *This scenario is the same setting as described in detail in the preceding Example 1. In contrast to the first example, we set the noise rate to* $\theta = 0.9$. *Now, assume that the reward for going* right *always occurs with maximum negative noise, i.e.* $\widetilde{\rho}(s, \text{right}) = 2 - \theta \cdot 2 = 0.2$. *Also assume the opposite for going* left, *i.e. always add maximum positive noise,* $\widetilde{\rho}(s, \text{left}) = 1 + \theta \cdot 1 = 1.9$.

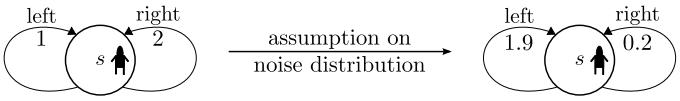


Fig. 7.2 Simple learning task, where the assumed noise distribution changes the optimal action.

Under these assumptions on the noise distribution, the noise clearly changes the optimal action, as the suboptimal action left *always leads to a higher reward than going* right. *Accordingly, the optimistic update rule cannot learn the optimal policy for the underlying original MDP.*

Although this example shows that the optimistic update rule in general cannot lead to convergence to an optimal policy for the original MDP under noise, it will do so with high probability under less artificial noise distributions. In particular, it has to be ensured that the maximum positive noise value will be experienced with a non-zero probability, as it is the case, e.g., if the noise is distributed according to a uniform or normal distribution. To prove this, we first show the following lemma:

Lemma 2. *Under assumptions A1, consider Q-Learning with update rule (7.3) and an arbitrary state-action pair* (s, a). *Let* t *denote a time step for which the maximum positive noised reward* $\widetilde{\rho}^{\max}(s, a) = \rho(s, a) + \theta\rho(s, a)$ *is observed. Then, the Q-estimates are updated according to*

$$\widehat{Q}_{t+1}(s, a) = \widetilde{\rho}^{\max}(s, a) + \gamma \max_{a' \in A} \widehat{Q}_t(s', a'). \tag{7.4}$$

Proof. Before we prove the lemma note that the Q-estimations $\widehat{Q}$ are monotonically increasing. This follows from update rule (7.3) and because rewards are non-negative and all $\widehat{Q}_0(s, a)$ are initially 0. Without loss of generality, we consider one state-action pair (s, a) and prove (7.4) by contradiction.

According to update rule (7.3), the estimate for (s, a) after iteration t is updated by

$$\widehat{Q}_{t+1}(s, a) = \max \left\{ \widehat{Q}_t(s, a), \widetilde{\rho}^{\max}(s, a) + \gamma \max_{a' \in A} \widehat{Q}_t(s', a') \right\}. \tag{7.5}$$

Assume the maximum argument in (7.5) evaluates to $\widehat{Q}_t(s, a)$, i.e.

$$\widehat{Q}_t(s, a) \geq \widetilde{\rho}^{\max}(s, a) + \gamma \max_{a' \in A} \widehat{Q}_t(s', a'). \tag{7.6}$$

Let $t' < t$ denote the iteration after which the last update to (s, a) occurred for a non-maximal positive noised reward, i.e. when

$$\rho_{t'}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_{t'}(s',a') = \widehat{Q}_{t'+1}(s,a) = \widehat{Q}_{t'+2}(s,a) = \ldots = \widehat{Q}_t(s,a).$$

Accordingly, (7.6) can be rewritten as follows

$$\rho_{t'}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_{t'}(s',a') \geq \widetilde{\rho}^{\max}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_t(s',a'). \tag{7.7}$$

By construction, $\rho_{t'}(s,a) < \widetilde{\rho}^{\max}(s,a)$ holds. Further, from the monotonicity of the estimates it follows that

$$\gamma \max_{a' \in A} \widehat{Q}_{t'}(s',a') \leq \gamma \max_{a' \in A} \widehat{Q}_t(s',a').$$

Accordingly, the right hand side of (7.7) is larger than the left hand side. This contradicts the assumption that the maximum argument in the update rule (7.5) evaluates to $\widehat{Q}_t(s,a)$. Thus,

$$\begin{aligned} \widehat{Q}_{t+1}(s,a) &= \max \left\{ \widehat{Q}_t(s,a), \widetilde{\rho}^{\max}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_t(s',a') \right\} \\ &= \widetilde{\rho}^{\max}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_t(s',a') \end{aligned}$$

and the lemma follows. □

Using this result, we now are able to prove the aforementioned statement:

Theorem 7. *Under assumptions A1, let $\widetilde{\mathrm{M}} = \langle \mathrm{M}, \widetilde{\rho}, ND, \theta \rangle$ be a noisy Markov Decision Process. If the maximal positive noised reward for all state-action pairs (s,a) is observed with a non-zero probability, i.e. if $\Pr(\widetilde{\rho}(s,a) = \rho(s,a) + \theta\rho(s,a)) > 0$, then Q-Learning using update rule (7.3) converges to $Q^{\star}_{\widetilde{\mathrm{M}}}(s,a)$ for all (s,a) as defined in (7.1).*

Proof. Without loss of generality, consider an arbitrary but fixed state-action pair (s,a). By assumption there exists an infinite sequence of time steps $T_{(s,a)} = \{\tau_1, \tau_2, \ldots\}$, where the maximal positive noised reward $\widetilde{\rho}^{\max}(s,a) = \rho(s,a) + \theta\rho(s,a)$ is observed.

If $\widetilde{\rho}^{\max}(s,a)$ is observed, we know from Lemma 2 that the corresponding Q-estimate is updated according to

$$\widehat{Q}_{\tau+1}(s,a) = \widetilde{\rho}^{\max}(s,a) + \gamma \max_{a' \in A} \widehat{Q}_{\tau}(s',a')$$

where $\tau \in T_{(s,a)}$.

Note that this update rule corresponds to the one used in deterministic Q-Learning (cf. Sect. 3.2.3). Considering the sequence of maximal positive noised reward observations $T_{(s,a)}$ and because $\widetilde{\rho}^{\max}(s,a)$ is fixed, the convergence of $\widehat{Q}_{\tau}(s,a) \to Q^{\star}_{\widetilde{\mathrm{M}}}(s,a)$ as $\tau \to \infty$ follows from the convergence of deterministic Q-Learning for deterministic MDPs (cf. Theorem 1 on page 31; a convergence proof is presented in [Mit97, Theorem 13.1]).

Because all $Q^{\star}$ values are optimal values based on $\widetilde{\rho}^{\max}(s,a)$ and due to the monotonicity of the $\widehat{Q}$-estimates, no other rewards can lead to convergence to values

larger than $Q^\star$. Thus, the convergence of Q-Learning using update rule (7.3) in noisy MDPs follows. □

From Lemma 1, we immediately obtain the following result:

Corollary 1. *Under assumptions A1, let* $\widetilde{\mathrm{M}} = \langle \mathrm{M}, \widetilde{\rho}, ND, \theta \rangle$ *be a noisy Markov Decision Process. Given that the maximal positive noised reward for all state-action pairs is observed with a non-zero probability, then Q-Learning with update rule (7.3) applied in* $\widetilde{\mathrm{M}}$ *converges to an optimal policy for the original noise-free Markov Decision Process* M.

Before we continue, note that one might generalize Theorem 7 to show the convergence to $\widehat{Q^\star}(s,a)$ values that are defined over the maximal positive noised reward observed with probability greater than zero for each (s,a).

Given noisy rewards, we just have seen under which conditions convergence to an optimal policy can be achieved. Next, we want to conclude this section by returning to Example 2. From that example, one might assume that using noise robust rewards can avoid such problems and can ensure convergence to optimal policies. Clearly, this holds for the simple setting considered in that example, as basically a single state has to be examined. However, noise robust rewards are not a general means to ensure convergence to optimal policies as shown in the third example:

Example 3. *Consider the simple MDP illustrated on the left side of Fig. 7.3 and assume a discount factor* $\gamma = 0.9$. *The agent should learn to reach goal state* s_2. *Then, the optimal Q-values can easily be calculated as follows:*

$$Q^\star(s_0,a) = \rho(s_0,a) + \gamma \cdot \max_{a'} Q^\star(s_1,a') = 1 + 0.9 \cdot 20 = 19$$
$$Q^\star(s_0,c) = \rho(s_0,c) + \gamma \cdot \max_{a'} Q^\star(s_2,a') = 20 + 0.9 \cdot 0 = 20$$
$$Q^\star(s_1,b) = \rho(s_1,b) + \gamma \cdot \max_{a'} Q^\star(s_2,a') = 20 + 0.9 \cdot 0 = 20$$

Thus, the optimal policy is given by $\pi^\star(s_0) = c$ *and* $\pi^\star(s_1) = b$.

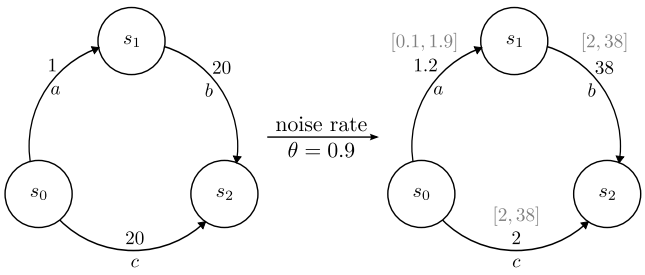


Fig. 7.3 A simple learning task, where the assumed noise distribution changes the optimal action.

As it can immediately be inferred from the gray intervals shown in the figure, the reward values in this example are noise robust for $\theta = 0.9$. *The black values presented on the right side of of Fig. 7.3 quantify the maximal positive noised rewards observed in this noisy MDP. Hence, by Theorem 7, Q-Learning with the optimistic update rule and noisy rewards will converge to the following values:*

$$\widehat{Q^\star}(s_0, a) = \rho(s_0, a) + \gamma \cdot \max_{a'} \widehat{Q^\star}(s_1, a') = 1.2 + 0.9 \cdot 38 = 35.4$$
$$\widehat{Q^\star}(s_0, c) = \rho(s_0, c) + \gamma \cdot \max_{a'} \widehat{Q^\star}(s_2, a') = 2 + 0.9 \cdot 0 = 2$$
$$\widehat{Q^\star}(s_1, b) = \rho(s_1, b) + \gamma \cdot \max_{a'} \widehat{Q^\star}(s_2, a') = 38 + 0.9 \cdot 0 = 38$$

Accordingly, the greedy policy will be $\widehat{\pi^\star}(s_0) = a$ *and* $\widehat{\pi^\star}(s_1) = b$.

Since $\pi^\star(s_0) \neq \widehat{\pi^\star}(s_0)$*, noise robust rewards cannot preserve the characteristics of general MDPs under noise. Hence, using deterministic Q-Learning with update rule (7.3) plus having noise robust rewards, in general, does not ensure to learn an optimal policy for the underlying MDP. Only if the condition on the noise distributions stated in Theorem 7 are met, the approach will learn optimal policies. In that case, however, we do require noise robust rewards.*

Despite the negative results concerning noise robust rewards, we will reconsider them later in this work. In particular, they will be useful when we deal with multiagent learning in sequential stage games (see Sect. 8.6).

7.4 Conclusion

In this chapter, we investigated the influence of randomly noised rewards in deterministic environments. We provided a formal model for this and showed that an optimal policy obtained under noise is also optimal for the noise-free process.

There are two main contributions in this chapter:

1. Noise robust rewards (i.e. rewards whose possible noised values do not overlap) sound like a promising property for learning under noised reward perceptions. However, we showed that this property cannot ensure convergence towards optimal policies in deterministic Q-Learning with noised rewards.
2. Additionally, we proposed to use an optimistic Q-value update rule that is known from multiagent reinforcement learning. Even under this update rule, noise robust rewards do not lead to optimal policies. Instead, an assumption on observing maximal positive noised rewards with a probability greater than zero is required to provably converge to optimal policies under noise using the optimistic update rule.

The assumption on observing the maximal positive noised rewards is not really a limitation, as it is generally fulfilled by standard distributions like the uniform or the Gaussian distribution. Accordingly, we proposed a simple algorithm that is able to learn an optimal policy in deterministic MDPs under noised rewards.

A further interesting remark is the following. Although we dealt with noisy rewards that are drawn from intervals, the results of this section do not depend on the intervals as a whole. Instead the endpoints of the intervals determine the proven properties.

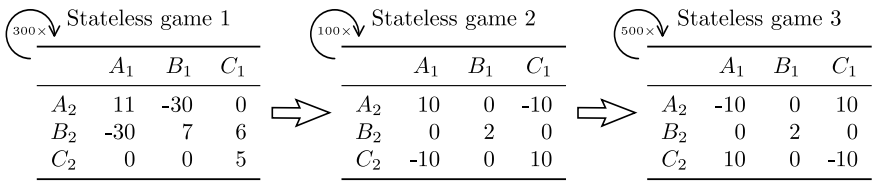
Finally, we want to relate the provided convergence results for deterministic Markov Decision Processes with noisy rewards to the well-known Q-Learning algorithm for non-deterministic environments (cf. Sect. 3.2.3). Our approach en-

sures convergence for probabilistic reward functions and deterministic state transition functions. In contrast, Q-Learning for non-deterministic environments ensures convergence for MDPs with probabilistic reward *and* state transition functions [WD92, Tsi94, Mit97]. Thus, the latter algorithm may also be used in our setting. However, as explained earlier in Sect. 3.2.3, it requires some assumptions on the learning rate (cf. (3.12)), which one has to take care of. In detail, the rate has to be decayed over time such that the sum of learning rates approaches infinity and such that the sum of their squares is a finite value. Besides this, the way how the learning rate is decayed also strongly influences the convergence speed. Even-Dar and Mansour [EDM03] showed, for instance, that the convergence speed may vary between a polynomial and an exponential, both depending on $\frac{1}{1-\gamma}$, where γ denotes the used discount factor. Contrary to that, our approach does not require a learning rate. As future work, it seems to be interesting to compare the convergence speeds of our approach with that of non-deterministic Q-Learning.

Chapter 8

Distributed Stateless Learning

In Chapter 4, we motivated and introduced a novel class of games, called sequential stage games. A sequential stage game consists of a set of stateless games that are played repeatedly, one after the other, each for a certain but to the agents unknown number of iterations. Accordingly, the term stage game reflects that a stateless game is played repeatedly for some time period. Figure 8.1 shows a simple example for two agents.



	A_1	B_1	C_1
A_2	11	-30	0
B_2	-30	7	6
C_2	0	0	5

	A_1	B_1	C_1
A_2	10	0	-10
B_2	0	2	0
C_2	-10	0	10

	A_1	B_1	C_1
A_2	-10	0	10
B_2	0	2	0
C_2	10	0	-10

Fig. 8.1 Simple cooperative sequential stage game with three stateless games for two agents (agent i can execute actions A_i, B_i, C_i) . The stateless games are played repeatedly for 300, 100, resp. 500 iterations before the next game in the sequence is played.

Such kinds of games are interesting because of their particular structure: agents face a special situation only for some time during which they have to learn how to deal with it in the best possible way. Afterwards, they will be confronted with a new situation for some time, to which they have to adapt again. In general, strategies for previous situations become useless, if consecutive stateless games have different characteristics. Also, it is highly unlikely to be confronted with the same situation for more than one time period in more complex games, e.g. in the Iterative Agent Partitioning Problem. This means, that storing old strategies is needless as this mainly produces costs that won't pay off because they most likely won't be used again.

In this chapter, we present an approach towards distributed learning in such stateless game sequences, i.e. in cooperative sequential stage games. The idea is to adopt Lauer's and Riedmiller's Distributed Q-Learning [LR00] to stateless settings and to combine it with engineered rewards that are transformed from a sequence of stage games. We describe the approach in detail and analyze it theoretically and experimentally. In addition, we formally investigate its behavior under noised rewards.

The main contributions of this chapter are as follows:

- We present the first multiagent reinforcement learning (MARL) approach tailored to cooperative sequential stage games. It is space and time efficient, as agents ignore other agents and only have to store one value for each action. Also, they only store one strategy that is adapted in an online manner to changing conditions.
- We prove convergence properties of our approach, i.e. we show that the approach converges under some assumptions to (near-)optimal solutions. The theoretical convergence results are underlined by empirical evaluations.
- We investigate the conditions for learning in scenarios with noised rewards and prove several convergence results.

This chapter is organized as follows. In Sect. 8.1, we start with an introduction and briefly present the idea of the proposed approach. The approach itself will be introduced in Sect. 8.2, followed by a formal analysis in Sect. 8.3, and an empirical evaluation in Sect. 8.4. Then, Sect. 8.5 discusses the time and space complexity of our approach and compares it to other algorithms. Section 8.6 considers our learning approach under noised rewards and provides some formal convergence results. Finally, we conclude this chapter with a discussion in Sect. 8.7.

Parts of this chapter are based on and have already been published in [KKB12], which is an extended and revised version of [KKB11a]. One additional result has been published in [KKB11b]. In particular, the presented details (e.g. lemmas, theorems, proofs and algorithms) in the first half of this chapter originate from those publications and are (partly) reformulated or extended.

8.1 Introduction

In this chapter, we present the *Distributed Stateless Learning* (DSL) approach for cooperative sequential stage games. The approach is an online learning technique, in which agents adapt their strategies to constantly changing settings that are defined through the sequence of stage games in a cooperative SSG. Figure 8.2 illustrates the keystones that are used in the DSL approach.

The proposed approach deals with cooperative sequential stage games (CSSG), which have been introduced and defined formally in Sect. 4.1. The first keystone is the Distributed Q-Learning (DQ) algorithm. It was already introduced in detail in Sect. 3.3.2.1. Recall that it converges to an optimal joint strategy for deterministic cooperative stochastic games with positive rewards. Therefore, it uses an *optimistic* assumption and updates values and strategies solely on improvements. The second keystone, the concept of engineered rewards, was presented in detail in Sect. 4.2. It basically allows constructing rewards that may encode (additional) information such that it is sufficient to observe only the engineered reward rather than a reward and a state signal. Finally, the third keystone describes a transformation of a cooperative sequential stage game using the concept of engineered rewards. This transformation, which we will describe later in this chapter, ensures that the optimal joint strategy is the same in both games, the original CSSG and the transformed one. In addition, the transformation has to ensure one condition on the relations of rewards for consecutive stage games. Details on this will follow in the next paragraph.

Based on these three keystones, the basic approach can be described as follows:

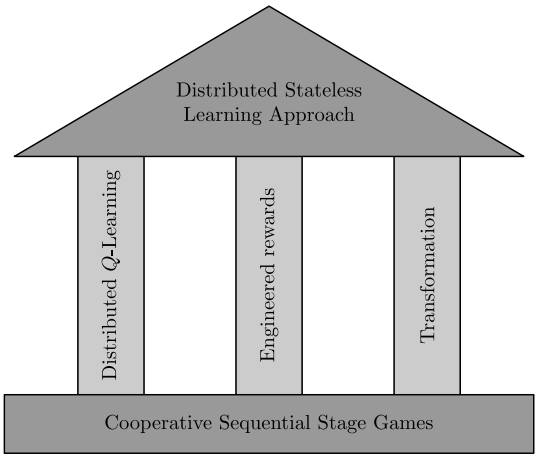


Fig. 8.2 Basement and keystones of the Distributed Stateless Learning (DSL) approach.

1. Consider an arbitrary cooperative sequential stage game Γ with a set of consecutively played stage games $\mathcal{G} = \{\langle G_0, n_0\rangle, \langle G_1, n_1\rangle, \ldots, \langle G_m, n_m\rangle\}$, where each game G_j is played n_j times. Using a universal transformation function that we provide later on, this game set can be transformed into a set of engineered stage games $\mathcal{G}' = \{\langle G'_0, n_0\rangle, \langle G'_1, n_1\rangle, \ldots, \langle G'_m, n_m\rangle\}$. The term "engineered stage game" reflects that the stage games are constructed such that the lowest reward in a succeeding stage game with index $j+1$ is larger than the largest reward in a preceding stage game with index j. Note that a transformation that creates such engineered games has to ensure that the sets of optimal joint strategies are equal for $\mathcal{G}$ and $\mathcal{G}'$.
2. Instead of playing the original game Γ with set $\mathcal{G}$, play the transformed game with game set $\mathcal{G}'$. Therefore, use a variant of the Distributed Q-Learning (DQ) algorithm which does not store $q(s,a)$ values for each state action pair but only $q(a)$ values for each action. Since, by construction, each reward in a succeeding stage game is larger than the largest reward in any preceding stage game, it is ensured that the optimistic update of DQ is able to adopt the agents strategies. By this argument, and because of the convergence properties of DQ, convergence towards (near-)optimal joint strategies for each stage game in $\mathcal{G}'$ follows. Since the sets of optimal joint strategies are equal for $\mathcal{G}$ and $\mathcal{G}'$, the agents also behave optimally in the original game.

Note that the concept of engineered rewards and the transformation, which maintains optimal joint strategies, are key requirements to enable learning in a cooperative SSG based on the Distributed Q-Learning algorithm. This follows, because the ordinary Distributed Q-Learning algorithm without these two concepts is unable to converge to (near-)optimal joint strategies in cooperative SSGs. The reason therefore can be found in the fact that values, and thus also strategies, will not be updated if a previous stage game has led to larger values than a succeeding stage game.

Since our approach adjusts agent strategies online, and hence does not need to distinguish explicitly between different states, it is also more space efficient than other approaches for general cooperative stochastic games.

Accordingly, the DSL approach is well tailored to the considered class of games.

8.2 Approach

As stated in the preceding section, the *Distributed Stateless Learning* (DSL) approach consists of two main steps. Given an arbitrary cooperative sequential stage game Γ, this game first has to be transformed into an optimal joint policy equivalent game Γ'. Then, a variant of the Distributed Q-Learning algorithm is used to play the engineered game Γ'. In this section, we will present the transformation function and the pseudocode of that algorithm, both mainly based on [KKB12] and an earlier version of that paper [KKB11b]. In order to concentrate on the approach itself, we shift its analysis to a separate section.

Let $\mathrm{CCSG}(\mathcal{A}, U)$ denote the set of cooperative common static games with the same set of agents $\mathcal{A}$ and joint actions U (cf. Def. 11). Since the approach deals with cooperative SSGs, the considered stage games are also cooperative and hence from the set $\mathrm{CCSG}(\mathcal{A}, U)$. For brevity, we will write CCSG if we refer to $\mathrm{CCSG}(\mathcal{A}, U)$.

In [LR00], Lauer and Riedmiller defined the optimistic assumption for cooperative deterministic stochastic games using individual, local q_i tables for each agent i. In detail, they use

$$q_i(s, a) = \max_{\substack{u=\langle a_0,\ldots,a_i,\ldots,a_{n-1}\rangle \\ a_i=a}} Q(s, u)$$

where $Q(s, u)$ corresponds to the Q-table in the centralized Q-Learning approach, that reduces the stochastic game to a MDP with actions that correspond to joint actions (cf. Sect. 3.3.2). Since we consider CSSGs, there is no state signal and our local q values reduce to

$$q_i(a) = \max_{\substack{u=\langle a_0,\ldots,a_i,\ldots,a_{n-1}\rangle \\ a_i=a}} Q(u).$$

We now begin with the presentation of a transformation function $\mathtt{t}$ for two consecutively played cooperative common static games [KKB12]:

Definition 24 (Transformation function $\mathtt{t}$). Given two arbitrary consecutively played games $G_j = \langle \mathcal{A}, U, \rho_{G_j}\rangle$ and $G_{j+1} = \langle \mathcal{A}, U, \rho_{G_{j+1}}\rangle$ from CCSG. Then, transformation function $\mathtt{t}$, based on G_j, transforms G_{j+1} into a new game G'_{j+1}. Formally $\mathtt{t}(G_j, G_{j+1}) = G'_{j+1}$, where $G'_{j+1} = \langle \mathcal{A}, U, \rho_{G'_{j+1}}\rangle$ uses reward function $\rho_{G'_{j+1}}$ which is calculated according to

$$\rho_{G'_{j+1}}(u) = \max_{u' \in U} \left\{ |\rho_{G_j}(u')| \right\} + \max_{u' \in U} \left\{ |\rho_{G_{j+1}}(u')| \right\} + \rho_{G_{j+1}}(u) \tag{8.1}$$

Based on two games $G_j, G_{j+1} \in \mathrm{CCSG}$, the transformation hence returns a new game $G'_{j+1} \in \mathrm{CCSG}$ for which all rewards are greater or equal to the largest reward in G_j. Furthermore, the set of optimal joint strategies for G_{j+1} and its engineered counterpart G'_{j+1} are equal, as we will prove later on. Based on the transformation $\mathtt{t}$, we can define a function T that transforms a complete cooperative sequential stage game, such that the aforementioned properties hold for any consecutively played pair of stage games. Since it is necessary to clearly distinguish between the transformation of two (cooperative common) static games using transformation function $\mathtt{t}$ and function T that applies $\mathtt{t}$ to transform an entire cooperative sequential stage game, we will hereafter refer to T as "CSSG-transformation function":

Definition 25 (CSSG-Transformation Function T).
Let $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle \in$ CSSG, and let $\mathbb{G}(\mathcal{G}) = \{G_j \mid \langle G_j, n_j \rangle \in \mathcal{G}\}$ denote the set of cooperative common static games contained in the tuples $\langle G_j, n_j \rangle$ of $\mathcal{G}$. Furthermore, define a cooperative common stage game $1_G = \langle \mathcal{A}, U, \rho^1 \rangle$, having a constant reward of 1, i.e. $\rho^1(u) = 1, \forall u \in U$. Then, the CSSG-transformation function $\mathsf{T} : \text{CSSG} \rightarrow \text{CSSG}$ applied to Γ calculates a new game $\Gamma' = \langle \mathcal{A}, U, \mathcal{G}', \langle G'_0, n_0 \rangle, g \rangle$, having $\mathcal{G}' = \{\langle G'_0, n_0 \rangle, \langle G'_1, n_1 \rangle, \langle G'_2, n_2 \rangle, \ldots, \langle G'_m, n_m \rangle\}$ with $G'_0 = \mathsf{t}(1_G, G_0)$, and $G'_j = \mathsf{t}(G'_{j-1}, G_j), G'_{j-1} \in \mathcal{G}', G_j \in \mathbb{G}(\mathcal{G}), j \geq 1$.

Different to an earlier definition of T in [KKB12], the above defined function is refined so as to transform the first game, too. In particular, it uses a special stage game 1_G, which returns 1 for any joint action, in order to transform the first stage game. As we will see later, this construction enables our approach to be applicable in stage games with arbitrary bounded rewards—particularly also negative rewards, which are not allowed in DQ-Learning. Therefore, using a game that always returns 0 would also have been sufficient. However, we will extend our approach to deal with noisy rewards like those considered in the previous chapter. It is then required to have positive bounded rewards that are greater than zero. Accordingly, we decided to use 1_G right from the start in order to avoid confusions later on.

Let us return to the approach itself. A cooperative SSG that is obtained from the CSSG-transformation T serves as input for the Distributed Stateless Learning algorithm as specified in Alg. 2. As before, $\gamma \in [0, 1)$ denotes the discount factor and A_i the set of actions of agent i. The iteration limit $t_{\max}$ is the sum over all n_j of the CSSG. As one can see, the algorithm basically corresponds to a simplified Distributed Q-Learning, which does not take a state signal into account. Figure 8.3 shows a flowchart for the DSL approach.

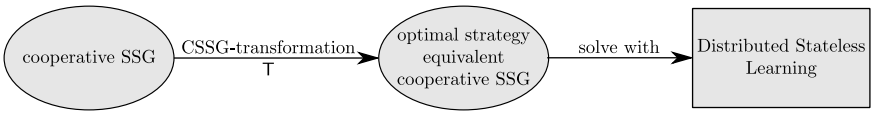


Fig. 8.3 An arbitrary CSSG is transformed using T. The resulting game with its engineered rewards is used as input for the DSL algorithm.

In the remainder, if we talk about the DSL *algorithm*, we mean the pseudocode presented in Alg. 2. If we talk about the DSL *approach*, we refer to the entire approach including engineered rewards and transformed games. In the next section, we will formally analyze the properties of the presented approach.

8.3 Formal Analysis

We start our analysis with a simple, yet important property of transformation function t that will be used later in this section:

Lemma 3. *Let game $G'_{j+1} = \mathsf{t}(G_j, G_{j+1}) \in$ CCSG be the result of the transformation function t as defined in Def. 24 for two* CCSG*s $G_j = \langle \mathcal{A}, U, \rho_{G_j} \rangle$ and $G_{j+1} = \langle \mathcal{A}, U, \rho_{G_{j+1}} \rangle$. Then all rewards in game $G'_{j+1} = \langle \mathcal{A}, U, \rho_{G'_{j+1}} \rangle$ are non-negative, i.e. $\rho_{G'_{j+1}}(u) \geq 0, \forall u \in U$.*

Algorithm 2 DSL (executed by each agent $i \in \mathcal{A}$) [KKB11b]

Input: Cooperative SSG obtained via CSSG-transformation T
procedure DISTRIBUTEDSTATELESSLEARNING
  $t \leftarrow 0$
  $\forall a \in A_i : q_0^i(a) \leftarrow 0$ ▷ initialize local q-function
  $\sigma_i \leftarrow$ choose arbitrary action from A_i ▷ initialize strategy
  while $(t < t_{\max})$ **do**
    Select action $a_t^i \in A_i$ (e.g. ε-greedy selection)
    Execute action a_t^i ▷ leads to joint action u_t
    Observe reward $\rho_t(u_t)$ ▷ reward of joint action u_t
    $\mathsf{maxq}_t^i \leftarrow \max_{a \in A_i}\{q_t^i(a)\}$
    $q_{t+1}^i(a) \leftarrow q_t^i(a), \forall a \in A_i \setminus \{a_t^i\}$
    $q_{t+1}^i(a_t^i) \leftarrow \max\{q_t^i(a_t^i), \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i\}$
    if $q_{t+1}^i(a_t^i) > \mathsf{maxq}_t^i$ **then**
      $\sigma_i \leftarrow$ choose a_t^i ▷ update strategy
    $t \leftarrow t + 1$

Proof. Recall (8.1) of transformation $\mathtt{t}$:

$$\rho_{G'_{j+1}}(u) = \max_{u' \in U} \left\{|\rho_{G_j}(u')|\right\} + \max_{u' \in U} \left\{|\rho_{G_{j+1}}(u')|\right\} + \rho_{G_{j+1}}(u).$$

Clearly, the first maximum argument leads to a value ≥ 0. The second part of that equation, $\max_{u' \in U} |\rho_{G_{j+1}}(u')| + \rho_{G_{j+1}}(u)$, is also ≥ 0 since adding any (negative) reward $\rho_{G_{j+1}}(u)$ to the largest absolute value of the same game is at least 0. □

We continue by proving Lemma 4, which states that the rewards in a transformed stage game obtained from transformation $\mathtt{t}$ are greater than or equal to the largest reward in the preceding stage game. This property will be required to show the convergence of the DSL approach later on.

Lemma 4 ([KKB12]). *Let game $G'_{j+1} = \mathtt{t}(G_j, G_{j+1}) \in$ CCSG be the result of the transformation function $\mathtt{t}$ as defined in Def. 24 for two games $G_j, G_{j+1} \in$ CCSG. Then all rewards in game G'_{j+1} are greater than or equal to those in game G_j, formally $\rho_{G'_{j+1}}(u) \geq \rho_{G_j}(u), \forall u \in U$.*

Proof. Recall (8.1) of transformation $\mathtt{t}$:

$$\rho_{G'_{j+1}}(u) = \max_{u' \in U} \left\{|\rho_{G_j}(u')|\right\} + \max_{u' \in U} \left\{|\rho_{G_{j+1}}(u')|\right\} + \rho_{G_{j+1}}(u).$$

Let us assume there is one $u \in U$ for which $\rho_{G'_{j+1}}(u) < \rho_{G_j}(u)$ holds, then:

$$\max_{u' \in U} \left\{|\rho_{G_j}(u')|\right\} + \max_{u' \in U} \left\{|\rho_{G_{j+1}}(u')|\right\} + \rho_{G_{j+1}}(u) < \rho_{G_j}(u).$$

This can be transformed to

$$\underbrace{\max_{u' \in U} \left\{|\rho_{G_{j+1}}(u')|\right\} + \rho_{G_{j+1}}(u)}_{\geq 0} < \underbrace{\rho_{G_j}(u) - \max_{u' \in U} \left\{|\rho_{G_j}(u')|\right\}}_{\leq 0} \tag{8.2}$$

which clearly is a contradiction, since the left side even for negative rewards evaluates to ≥ 0 and the right side to ≤ 0 since $\max_{u' \in U} \left\{|\rho_{G_j}(u')|\right\}$ is the largest absolute reward value in game G_j. Hence, the lemma follows. □

Since the proposed approach plays a game with transformed, i.e. engineered rewards, it is important to assure that the learned (near) optimal strategies are also (near) optimal in the source game. This property is proven in Lemma 6. For that proof, we need to show that an optimal joint strategy remains optimal if a constant value is added to each reward. This is done in Lemma 5.

Lemma 5 ([KKB12]). *An optimal joint strategy $\boldsymbol{\sigma}^\star$ for a game G remains optimal if a constant value is added to all rewards.*

Proof. Since a joint strategy in a static stateless game defines a particular joint action, let us agree for the course of this proof that $\rho(s, \boldsymbol{\sigma}, s')$ corresponds to the reward that follows from executing the joint action defined by the joint strategy $\boldsymbol{\sigma}$ in state s and transitioning to s'.

By definition, an optimal joint strategy $\boldsymbol{\sigma}^\star$ maximizes the summed discounted rewards, i.e. for any optimal joint strategy $\boldsymbol{\sigma}^\star$ it holds that

$$\sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}^\star, s') \geq \sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}, s'), \forall \boldsymbol{\sigma} \in \boldsymbol{\Sigma},$$

where γ is the discount factor, ρ the reward function, and $\boldsymbol{\Sigma}$ the set of all joint strategies.

Let us assume that an optimal joint strategy $\boldsymbol{\sigma}^\star$ becomes suboptimal if a constant value is added to all rewards. Hence, there must be another joint strategy $\boldsymbol{\sigma}'$ that maximizes the summed discounted rewards:

$$\begin{aligned}
\sum_{t=0}^{\infty} \gamma^t (\rho(s, \boldsymbol{\sigma}^\star, s') + c) &< \sum_{t=0}^{\infty} \gamma^t (\rho(s, \boldsymbol{\sigma}', s') + c) \\
\sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}^\star, s') + \sum_{t=0}^{\infty} \gamma^t c &< \sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}', s') + \sum_{t=0}^{\infty} \gamma^t c \\
\sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}^\star, s') &< \sum_{t=0}^{\infty} \gamma^t \rho(s, \boldsymbol{\sigma}', s')
\end{aligned}$$

The lemma follows since the above inequality is a contradiction to the assumption that $\boldsymbol{\sigma}^\star$ was an optimal joint strategy before the constant was added. □

Lemma 6 ([KKB12]). *Let game $G'_{j+1} = \mathtt{t}(G_j, G_{j+1}) \in$ CCSG be the result of the transformation function* t *as defined in Def. 24 for two games $G_j, G_{j+1} \in$ CCSG. Then any optimal joint strategy for game G_{j+1} is also an optimal joint strategy for G'_{j+1} and vice versa.*

Proof. Let $c = \max_{u' \in U} \left\{ |\rho_{G_j}(u')| \right\} + \max_{u' \in U} \left\{ |\rho_{G_{j+1}}(u')| \right\}$. Then, (8.1) of transformation function t can be rewritten as $\rho_{G'_{j+1}}(u) = c + \rho_{G_{j+1}}(u)$, where c is constant for any two fixed games G_j and G_{j+1}. Thus, the reward function $\rho_{G'_{j+1}}$ for game G'_{j+1} is obtained by adding a constant to the rewards of game G_{j+1}. In combination with Lemma 5, any optimal joint strategy $\boldsymbol{\sigma}^\star$ for game G_{j+1} hence is also an optimal joint strategy for game G'_{j+1}. By the same arguments, the other direction of the proof follows, as $\rho_{G_{j+1}}(u) = \rho_{G'_{j+1}}(u) - c$. □

Since the function T for transforming complete cooperative SSGs is defined based on t, the optimal joint strategies for the sequence of stage games in a transformed CSSG and its source game are obviously also identical.

Let us now prove some convergence properties of our approach. First of all, We show that it converges to an optimal joint strategy for the last stage game contained in a CSSG. Therefore, consider Corollary 2 which follows from the convergence of Distributed Q-Learning (DQ) in general deterministic cooperative stochastic games with positive rewards:

Corollary 2 ([KKB12]). *Distributed Q-Learning converges to an optimal joint strategy for a cooperative common stage game that is obtained from transformation* $\mathtt{t}$*, if each action is performed infinitely often.*

By demanding that the stage game in Corollary 2 is obtained via transformation $\mathtt{t}$, we fulfill the convergence requirement of Distributed Q-Learning regarding positive rewards, as shown in Lemma 3. Clearly, we deal with "deterministic" stage games, since there are no state transitions in a single stage game and the reward functions are also deterministic. Hence, all assumptions for convergence made by DQ are fulfilled. Using this corollary, we are able to prove our first convergence result for the DSL approach:

Theorem 8 ([KKB12]). *Let Γ be a cooperative sequential stage game whose last cooperative common static game G_m is played $n_m \to \infty$ times. Then, DSL for $\Gamma' = \mathsf{T}(\Gamma)$ converges to an optimal joint strategy $\boldsymbol{\sigma}^\star$ for G_m, if each joint action is visited infinitely often.*

Proof. Since $\sum_{j=0}^{m} n_j \to \infty$ as $n_m \to \infty$, each joint action is visited infinitely often. From Def. 25 of the CSSG-transformation function T, we obtain game G'_m, which is an optimal joint strategy equivalent game of G_m according to Lemma 6. From Lemma 4 and Def. 25, we know that the reward for each joint action in G'_m is greater than or equal to the rewards obtained in all previous stage games played by DSL on Γ'. Since this is a required condition for updating the strategies, the convergence to an optimal joint strategy $\boldsymbol{\sigma}^\star$ for G'_m follows from Corollary 2. Since G'_m and G_m are joint strategy equivalent games, the proof is completed. □

Clearly, the goal of any approach for sequential stage games should be to find optimal or at least near-optimal joint strategies for each stage game. The next theorem proves such a property for the DSL approach in a transformed CSSG and states some required conditions:

Theorem 9 ([KKB12]). *Let $\Gamma' = \mathsf{T}(\Gamma)$ be the cooperative sequential stage game obtained from a* CSSG*-transformation of a cooperative SSG Γ with $m+1$ stage games. Then, for all stage games $G'_j, 0 \leq j \leq m$ and $\forall \varepsilon, \delta > 0$, there exists an $n_j(\varepsilon, \delta) : n_j(\varepsilon, \delta) < n_j < \infty$ such that DSL for Γ' successively converges to an ε-optimal joint strategy $\boldsymbol{\sigma}^\star_j$ for each stage game G'_j with probability $> 1 - \delta$ if and only if G'_j is played n_j times and each joint action is visited* sufficiently *often.*

Proof. From the application of $\mathsf{T}(\Gamma)$ we obtain Γ' with the set of cooperative common static games $\mathcal{G} = \{\langle G'_0, n_0\rangle, \langle G'_1, n_1\rangle, \ldots, \langle G'_m, n_m\rangle\}$. From Lemma 3 it follows that we only deal with non-negative reward values. By Lemma 4 it follows that all rewards in G'_j are greater than or equal to those of the previous game G'_{j-1}. Accordingly, the algorithm is able to update the q-values and the strategy if any optimal joint action in G'_j is found, as the reward will be greater or equal than the local q-value that emerged from previous games (cf. Alg. 2, lines 9-13). Hence, it is sufficient to prove the above proposition for an arbitrary but fixed game G'_j.

Let $q_t^i(a)$ denote the local q-value for agent i and action $a \in A_i$ in iteration t of the algorithm. In addition, and without loss of generality, let $q^\star(j,a)$ be the optimal q-value for a fixed stage game G'_j and action a.

Then, we have to show that for all $\varepsilon, \delta > 0$ there exists an $n_j(\varepsilon,\delta)$ such that if $n_j > n_j(\varepsilon,\delta)$ (and $n_j < \infty$) then $\forall a \in A_i, \forall i \in \mathcal{A}$:

$$\Pr(|q^i_{n_j}(a) - q^\star(j,a)| < \varepsilon) > 1 - \delta.$$

From [LR00], we know that the q-value update as performed in line 11 of the DSL algorithm leads to monotonically increasing q-values for non-negative rewards $\rho_i(a) \geq 0, \forall a \in A_i, \forall i \in \mathcal{A}$. Also from [LR00], we know that $q_t^i(a)$ with probability 1 converges to $q^\star(j,a)$ as $t \to \infty$ and given that each joint action is visited often enough. By these two arguments, we can conclude the existence of an $n_j(\varepsilon,\delta)$ that meets our requirements. □

By Lemma 6 and Def. 25 of the CSSG-transformation function T, the sets of optimal joint strategies for two CSSGs Γ and $\mathsf{T}(\Gamma) = \Gamma'$ are equal. From Lemma 3, we know that the transformed CSSG only uses non-negative rewards. Thus, an immediate conclusion from Theorem 9 is that the DSL approach is able to converge to ε-optimal joint strategies for any cooperative SSG, as stated in Corollary 3.

Corollary 3 ([KKB12]). *Let $\delta, \varepsilon > 0$, then the* DSL *approach converges with probability $> 1 - \delta$ to ε-optimal joint strategies for each stage game of any cooperative sequential stage game Γ if played on the transformed cooperative* SSG $\Gamma' = \mathsf{T}(\Gamma)$ *and if each stage game is played n_j times with $0 < n_j(\varepsilon,\delta) < n_j < \infty$ as in Theorem 9 and such that each joint action is visited sufficiently often.*

We just have shown the convergence properties of the described general DSL approach. Now, let us briefly point out what *basic* requirements have to be fulfilled such that convergence can be ensured, if the approach is realized with different components. For instance, one could use different transformation functions or replace the ϵ-greedy action selection mechanisms (cf. Alg. 2, line 6) by more sophisticated techniques as we will do in Ch. 9. In detail, the basic assumptions that have to be fulfilled are:

Assumptions 2 *Given a cooperative* SSG $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0\rangle, g\rangle$*, with the set of stage games $\mathcal{G} = \{\langle G_0, n_0\rangle, \langle G_1, n_1\rangle, \ldots, \langle G_m, n_m\rangle\}$. Then, the* DSL *approach converges if the following holds:*

1. *Local q-values for all agent actions must be initialized to zero.*
2. *The transformation of Γ to an engineered game Γ' has to ensure that:*
   - *All engineered rewards are bounded and non-negative for all stage games.*
   - *For any pair of consecutively played (transformed) stage games G'_j, G'_{j+1}, the smallest reward in G'_{j+1} has to be greater than or equal to the largest reward in the preceding stage game G'_j.*
3. *All joint actions in each stage game have to be visited* sufficiently *often to ensure convergence to ε-optimal joint strategies for each stage game (Corollary 3), or*
4. *the last stage game has to be played for $n_m \to \infty$ many times and each joint action has to be visited infinitely often to ensure convergence to an optimal joint strategy for the last stage game G_m (Theorem 8).*

Next, we investigate the update behavior of the agents. In particular, we will prove that all agents simultaneously update their strategies. Therefore, we first show that the maximal stored local q-value over all actions and agents is equal at any point in time:

Lemma 7 ([KKB11b]). *Let $\mathcal{A}$ denote the set of agents, A_i the set of actions available to agent i and $q_t^i(a_t^i)$ the local q-value at time t for agent i and action a_t^i. If the discount factor γ is equal for all agents $i \in \mathcal{A}$, then the maximum q-value maxq_t^i at time step t is equal for all agents. Formally, $\forall t, \forall i, j \in \mathcal{A} : \mathsf{maxq}_t^i = \mathsf{maxq}_t^j$.*

Proof. We use an inductive proof over t.

$t = 0$: Initially, all q-values are zero, thus, the induction hypothesis trivially holds, i.e. $\mathsf{maxq}_0^i = 0, \forall i \in \mathcal{A}$.

$t \rightsquigarrow t + 1$: Let $u_t = (a_t^0, \dots a_t^i, \dots, a_t^{n-1})$ denote the joint action of n agents at time t, where a_t^i is the action of agent i. Without loss of generality select any agent $i \in \mathcal{A}$ and consider the reward $\rho_t(u_t)$ obtained at t for that joint action. The q-values for all actions other than a_t^i remain: $q_{t+1}^i(a) = q_t^i(a), \forall a \in A_i \setminus \{a_t^i\}$. Thus, maxq_{t+1}^i can only be changed if the q-value of action a_t^i is updated by

$$q_{t+1}^i(a_t^i) = \max\{q_t^i(a_t^i), \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i\}. \tag{8.3}$$

Due to the induction hypothesis, maxq_t^i is equal for all agents. Additionally, by the assumption on γ and since $\rho_t(u_t)$ is a common reward, it follows that $\rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i$ is equal for all agents as well. Note that $q_t^i(a_t^i) \leq \mathsf{maxq}_t^i$ holds by definition of maxq_t^i. Let us now consider the following three cases regarding (8.3):

1. $\rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i < \mathsf{maxq}_t^i$: Since $q_t^i(a_t^i) \leq \mathsf{maxq}_t^i$, the q-value $q_{t+1}^i(a_t^i)$ might become larger than $q_t^i(a_t^i)$ but will remain $\leq \mathsf{maxq}_t^i$. Hence, $\mathsf{maxq}_{t+1}^i = \mathsf{maxq}_t^i$.
2. $\rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i = \mathsf{maxq}_t^i$: Since $q_t^i(a_t^i) \leq \mathsf{maxq}_t^i$, the new q-value becomes $q_{t+1}^i(a_t^i) = \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i$ and thus the maximum value remains, i.e. $\mathsf{maxq}_{t+1}^i = \mathsf{maxq}_t^i$.
3. $\rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i > \mathsf{maxq}_t^i$: Since $q_t^i(a_t^i) \leq \mathsf{maxq}_t^i$, the new q-value becomes $q_{t+1}^i(a_t^i) = \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i$ and thus the maximum value is increased to $\mathsf{maxq}_{t+1}^i = \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i$.

All agents will update their maximum values according to one of these three cases. Since maxq_t^i and $\rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i$ are equal for all agents as explained before, all agents will perform the same update. Thus $\forall i, j \in \mathcal{A} : \mathsf{maxq}_{t+1}^i = \mathsf{maxq}_{t+1}^j$. □

Hence, we can conclude that all agents update their strategies simultaneously:

Corollary 4 ([KKB11b]). *If the discount factor γ is equal for all agents, then all agents update their strategies simultaneously.*

Proof. The statement follows immediately from the strategy update condition in line 12 of the DSL algorithm and Lemma 7. □

Note that this insight will become valuable later on, as it allows us to reduce the computational effort in a coordination technique proposed in Ch. 9.

In the next section, we will empirically evaluate the approach in a simple two-agent setting.

8.4 Evaluation

In order to underline the theoretical convergence properties of our approach, we performed several experiments and analyze the observed results in this section. We use the same exemplary CSSG as in [KKB12], but provide a broader analysis. In addition, this section differs in the engineered reward values due to the slight extension of the CSSG-transformation function T described earlier.

We investigated a cooperative SSG with three alternating static games that build a sequence of nine stage games. The *Climbing Game (CG)* and the *Penalty Game (PG)* are taken from [CB98], and the *Mirrored Penalty Game (MPG)* corresponds to a mirrored version of the PG. Tables 8.1–8.3 show the rewards of these three two agent games. The game set of the considered CSSG is described by $\mathcal{G} = \{\langle CG0, n_0\rangle, \langle PG1, n_1\rangle, \langle MPG2, n_2\rangle, \langle CG3, n_3\rangle, \langle PG4, n_4\rangle, \langle MPG5, n_5\rangle, \langle CG6, n_6\rangle, \langle PG7, n_7\rangle, \langle MPG8, n_8\rangle\}$, and those games that contain parameter k use $k = -30$. Note that learning optimal joint strategies for each stage game in this sequence involves hard turnarounds. For instance, the optimal joint action $\langle A_1, A_2\rangle$ in PG becomes the worst joint action in MPG. There, in contrast, the previously worst joint action is the optimal one. Also, the transition from MPG to CG involves such a hard turn. Accordingly, the considered CSSG demands high standards of the learning algorithm.

Table 8.1 Climbing game

	A_1	B_1	C_1
A_2	11	-30	0
B_2	-30	7	6
C_2	0	0	5

Table 8.2 Penalty game ($k \leq 0$)

	A_1	B_1	C_1
A_2	10	0	k
B_2	0	2	0
C_2	k	0	10

Table 8.3 Mirrored penalty game ($k \leq 0$)

	A_1	B_1	C_1
A_2	k	0	10
B_2	0	2	0
C_2	10	0	k

To run our experiments, we first need to apply the CSSG-transformation T. In order to clarify the transformation process, we shortly describe its beginning. Recall from Def. 25 that T defines a special game 1_G, which returns a reward of 1 for each joint action. Consider the transformation of the first stage game $CG0$ based on 1_G. The engineered reward for joint action $\langle A_1, A_2\rangle$ according to (8.1) becomes:

$$\begin{aligned}\rho_{CG0'}(\langle A_1, A_2\rangle) &= \max_{u' \in U}\{|\rho_{1_G}(u')|\} + \max_{u' \in U}\{|\rho_{CG0}(u')|\} + \rho_{CG0}(\langle A_1, A_2\rangle)\\ &= 1 + \max\{|-30|, |-30|, 11, 7, 6, 5, 0, 0, 0\} + 11\\ &= 42.\end{aligned}$$

The engineered reward for $\langle B_1, A_2\rangle$ becomes:

$$\begin{aligned}\rho_{CG0'}(\langle B_1, A_2\rangle) &= \max_{u' \in U}\{|\rho_{1_G}(u')|\} + \max_{u' \in U}\{|\rho_{CG0}(u')|\} + \rho_{CG0}(\langle B_1, A_2\rangle)\\ &= 1 + \max\{|-30|, |-30|, 11, 7, 6, 5, 0, 0, 0\} + (-30)\\ &= 1.\end{aligned}$$

Thus, the transformation eliminated the negative value and, as shown in the final reward table 8.4, all values are greater than or equal to the values of the (artificial) previous game 1_G. The stage game $PG1'$ is obtained from transformation of $CG0'$

Table 8.4 Transformed stage game $CG0'$

	A_1	B_1	C_1
A_2	42	1	31
B_2	1	38	37
C_2	31	31	36

Table 8.5 Transformed stage game $PG1'$

	A_1	B_1	C_1
A_2	82	72	42
B_2	72	74	72
C_2	42	72	82

and $PG1$, i.e. $PG1' = \mathtt{t}(CG0', PG1)$. Table 8.5 presents the engineered rewards, whose values follow since the maximum absolute reward in $CG0'$ is 42, and the maximum absolute value in $PG1$ is 30.

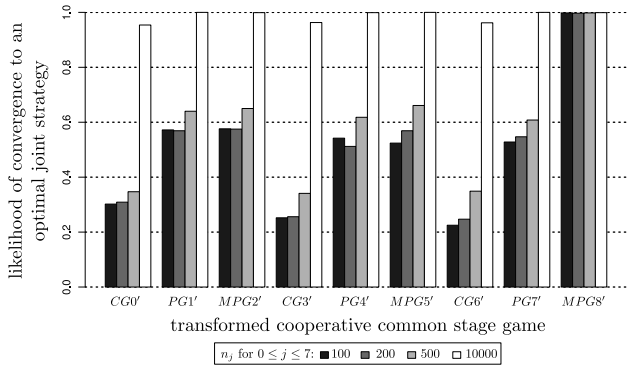
We performed experiments with the transformed CSSG, ϵ-greedy action selection using $\epsilon \in \{0.05, 0.2, 0.5\}$, a discount factor of $\gamma = 0.8$, and repeated each setting 1000 times. The number of iterations for each of the first eight stage games has been $n_j \in \{100, 200, 500, 10000\}$. The last stage game $MPG8'$ was always played for 10000 iterations. Figure 8.4 visualizes the simulation results.

In general and independent from ϵ, convergence to an optimal joint strategy for the last stage game $MPG8'$ can be observed with probability 1. This result strongly underlines Theorem 8, which states that this behavior will be observed with probability close to 1 if the last stage game is played *often* ($n_m \to \infty$, to be precise).

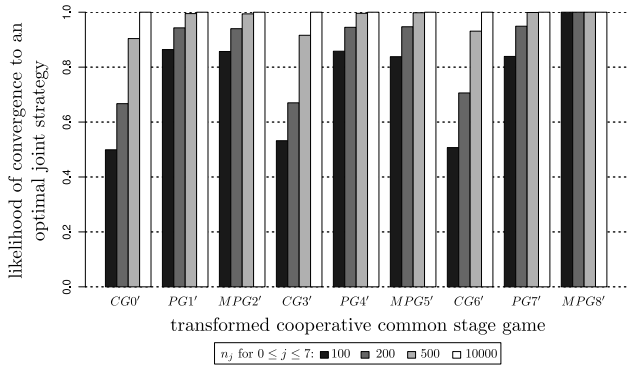
Now turn your attention to Theorem 9, which states that, if each stage game is played long enough and thereby also each joint action is executed often enough, then convergence to near optimal joint strategies for each stage game will be reached with probability close to 1. This theoretical insight is supported by our experiments as shown in Fig. 8.4(b). The figure reveals that given an increasing number of iterations per stage game, the likelihood of converging towards an optimal joint strategy in each stage game also increases. The reason for this is that an optimal joint strategy is more likely to be played under such settings.

Finally, let us focus on the likelihood of convergence in each stage game with respect to different ϵ-greedy strategies. Here, Fig. 8.4 indicates that the likelihood of convergence increases if more exploration is performed—i.e. for larger ϵ values. This is not surprising, since the exemplary game involves only nine joint actions. Thus, executing an optimal joint action by chance in settings with low numbers of iterations per stage game is more probable with increased exploration (cf. Figs. 8.4(a) and 8.4(c)). However, DSL is built for maximizing the agents performance *during runtime* rather than returning an optimal joint strategy in the end. This desired behavior originates from the inherent property of CSSGs, where optimal joint strategies are only valid for a certain timespan and get replaced by new strategies for new situations later on. Accordingly, it is interesting to investigate the average reward per iteration during the course of playing a CSSG. Table 8.6 shows the corresponding values. It is clear that, although the likelihood of convergence to an optimal joint strategy in any stage game is increased with larger ϵ, the average reward per iteration is higher if less exploration is performed. Accordingly, the tradeoff between exploration and exploitation is a particularly important issue in the DSL approach and also in CSSGs in general. We will dedicate Ch. 9 to this topic, where we will propose and investigate different coordination strategies that should help to quickly find optimal joint strategies and, once found, to stick to them.

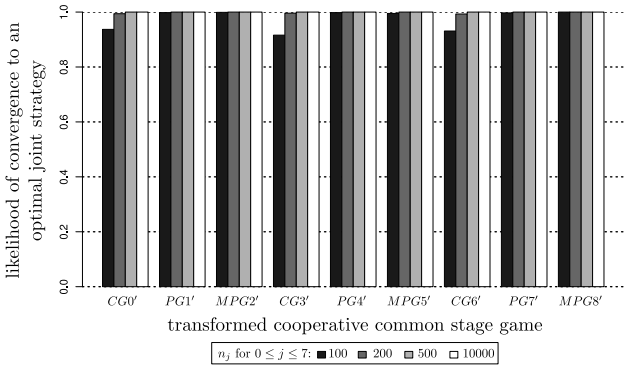
Having evaluated our approach, we want to briefly examine why using the FMQ heuristic [KK02] (see also Sect. 3.3.2) instead of Distributed Q-Learning as basis for



(a) ϵ-greedy action selection with $\epsilon = 0.05$.



(b) ϵ-greedy action selection with $\epsilon = 0.2$.



(c) ϵ-greedy action selection with $\epsilon = 0.5$.

Fig. 8.4 Simulation results for the DSL approach in an artificial cooperative sequential stage game with different ϵ-greedy selection probabilities and different numbers of iterations per stage game.

Table 8.6 Average reward per iteration.

ϵ	n_j			
	100	200	500	10000
0.05	347.06	335.36	308.30	199.89
0.20	343.53	331.94	305.15	196.00
0.50	337.68	326.10	299.31	189.98

our approach would not be successful in general. Recall that FMQ, in contrast to DQ, is able to deal with stochastic rewards to some extent, which potentially could result in an extended applicability of DSL. However, the success of FMQ with respect to converging "almost certainly" to optimal joint actions comes at a price, namely a game-depending weight factor within the heuristic. Since the agents in our settings can only indirectly learn from a game transition due to the lack of a state signal, they also can hardly determine when to change such a game-depending weight parameter. In addition, the next problem would be how to set the weight parameter, if no a priori knowledge about the game is available. Also, our approach would properly lose its convergence properties. Hence, FMQ can hardly be applied in the DSL approach.

In the next section, we will consider the time and space complexity of our approach.

8.5 Complexity

This section is partly based on [KKB12] and will briefly analyze the time and space complexity of the DSL approach.

For the runtime analysis let us assume that we use a hash table to store the q-values. Then, accessing the values requires time $\mathcal{O}(1)$ on average under reasonable assumptions (cf. [CLRS01] for details). The initialization of the local q-values requires time $\mathcal{O}(|A_i|)$, since each action value has to be set to zero. A single iteration of the "while" loop in line 5 of Alg. 2 strongly depends on the action selection mechanism and the used data structures. Let us assume a simple ϵ-greedy action selection procedure. Its runtime is in $\mathcal{O}(1)$, since it only involves some simple operations like if-statements, or access to q-values, and the generation of a random number. With the help of a poly-time initialization method, the generation of such a random number can be accomplished in constant time according to [Vos91]. Executing an action is assumed to take constant time, as an action in terms of this analysis is just a concept. The same applies for observing the reward. Since maximum values can also be determined in constant time, the overall runtime for one iteration of the loop under our assumptions is in $\mathcal{O}(1)$. As the loop is executed for $t_{\max}$ iterations, we obtain an overall runtime in $\mathcal{O}(|A_i| + t_{\max})$ for the DSL algorithm.

Besides the runtime of the DSL algorithm, we also need to consider the runtime of the transformation. The transformation of a single stage game requires time in $\mathcal{O}(|U|)$. This follows because each joint action $u \in U$ is assigned a reward $\rho(u)$ that has to be transformed using transformation t, whose complexity is determined by twice finding a maximum value in the set of all rewards. Since CSSG-transformation T transforms one sequential stage game consisting of m stage games, the whole transformation process hence requires time in $\mathcal{O}(m|U|)$. Note that this transformation has not to be performed by the agents, but it is part of a preprocessing or done at runtime within the environment.

Concerning space complexity, we can state that the algorithm on a single agent only requires space in $\mathcal{O}(|A_i|)$. It has to store one q-value for each agent action $a \in A_i$, and requires constant space for the observed reward, the maximum q-value maxq, and the current strategy.

The space required in the environment to store the sequential stage game remains unchanged by the transformation and is in $\mathcal{O}(m|U| + m)$, because for each of the m

stage games, we need to store rewards for all possible joint actions plus the space required for the game transition function.

Now, let us briefly compare our approach to ordinary algorithms for cooperative stochastic games. Consider the construction used in Prop. 3 on page 50 to show that SSG $\subset$ SG. Given such a stochastic game that corresponds to a sequential stage game, we observe much higher space requirements. In detail, because each iteration of any stage game of the SSG corresponds to one state in the stochastic game, the space for storing the game structures increases from $\mathcal{O}(m|U| + m)$ to $\mathcal{O}(\sum_{j=0}^{m} n_j|U| + \sum_{j=0}^{m} n_j)$. Given such a cooperative SG, the space required by an agent in Distributed Q-Learning is in $\mathcal{O}(\sum_{j=0}^{m} n_j|A_i| + \sum_{j=0}^{m} n_j)$, as it has to store q-values for each state and action pair plus a strategy for each state. Agents in joint action learners like OAL [WS03] require even more space, since they need to store values for each joint action. Despite the large space requirements, DQ for instance, is unable to converge to optimal joint strategies in the constructed SSG-equivalent stochastic game since each state action pair is visited only once.

8.6 Distributed Stateless Learning with Noisy Rewards

In this section, we provide some insights into the behavior of the Distributed Stateless Learning (DSL) approach under noisy reward perceptions by the agents. Therefore, we extend the results of Ch. 7 for Q-Learning with randomly noised rewards to the multiagent case.

We first define a corresponding multiagent model, where agents can observe individually noised rewards (Sect. 8.6.1). In Sect. 8.6.2, we investigate DSL under different settings with noisy rewards. Finally, we conclude this section with an overview and a discussion of the results in Sect. 8.6.3.

8.6.1 Definitions

Let us begin with the definition of the model that is used for learning in a cooperative sequential stage game under noisy rewards:

Definition 26 (Noisy Cooperative Sequential Stage Game). Consider an ordinary cooperative SSG $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$. The corresponding *noisy cooperative sequential stage game* (NCSSG) differs from Γ in the reward perception of the agents. In detail, each agent i has an individual noise rate $\theta_i \in [0, 1)$. In a cooperative stage game $G_j = \langle \mathcal{A}, U, \{\rho_i^j\}_{i \in \mathcal{A}} \rangle$, an agent perceives a random value as noised reward for joint action $u \in U$ from the interval $[\rho_i^j(u) - \theta_i \rho_i^j(u), \rho_i^j(u) + \theta_i \rho_i^j(u)]$. The random value is drawn according to an agent specific noise distribution ND_i. A cooperative stage game where agents perceive such noised rewards is called *noised cooperative stage game*.

Given that all agents $i \in \mathcal{A}$ have a noise rate $\theta_i = 0$, the noisy CSSG reduces to an ordinary CSSG. Note that the above definition also explains, why we decided that the general CSSG-transformation function (cf. Def. 25) is not allowed to return zero-valued rewards, since a reward of 0 would remain zero under noise as defined above.

Analog to the definition of an optimal policy in a noisy MDP (cf. Ch. 7), we need to define optimal strategies for a noisy cooperative sequential stage game, too. Since such a NCSSG consists of a set of noised cooperative stage games, we can condense the definition to optimal strategies for that class of stateless games.

As each agent makes its own noised observation of the common reward $\rho(u)$, the optimal local q_i-values are agent specific, as indicated by the index. Analog to noisy MDPs, the Bellman optimality equations for a single agent i in a NCSSG become:

$$\begin{aligned} q_i^\star(a) &= \max_{\substack{u=\langle a_0,\dots,a_i,\dots,a_{n-1}\rangle \\ a_i=a}} \left\{ [\rho_i(u) + \theta_i \rho_i(u)] + \gamma \max_{a' \in A} q_i^\star(a') \right\} \\ &= \max_{\substack{u=\langle a_0,\dots,a_i,\dots,a_{n-1}\rangle \\ a_i=a}} \left\{ \widetilde{\rho}_i^{\max}(u) + \gamma \max_{a' \in A} q_i^\star(a') \right\} \end{aligned} \tag{8.4}$$

For brevity, we will use $\widetilde{\rho}_i^{\max}(u) = \rho_i(u) + \theta_i \rho_i(u)$ to denote the maximal positive noised reward value for a joint action observed by an agent. If the context is clear, we might also skip the agent index and write $q^\star$ if we refer to $q_i^\star$.

Based on the $q_i^\star$ values, the optimal joint strategy for a noised cooperative stage game is defined by $\boldsymbol{\sigma}^\star = \langle a_0, a_1, \dots, a_{n-1} \rangle$ having $a_i = \arg\max_{a \in A_i} q_i^\star(a)$. An optimal joint strategy $\boldsymbol{\sigma}^\star$ for a noised game is called *globally optimal joint strategy*, if it corresponds to the optimal joint strategy for the noise-free stage game.

8.6.2 Convergence Results

In this section, we provide some insights into the behavior of the DSL approach under noisy rewards in cooperative stage games. We proceed stepwise and, first of all, show that DSL learns optimal joint strategies, if a common and globally noised reward is used. Furthermore, we prove that DSL cannot learn optimal solutions if agents observe individually noised rewards under the given assumptions. Given the property of noise robust rewards, as introduced in the single agent settings, we then show that DSL under noise is able to learn optimal solutions for games with a single optimal joint action. A general method to create such noise robust rewards for any cooperative stage game is presented. Finally, we propose an extension that allows DSL to learn even if multiple optimal joint actions exist.

8.6.2.1 Globally Noised Common Rewards

Let us begin with a rather simple case in which a common global reward is augmented with noise, which results in a noised reward that is distributed to all agents.

Proposition 6. *Consider a noised cooperative stage game G with positive rewards. The* DSL *approach converges to a globally optimal joint strategy for G if:*

1. *at any time step t the observed noised reward is equal for all agents, i.e. $\widetilde{\rho}_i(u_t) = \widetilde{\rho}_j(u_t), \forall i, j \in \mathcal{A}, i \neq j$,*
2. *all agents use the same discount factor γ,*
3. *each joint action is executed infinitely often, and*

4. *the maximal positive noised reward* $\widetilde{\rho}_i^{\max}(u) = \rho_i(u) + \theta\rho_i(u)$ *is observed with probability* > 0 *for each joint action* $u \in U$.

Proof (Sketch). Note that DSL uses the same optimistic q-value update rule as the single agent Q-Learning based approach in Ch. 7. That approach was shown in Theorem 7 to learn optimal strategies under noise by assuming that the maximal positive noised reward is observed with non-zero probability. Also, the definitions of optimal q-/Q-values are similar. Due to this analogy, we can re-use this result as shown in the following proof sketch:

- The q update function used in the DSL algorithm ensures monotonically increasing q_i values.
- The non-zero probability of observing the maximal positively noised reward together with demanding infinitely many visits to each joint action leads to a sequence of maximum noised reward observations. Accordingly, the local q_i values for each agent will converge to $q^\star$ as defined in (8.4) by the same arguments as in Theorem 7.
- It remains to show, that all agents agree on the same globally optimal joint strategy. This follows from the convergence to optimal q values and from the simultaneous strategy updates of all agents on improvement as proved in Corollary 4, given that all agents observe the same reward and use the same discount factor.

□

8.6.2.2 No Convergence under Individually Noised Perceptions

The above covered case of a "globally noised" common reward is not very realistic due to agent individual perceptions. However, given that agents have such individually noised perceptions, DSL cannot guarantee convergence to optimal joint strategies. To prove this statement, we first have to consider the following example. It demonstrates that agents—which actually play the same cooperative stage game—individually might believe to play different stage games due to their individual reward perception:

Example 4. *First, recall that a noised reward is a random value from* $[(1-\theta)\rho, (1+\theta)\rho]$*, where* $\theta \in [0, 1)$ *denotes a noise rate and* ρ *a clear reward value.*

Next, consider the sample game shown in Tab. 8.7. It is played by two agents with IDs 1 and 2. The reward sensors of both agents have a noise rate of $\theta = 0.1$*. While playing that sample game, let us assume that agent 1 always observes noised rewards from the noise intervals shown in Tab. 8.8, and agent 2 those shown in Tab. 8.9. It is easily verified that both tables are valid according to the noise rates of the agents' sensors and given a corresponding noise distribution. In particular, the intervals show all noised rewards that are observed with a non-zero probability. (Clearly, this noise distribution is artificial, but nevertheless valid.)*

In this setting, agent 2 then considers the joint action $\langle B_1, B_2\rangle$ *to be optimal, because it always leads to the highest reward according to its perception. By the same arguments, agent 1 considers* $\langle A_1, A_2\rangle$ *to be the optimal joint action. Accordingly, this example has demonstrated that agents, which play the same cooperative stage game, actually play different stage games due to their individual noisy reward perception.*

Table 8.7 Sample game: reward function ρ

	A_1	B_1
A_2	10	1
B_2	1	9

Table 8.8 Possible reward intervals for agent 1.

	A_1	B_1
A_2	$[10.0, 11.0]$	$[0.9, 1.1]$
B_2	$[0.9, 1.1]$	$[8.1, 9.9]$

Table 8.9 Possible reward intervals for agent 2.

	A_1	B_1
A_2	$[9.0, 9.5]$	$[0.9, 1.1]$
B_2	$[0.9, 1.1]$	$[9.6, 9.9]$

Having this example in mind, we can next state the following:

Proposition 7. *Consider any noised cooperative stage game and assume that agents only know their action resp. the joint action and their individual (noised) reward. Without additional knowledge, no deterministic MARL algorithm, which learns an optimal joint strategy by maximizing the (expected cumulative discounted) reward, can converge to an optimal joint strategy in general.*

Proof. Without loss of generality, consider an arbitrary noised cooperative stage game G with two agents, agent 1 and agent 2, and an arbitrary deterministic MARL algorithm Alg as described in the proposition that is executed by both agents.

Let us assume, that the noised rewards observed by agent 1 describe a game whose optimal joint action, i.e. the joint action with the highest reward, corresponds to the optimal joint action of the actual game G. Since Alg identifies the optimal joint action by maximizing the (expected cumulative discounted) reward, we can assume that algorithm Alg converges to that optimal joint action. Now, from Example 4, we know that agents individually might perceive different "games" due to their noised perception. Thus, we can assume that agent 2 observes noised rewards which lead to a different optimal joint action than for agent 1. To be perceived as an optimal joint action, that joint action has to lead to the highest reward for agent 2. Since algorithm Alg on agent 1 finds the optimal joint action by identifying the action which leads to the highest reward, the algorithm has to do so on agent 2, too. Accordingly, Alg on agent 2 considers another joint action to be optimal than on agent 1.

As we have seen in Example 4, the combination of individual actions according to different learned joint actions can lead to a suboptimal joint action for the actual game. It follows that Alg cannot be guaranteed to converge to an optimal joint strategy for the actual game G.

By assumption, Alg is a deterministic MARL algorithm and only has access to inputs consisting of (joint) actions and individual rewards. It also has no additional knowledge on the game or the behavior of the other agents. Thus, the algorithm cannot behave differently on both agents, i.e. it cannot once select as optimum the joint action which led to the highest reward on agent 1, and at the same time choose on agent 2 a suboptimal joint action that did not lead to the highest reward. Accordingly, no deterministic MARL algorithm, which learns an optimal joint strategy by maximizing the (expected cumulative discounted) reward, is able to learn an optimal joint action in general for noised cooperative stage games, if no addition knowledge is available. □

From this general result, we hence can conclude that DSL is unable to learn an optimal joint strategy under agent-individual noised rewards if no additional properties hold.

8.6.2.3 Games With Unique Optimal Joint Action

In this section, we consider such an additional property on rewards that was discussed before. Definition 23 in Ch. 7 introduced *noise robust rewards*, i.e. rewards whose noise intervals do not intersect. This property, however, was shown to be insufficient for learning optimal policies under noise in the considered general single agent settings in the same chapter. Nevertheless, given that only a single state is considered, noise robust rewards can become useful as one could deduce from Example 2 on page 98. Since we deal with stage games in our multiagent context, we will next investigate the influence of noise robust rewards on DSL.

In particular, we argue that noise robust rewards allow DSL to converge to an optimal joint strategy under agent-individual noise perceptions if there is only a single optimal joint strategy. In order to prove this, first consider the following lemma, which states that θ-noise robust rewards preserve game characteristics under agent individual noise rates, if θ corresponds to the largest individual noise rate over all participating agents:

Lemma 8. *Let $\Theta = \{\theta_0, \ldots, \theta_{n-1}\}$ be a set of n different agent-individual noise rates, and let $\theta^{\max} = \max_{\theta \in \Theta} \theta$ denote the maximal noise rate. Given a cooperative stage game $G = \langle \mathcal{A}, U, \rho \rangle$ with $|\mathcal{A}| = n$ agents and $\theta^{\max}$-noise robust rewards. Furthermore, assume that each joint action is executed at least once.*

Then, the game characteristics observed by each agent $i \in \mathcal{A}$ under its individual noise rate θ_i correspond to the original game. Formally, let $\rho(u_0) \leq \rho(u_1) \leq \ldots \leq \rho(u_x)$ define an ordering over all joint actions in U according to the noise-free reward. Then, each agent can deduce the same ordering from its noised reward perception.

Proof. Let $R = \{\rho(u) \mid u \in U\}$ be the set of all distinct reward values, which by assumption are $\theta^{\max}$-noise robust. Without loss of generality, consider an arbitrary agent $i \in \mathcal{A}$.

Let $\rho \in R$ be an arbitrary reward, and let $I = [(1 - \theta^{\max})\rho, (1 + \theta^{\max})\rho]$ denote the interval of possible values under the maximum noise rate. Then the interval denoting all possible noised perceptions of ρ for agent i under its individual noise rate θ_i is fully included in I, formally:$[(1 - \theta_i)\rho, (1 + \theta_i)\rho] \subseteq I$. Figure 8.5 gives an example for this using $R = \{1, 3, 8, 20\}$, a maximum noise rate of $\theta^{\max} = 0.4$, and an agent with noise rate $\theta_i = 0.2$.

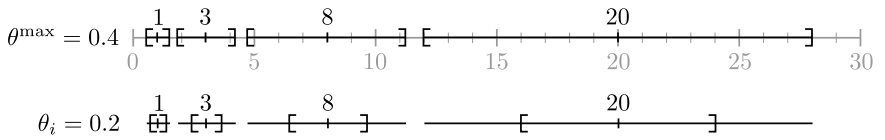


Fig. 8.5 Influence of different noise rates on an agent's individual noise perception.

Thus, independent from the actually observed noised reward, each agent can construct the same ordering of joint actions as defined in the noise-free game, if it observes each joint action at least once. □

Based on this result, we can show the mentioned convergence result:

Proposition 8. *Given a cooperative stage game $G = \langle \mathcal{A}, U, \rho \rangle$ with $\theta^{\max}$-noise robust rewards, where $\theta^{\max}$ is the maximal noise rate over all agent individual noise rates. Then, the* DSL *approach converges to a globally optimal joint strategy if:*

1. *the stage game G has only one optimal joint action, and*
2. *each joint action is executed infinitely often.*

Proof (Sketch). From the perspective of any agent $i \in \mathcal{A}$, it seems that i plays a single state noisy MDP. This follows a) from the fact that the agent has a noised reward perception and b) because the observed noised reward also depends on the actions of other agents that agent i is not aware off. According to the results for single agent learning in noisy MDPs, using the same update rule as in DSL, it hence follows that agent i's q-values will converge to the optimal q-values that are determined by the maximal noised observed rewards for each action (cf. Theorem 7 and its implications mentioned on page 100). Since the rewards are $\theta^{\max}$-noise robust, even the lowest noised reward value for the unique optimal joint action, i.e. that joint action that results in the highest reward, is larger than any other noised reward for any other joint action for each agent. Thus, each agent i finds its best action $a^\star$, i.e. the action that it has to execute according to the unique optimal joint action[1] in the game, it believes to play according to its individual noised reward perception.

Despite these individual game perceptions, we know from Lemma 8, that all agents agree on the same game characteristics, i.e. the ordering of all joint actions will be the same for all agents with respect to the achievable reward.

The proposition directly follows from these arguments and the fact that each joint action is played infinitely often.

To clarify the outlined argumentation, consider the example game with two agents shown in Tab. 8.10 and assume the noise rates to be $\theta_1 = 0.4$ and $\theta_2 = 0.2$. Note that, these rewards are 0.4-noise robust (cf. Fig. 8.5).

Table 8.10 Stage game G.

	A_1	B_1
A_2	1	3
B_2	8	20

Table 8.11 shows the possible noised reward observations according to the corresponding noise rates of both agents. Under the (extreme) assumption that all rewards occur only maximal positive noised except for the reward of the optimal joint action, which occurs with the maximal negative noised value only (12 resp. 16), Tab. 8.12 shows the resulting maximal observed individually noised rewards for all agent's actions. From the convergence results of single agent Q-Learning in noisy MDPs, and by using the optimistic update rule, i.e. the same rule as used in DSL, both agents will converge to the optimal action B. Obviously, the agents agree in the same game characteristics, i.e. both prefer action B over A. Hence, DSL converges to the unique optimal joint action under agent-individual noise. □

[1] Recall that an optimal joint strategy in a stateless setting reduces to an optimal joint action.

Table 8.11 Possible noise intervals for agent specific actions in stage game G shown in Tab. 8.10.

	Action A	Action B
Agent 1	$[0.6, 1.4] \cup [4.8, 11.2]$	$[1.8, 4.2] \cup [12, 28]$
Agent 2	$[0.8, 1.2] \cup [2.4, 3.6]$	$[6.4, 9.6] \cup [16, 24]$

Table 8.12 Maximal noised rewards observed by the agents for their individual actions in stage game G shown in Tab. 8.10.

	Action A	Action B
Agent 1	11.2	12
Agent 2	3.6	16

Accordingly, DSL can learn to solve a single cooperative stage game with an unique optimal joint action if rewards are noise robust. Given that we apply the ordinary CSSG-transformation function to create engineered rewards for a cooperative sequential stage game, DSL is able to learn in such cooperative settings, if each stage game contains an unique optimal joint action.

However, rewards in general are not noise robust. Hence, it is necessary to create such noise robust rewards based on the original rewards without changing the game characteristics. A simple general approach therefore is proposed in the following definition:

Definition 27 (Noise Robust Reward Creator). Consider a reward function $\rho : U \to \mathbb{R}$ that maps each joint action $u \in U$ to a real valued reward and a list L of all distinct reward values sorted in ascending order. Let $\theta \in [0, 1)$ be a noise rate and $z > \frac{1+\theta}{1-\theta}$ be a constant value. The mapping function $\text{INDEX} : \mathbb{R} \to \mathbb{N}$ returns the index of a reward value $\rho(u) \in \mathbb{R}$ in L starting at 0.

The *θ-noise robust reward creator function* $\rho_\theta : U \to \mathbb{R}$ then is defined by

$$\rho_\theta(u) = z^{\text{INDEX}(\rho(u))}, \forall u \in U.$$

Obviously, the relations between the original rewards sorted in list L and the created noise robust rewards are preserved. This follows due to the increasing power value. Next, it is left to prove that this creator function actually produces θ-noise robust rewards. This is done in Lemma 9.

Lemma 9. *Let $\theta \in [0, 1)$ be a noise rate, $z > \frac{1+\theta}{1-\theta}$ be a constant value, and $L = (z^0, z^1, \ldots, z^{n-1})$ be a list of n values that are powers of z. Then for noise rate θ and any pair of values $z^i, z^j \in L$ with $i \neq j$ it holds that the noise intervals of both values do not intersect, formally*

$$[(1-\theta)z^i, (1+\theta)z^i] \cap [(1-\theta)z^j, (1+\theta)z^j] = \emptyset.$$

Proof. First note that any two intervals of succeeding values z^i, z^{i+1} have an empty intersection if and only if $z^i + \theta z^i < z^{i+1} - \theta z^{i+1}$. If the intervals for z^i and z^{i+1} for $0 \leq i < n-1$ do not intersect, then there can also be no intersection for any other two values. Thus, it is sufficient to prove the property for an arbitrary succeeding value pair $z^i, z^{i+1} \in L$.

Without loss of generality, select an index i with $0 \leq i < n-1$ and subsequently consider the two values $z^i, z^{i+1} \in L$. Under the assumption on z as stated above, assume additionally that the noise intervals intersect. Then the following transformations hold, where the last step follows since $\theta \in [0, 1)$:

$$
\begin{aligned}
z^i + \theta z^i &\geq z^{i+1} - \theta z^{i+1} \\
z^i + \theta z^i &\geq z^i z - \theta z^i z \\
1 + \theta &\geq z - \theta z \\
1 + \theta &\geq z(1 - \theta) \\
\frac{1+\theta}{1-\theta} &\geq z
\end{aligned}
$$

Accordingly, the two intervals only intersect if $z \leq \frac{1+\theta}{1-\theta}$. However, $z > \frac{1+\theta}{1-\theta}$ holds as a precondition and, thus, the two intervals never intersect. The proof is completed since this is a contradiction to the assumption. □

An immediate conclusion from this lemma is that the noise robust creator function presented in Def. 27 creates noise robust rewards.

8.6.2.4 Reward Class Extension

The assumption that a cooperative stage game only has a single optimal joint action is quite restrictive and quickly becomes unrealistic in large games. Thus, we next briefly present a concept that enables DSL to learn in general cooperative stage games with multiple optimal joint actions.

The basic idea is based on reward classes. That is, if an agent perceives a noised reward, then the value's order of magnitude allows the agent to unambiguously decide to which class this reward belongs. Instead of learning from the noised reward, the agent learns from a numerical class identifier. Since these identifiers are crisp values, the usual convergence results for DSL hold. Before we clarify this approach using a simple example, we first present some more technical details.

In particular, we assume that rewards are obtained via the noise robust reward creator method defined in Def. 27. Given any resulting reward z^ℓ, the corresponding reward class is described by the value of ℓ. Given that an agent i observes a noised reward $\widetilde{\rho} \in [(1-\theta_i)z^\ell, (1+\theta_i)z^\ell]$, the reward class extension of DSL enables the agent to infer ℓ from $\widetilde{\rho}$. Then, the agent uses ℓ as reward value and applies the ordinary DSL approach. Note that, in order to calculate ℓ from $\widetilde{\rho}$, different techniques can be applied. A simple approach would assume that each agent knows z, such that each agent i can determine the corresponding reward class ℓ from $\widetilde{\rho}$ by iteratively calculating a value $v(\ell) = (1-\theta_i)z^\ell$ by increasing ℓ until $v(\ell) \leq \widetilde{\rho} < v(\ell+1)$.

Consider the following example to clarify the presented idea:

Example 5. *Assume two agents, 1 and 2, with noise rates $\theta_1 = 0.4$ and $\theta_2 = 0.2$ and the game shown in Tab. 8.13 that contains two optimal joint actions ($\langle A_1, A_2\rangle$ and $\langle B_2, B_2\rangle$). Table 8.14 shows a stage game that is obtained from G using the robust reward creator method defined in Def. 27 with $z = 3$. Note that $z > \frac{1+\theta_1}{1-\theta_1}$ holds, i.e. the created rewards are robust for the largest considered noise rate.*

Furthermore, assume that the agents play any action, e.g. $u = \langle A_1, B2\rangle$ which results in a crisp reward of 3. Due to the agent individual noise perception, agent 1 might observe a noised reward $\widetilde{\rho}_1(u) = 1.98 \in [(1-\theta_1)\cdot 3, (1+\theta_1)\cdot 3] = [1.8, 4.2]$ and agent 2 a reward of $\widetilde{\rho}_2(u) = 2.78 \in [(1-\theta_2)\cdot 3, (1+\theta_2)\cdot 3] = [2.4, 3.6]$. As the agents infer the reward class from their observation, consider for instance agent

Table 8.13 Original stage game G

	A_1	B_1
A_2	5	2
B_2	3	5

Table 8.14 Created stage game with noise robust rewards using reward classes.

	A_1	B_1
A_2	$9 = z^2$	$1 = z^0$
B_2	$3 = z^1$	$9 = z^2$

Table 8.15 Assumed stage game with reward classes.

	A_1	B_1
A_2	2	0
B_2	1	2

1 and assume that it uses the proposed iterative approach. The agent then calculates the following values:

$$v(0) = (1 - 0.4) \cdot 3^0 = 0.6$$
$$v(1) = (1 - 0.4) \cdot 3^1 = 1.8$$
$$v(2) = (1 - 0.4) \cdot 3^2 = 5.4$$

Accordingly, the reward class is $\ell = 1$ *as* $v(1) \leq \widetilde{\rho}_1(u) < v(2)$ *holds. The other agent will come to the same conclusion. Hence, both agents assume a crisp reward of* $\rho(u) = 1$ *as shown in Tab. 8.15.*

It follows that the agents assume to play the game shown Tab. 8.15, which has the same set of optimal joint actions as the original game presented in Tab. 8.13. The convergence to optimal solutions then trivially follows from the convergence behavior of DSL*, because the assumed game is an ordinary cooperative stage game.*

Finally, we want to highlight that noise robust rewards alone are not sufficient. The key additional idea is to use a special structure within the noise robust rewards, i.e. each reward has the form z^ℓ, such that all agents at the same time can deduce the same reward class from their individually noised observations.

8.6.3 Summary

In this section we investigated the behavior of the DSL approach under noisy reward perceptions. We basically showed that DSL is able to learn optimal or near-optimal solutions in cooperative stage games with agent-individual noised rewards under some assumptions. A comprehensive overview on the obtained results is presented in Tab. 8.16.

In some settings, noise robust rewards and particular structures are required to ensure convergence to optimal solutions. These properties can be obtained by applying special reward transformation functions. If these functions are applied as part of the CSSG-transformation function, the presented convergence results can be extended to cooperative sequential stage games, too.

8.7 Discussion

The main contributions of this chapter are a novel multiagent reinforcement learning approach for a new class of cooperative games with a special structure and its theo-

Table 8.16 Overview of the results for convergence towards optimal joint strategies in the DSL approach under noised reward perceptions. (Legend: ✓= yes, ×=no, ⊖=irrelevant)

Same, globally noised reward	Noise robust rewards	Reward classes	Convergence properties	Reference
✓	⊖	⊖	converges, if maximum positive noised rewards occur with non-zero probability	Proposition 6
×	×	⊖	no convergence to optimal joint action in general	Proposition 7
×	✓	×	converges, if the stage game has only one unique optimal joint strategy	Proposition 8
×	✓	✓	convergence to optimal joint strategy under usual assumptions	Example 5

retical analysis, including settings with noised rewards. In detail, the contributions include the following:

- We presented the Distributed Stateless Learning (DSL) approach for learning in cooperative sequential stage games (CSSG).
- We proved that DSL converges to optimal joint strategies for the last stage game of a CSSG, if each joint action is played infinitely often.
- It was shown that near-optimal solutions can be expected for each stage game if played sufficiently long.
- We provided several results for the behavior of DSL under agent-individual noised reward observations. It was shown that noise robust rewards allow agents to find (near-)optimal solutions if there is only one single optimal strategy. Given noise robust rewards with a special structure, we also showed that learning in general cooperative stage games under noise is feasible.
- We analyzed the space and time complexity of the proposed approach and showed its efficiency.
- The theoretical convergence properties have been underlined with an empirical analysis.

With the help of the DSL approach, we allow the agents in a cooperative MAS to learn a (near-)optimal joint strategy for the current situation without the need of observing all actions of the other agents. Since the approach realizes the concept of engineered rewards, our agents can further recognize new (global) situations without having to observe the entire system. Hence, the agents are able to adjust their strategies to any new situation without observing other agents or the system's state and to learn solely from the supplied global reward value. Under some assumptions, even a noised reward perception does not hinder the approach from learning optimal solutions. Given that the agents basically ignore their environment as well as other agents and solely learn from observed rewards, the DSL approach is highly flexible. For instance, it allows to drastically changing important parameters of a problem scenario, e.g. the number of agents, without the need of manually notifying or adjusting the learners. Also, because the agents do not maintain values for specific situations but only for the current situation, the approach is very space efficient.

Despite these positive properties of the DSL approach, we also want to point out the following two issues. First of all, the approach is able to learn in consecutive stage games due to the engineered rewards which ensure that rewards in a new situation are

always larger than any other previously seen reward. In general, this special reward structure usually is not given a priori but has to be "engineered", for instance, by using the described reward transformation functions. However, their application can become impractical, e.g., because the joint action space grows exponentially in the number of agents. Hence, one might decide to perform the transformation during the learning process, e.g. through a reward-giving central instance, instead of creating the engineered rewards as part of a preprocessing. Another approach would be to let the agents locally decide when to transform the rewards—an idea that we will pursue later in Sect. 9.2 and evaluate in Sect. 10.4.2 in the context of the IAPP.

The second issue is concerned with the speed of convergence. Given small problem settings as those evaluated in this chapter, the agents quickly will agree on an optimal joint action. However, in a large MAS that involves several hundreds or even thousands of agents, the number of possible joint actions grows too quickly. In that case, additional coordination techniques are required in order to improve the speed of convergence. We will present some techniques and evaluate them in Chapters 9, 10, and 11.

According to the results of this chapter, it is finally left to state, that DSL is a promising multiagent reinforcement learning approach for large cooperative systems.

Chapter 9

Coordination in Sequential Stage Games

Besides learning itself, coordination is another important aspect in multiagent systems (MAS). According to e.g. [Vla07, Ch.4] and [Bou99], coordination techniques in cooperative MAS can be based on social conventions (or laws), roles defined by subsets of actions, coordination graphs which decompose complex problems, communication, or learning.

The coordination strategies proposed in this chapter extend the learning approach introduced in the previous chapter. The goal of those strategies is to direct the agent individual action selection in order to quickly find a good joint action. All coordination strategies presented in the following sections can be used in any cooperative sequential stage game. However, some strategies need to be adjusted to match properties of the considered problem domain. We will investigate the proposed strategies in our Iterative Agent Partitioning Problem (IAPP) variant. In order to apply the Distributed Stateless Learning (DSL) approach in this context, we will also show how to model the IAPP as cooperative sequential stage game and propose three different ways of calculating a reward that can be used for learning.

The main contributions of this chapter are as follows:

- We propose and discuss five different coordination strategies.
- By showing how the IAPP can be modeled as sequential stage game, we provide a first useful and large-scale example for that class of games.
- We present three different reward concepts:
  - Rewards that are calculated and transformed by a central instance.
  - Rewards that are calculated and transformed locally, i.e. within the agents.
  - Rewards that are calculated by a central instance, but transformed locally.
- We show that locally calculated rewards, in our concrete IAPP implementation and also in general, cannot guarantee convergence to (near) optimal solutions.

The rest of this chapter is organized as follows. Section 9.1 introduces the developed coordination strategies in detail. Section 9.2 models the IAPP as cooperative sequential stage game and discusses the different reward calculation concepts. Finally, Sect. 9.3 briefly concludes this chapter.

9.1 Coordination Strategies

In this section, we provide detailed descriptions of the proposed coordination techniques. These methods realize coordination by influencing the action selection process within the DSL approach. Figure 9.1 illustrates some general characteristics of action selection strategies and classifies the proposed strategies.

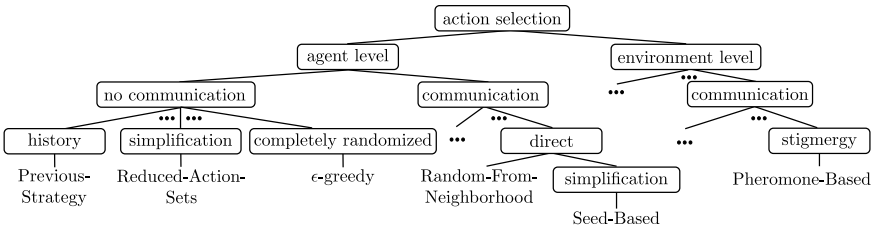


Fig. 9.1 Characteristics of the proposed action selection strategies.

The ideas of the following coordination strategies have been published in [KKB11b]. In contrast to that work, we provide a more detailed description including pseudocode and analysis. Furthermore, the three communication based strategies (Random-From-Neighborhood, Seed-Based, Pheromone-Based) are extended compared to those in [KKB11b] by exploitation of local knowledge after movement.

In order to ensure convergence according to the theoretical results in the previous chapter, the coordination strategies have to ensure that assumptions A2 (cf. page 111) are fulfilled. Assumptions A2.1 and A2.2 are not influenced by the action selection procedure and hence do not have to be considered here. In order to fulfill assumptions A2.3 and A2.4, it must at least be ensured that each action is selected with a non-zero probability. Otherwise, one neither can execute an action sufficiently often (A2.3) nor infinitely often (A2.4).

9.1.1 Pheromone-Based Strategy

The *Pheromone-Based Strategy (*PHEROMONE*)* is inspired by social insect colonies like termites and ants, or to be more precise, by a concept called *stigmergy* that can be found in such societies and which is often realized using so-called *pheromones*. In Sect. 2.4, we already provided a detailed overview on stigmergy and pheromones. However, the next paragraph briefly repeats necessary foundations to make sure that this section is consistent in itself.

The idea of stigmergy is to coordinate individuals via indirect communication through modifications in the environment [BDT99]. Ants, for instance, use this concept in order to find shortest paths between a nest and a food source. Each ant randomly wanders around and searches for a path to the food. Along its way it deposits a substance which is called *pheromone*. If an ant smells a strong pheromone trail it follows that trace with a higher probability compared to a trace with weaker pheromones. Besides reinforcing traveled paths, it is also important to note that pheromones evaporate over time. Since a shorter path can be traveled more often

in the same time span than a longer path, there will be more reinforcements on the shorter path. Thus, the pheromone concentration on that path increases, which in turn attracts more agents to follow that path. In the long run, the pheromones on longer paths will disappear and the ants will find a shortest path to the food. Thus, by modifying the environment and by reacting to changes, ants can agree on a shortest path without direct communication.

Inspired by this general idea, artificial pheromones stored in an environment in order to direct decision making have been intensively used in Ant Colony Optimization (ACO) algorithms (see e.g. [BDT99] or [DS04]). The proposed *Pheromone-Based Strategy* adopts the idea in the context of DSL. It introduces one pheromone type for each agent action. In contrast to the above described approach, pheromones do not evaporate automatically, but are decreased manually by agents (a similar local pheromone update is used in [DG97]). If an agent decides to place/update pheromones within a certain area of its environment, it reinforces the pheromone that corresponds to its lastly executed action by adding a constant value to the existing pheromone. All other pheromones are weakened according to a decay rate. The action selection of an agent is a modified ϵ-greedy approach, where the own strategy is followed with probability $1 - \epsilon$. With probability ϵ, the agent executes a random action that is determined using a proportional selection procedure (e.g. Roulette Wheel Selection [Eng05]) based on the different pheromone concentrations sensed at the agent's position. If the agent cannot sense any pheromones then it falls back and executes a nested ϵ-greedy action selection. Given that an agent has moved to a new position, it relies on the pheromone knowledge in the environment to choose its next action. Therefore, it follows the same process as in the aforementioned ϵ case.

According to the above general description, it can be concluded that the approach combines the individual learning process and the commonly influenced artificial pheromones in order to direct the exploration of the agents. Algorithm 3 summarizes this general approach.

Algorithm 3 Pheromone-Based Strategy (executed on each agent $i \in \mathcal{A}$)

Input: selection pressure α, probability $\epsilon \in (0, 1]$
Output: an action a to execute

```
1:  procedure PHEROMONEBASEDSTRATEGY
2:      if random value ∈ [0, 1] < ε or agent has moved then
3:          PI ← get pheromone information at agent position
4:          if PI ≠ ∅ then
5:              return ROULETTEWHEELSELECTION(α, PI)
6:          else                ▷ use ordinary ε-greedy strategy if there are no pheromone information
7:              action a ← σ_i
8:              if random value ∈ [0, 1] < ε then
9:                  a ← random action
10:             return a
11:     else
12:         return σ_i                                          ▷ current strategy
```

The exact way of how and where pheromones are stored and updated depends on the considered problem. In the context of our IAPP variant, we use a grid environment and thus consider a location based approach as shown in Alg. 4. In detail, there are different pheromone types φ_a, one type for each target selection action. Agents initially place pheromones for each of their own actions within a radius r around

their position. The initial strength (or value) of a pheromone φ_a for action $a \in A_i$ at position p is set to $v(\varphi_a, p) = \mathrm{p_{init}} > 0$. Whenever an agent updates pheromones, it reinforces the pheromone value for the lastly executed action a by adding a constant value $c \in \mathbb{R}^+$ to $v(\varphi_a, p)$ on all positions p within radius r. At the same time, the pheromone values of all other—not executed—actions $\hat{a} \in A_i \setminus \{a\}$ are weakened using a decay rate $\Delta \in [0, 1)$, such that the new value becomes $v(\varphi_{\hat{a}}, p) = (1 - \Delta)v(\varphi_{\hat{a}}, p)$. This update process corresponds to the generic ACO update rule as described for example in [DZMB02].

Algorithm 4 Pheromone update strategy (executed on each agent $i \in \mathcal{A}$)

Input: initial pheromone value $\mathrm{p_{init}} > 0$, reinforcement value $c \in \mathbb{R}^+$, pheromone decay rate $\Delta \in [0, 1)$

```
1:  procedure UPDATEPHEROMONES
2:      P ← all positions p within radius r around the agent
3:      for all position p ∈ P do
4:          PI ← get pheromone information at position p
5:          if PI = ∅ then
6:              for all a ∈ A_i do
7:                  v(φ_a, p) ← p_init          ▷ place initial pheromone φ_a for action a
8:          else
9:              for all φ_a in PI do
10:                 if action a decoded by φ_a corresponds to σ_i then
11:                     v(φ_a, p) ← v(φ_a, p) + c      ▷ reinforce current strategy's pheromone
12:                 else
13:                     v(φ_a, p) ← (1 − Δ)v(φ_a, p)      ▷ decay other pheromones
```

The action probabilities required to realize the Roulette Wheel Selection in line 5 of the general approach (Alg. 3) are calculated based on the pheromone information at an agent's position. In detail, an agent located at p calculates a probability for each of its actions $a \in A_i$ for which pheromone information are available. The probability is calculated according to (9.1), where α is a parameter that influences the selection pressure.

$$\Pr(a) = \frac{v(\varphi_a, p)^\alpha}{\sum_{\hat{a} \in A_i}^{\varphi_{\hat{a}}} v(\varphi_{\hat{a}}, p)^\alpha} \tag{9.1}$$

We have not yet exactly stated when pheromones are placed or updated. An obvious approach is to update pheromones in every iteration which results in a large overhead. Another approach is to perform updates only in special situations, e.g. if a better (joint) strategy is found. In the context of the IAPP, we know from Corollary 4 that all agents simultaneously perform strategy updates under some assumptions. Since the idea of using pheromones is to guide exploration towards good strategies, pheromone updates on improvements seem to be a good alternative compared to permanent updates. In [KKB11b], we compared the outcome of using updates only on improvement to updates in every iteration and found no significant difference in solution qualities. However, the number of updates in that work could in average be reduced from 45000 to 867, i.e. a reduction by 98.07%. Hence, we will only consider the "update on improvement strategy" in this thesis.

It is left to show that PHEROMONE fulfills assumptions A2.3 and A2.4. This follows, because all agents initially place pheromones with strength $\mathrm{p_{init}} > 0$ for all their actions. Due to the a decay rate $\Delta \in [0, 1)$, and a non-negative reinforcement

$c \in \mathbb{R}^+$, pheromone values always are > 0. Thus, there is a non-zero probability for each pheromone (action) of being selected in the Roulette Wheel Selection. Since $\epsilon > 0$, there also is a non-zero probability of using pheromones or executing an ϵ-greedy action selection. Accordingly, the assumptions required to ensure convergence are fulfilled.

The idea of using pheromones in reinforcement learning is not new. For instance, Chang [Cha04] proposes a pheromone-based exploration-exploitation strategy for single agent reinforcement learning. In contrast to our PHEROMONE approach, Chang's single agent creates virtual ants which walk through a graph-based model of an MDP and then backtrack to update the exploration-exploitation behavior of the agent.

9.1.2 Reduced-Action-Set Strategy

One way to reduce the search space during learning is to neglect actions that most probably will not be part of any optimal solution. The *Reduced-Action-Set Strategy (*RAS*)* realizes this idea by using a heuristic H that determines, which actions can be neglected. Based on the obtained reduced action set, the agent then executes an ϵ-greedy action selection. The frequency of using heuristic H to reduce the action set depends on the considered problem and can be controlled by firing an event. Algorithm 5 summarizes this approach.

In the context of the IAPP, we investigate a simple heuristic that reduces an agent's action set of size $|A_i|$ to some number $N < |A_i|$. Therefore, only the N actions that correspond to selecting the N nearest targets to an agent are part of the reduced action set. The action set can be refreshed with different frequencies in dynamic settings with movement. We will evaluate one approach that refreshes the action set whenever an agent has moved.

Algorithm 5 Reduced-Action-Set Strategy (executed on each agent $i \in \mathcal{A}$)

Input: heuristic H, probability $\epsilon \in (0, 1]$
Output: an action a to execute
1: **procedure** REDUCEDACTIONSETSTRATEGY
2: **if** not initialized **then**
3: $A_i^{\mathrm{RAS}} \leftarrow$ H.GETREDUCEDACTIONSET(A_i) ▷ initialize reduced action set
4: **if** refreshOnEvent and agent has moved since last call **then**
5: $A_i^{\mathrm{RAS}} \leftarrow$ H.GETREDUCEDACTIONSET(A_i) ▷ update reduced action set
6: action $a \leftarrow \sigma_i$
7: **if** random value $\in [0, 1] < \epsilon$ or $a \notin A_i^{\mathrm{RAS}}$ **then**
8: $a \leftarrow$ random action from A_i^{RAS}
9: **return** a

In contrast to the other coordination strategies in this chapter, the RAS strategy cannot ensure convergence to optimal solutions in general. Obviously, this follows since the neglected actions might be part of an optimal solution and also because the formal requirements, i.e. a non-zero probability of being executed, are not fulfilled for those neglected actions.

Conceptually, using only a subset of actions is related to assigning roles to agents. Designing such explicit roles can be used to facilitate the decision making process

given that agents know about the roles of other agents [Vla07, Ch. 4.3]. The RAS approach, in contrast, does not model such explicit roles, but simply reduces the number of actions.

In a student's work [Swa10] under our supervision, a similar strategy in the context of the (static) Online Partitioning Problem was considered. Like in RAS, the idea in that work also was to consider reduced action sets. In contrast to RAS, the action sets have been reduced dynamically, depending on the learning process. The obtained results indicate that dynamically adjusted action sets can help to improve the speed of learning.

9.1.3 Seed-Based Strategy

The *Seed-Based Strategy (*SEED*)* tries to stabilize the learning environment by allowing only a subset of all agents to explore, i.e. to try different actions, while the rest follows a fixed strategy. In detail, SEED is a communicative, phase-based approach. At the beginning of a phase, each agent decides to become a "seed agent" with probability $p_{\text{seed}} \in (0, 1]$. During the course of a phase, all non-seed agents do not explore by (randomly) executing different actions. Instead, they stick to their strategy to stabilize the learning environment. On the other hand, seed agents explore the learning environment by following the ϵ-greedy action selection method. Shortly before the end of a phase, which lasts $\mathrm{phase}_{\mathrm{Dur.}}$ iterations, all non-seed agents try to contact an arbitrary seed agent in their communication radius. If they can communicate with a seed, they request the seed's current strategy and execute it. Otherwise, they follow their own strategy again. Note that there is one case in which we allow all agents, in particular also non-seed agents, to change their behavior during a phase, namely if an agent has moved. In that case it behaves as if a phase is over and tries to follow the strategy of a seed agent nearby. Algorithm 6 summarizes this idea.

Algorithm 6 Seed-Based Strategy (executed on each agent $i \in \mathcal{A}$)

Input: number of iterations in a phase $\mathrm{phase}_{\mathrm{Dur.}}$, probabilities $\mathrm{p}_{\mathrm{seed}} \in (0, 1], \epsilon \in (0, 1]$
Output: an action a to execute

```
1:  procedure SEEDBASEDSTRATEGY
2:      if begin of new phase then
3:          become seed agent with probability p_seed          ▷ Synchronized with other agents
4:      if agent has moved or is not a seed agent then
5:          if is last iteration of phase or agent has moved then ▷ Phase lasts phase_Dur. iterations
6:              j ← arbitrary seed agent within communication radius
7:              if j ≠ NULL then
8:                  return σ_j                  ▷ Follow strategy of a random reachable seed agent
9:          return σ_i          ▷ Follow own current strategy if no seed reachable or phase not over
10:     else                                    ▷ If seed agent, then use ordinary epsilon greedy strategy
11:         action a ← σ_i
12:         if random value ∈ [0, 1] < ε then
13:             a ← random action
14:         return a
```

The communicative overhead of this strategy can be influenced by the length of a phase $\mathrm{phase}_{\mathrm{Dur.}}$ and the number of seed agents which depends on p_{seed}.

Clearly, SEED fulfills assumptions A2.3 and A2.4, because there is a non-zero probability that each agent can become a seed agent, and because seed agents perform an ϵ-greed action selection procedure with $\epsilon > 0$.

9.1.4 Random-From-Neighborhood Strategy

The *Random-From-Neighborhood Strategy (*RFN*)* is an extended ϵ-greedy action selection mechanism. Like in the ordinary ϵ-greedy strategy, each agent selects a random action with probability ϵ. Different to the ordinary ϵ-greedy strategy, an agent in RFN follows its own strategy with a probability of $p(1-\epsilon)$, and in the remaining cases, i.e. with probability $(1-p)(1-\epsilon)$ or if the agent has moved, it follows a strategy from a random reachable neighbor. If no neighbor is reachable, or if the neighbor also just has moved, then the agent again relies on its own strategy.

Hence for small ϵ, the probability $p \in [0,1]$ influences the communicative overhead and lets the agent either trust in its own strategy ($p \to 1$) or follow a neighbor's strategy ($p \to 0$). Given $p = 1$, the strategy is reduced to the ordinary ϵ-greedy strategy. Algorithm 7 gives a detailed overview on RFN.

Algorithm 7 Random-From-Neighborhood Strategy (executed on each agent $i \in \mathcal{A}$)

Input: probabilities $p \in [0,1], \epsilon \in (0,1]$
1: **procedure** RANDOMFROMNEIGHBORHOOD
2: $\mathrm{rnd} \leftarrow$ random value from $[0,1]$
3: **if** $\mathrm{rnd} < (1-p)(1-\epsilon)$ or agent has moved **then**
4: **if** there are reachable neighbors in communication radius **then**
5: **for all** reachable neighbor j **do**
6: **if** j has not just moved **then** ▷ rely on "static" agents
7: **return** current strategy of j
8: **return** σ_i ▷ follow own strategy if no or no "static" neighbors reachable
9: **else if** $\mathrm{rnd} < (1-\epsilon)$ **then**
10: **return** σ_i ▷ current strategy
11: **else**
12: **return** random action

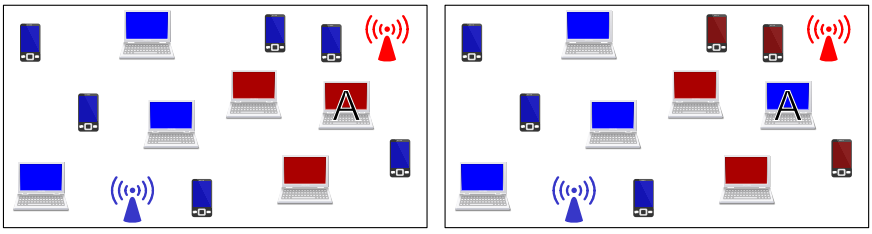
Since $\epsilon > 0$, agents will execute random actions with a non-zero probability. Accordingly, RFN fulfills assumptions A2.3 and A2.4 and hence preserves the convergence properties of the DSL approach.

9.1.5 Previous-Strategy

The *Previous-Strategy (*PS*)* is based on an observation made in the IAPP context. Let us consider Fig. 9.2 to briefly pinpoint that observation. As illustrated in Fig. 9.2(a), device A selected the red access point which is also the closest to A. Some of its neighbors, however, selected the blue access point, although it would be better to choose the red one according to the objectives of the IAPP. Now, consider the next

time step shown in Fig. 9.2(b). When A decided to deviate from its previous decision, the neighbors selected the best access point by coincidence. According to the IAPP objective function, the partitioning in this time step nevertheless is better than before. Thus, the DSL algorithm will force all agents to update their strategy. Particularly, the strategy of A will change from choosing the red access point to choosing the blue access point. Clearly, it changes from a globally optimal action to a suboptimal action, which just *seems* to be better due to the changed behavior of the other devices. In such situations, it hence can be beneficial to stick to the previous strategy with some probability. In that case, A might select the red access point in the next iteration, which might lead to further improvements. Hence, sticking to the previous strategy with some probability is the main idea behind the proposed strategy.

However, the success of this strategy is questionable. Although sticking to the previous action for A makes sense in the above scenario, its neighbors probably will do so, as well. Accordingly, the success of this strategy does not only depend on the likelihood of such scenarios, but also on the used probabilities.



(a) Iteration t. (b) Iteration $t+1$.

Fig. 9.2 Uncoordinated action selection.

In detail, each agent stores its previous strategy before it changes its strategy to the last action that led to a strategy update. This previous strategy is executed with a high probability (e.g. $p = 0.9$), given that a previous strategy already exists. Otherwise, with probability $1-p$, a ϵ-greedy action selection is executed and the agent follows its current strategy with probability $1-\epsilon$ and a random action with probability ϵ. Algorithm 8 summarizes this strategy.

Clearly, PS fulfills assumptions A2.3 and A2.4, since $p < 1$ and $\epsilon > 0$ lead to a non-zero probability for executing a random action.

Algorithm 8 Previous-Strategy (executed on each agent $i \in \mathcal{A}$)

Input: previous strategy $\widehat{\sigma}_i$, probabilities $p \in [0,1), \epsilon \in (0,1]$
Output: an action a to execute

```
1: procedure PREVIOUSSTRATEGY
2:     if random value ∈ [0, 1] < p then
3:         if σ̂_i ≠ NULL then
4:             return σ̂_i                          ▷ return previous strategy
5:     if random value ∈ [0, 1] < ε then
6:         return random action
7:     return σ_i                                   ▷ current strategy
```

9.2 Modeling the IAPP as Sequential Stage Game

Modeling an instance of our Iterative Agent Partitioning Problem (IAPP) variant as cooperative sequential stage game $\Gamma = \langle \mathcal{A}, U, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$ is straightforward. Each agent $i \in \mathcal{A}$ has one distinct action to select each target, i.e. $A_i = \{\text{select}_T^i \mid \forall T \in \mathcal{T}\}$, from which the joint action set U follows. The set of games $\mathcal{G}$ contains all games that occur within the k iterations of the IAPP. In detail, a new game occurs whenever the evaluation function for the same partitioning results in a different value, e.g. due to moving agents or if, e.g., target positions change. Accordingly, the number of repetitions of each stage game can be derived, e.g., from the length of the agents' jobs in the IAPP, and/or the frequency of changing settings. Based on this, the game transition function g follows. Note that any joint action realizes a partitioning $\boldsymbol{p}$ as defined in Def. 19 on page 62. Thus, the global common reward for any joint action in any stage game can easily be calculated using evaluation function (5.2).

It has to be ensured, that rewards are transformed such that the convergence requirements (see assumptions A2.2) are fulfilled and DSL can work correctly. In general, a preprocessing in order to generate engineered rewards like the transformation function proposed in Ch. 8 may be impractical in large systems like the IAPP. Instead, it would be nice to create engineered rewards that possess the desired properties online, i.e. while the agents are learning. Therefore, we will propose, discuss, and evaluate three different approaches in the remainder of this section.

9.2.1 Rewards

In this section, we identify and describe different concepts that can be used to generate rewards for learning in DSL. Although we describe these concepts in close relation to our IAPP variant, the general concepts can also be applied in other problem domains.

First, in Sect. 9.2.1.1, we propose a simple central reward calculation and transformation approach that preserves the theoretical convergence properties of DSL. We briefly discuss how a common global reward can be calculated given that there is no central instance that corresponds to *the environment* used in the schematic view on MARL. In Sect. 9.2.1.2, we present an alternative reward that is calculated individually by each agent and that relies only on (semi-)local information. Finally, in Sect. 9.2.1.3, we propose another approach that uses the globally calculated, but not transformed rewards, such that agents individually transform these rewards according to their own local observations.

9.2.1.1 Global Reward

The concept of a *global reward* is defined by a central instance that is able to calculate and transform a common reward that subsequently can be observed by all agents.

In our IAPP variant, a concrete implementation of this concept looks as follows. Given a partitioning $\mathcal{S}_{\mathcal{A},\mathcal{T}}^t$ at iteration t. Assume a central instance that is capable of calculating the partitioning quality $f(\mathcal{S}_{\mathcal{A},\mathcal{T}}^t)$ in each iteration according to equation (5.2). This partitioning quality value can be used as reward for the joint action which led to the partitioning. In order to fulfill the convergence requirements, however, the

reward must be transformed to reflect changing scenarios that occur if agents move. Let $L = \langle t_0, t_1, t_2, \ldots \rangle$ be a list of iterations at which at least one agent has moved. The transformed reward $\bar{\rho}_t$ at iteration t with $t_c \leq t < t_{c+1}$ and $t_c, t_{c+1} \in L$ is given by $\bar{\rho}_t = f(\mathcal{S}^t_{\mathcal{A},\mathcal{T}}) + c \cdot \Phi$, where $\Phi \geq 1$ is a positive transformation additive. Hence, the basic idea of the transformation is to increase the original partitioning quality value using an additive value that grows with every new scenario (resp. stage game). Since $f(\mathcal{S}^t_{\mathcal{A},\mathcal{T}}) \in [0, 1]$, we will use $\Phi = 1$ in the empirical evaluation presented in Ch. 10.

This globally transformed reward preserves the convergence guarantees of DSL, as it fulfills the assumptions stated in A2.2: First note that all agents observe the same bounded positive reward for any sequential stage game with less than infinitely many stage games. Secondly, this reward is engineered such that the actual reward in any stage game is larger than any other reward observed in a previous stage game due to its additive construction.

Generally, however, there is usually no computational central instance in a MAS that could calculate a global reward. Thus, a decentralized approach is desirable. One way to obtain a locally calculated global reward is to use a simple yet intensive communication protocol. Therefore, each agent broadcasts its current assignment and its target distances to all other agents. With the help of these information, each agent can calculate the (real) global reward value by itself. However, this strategy makes several assumptions, e.g. arbitrarily large storage capacity at each agent or a fully connected communication graph, which might be hard to realize. More sophisticated communication based protocols can help to estimate the global partitioning quality. In a bachelor's thesis under our supervision, we evaluated different communication protocols that might be used to estimate the partitioning quality [Kli10]. Based on this evaluation, the student implemented two different algorithms and evaluated them in a static IAPP scenario. The hypothesis of the first approach states that the partitioning quality can be approximated if each agent has information from a *well* chosen randomly distributed subset of agents. In order to realize this approach, a *gossip-based* protocol is used as these protocols can generate such random local subsets (cf. e.g. [KDG03] [JGKvS04] [JVG+07] [KvS07]). Depending on different factors, like the actual distribution of agents, the simulation results revealed a good approximation quality which showed a constant deviation from the real partitioning quality. The second approach uses a sophisticated hierarchy-based protocol inspired by the work of Heinzelman et al. [HCB00]. In this approach, agents determine cluster heads and submit their partitioning information to these cluster heads. The latter in turn compactly summarize these information and submit them to a (central) cluster head. This super-agent calculates the overall partitioning quality and submits the resulting approximation value to the cluster heads, which finally redistribute the value to all agents. The empirical evaluation showed an optimal approximation of the actual partitioning quality. Since both protocols still are communication intensive and have several requirements, we do not want to go into detail here. Instead, we refer the reader to [Kli10]. However, it has to be noted that one might lose the convergence guarantees of the DSL approach if approximated solution qualities are used as basis for rewards.

9.2.1.2 Local Reward

The concept of a *local reward* is defined by agents that calculate and transform an individual reward based on their own local observations.

An implementation of this concept in our IAPP variant is presented next. It requires each agent to communicate only with its current target, rather than to interact with other agents. First, an agent $i \in \mathcal{A}$ calculates a local approximation $\widehat{f}^i_{\text{loc.}}(\mathcal{S}^t_{\mathcal{A},\mathcal{T}})$ of the partitioning quality for a partitioning $\mathcal{S}^t_{\mathcal{A},\mathcal{T}}$ described by partitioning function $\boldsymbol{p}^t : \mathcal{A} \to \mathcal{T}$ according to

$$\widehat{f}^i_{\text{loc.}}(\mathcal{S}^t_{\mathcal{A},\mathcal{T}}) = \alpha \cdot \text{DISTRIBUTION}(\boldsymbol{p}^t(i)) + \beta \cdot \text{DISTANCE}(\boldsymbol{p}^t(i)) \tag{9.2}$$

using the same weight factors α, β as in the globally calculated reward and having

$$\text{DISTRIBUTION}(\boldsymbol{p}^t(i)) = 1 - \frac{\text{\# agents at target } \boldsymbol{p}^t(i)}{|\mathcal{A}|} \tag{9.3}$$

as local DISTRIBUTION objective value and

$$\text{DISTANCE}(\boldsymbol{p}^t(i)) = 1 - \frac{\boldsymbol{\delta}(i, \boldsymbol{p}^t(i))}{\sum_{T' \in \mathcal{T}} \boldsymbol{\delta}(i, T')} \tag{9.4}$$

as local DISTANCE objective value. In order to calculate the distribution value, it has to be ensured that each agent registers itself at its currently selected target, such that the target has information about the number of clients. It is also required that all agents know the total number of agents in the system.

The approximated distance value can be calculated since each agent knows its distances to the targets. Furthermore, (9.4) directly represents the DISTANCE objective, since the closest target to agent i will result in the highest reward. The approximated distribution value, in contrast, requires more knowledge on the global state in form of the *number* of agents that are registered at the agent's current target. Note that this information is quite abstract because it only describes the number of agents at one target and does not include any additional global information (e.g. which agent selected this target, or how many agents selected other targets)[1]. From (9.3), it follows that each agent would obtain the highest DISTRIBUTION value if it is the only client at its target. Indeed, all agents have to select a target, which implies that the highest DISTRIBUTION value for each agent is achieved if agents are equally distributed to the targets. Accordingly, the proposed local approximation of the partitioning quality reflects the properties of the global evaluation function (5.2) in a good way.

Based on this local approximation of the partitioning quality, the agent can construct a reward for learning. Therefore, and in order to properly enable learning using the DSL approach, it should additionally ensure to transform the reward if a new

[1] In general, we argue that one cannot optimize the DISTRIBUTION objective without additional knowledge on the current system state. Hence completely locally calculated rewards are implausible. Instead, one should speak about *semi-local rewards* in order to reflect the required *abstracted* global distribution knowledge. This corresponds to observations in real life situations. Consider, for instance, the task of dividing a group of pupils into equally sized teams in a sports class. In this case, each pupil does not necessarily have complete knowledge about the actual numbers, but he gets an impression on the different team sizes. This abstracted global knowledge helps him to select a group to join. Without this abstract distribution knowledge, pupils can hardly manage to estimate the current team sizes.

stage game arises. For this transformation, we adapt the same approach as described in the previous section for the global reward. The only difference is that each agent i maintains its own transformation additive Φ_i and increases it by 1 whenever the agent itself has moved.

Since we will use this local reward calculation in the upcoming experiments in Ch. 10, let us briefly show that the DSL approach loses its convergence guarantees under such rewards. Therefore, have a look at the following example:

Example 6. *Consider the scenario shown in Fig. 9.3. Let us assume weight factors* $\alpha = \beta = 0.5$ *for the global reward function according to (5.2), a discount factor of* $\gamma = 0.0$*, and that we use* DSL *in its basic form with the* ϵ*-greedy action selection.*

The resulting local DISTANCE *values are shown in Tab. 9.1. Table 9.2 presents the local* DISTRIBUTION *values and also the resulting local rewards for all possible partitionings.*

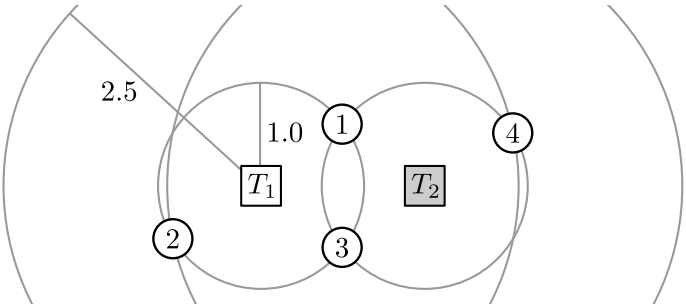


Fig. 9.3 Scenario.

Table 9.1 Local DISTANCE$(\boldsymbol{p}(i))$ value for the example shown in Fig. 9.3.

Agent i	$\boldsymbol{p}(i) = T_1$	$\boldsymbol{p}(i) = T_2$
1	0.50	0.50
2	0.71	0.29
3	0.50	0.50
4	0.29	0.71

From the bold values in Tab. 9.2, one can deduce the following list of optimal actions for the agents according to their maximum local rewards:

Agent 1: T_1, T_2
Agent 2: T_1
Agent 3: T_1, T_2
Agent 4: T_2

From this list, four different joint actions can be constructed, among which only two are optimal according to the global reward function. Table 9.3 shows these joint actions.

Table 9.2 Local DISTRIBUTION($\boldsymbol{p}(i)$) value for all possible assignments (states) in the example of Fig. 9.3. For brevity, DB(i) stands for DISTRIBUTION($\boldsymbol{p}(i)$) and state $(1,2,3,4) = (T_2, T_1, T_1, T_1)$ means that agent 1 is assigned to Target T_2, agent 2 to T_1, etc. $\widehat{f}^i_{\text{loc.}}$ stands for the local reward (according to (9.2)), that agent i gets for the respective state using the distance values from Tab. 9.1. Rows marked with ★ correspond to optimal solutions according to (5.2).

State i $(1,2,3,4)$	Agent 1		Agent 2		Agent 3		Agent 4		Global reward acc. to (5.2)	
	DB(1)	$\widehat{f}^1_{\text{loc.}}$	DB(2)	$\widehat{f}^2_{\text{loc.}}$	DB(3)	$\widehat{f}^3_{\text{loc.}}$	DB(4)	$\widehat{f}^4_{\text{loc.}}$		
(T_1,T_1,T_1,T_1)	0.00	0.25	0.00	0.36	0.00	0.25	0.00	0.14	0.36	
(T_1,T_1,T_1,T_2)	0.25	0.38	0.25	0.48	0.25	0.38	0.75	**0.73**	0.88	
(T_1,T_1,T_2,T_1)	0.25	0.38	0.25	0.48	0.75	**0.63**	0.25	0.27	0.74	
(T_1,T_1,T_2,T_2)	0.50	0.50	0.50	0.61	0.50	0.50	0.50	0.61	1.00	★
(T_1,T_2,T_1,T_1)	0.25	0.38	0.75	0.52	0.25	0.38	0.25	0.27	0.66	
(T_1,T_2,T_1,T_2)	0.5	0.5	0.5	0.39	0.5	0.5	0.5	0.61	0.86	
(T_1,T_2,T_2,T_1)	0.5	0.5	0.5	0.39	0.5	0.5	0.5	0.39	0.79	
(T_1,T_2,T_2,T_2)	0.75	**0.63**	0.25	0.27	0.25	0.38	0.25	0.48	0.74	
(T_2,T_1,T_1,T_1)	0.75	**0.63**	0.25	0.48	0.25	0.38	0.25	0.27	0.74	
(T_2,T_1,T_1,T_2)	0.50	0.50	0.50	0.61	0.50	0.50	0.50	0.61	1.00	★
(T_2,T_1,T_2,T_1)	0.50	0.50	0.50	0.61	0.50	0.50	0.50	0.39	0.86	
(T_2,T_1,T_2,T_2)	0.25	0.38	0.75	**0.73**	0.25	0.38	0.25	0.48	0.88	
(T_2,T_2,T_1,T_1)	0.50	0.50	0.50	0.39	0.50	0.50	0.50	0.39	0.79	
(T_2,T_2,T_1,T_2)	0.25	0.38	0.25	0.27	0.75	**0.63**	0.25	0.48	0.74	
(T_2,T_2,T_2,T_1)	0.25	0.38	0.25	0.27	0.25	0.38	0.75	0.52	0.66	
(T_2,T_2,T_2,T_2)	0.00	0.25	0.00	0.14	0.00	0.25	0.00	0.36	0.36	

Table 9.3 Possible joint actions.

Joint action	Global reward
$\langle T_1,T_1,T_1,T_2\rangle$	0.88
$\langle T_1,T_1,T_2,T_2\rangle$	1.00
$\langle T_2,T_1,T_1,T_2\rangle$	1.00
$\langle T_2,T_1,T_2,T_2\rangle$	0.88

To which joint action DSL *converges depends on the order of experiencing the maximum local reward at each agent. Accordingly, this example has shown that* DSL *is not guaranteed to converge to an optimal solution under the constructed local rewards.*

Basically, this example illustrates the inherent coordination problem in learning MAS. This problem also occurs if a common reward function is used by a set of agents as described, e.g., in [BBDS08].

Besides from the above stated specific example, the loss of the convergence properties under general (semi-)local rewards follows indirectly from Proposition 7. To clarify this, note that agents that use such a (semi-)local reward most probably will observe individually different rewards at some points in time. The same argument also holds under the settings stated in Proposition 7. Only the reasons for observing individually different rewards differ, once it is due to noised rewards and once due to local reward calculation.

9.2.1.3 Locally Transformed Global Reward

The concept of *locally transformed global (LTG)* rewards is based on a central instance that calculates a common global reward, which is observed by all agents. The agents use this common reward and individually decide, based on own local observations, when and how to create an engineered reward from the given global value. The advantage of this method is that the environment can calculate an ordinary reward and does not need to realize the transformation, which is solely a part required by the DSL approach. Accordingly, this concept requires fewer assumptions on the environment.

In our concrete implementation, we use the same global reward as described in Sect. 9.2.1.1, i.e. we use evaluation function (5.2) to calculate a common global reward. This value is not transformed and given unmodified to the agents. Agents locally transform the received value to reflect changes according to their own local perceptions. Therefore, like in the local reward approach in the previous section, each agent uses a constant that is added to the clear global reward value. If the agent has moved, the additive is increased by 1.

By the same arguments as in the previous section on local rewards, DSL loses its convergence guarantees under locally transformed global rewards.

9.3 Summary

The purpose of this chapter was the presentation of different coordination techniques that can be used for action selection in the Distributed Stateless Learning approach. We described the ideas of five coordination techniques and presented detailed pseudocodes for each of them. In addition, we provided a first large-scale application of the cooperative sequential stage game model by presenting a concrete implementation of our IAPP variant. We also proposed three general concepts for the calculation of rewards in the context of DSL and provide concrete implementations for our IAPP variant. Furthermore, we showed that locally calculated rewards cannot preserve the theoretical convergence guarantees of the DSL approach.

Chapter 10

Empirical Analysis

In this chapter, we empirically[1] analyze the Distributed Stateless Learning (DSL) approach and the proposed coordination techniques in the context of our Iterative Agent Partitioning Problem (IAPP) variant.

The main contributions of this chapter are as follows:

- We provide an exhaustive empirical analysis of the DSL-based approaches presented in Ch. 9.
- DSL-based approaches do not only work well in small problems with two agents, but can also successfully be applied in large MAS involving thousands of agents with huge joint action spaces.
- Additional coordination techniques can significantly improve the performance of the basic learning approach. Particularly, communicative strategies are successful.
- Although sequential stage games "remove the state" from the learning system, DSL still suffers from the (conceptual) curse of dimensionality in the sense that it can be found in joint action spaces rather than in state-joint-action spaces.
- Simulations confirm the theoretically proven convergence properties of DSL.
- Although locally calculated rewards cannot guarantee convergence to (near) optimal solutions in general, we show that they still result in high quality solutions in practice, which seem to be at least close to optimal in the considered settings.
- We showed that, in order to enable learning in large dynamic systems, agents require, besides time, a learning environment that is somewhat stable in the sense that not too many agents explore it at the same time.
- DSL-based approaches are able to outperform a special purpose state-of-the-art algorithm in the IAPP domain.

The chapter is organized as follows. In Sect. 10.1, we present the general settings used for the simulations if not stated otherwise. We analyze the BASIC DSL approach in our IAPP variant with many agents in Sect. 10.2. Section 10.3 provides some insights into the coordination techniques proposed in the previous chapter. It then follows Sect. 10.4, which analyzes the coordination techniques under three different reward concepts and under three different job generation patterns. Finally, some selected experiments in different scenarios are presented in Sect. 10.5, before we conclude this chapter in Sect. 10.6.

[1] The simulation tool is available for download [Kem12].

10.1 Settings

Clearly, an empirical analysis can only provide hints on how the subject under investigation behaves in general. Nevertheless, appropriate settings should be investigated to obtain valuable insights. Based on lots of manual experiments, we hence decided to use the following settings as a basis for our investigations. The chosen parameters reflect a non-trivial setting that is already large, but which does not require too much computational time. For instance, since the problem can be solved for two targets but no efficient algorithm for three targets is known, we decided to consider at least three targets. In addition, we will extend this basic scenario to investigate other settings whenever we believe it to be appropriate.

In detail, and if not stated otherwise, we will use the following basic parameter values. We consider $n = 500$ agents, $m = 3$ targets, a grid environment with 50×50 cells, weight factors $\alpha = \beta = 0.5$ for the evaluation function (5.2), $k = 6000$ iterations, and a discount factor $\gamma = 0.0$. Each agent i considers the nearest $\mathsf{k} = 8$ agents within its communication radius $r_i = 5$ as neighbors. Random numbers are chosen uniformly at random from their corresponding intervals. The presented results are mainly average values over 30 repetitions for each experiment with different random seeds. We will usually provide graphs based on these averages and do not provide additional confidence intervals or error bars. Based on our experience, we can state that the averages over 30 repetitions already well reflect the properties of the curves. More repetitions mainly lead to smaller confidence intervals. As an example consider Fig. 10.1, which illustrates confidence intervals for BASIC DSL with different numbers of repetitions.

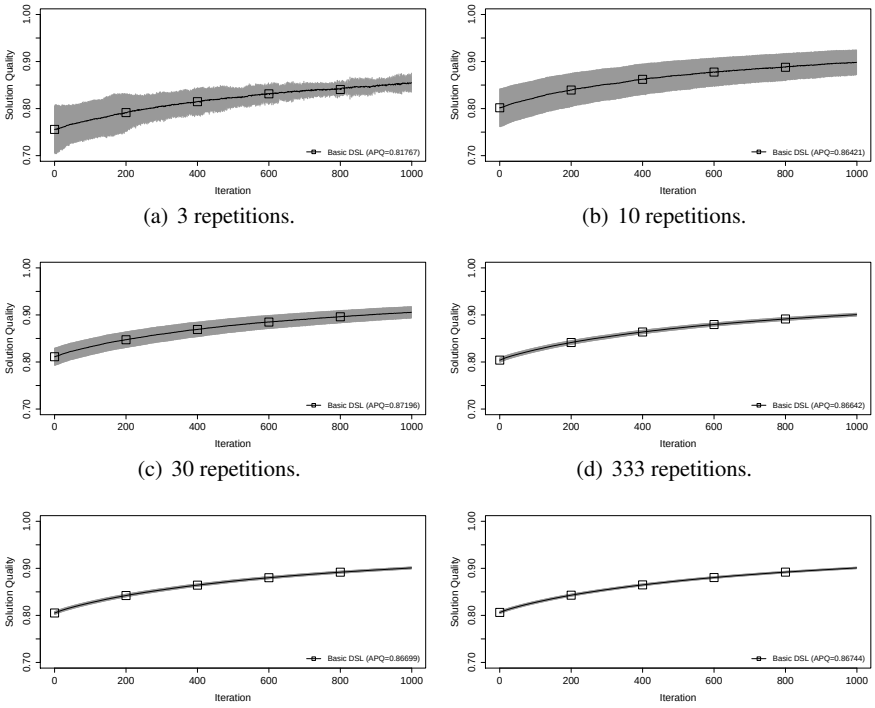
In our empirical analysis we will combine different reward calculation strategies with different job generation patterns. The latter can basically be divided into two classes:

1. Random (R): Jobs are created at random positions in the environment.
2. Circle (C): Given an agent's old job at position p, the agent's new job will be located at a random position within a circle with radius r centered at p.

Let $\mathrm{R}(x, y)$ describe a random job generation with jobs that have a random duration drawn from the interval $[x, y]$, and let $\mathrm{R}(x)$ denote the random strategy, where all jobs have a duration of x, i.e. $d_{\min} = d_{\max} = x$. Identifiers for the circle job generation approach are defined correspondingly. Throughout the evaluation, we will use a radius of $r = 5$ for the circle job generation strategy.

Due to the vast number of possible parameter combinations that arise from different scenarios settings (e.g. number of agents, targets, job generation strategies) and algorithm related parameters (e.g. reward types, probabilities, pheromone values, etc.), a comprehensive analysis is a time consuming and tedious task. Thus, we constructed a set of different scenarios which we believe to lead to an appropriate analysis of the DSL approach and the different coordination strategies. We will compare our results to the state-of-the-art Exchange Target Strategy (ETS) and to a simple approach called Random Target Strategy (RTS) [Goe07], in which every agent in each iteration simply selects a random target.

For the simulations, our framework assures that agents execute their actions in a random order in each iteration. This assumption reflects characteristics of real distributed systems better than a fixed ordering, which might even be harder to realize in large systems. This particularly holds for communicative strategies, as there generally is no guarantee for receiving messages in a predefined order.



(a) 3 repetitions. (b) 10 repetitions.

(c) 30 repetitions. (d) 333 repetitions.

(e) 666 repetitions. (f) 1000 repetitions.

Fig. 10.1 Influence of the number of repetitions to the graph for BASIC DSL. Gray areas correspond to confidence intervals with confidence level of $\alpha = 0.05$.

10.2 Basic Distributed Stateless Learning

We begin our evaluation with the standard BASIC DSL approach that uses the ϵ-greed action selection and the global reward transformation.

10.2.1 Exploration Probability

As we have already seen in Fig. 8.4 in Sect. 8.4, the performance of BASIC DSL in a simple two-agent CSSG depends on ϵ. To determine the impact of ϵ in our IAPP setting, we investigate the basic approach for $\epsilon \in \{0.001, 0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9\}$. Therefore, we will consider the behavior in a static scenario without changing jobs and another slightly more dynamic setting that uses $\mathrm{R}(10000)$ for the creation of jobs.

Clearly, given too high ϵ values, the learning environment will be unstable, as there is a high probability for many agents to execute a random action instead of following their current strategy. Accordingly, we expect low solution qualities for high ϵ values. However, with decreasing ϵ values this instability should diminish due to a smaller number of exploring agents. In that case, many agents will stick to

their currently best strategy and, since only a small number of agents explores, the probability of finding a better joint strategy should increase.

The results shown in Figs. 10.2 and 10.3 support these hypotheses and outline the importance of ϵ. In particular, it has a strong impact on the convergence speed. Also, it seems to influence the level of convergence.

Although the results show the highest solution quality (not the highest average solution quality) for $\epsilon = 0.001$ in Fig. 10.2, one can see a disadvantage of this quite small value in dynamic settings like those evaluated in Fig. 10.3. A smaller ϵ-value leads to a slow increase in solution qualities, which leads to a low average partitioning quality. Thus, we will use $\epsilon = 0.01$ for all further experiments, since it quickly results in higher values than $\epsilon = 0.001$, but still achieves a comparable maximum solution value.

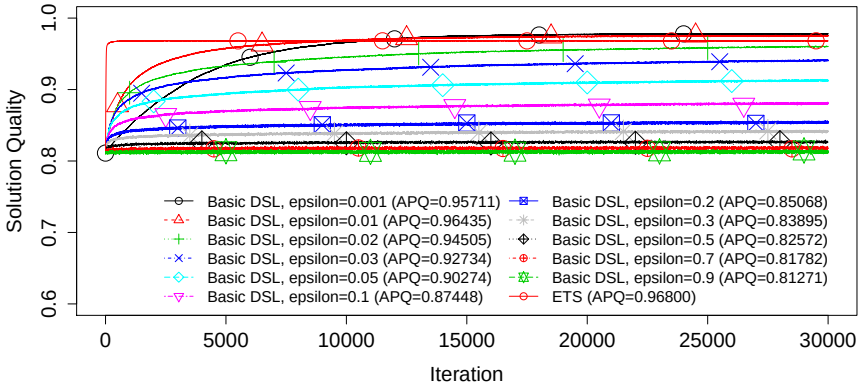


Fig. 10.2 BASIC DSL with varying ϵ values in a single stage game with 30000 iterations.

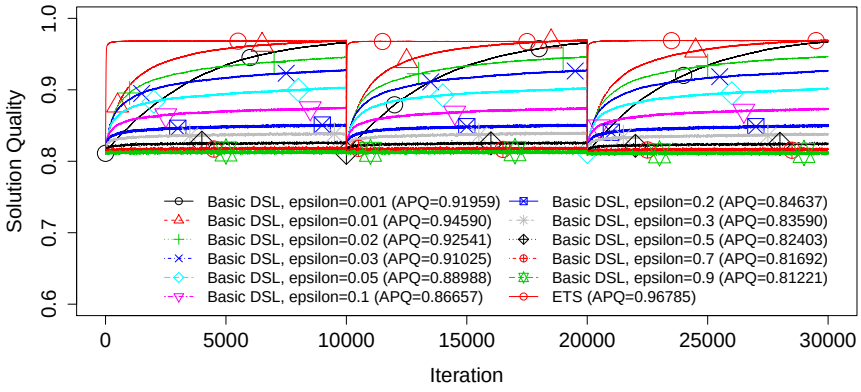


Fig. 10.3 BASIC DSL with varying ϵ values and random movement at the end of each 10000 iteration lasting stage game.

10.2.2 Discount Factor

In general MARL approaches, the discount factor γ is used to rate the value of future rewards, i.e. receiving a future reward is worth only parts of what it is worth on immediate reception [SB98]. In a sequential stage game, however, agents do not know when a stage game ends. Accordingly, they can be myopic and try to maximize their immediate reward ($\gamma \to 0$).

Figure 10.4 shows the development of solution qualities for a single stage game with fixed agent positions and varying discount factors $\gamma \in \{0.0, 0.1, 0.5, 0.7, 0.9\}$. As we can see, a low discount factor leads to a better average partitioning quality.

Based on this result, and since agents in general do not know how long a stage game lasts, we will set the discount factor to $\gamma = 0.0$ in all upcoming simulations.

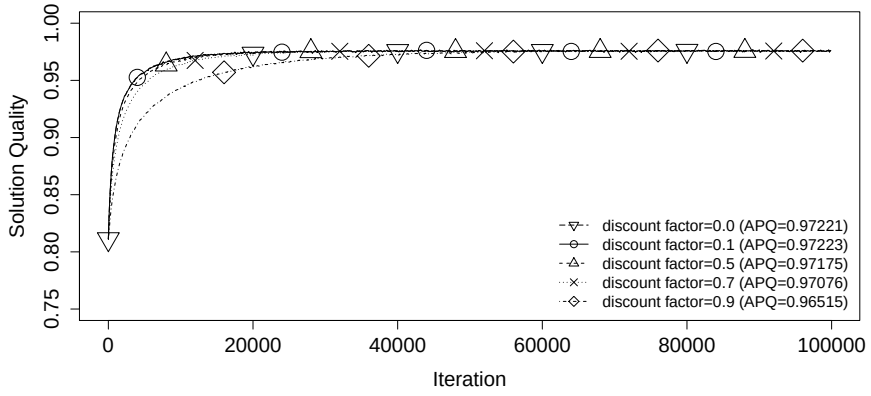


Fig. 10.4 Influence of the discount factor in a static scenario ($\mathsf{APQ}(100)$[2]).

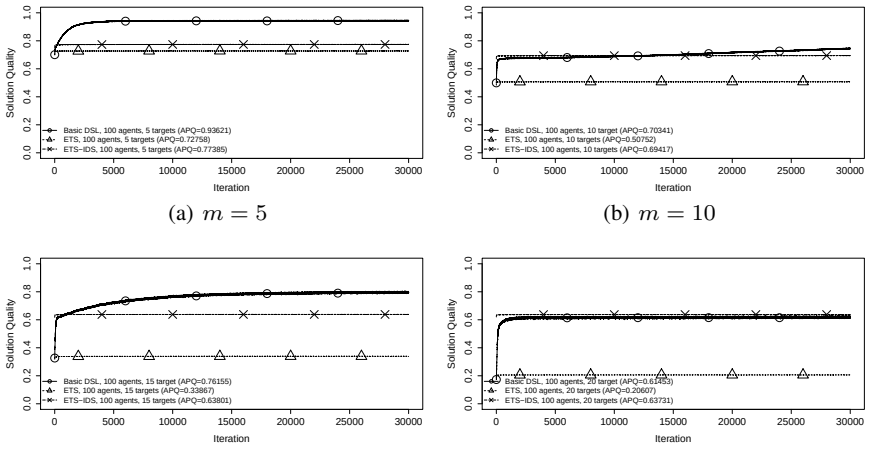
10.2.3 Robustness

In this section, we intend to show that BASIC DSL is able to learn good solutions under settings, where state-of-the-art ETS performs poorly. In detail, we consider scenarios with different agents per target ratios using $n = 100$ agents and $m \in \{5, 10, 15, 20\}$ targets. As we have already seen in the analysis of the solution quality function in Fig. 5.6 in Sect. 5.3.1, a low agent per target ratio leads to quickly declining distribution values for non-uniform partitionings. Thus, we can expect decreasing solution qualities for an increasing number of targets. For ETS with random target initialization this means that, due to the low agents per target ratio, there is a high probability of having a non-uniform partitioning. Since ETS does not

[2] As a tradeoff between information gain and computational time, some of the average partitioning qualities values (APQ) and the graphs are not based on the partitioning quality of every single iteration. Whenever not each iteration is considered, we mark the corresponding results with a short hint in the form of $\mathsf{APQ}(x)$, where x denotes that only every x-th iteration was used to calculate the APQ-value, resp. to plot the graph.

change the initial distribution value, as it only optimizes the distance, it will most likely result in low quality solutions under such settings.

Figure 10.5 shows our simulation results. There, we compare our approach to the ordinary ETS with random target initialization and to an ETS variant proposed in [Goe07] that uses an ID-dependent initialization strategy (IDS). Note that IDS ensures an optimal DISTRIBUTION value, but it requires consecutively numbered agents which is a non-trivial problem in large systems (cf. e.g. [OAVBFR09] for the unique ID assignment problem in Wireless Sensor Networks). As we can see in the figure, our learning approach finds significantly better partitionings than ordinary ETS. DSL also finds better solutions compared to ETS-IDS for $m \in \{5, 10, 15\}$. Given 20 targets, ETS-IDS outperforms DSL within the simulated number of iterations. But, from our theoretical analysis, we know that DSL will be able to find a solution as least as good as ETS-IDS given more iterations. In contrast to ETS-IDS, however, our approach does not require consecutively numbered agents. Thus, it is more robust if the agent set is changed during runtime, e.g. if agents break down or new ones are added.



(a) $m = 5$ (b) $m = 10$

(c) $m = 15$ (d) $m = 20$

Fig. 10.5 Influence of different agents per target ratios using $n = 100$ agents and $m \in \{5, 10, 15, 20\}$ targets.

10.2.4 Dynamic Scenarios

Next, we consider the behavior of BASIC DSL in dynamic settings with $\epsilon = 0.1$. We use $\mathrm{R}(5, 100)$ and $\mathrm{C}(5, 100)$ to generate jobs, such that it is almost certain to have at least one moving agent per iteration after the first five iterations are over.

Table 10.1 shows the resulting average partitioning qualities and compares them to the Random Target Strategy (RTS). Apparently, our approach is unable to learn in such highly dynamic settings. The reason for this behavior is obvious: since there is at least one moving agent per iteration, each iteration corresponds to a new stage game.

Due to the global reward transformation, all rewards are larger than the rewards in the previous game. Thus, no matter which action an agent performs, it will obtain steadily increasing rewards. Accordingly, the learning degenerates to a random action selection. Support for this argument can be concluded from the corresponding results of the RTS approach.

Table 10.1 Average partitioning qualities of BASIC DSL and RTS in highly dynamic scenarios.

Approach	Job Generation	APQ
BASIC DSL	$C(5, 100)$	0.80975
	$R(5, 100)$	0.81045
RTS	$C(5, 100)$	0.80994
	$R(5, 100)$	0.81054

10.2.5 Changing Number of Agents and Target Positions

In this section, we investigate how BASIC DSL behaves when the number of agents changes from stage game to stage game and, in addition, targets also change their positions. Therefore, we consider a scenario with 30000 iterations that changes significantly every 10000 iterations. During the first 10000 iterations, only 100 agents and three targets are located at fixed positions in the environment. In iteration 10000, 300 additional agents are added to the system and the targets swap their IDs, i.e. all targets change their positions. After another 10000 iterations, another additional 200 agents are added and the targets again change their positions by swapping IDs. Note that once an agent has been placed in the environment, it is not allowed to move. We investigated this scenario with BASIC DSL and a globally transformed reward. Furthermore, we also considered the behavior of the algorithm using an untransformed reward, i.e. the pure partitioning quality according to evaluation function (5.2). Figure 10.6 visualizes the average solution qualities over 30 independent runs per setting.

As the figure reveals, BASIC DSL with globally transformed rewards is able to learn high quality solutions in each of the three strongly different consecutive IAPP stage games. In particular, all "old" agents can adapt to settings that occur if "new" agents are added to the system, because the observed rewards are guaranteed to be at least as big as the maximum reward observed in the preceding stage game.

Without globally transformed rewards, however, only newly added agents are able to learn since their local q-tables are empty. As old agents already observed larger rewards than those that typically occur at the beginning of a new stage game, they most probably will not change their strategy. In extreme cases, where the maximal reward value is smaller than a previously received reward, agents even cannot learn a (near-)optimal solution. Accordingly, this experiment also underlines the importance of transformed rewards for enabling agents to adapt to changing scenarios, i.e. to changing stage games.

Now, let us briefly discuss one question that arises from this experiment: If "old" agents are unable to learn due to "old" q-values, shouldn't they simply reset the q-tables at the beginning of a new stage game? The answer to this question is manifold.

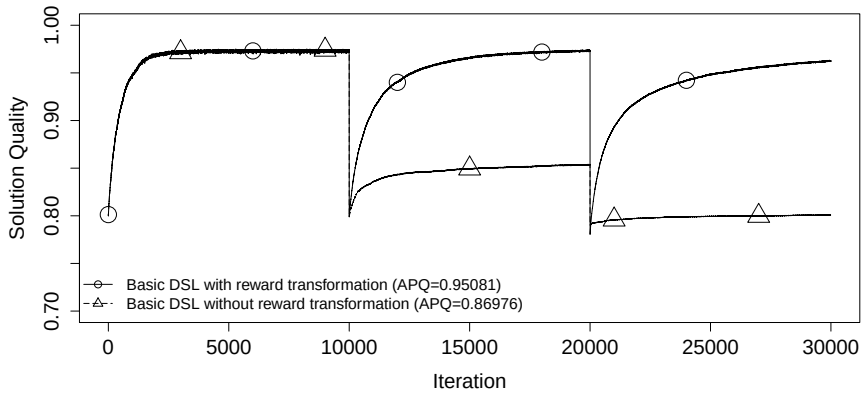


Fig. 10.6 Influence of global reward transformation to BASIC DSL in setting with changing numbers of agents and changing target positions.

First of all, note that a global transformation of rewards is related to resetting q-values. In detail, by assuring that the smallest reward in a new stage game is at least as large as the largest reward in the preceding stage game, each agent is able to update its q-values and thus also its strategy. This follows due to the "optimistic" update rule, that changes values and strategies only on improvements. Accordingly, resetting the q-tables and the transformed (engineered) rewards are equivalent with respect to enabling agents to learn in changed settings.

However, in order to enable an agent to decide on when to reset its q-table, the agent requires additional information, e.g. about the duration of each stage game, or, as an alternative, it needs to be able to observe the entire system state. In contrast, when using the framework of cooperative sequential stage games and the DSL approach with engineered rewards, agents are neither required to observe the environment nor do they need to know the structure of the played game. Instead, the transformed rewards automatically enable adaptations to new settings. This is particularly beneficial in large MAS, because agents cannot observe the full environment, but obtain a common reward that is calculated by a central instance.

10.2.6 Summary

As we have shown in this section, the DSL approach is able to deal with large sequential stage games like the IAPP. DSL was also shown to be able to learn in open systems with varying numbers of agents. Accordingly, the results of this section support theoretical and empirical results given in Ch. 8. Although we investigated the discount factor and the robustness only in a single stage game, the results also hold for sequences of consecutive stage games like those obtained from using $\mathrm{R}(10000)$. This follows, since the transformation provably enables the agents to adapt to new settings given enough time. It is particularly noticeable that the standard DSL approach in its most basic form already is able to find partitionings with a higher quality than the state-of-the-art ETS. In addition, DSL does neither need inter-agent communication nor the ability of calculating distances to arbitrary targets. Since both is required in

ETS, DSL is more efficient, and hence also optimizes the third not formally captured cost objective of the IAPP.

However, we also observed that the basic approach is unable to learn in highly dynamic settings if globally transformed rewards are used. As part of Sect. 10.4, we will show that DSL in its basic variant can also learn in such dynamic settings when different (non-global) reward calculation strategies are used.

10.3 Strategy Specific Simulations

Having analyzed the BASIC DSL approach in our IAPP variant, we use this section to analyze different properties of the proposed coordination strategies. If not stated otherwise, the simulations are carried out according to the aforementioned settings and using the standard values shown in Tab. 10.2.

Table 10.2 Standard parameter settings that are used in this evaluation.

Strategy	Parameter (variable identifier)	Value
PHEROMONE	pheromone disposition radius (r)	5
	reinforcement (c)	20
	pheromone decay rate (Δ)	0.1
	selection pressure (α)	2
	initial pheromone value ($\mathrm{p_{init}}$)	100
	pheromones only on strategy update	TRUE
	ϵ	0.6
PS	probability for previous strategy (p)	0.9
SEED	seed probability ($\mathrm{p_{seed}}$)	0.1
	phase duration ($\mathrm{phase_{Dur.}}$)	10
RAS	number of actions (N)	2
	refresh on movement	TRUE
RFN	probability (p)	$\frac{2}{3}$
ETS	flip probability ($\mathrm{p_{flip}}$)	0.1

Clearly, not all possible parameter values and variations shall be evaluated here, as this quickly would become tedious. Instead, we focus on some of the most interesting issues that we have discovered while working with the strategies. We provide a short summary for each strategy, and use the last subsection to briefly enumerate the most interesting conclusions.

10.3.1 Pheromone-Based Strategy

In the PHEROMONE strategy, multiple parameters can influence the solution quality. In addition, good parameter values are hard to define for general settings because they strongly depend on the considered scenario. For instance, a too low pheromone

evaporation rate in a highly dynamic scenario can lead to problems as pheromone traces may be unable to reflect environmental changes in time. In less dynamic settings, on the other hand, slowly changing pheromones might be desirable, as they might enable DSL to cope with small perturbations efficiently. Accordingly, we do not intend to analyze all dependencies in detail as this quickly becomes tedious. Instead, we will show that the parameter values chosen for the experiments in Sect. 10.4 are a good tradeoff for the considered scenarios.

As we will see in Fig. 10.19(a) on page 162, the PHEROMONE approach constantly improves over three different stage games in a setting with global reward, and finally also outperforms the state-of-the-art ETS approach—at least in terms of the solution quality in the last iteration. Here, we show that PHEROMONE can do even better than this. Therefore, parameters values have to be adjusted compared to the standard values given in Tab. 10.2. Figure 10.7 shows the simulation results for varying decay rates and using $\epsilon = 0.05$ instead of the standard value $\epsilon = 0.6$. Compared to Fig. 10.19(a), we observe an almost immediate increase in solution qualities, and hence also the APQ-values for all decay rates are larger than the one reached by ETS.

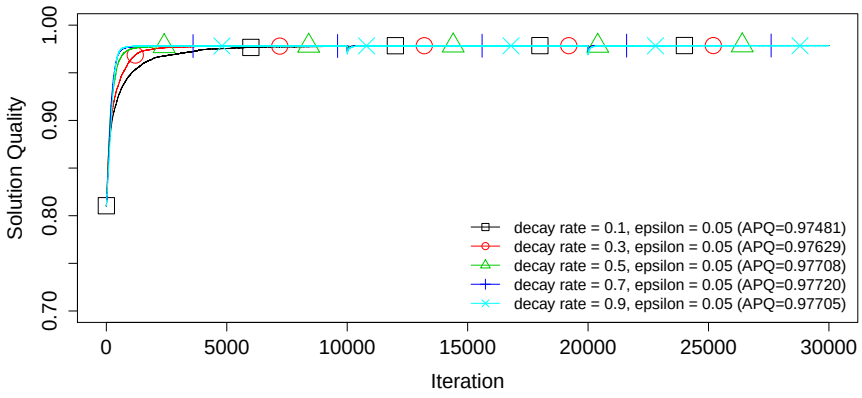


Fig. 10.7 PHEROMONE with different decay rates in a setting with three stage games, each lasting 10000 iterations, having 3 targets and global rewards.

In order to obtain statistically significant results for the influence of the decay rate on the convergence speed, more than 30 independent runs per experiment would be necessary. From the experience made during this thesis, however, we can state that the characteristics shown in the figure should be preserved under more experiments. Hence, we argue that an increased decay rate leads to quicker convergence.

Next, consider using $\epsilon = 0.05$ in settings with locally transformed global rewards and frequently moving agents. From experiments presented later in Figs. 10.22(b) and 10.22(c), we can conclude that PHEROMONE with standard parameters is able to learn in such scenarios. Figures 10.8 and 10.9, however, indicate that using $\epsilon = 0.05$ in the same settings leads to a poor performance. Since the value for ϵ is comparatively small, agents will seldom use pheromones to select actions. Hence, they cannot benefit in the same way as they do with the standard value $\epsilon = 0.6$.

From these examples and from comparison to experiments conducted in the following sections, it becomes clear that PHEROMONE is a powerful approach that

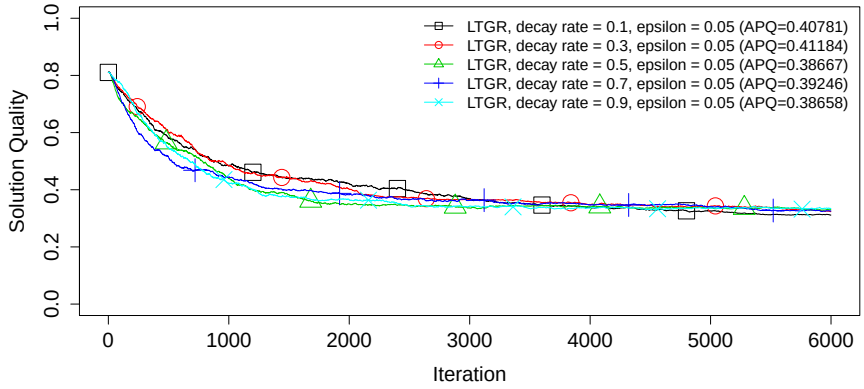


Fig. 10.8 PHEROMONE with varying decay rates in a setting with locally transformed global rewards and job generation pattern $\mathrm{R}(5, 100)$.

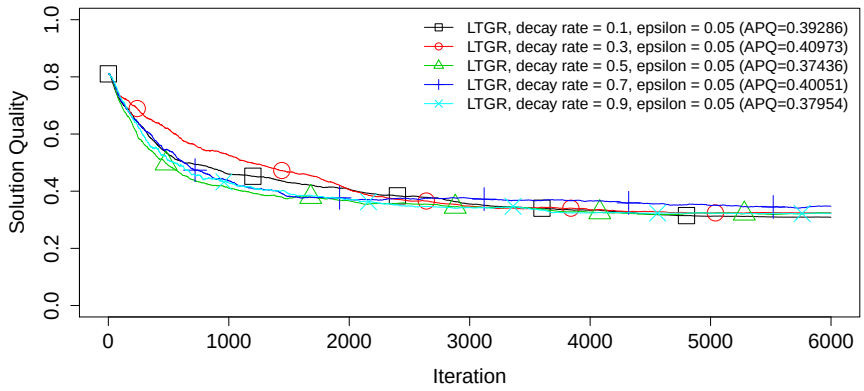


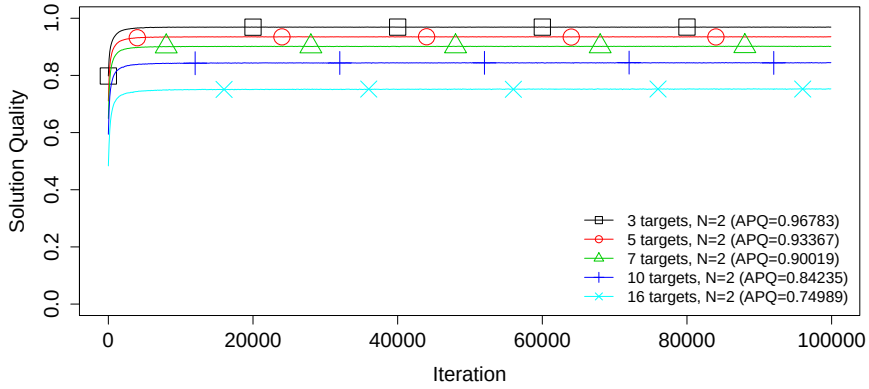
Fig. 10.9 PHEROMONE with varying decay rates in a setting with locally transformed global rewards and job generation pattern $\mathrm{C}(5, 100)$.

is able to deal with various settings. However, its performance strongly depends on properly chosen parameter values which depend on the considered scenarios.

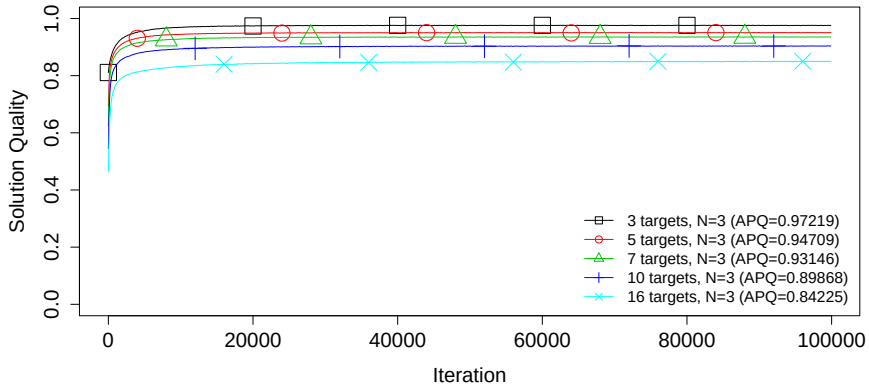
10.3.2 Reduced-Action-Set Strategy

In this section, we investigate the influence of the heuristic that reduces the number of actions in the Reduced-Action-Set strategy (RAS). Table 10.3 compares RAS with different numbers of actions, BASIC DSL, and the Random-Target Strategy (RTS) in terms of the average solution quality in the last (10^5-th) iteration of stage games with different numbers of targets.

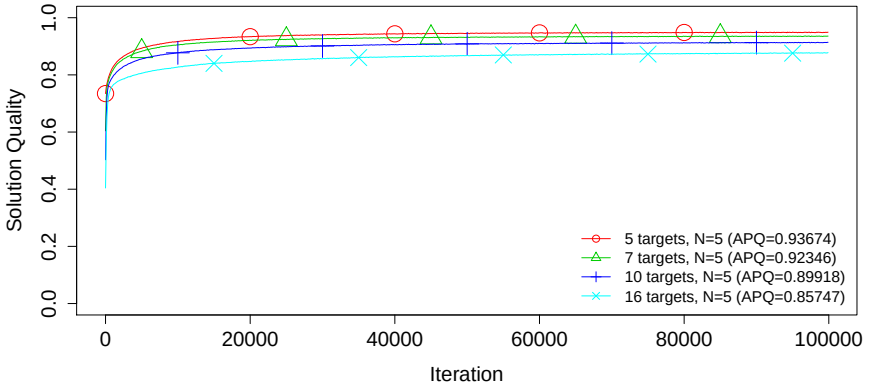
For an action reduction heuristic that results in $N = 2$ actions, RAS performs comparable to BASIC DSL in the three target setting, but it tends to become worse with increasing numbers of targets. Considering a heuristic that allows selecting the three nearest targets ($N = 3$), we observe a significant increase in the final solution



(a) RAS using a heuristic that allows selecting only between the $N = 2$ nearest targets (APQ(100)).



(b) RAS using a heuristic that allows selecting only between the $N = 3$ nearest targets (APQ(100)).



(c) RAS using a heuristic that allows selecting only between the $N = 5$ nearest targets (APQ(100)).

Fig. 10.10 Development of solution qualities of RAS with different heuristics.

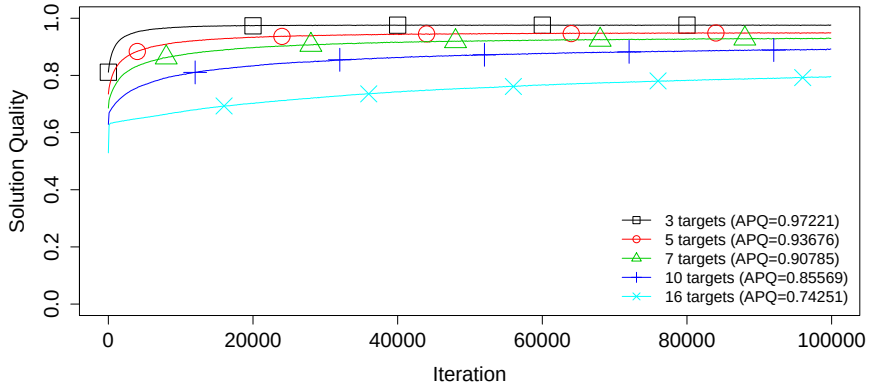
Table 10.3 Comparison of the average solution qualities in a fixed stage game after 10^5 iterations.

Targets	Strategy	Heuristic	Avg. Quality	Diff. to BASIC
3	RAS	$N = 2$	0.96898	- 0.00710
	RAS	$N = 3$	0.97607	- 0.00001
	BASIC		0.97608	
	RTS		0.80923	
5	RAS	$N = 2$	0.93506	- 0.01370
	RAS	$N = 3$	0.95065	+ 0.00189
	RAS	$N = 5$	0.94836	- 0.00040
	BASIC		0.94876	
	RTS		0.73407	
7	RAS	$N = 2$	0.90197	- 0.02788
	RAS	$N = 3$	0.93537	+ 0.00552
	RAS	$N = 5$	0.93519	+ 0.00534
	BASIC		0.92985	
	RTS		0.67935	
10	RAS	$N = 2$	0.84399	- 0.04717
	RAS	$N = 3$	0.90327	+ 0.01211
	RAS	$N = 5$	0.91218	+ 0.02102
	BASIC		0.89116	
	RTS		0.62056	
16	RAS	$N = 2$	0.75172	- 0.04293
	RAS	$N = 3$	0.85027	+ 0.05562
	RAS	$N = 5$	0.87730	+ 0.08265
	BASIC		0.79465	
	RTS		0.52180	

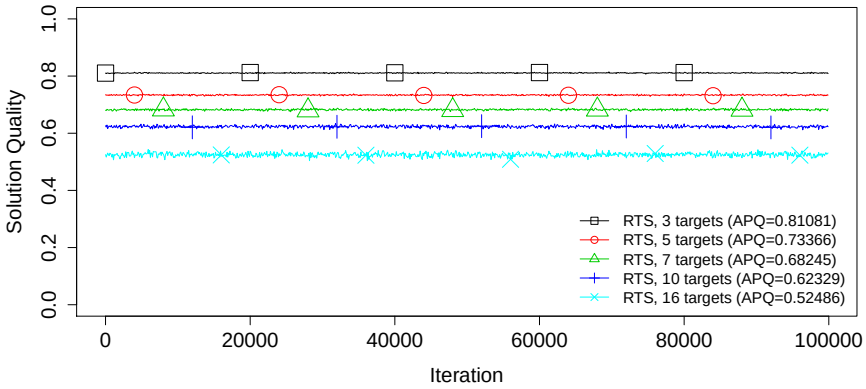
quality compared to the former heuristic. The $N = 3$ heuristic achieves even better solutions than BASIC DSL in settings with many targets. Adding even more actions, i.e. $N = 5$, again leads to an increased performance compared to $N = 3$. The improvement, however, is smaller compared to the step from using $N = 3$ instead of $N = 2$ actions.

Accordingly, it seems unnecessary to allow agents to choose between all possible targets in order to find a high quality (possibly optimal) solution. Instead, depending on the number of targets, a significantly smaller action set is sufficient to find good solutions. Clearly, one reason for this can be found in the properties of our IAPP variant. Since targets and agents are distributed uniformly at random, there is a high likelihood for each agent, that choosing any of the N nearest targets should not prohibit the agents from finding an optimal solution. Support for this statement also comes from comparing the performance of, e.g., RAS with $N = 2$ in the 16 targets case to the results of the random strategy RTS. Here, we can conclude that the heuristic—although being quite restrictive—does not lead to arbitrarily bad solutions.

A further observation can be made by considering the development of solution qualities in the above experiments over time, as visualized in Figs. 10.10 and 10.11. The time needed for convergence increases with the number of actions available to the agents. Compared to BASIC DSL, however, all three RAS heuristics show a quicker convergence to their final solution quality. This higher speed of convergence even improves the average partitioning quality, such that RAS can become competitive to BASIC DSL in terms of the APQ-value, although the latter reaches a higher solution quality in the end.



(a) Development of solution qualities using the BASIC DSL approach (APQ(100)).



(b) Development of solution qualities using the RTS approach (APQ(100)).

Fig. 10.11 Development of solution qualities.

To conclude, note that the heuristic should be chosen carefully depending on the number of targets in the system. However, as a rule of thumb, a relatively low number of actions is a good starting point.

10.3.3 Seed-Based Strategy

One key parameter of the SEED strategy is the phase duration $\mathrm{phase}_{\mathrm{Dur.}}$. Figure 10.12 visualizes the influence of this parameter. As we can see, learning takes longer for increased $\mathrm{phase}_{\mathrm{Dur.}}$ values, since in a longer phase less agents compared to a shorter phase are allowed to explore.

The second strategy specific parameter is the seed probability $\mathrm{p}_{\mathrm{seed}}$, which basically determines the number of seeds during any phase. For high values, more agents can be expected to become seeds than for lower values. Figure 10.13 captures the influence of this parameter. In the figure, we plot the behavior in a single stage game, because consecutive stage games behave identically. From the results, we can observe that the seed probability mainly influences the convergence speed. Additionally, it

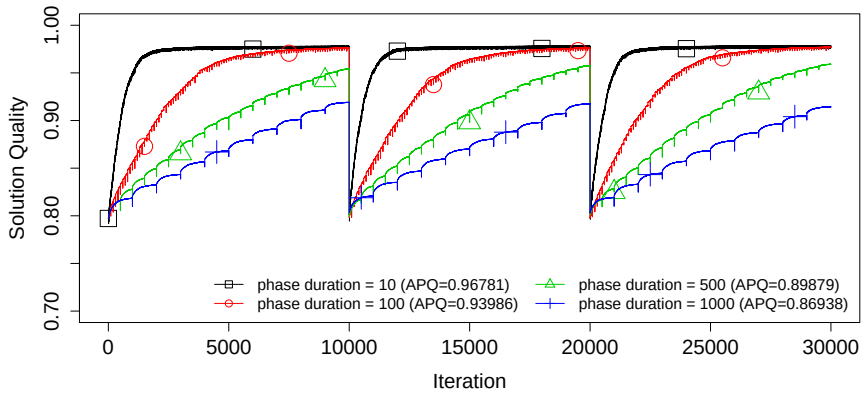


Fig. 10.12 Influence of the phase duration parameter in the SEED strategy.

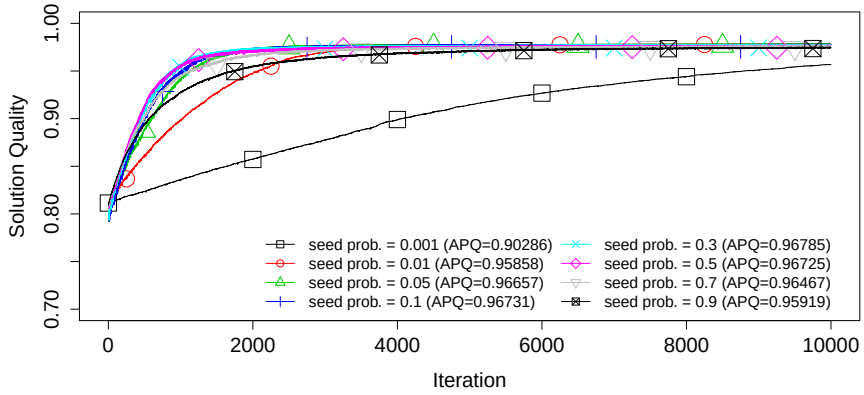


Fig. 10.13 Influence of different seed probabilities in a single stage game with a phase duration of 10.

slightly impacts the level of convergence. Given too low or too high seed probabilities, i.e. $p_{seed} \in \{0.001, 0.01\}$ or $p_{seed} \in \{0.7, 0.9\}$, we observe a slower improvement compared to more moderate probabilities like $p_{seed} \in \{0.05, 0.1, 0.3, 0.5\}$. Since the speed of reaching a high quality level also influences the average partitioning quality, the same relations as for the speed can be found in the APQ values as shown in Fig. 10.14. Clearly, given a too high seed probability, the strategy reduces to the BASIC DSL approach which explains the simulation results. On the other end, if a too low probability is used, only a small number of agents will be able to learn, i.e. to explore, which can explain the observed slower improvement.

In order to statistically underline the results given in Figs. 10.13 and 10.14, additional repetitions would be necessary. From our experience, however, we argue again that the characteristics shown after 30 iterations correspond to those found in simulations with more repetitions.

To conclude, a short phase duration as well as a moderate but small seed probability should be chosen. On the one hand, this combination creates a more stable learning environment because not too many simultaneous learners are present within each phase. On the other hand, a short phase duration increases the probability for

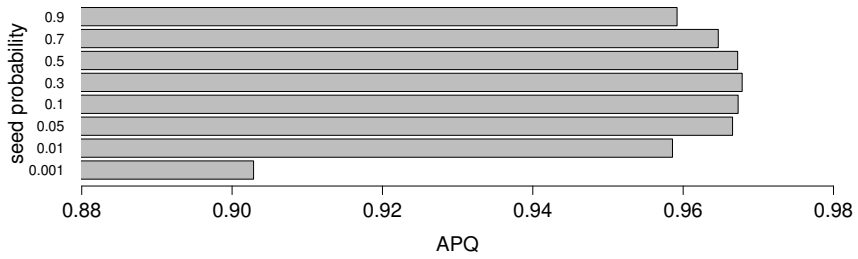


Fig. 10.14 Influence of the seed probability to on the average partitioning quality.

each agent to become a seed from time to time, and thus enables it to explore the search space and to find better solutions.

10.3.4 Random-From-Neighborhood Strategy

Besides the size of the neighborhood, the probability p of following a neighbor's strategy also influences the performance of the RFN strategy. We will focus on that probability in this section. Recall that $p \to 0$ lets the agents mainly follow strategies of their neighbors, whereas $p \to 1$ lets them behave more like in BASIC DSL, i.e. it lets the agents follow their own strategies.

Figure 10.15 shows some simulation results for $p \in \{0.1, 0.3, 0.\bar{6}, 0.9\}$ in a setting with three stage games and global rewards. Apparently, the APQ-value increases when more agents follow their own strategy. This is not surprising, since randomly following a neighbors strategy in a long lasting stage game either does not change anything, e.g. if all neighbors already follow the same strategy, or leads to perturbations as the learning environment becomes more unpredictable.

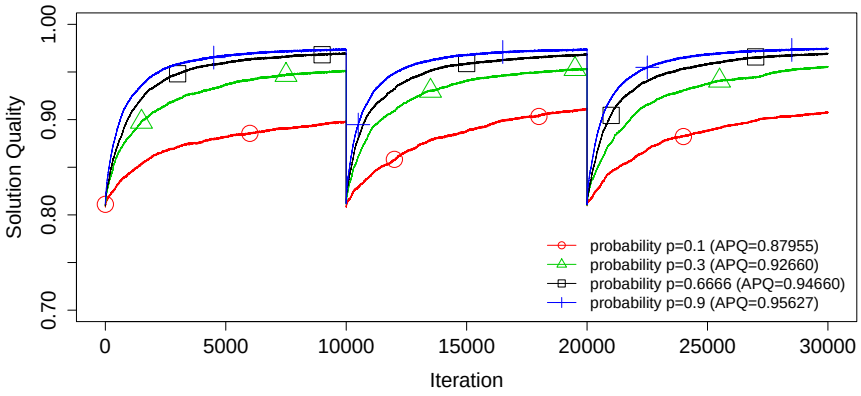


Fig. 10.15 Development of solutions for RFN in settings with three targets and global rewards.

As we will see in upcoming experiments described in Sect. 10.4.2, RFN performs well in highly dynamic environments with local rewards and job generation pattern

$\mathrm{R}(5, 100)$. Accordingly, we investigate the influence of probability p in such settings. The results shown in Fig. 10.16 indicate that p has no significant influence on the average partitioning quality and also almost no influence on the speed of reaching the final solution quality level. We only observe a slightly lower convergence speed if a neighbor's strategy is followed too often ($p = 0.9$).

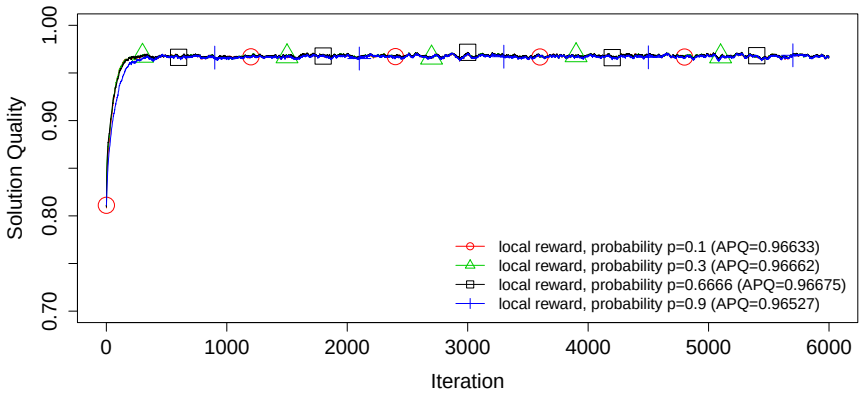


Fig. 10.16 Development of solution qualities of RFN in a setting job generation pattern $\mathrm{R}(5, 100)$ and local rewards.

Based on these experiments, we finally conclude that a low probability is a good starting point in settings with local rewards and frequently changing agents, as the additional communicative overhead for higher probabilities does not lead to better solutions. In settings with little or no movement and global rewards, a higher probability results in an improved performance. Since too high probabilities lead to a lot of communicative overhead, choosing $p = \frac{2}{3}$ seems to be a good tradeoff for the considered settings. In general, however, the best choice for p should be evaluated depending on the considered scenario.

10.3.5 Previous-Strategy

The only variable parameter in the PS approach is the probability of sticking to the previous strategy. Figure 10.17 visualizes our results for $p \in \{0.1, 0.3, 0.7, 0.9\}$. As we can see, the standard value $p = 0.9$ performed worst. For decreasing probabilities we obtain an increased performance. However, as $p \to 0$, the approach "degenerates" to BASIC DSL.

Accordingly, this strategy does not offer any benefits for learning as it can only become as good as the basic learning approach.

10.3.6 Summary

From these simulations, we draw the following conclusions:

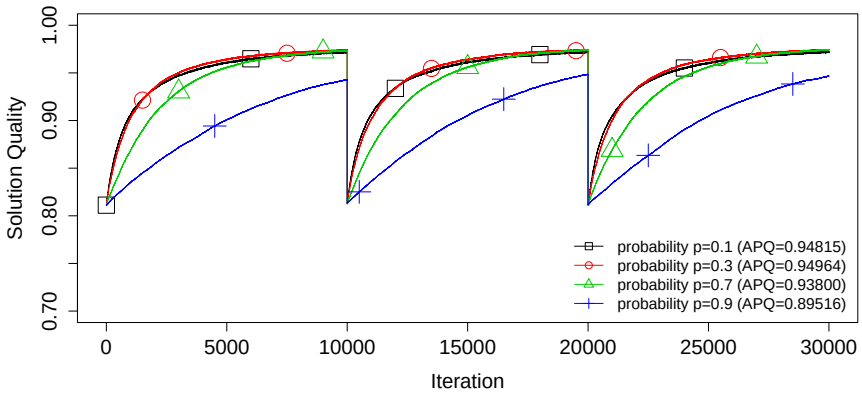


Fig. 10.17 Influence of probability p in PS.

- PHEROMONE is able to quickly find high quality solutions, but its performance depends on properly chosen parameters that may differ for different (job generation) scenarios and reward types.
- If global rewards are used, RAS does not degenerate to a random strategy, even if the number of possible actions is very low compared to the number of targets. The choice of the heuristic is important, but a small number of actions is a good starting point in our IAPP setting.
- SEED finds high quality solutions in general, but the speed of convergence and the solution quality level depend on proper parameter values. Short phase durations and low seed probabilities are a good trade-off.
- In settings with local rewards and frequently changing agent positions, RFN finds high quality solutions for all investigated neighborhood-interaction probabilities. Thus, small probabilities should be used to reduce the communicative overhead. In global reward settings, higher values are required to achieve good results.
- PS behaved best for parameter values that degenerate the strategy to BASIC DSL.

In general, the strategy specific simulations showed that parameter values strongly influence the results. Thus, all parameters should be chosen carefully.

10.4 Reward Types and Coordination Strategies

In this section, we will analyze the performance of the proposed coordination strategies. Therefore, we will compare them to each other and to ETS and RTS. Again, we will use the standard parameter values presented earlier in Tab. 10.2.

First, we consider a setting with globally transformed common rewards and a single stage game lasting 30000 iterations. Figure 10.18 shows the results from which we can observe that:

1. All strategies are able to learn a good partitioning given enough time.
2. Strategies show different convergence speeds.
3. Despite PHEROMONE, all strategies finish with a partitioning quality higher than ETS.

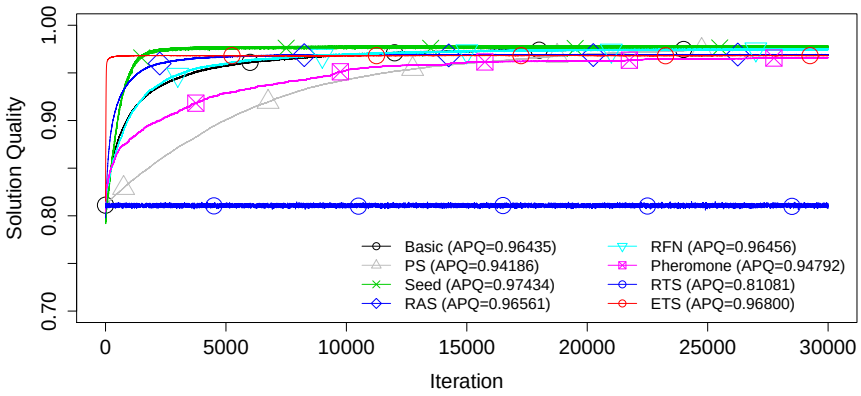


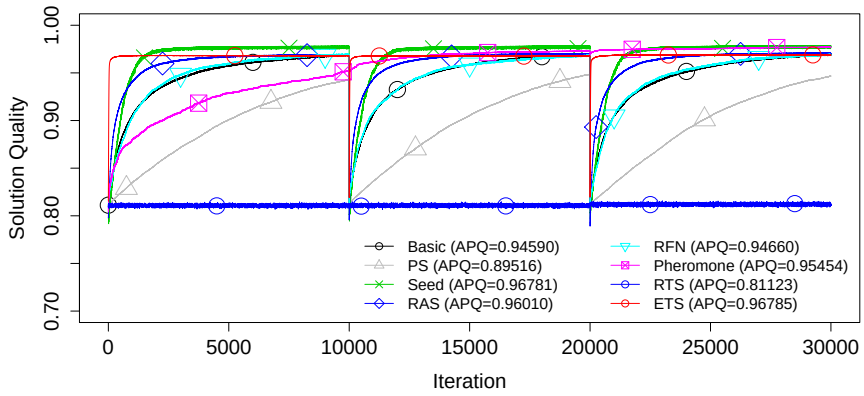
Fig. 10.18 Given enough time, all coordination strategies are able to reach approximately the same level compared to ETS.

Particularly the last observation is interesting since we know from our theoretical analysis that all strategies but RAS are guaranteed to converge to optimal solutions. In the above experiments, however, RAS performs even better than PHEROMONE. Basically, we can explain the good results of RAS by noting that not selecting the most distant target is not a hard restriction in a setting with only three targets. The suboptimal performance of PHEROMONE follows from the used standard parameter values, which are designed to slowly create a stable pheromone distribution. In preceding experiments (see Sect. 10.3), we have already seen that PHEROMONE can reach very high solutions qualities significantly faster.

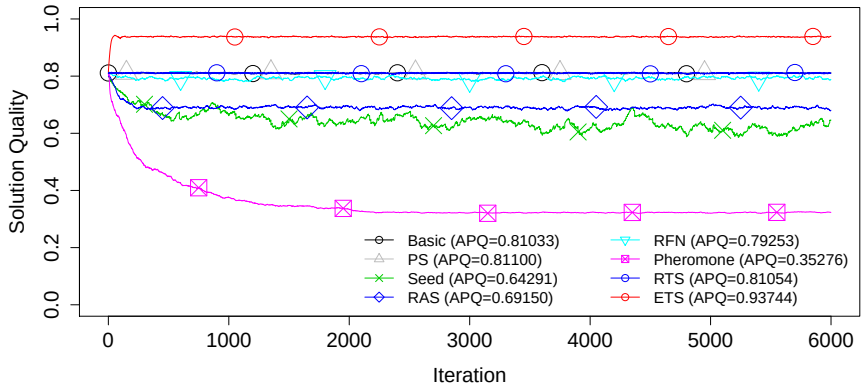
In the following, we will mainly investigate the different strategies according to the previously introduced reward concepts in three different job generation scenarios. The motivation of using $\mathrm{R}(10000)$ is to show how well the strategies perform given enough time for adaptation to randomly generated new stage games. Although agents in the second job setting with $\mathrm{R}(5, 100)$ most probably are also faced with new stage games in each iteration, this job generation pattern still provides some sort of stability through the longer lasting jobs. However, worse solution qualities can be expected, since agents may not have the time to learn a good action. By considering $\mathrm{C}(5, 100)$, we intend to further increase this stability. Note that this job creation pattern intends to make use of IAPP domain knowledge, since we already know from other experiments that good solutions contain regions, in which agents select the same target to optimize the partitioning quality. By restricting the movement area to a (small) circle around the last position, a previously good target selection is likely to remain optimal. Thus, we expect slightly better results in $\mathrm{C}(5, 100)$ compared to $\mathrm{R}(5, 100)$.

10.4.1 Global Reward

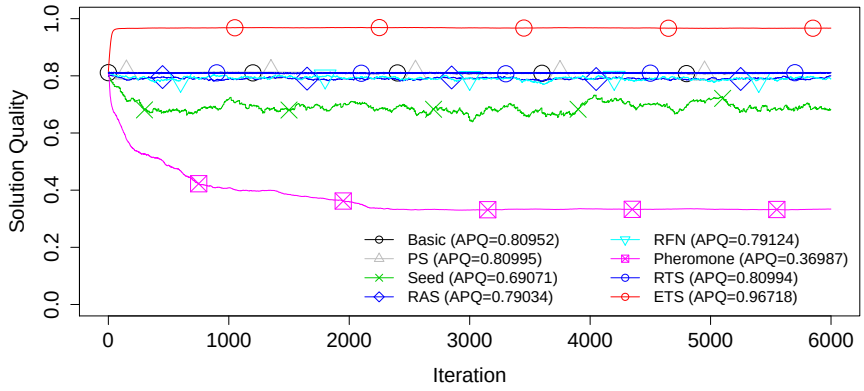
Figure 10.19 shows the results obtained from simulations with a globally transformed common reward in the three job generation settings. In Fig. 10.19(a), we can see for $\mathrm{R}(10000)$, that all coordination strategies perform significantly better than the Random Target Strategy (RTS). Although ETS is by far the fastest strategy that also



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.



(c) Job generation setting $\mathrm{C}(5, 100)$.

Fig. 10.19 Performance of the different strategies given a globally transformed common reward. (Details on the DISTRIBUTION and DISTANCE objective values can be found in the appendix in Figs. B.1 and B.2.)

converges at a high level, it does not lead to the best partitionings. Except for PS, all other coordination techniques achieve a comparable partitioning quality at the end of each stage game. The SEED and the PHEROMONE approaches even converge at higher levels. Note that, except for PHEROMONE, all coordination strategies basically restart in each stage game at the same level and then develop in the same ways as in the previous stage games. Indeed, the PHEROMONE strategy is the only approach whose solution quality improves continuously and almost independently from the two game transitions. This behavior follows from the fact that agents can use already existing pheromones at their new positions after movement. As we have seen in Sect. 10.3, the PHEROMONE approach can also converge at higher levels than ETS if different parameter values are used.

In contrast, let us now consider the dynamic settings with $\mathrm{R}(5, 100)$ in Fig. 10.19(b) and $\mathrm{C}(5, 100)$ in Fig. 10.19(c). We observe that no coordination strategy can outperform a random target selection. Indeed, PHEROMONE, SEED, RAS, and RFN perform even worse than RTS. To explain this behavior, note that a new cooperative stage game arises in almost each iteration, because there is at least one agent that has moved with a high probability in both dynamic job generation scenarios. Recall, that in this case, the global transformation function will always return a higher reward for the new stage game compared to any other previously observed value. Accordingly, it does not matter which joint action was executed. Indeed, each agent will always believe its action was good since it has obtained a higher reward. Consequently, this problem prevents any learning. The additional results shown in Fig. 10.20 for $\mathrm{C}(2500, 2600)$ as well as in Fig. 10.19(a) for $\mathrm{R}(10000)$, emphasize this statement, as both cases allow learning since the stage games last "long enough". After a while, however, agents in the $\mathrm{C}(2500, 2600)$ setting will also be confronted with stage games that change in every iteration, which follows due to unsynchronized job generation processes.

Although we do not intend to discuss every curve in detail, we want to explain why PHEROMONE particularly suffers from those too short stage games. Since all agents place pheromones for their lastly executed action within radius $r = 5$ in almost every iteration, and due to the high probability of following the pheromones ($\epsilon = 0.6$), one pheromone will quickly dominate all others. To obtain such a dominating pheromone

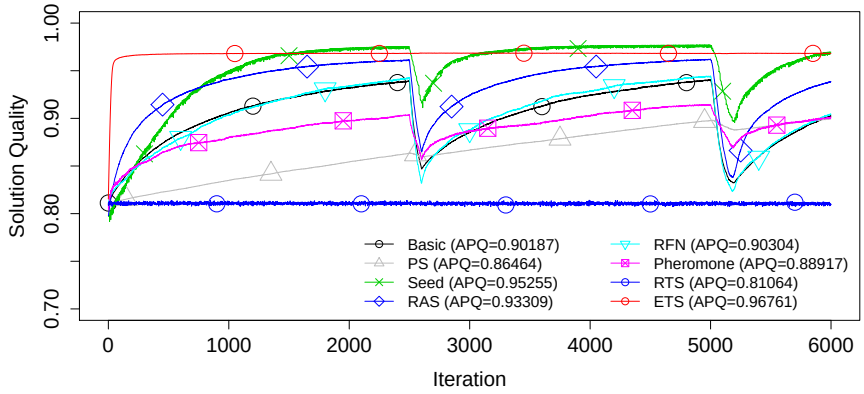


Fig. 10.20 Agents can learn good partitionings in long lasting stage games ($\mathrm{C}(2500, 2600)$) with globally transformed rewards.

it is sufficient if some neighbored agents select the same target by coincidence. Then their updates will increase the probability for other agents in their vicinity to also take this action and so on, until all agents select the same target. Given that all agents select the same target, one clearly obtains a DISTRIBUTION value of zero as shown in the corresponding Figs. B.1(b) and B.1(c) in the appendix, which finally leads to the low observed overall partitioning qualities.

For the SEED and RAS strategies, we observe a better performance in the circle scenario. This can be expected, since movement within a restricted area leads to slower changing overall settings compared to completely random movements.

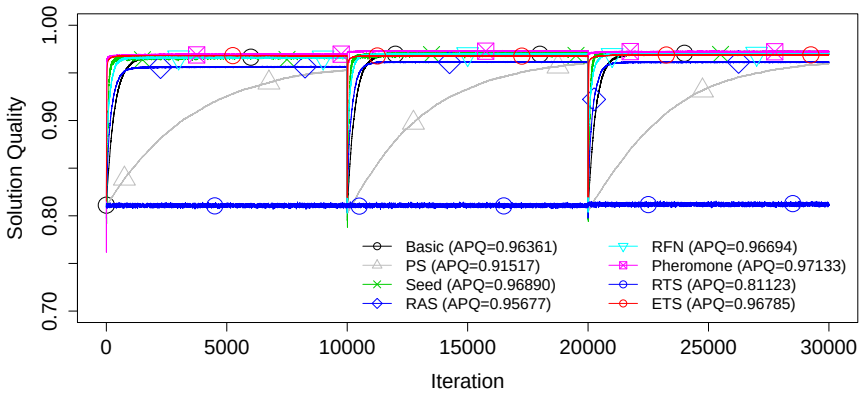
10.4.2 Local Reward

Figure 10.21 summarizes the results for the local reward calculation strategy in different settings. In Fig. 10.21(a) with the $\mathrm{R}(10000)$ job setting, we observe a significantly improved convergence speed for all coordination strategies compared to using global rewards. However, local rewards also lead to lower maximum partitioning qualities. Since it is hard to deduce this from the figures, have a look at Tab. 10.4, where we present the average reward over the last iterations of all three stage games for both reward settings. Note that the last iterations must not correspond to the best solutions obtained in each stage game, but they are at least an indicator that support the statement. Furthermore, it is worth noting that PHEROMONE quickly achieves a very high solution quality and then basically stays at this level, such that it results in a higher APQ-value than in the global reward setting. In general, the results for the considered $\mathrm{R}(10000)$ setting hence support the local reward as an appropriate replacement for a global reward, although we lose the convergence guarantees.

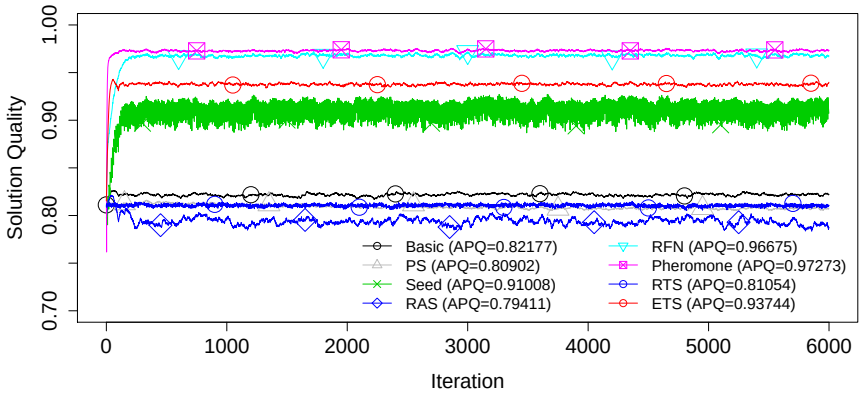
Next, consider the dynamic settings $\mathrm{R}(5, 100)$ in Fig. 10.21(b) and $\mathrm{C}(5, 100)$ in Fig. 10.21(c). Comparing these results to the corresponding figures for the global rewards (Figs. 10.19(b) and 10.19(c)) reveals a significant difference. In particular, in the $\mathrm{C}(5, 100)$ setting all strategies are able to learn better solutions than the RTS. In addition, the three communication-based strategies (PHEROMONE,SEED, and RFN) achieve the same level as ETS. In $\mathrm{R}(5, 100)$, all strategies, except RAS and PS, result in better average partitioning qualities than RTS. In addition, PHEROMONE and RFN even outperform the state-of-the-art ETS.

Table 10.4 Influence of global/local reward on partitioning qualities at the last iteration of each stage game. The ALI reward denotes the average reward over the last iterations of the three stage games, i.e. the average reward over partitioning qualities at iterations 9999,19999, and 29999.

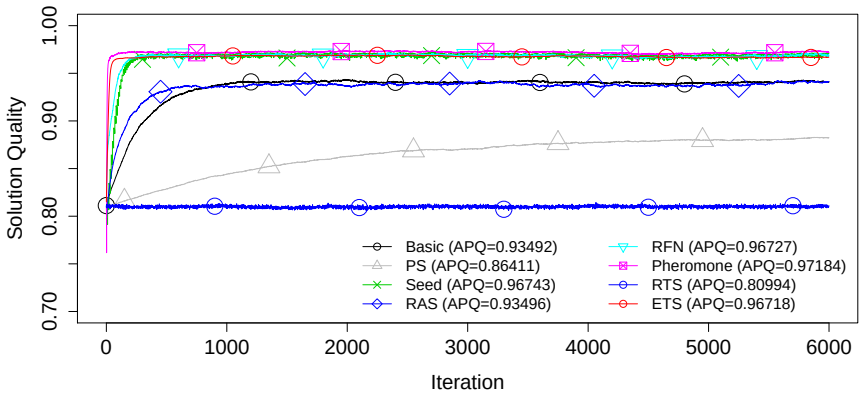
Approach	ALI Reward	
	global	local
BASIC	0.96934	0.96888
PS	0.94607	0.95766
SEED	0.97808	0.97093
RAS	0.96955	0.95969
RFN	0.96903	0.96874
PHEROMONE	0.96749	0.97130
RTS	0.81127	
ETS	0.96819	



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.



(c) Job generation setting $\mathrm{C}(5, 100)$.

Fig. 10.21 Performance of the different strategies given the local reward setting. (Details on the DISTRIBUTION and DISTANCE objective values are given in the appendix in Figs. B.3 and B.4.)

Next, we explain the shift in behavior for the dynamic settings compared to the global reward scenario. First, note that an agent does not increase its transformation additive whenever any other agent moves. Second, each agent learns independently using its own local reward. Thus, as long as the agent does not move, it basically plays a single agent game with non-deterministic rewards for its own actions.

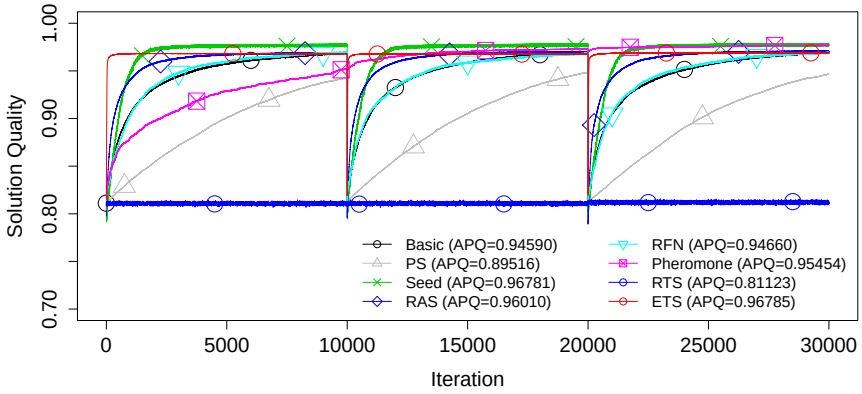
In that single agent game ("stateless MDP"), the learning process is less influenced and disordered by other agents compared to the globally transformed reward settings. This follows, since all agents' decisions and movements in the global reward setting influence the DISTANCE and the DISTRIBUTION value. In contrast, the local reward is much more stable for two reasons. First, changed positions of other agents do not influence the local distance reward. Second, although agents most probably create different partitionings in each iteration, only the number of agents assigned to the currently selected target is interesting from a single agent's point of view. As one can expect from the properties of the local distribution approximation function as discussed in Sect. 9.2.1.2, the corresponding value will stabilize close to an equal distribution. Figures B.3(b) and B.3(c) support the latter statement as the (global) DISTRIBUTION objective value according to (5.2) close to 1 in average. Accordingly, by trying to maximize its own local reward, each agent also maximizes the global solution quality.

Recall the theoretical results on single agent Q-Learning with optimistic updates in settings with noisy rewards, as presented in Sect. 7.3. From Corollary 1, we know that single agent deterministic Q-Learning with the same optimistic update rule as DSL, converges to an optimal policy if the maximum (positively noised) reward is observed "often enough". Additionally, notice that in case of a locally calculated reward, an agent basically plays such a noisy MDP with a single state. This holds because noisy rewards are related to non-deterministic ones, as agents in both cases observe different values for repeatedly executing the same action in the same state. It follows, that we can transfer the result of Corollary 1 to our locally calculated reward setting. Thus, we can expect that each agent will learn the action that maximizes its local reward. Due to this and due to the aforementioned properties of the local reward function, we then can expect a high solution quality according to evaluation function (5.2).

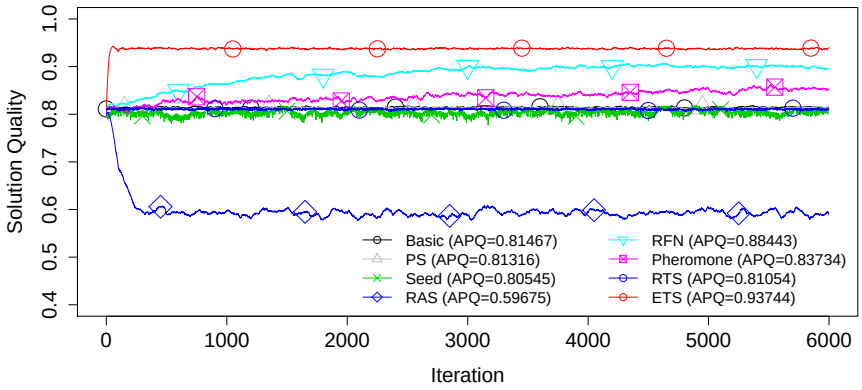
Finally, be reminded that DSL loses its convergence guarantees under local rewards. However, the transfer of the results from single agent Q-Learning with noisy rewards to the local reward calculation settings at least helps to explain the observed high quality solutions.

10.4.3 Locally Transformed Global Reward

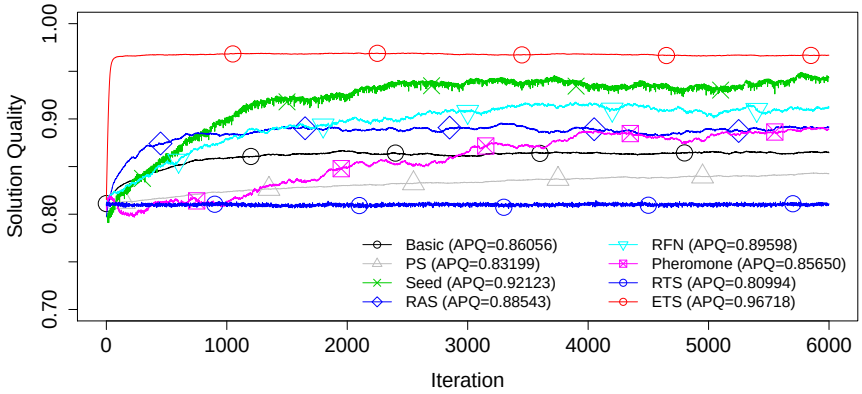
Figure 10.22 shows the simulation results obtained when using the *locally transformed global reward (LTG)*. First of all, notice that the results for $\mathrm{R}(10000)$ correspond exactly to those for the global reward in the same setting (cf. Figs. 10.19(a) and 10.22(a)). Clearly, the reason for this is that all agents move at the same time, and thus all agents locally perform the same transformation. The results for the dynamic scenarios, which use $\mathrm{R}(5, 100)$ and $\mathrm{C}(5, 100)$ to generate of jobs, are more interesting.



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.



(c) Job generation setting $\mathrm{C}(5, 100)$.

Fig. 10.22 Performance of the different strategies given the locally transformed global reward setting. (Details on the DISTRIBUTION and DISTANCE objective values can be found in the appendix in Figs. B.5 and B.6.)

Compared to the global reward, we observe that in $\mathrm{R}(5, 100)$ only RFN and PHEROMONE are able to learn. BASIC and PS perform as good as the random strategy and SEED performs slightly worse.

RAS attracts our attention due to its very low performance. A closer look at the DISTRIBUTION and the DISTANCE objective values in the corresponding figures in the appendix (Figs. B.5(b) and B.6(b)) reveals that its poor performance is rooted in the low DISTRIBUTION value. Naturally, this observation is explainable, as there are only three targets and just the two nearest ones can be selected through the agents. Thus, we frequently have situations like the one sketched in Fig. 10.24(a). There, one target can be selected by much more agents than the other two because it is at least the second closest to all agents. After moving, an agent will execute its previously best action with a probability of $1 - \epsilon = 0.99$, if that action still is part of its updated action set. In that case, it will obtain a higher reward than before due to the nature of the local transformation. Now assume that, after a move, the previously best action is not available anymore, then, by line 7 in Alg. 5, a random action will be executed. Since there is one target action which is available for most agents, the probability of executing that action by coincidence is higher under the random choice compared to the remaining actions. Again, executing this action will result in a high reward because of the movement. Due to the low exploration probability ϵ, many agents will stay with that strategy, which finally leads to declining DISTRIBUTION values. In order to support this explanation, i.e. the influence of ϵ, we conducted additional experiments with different ϵ values. The results are presented in Fig. 10.23, and indicate that RAS can avoid such declining DISTRIBUTION values with increasing ϵ values. However, it then degenerates to a random strategy.

In addition, situations like the one illustrated in Fig. 10.24(b) can also explain low solution qualities in some settings, because it may happen that a single target, or more generally, some targets are covered by other targets and thus are seldom considered to be valid actions.

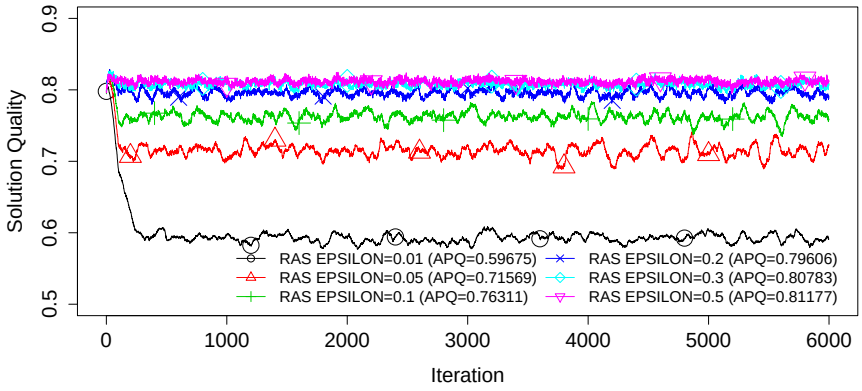
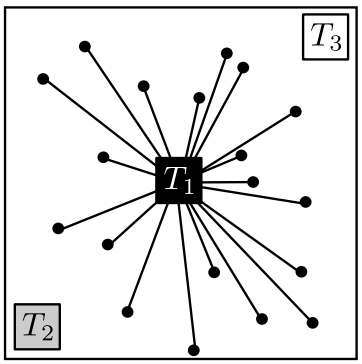


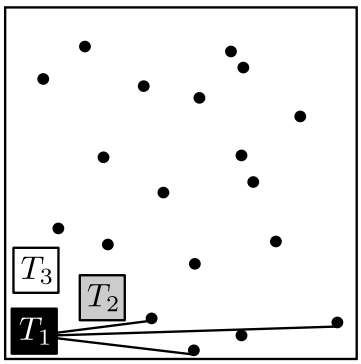
Fig. 10.23 RAS behavior under varying ϵ rates.

After this excursion, let us return to the analysis of the LTG reward in the circle setting $\mathrm{C}(5, 100)$ as shown in Fig. 10.22(c). Here, RAS is able to learn since the above described effect of randomly selecting an action, because the previous strategy is not available anymore, does not occur after every movement. Furthermore, all

strategies are able to learn, but none of them reaches the same level as ETS. The strategies also do not exhibit the same convergence speed and also not the same average partitioning quality as in the local reward scenario.



(a) Agents consider the middle target T_1 as possible choice, which leads to bad DISTRIBUTION values as stated in the text.



(b) Only some agents will consider target T_1 at all.

Fig. 10.24 RAS is likely to result in bad DISTRIBUTION objective values in the sketched scenarios.

Comparable to the locally calculated reward setting considered in the previous section, the LTG reward also leads to a "single agent MDP" with only one state. Contrary to the local reward, however, the influence of other agents on the common reward is much stronger for LTG rewards. In detail, the decisions of all agents and the movement of any agent influence the global reward, and hence the LTG reward, too. Accordingly, the used LTG reward for learning, compared to the local reward, is much more volatile on the one hand. On the other hand, compared to the globally transformed reward, it is less volatile. Altogether, this explains the results, which are in the middle between globally transformed reward settings and local reward calculation.

10.4.4 Summary

Let us briefly summarize the behavior of DSL and the coordination strategies in different reward and job generation settings. For clarity, we only list the main results:

- In order to enable learning in large dynamic systems, agents require, besides time, a learning environment that is locally stable, resp. shows a low dependence on actions of other agents.
- Using globally transformed rewards and given enough time for each stage game, all coordination strategies reach high quality solutions (cf. Figs. 10.18 and 10.19(a)).
- If a global reward is available, it should not be transformed in the environment, since this leads to low solution qualities in dynamic settings. Instead, it should be locally transformed by the agents. This maintains the guaranteed convergence to

ε-optimal solutions in each stage game in settings like $\mathrm{R}(10000)$, and allows at least some learning in more dynamic settings.
- Although the proposed local reward implies the loss of convergence guarantees, most strategies still perform very well with that reward. This follows, because the setting becomes comparable to single agent learning in noisy MDPs, for which convergence results exist. In particular, our strategies either outperform the state-of-the-art ETS, or reach the same level and produce less costs thereby.
- Overall, the PS coordination strategy was not convincing due to low convergence speed and degeneration to random target selection in dynamic settings.
- Communicative coordination strategies are the most successful approaches.

Finally, we have to clarify that these results may be limited to our variant of the IAPP in the considered scenarios. In other problem domains or scenarios, the strategies might behave differently. To get further insights, the next section will present some additional experimental results for various interesting scenarios.

10.5 Evaluation in Different Scenarios

Based on previous insights, we next investigate some selected DSL approaches in settings that differ from the standard scenario used earlier. The goal is to develop an even deeper understanding for the proposed learning methods, particularly including their limitations.

As we have already seen, DSL is able to adapt to changing stage games in our IAPP variant. Hence, we reduce the number of stage games to two instead of three, in order to save simulation time in $\mathrm{R}(10000)$ job generation settings. If not stated otherwise, we will re-use the standard parameter values from Tab. 10.2. Since the focus of this section is on high level properties of the approaches, we will not discuss strategy-specific parameter values. The used values have been chosen based on the insights gained so far and are supported by preliminary manual tests.

10.5.1 Large Joint Action Spaces

Note that MARL algorithms in the past often have only been evaluated empirically for small (repeated) games involving just two agents (e.g. [CB98] [LR00] [KK02]). In this section, we will evaluate our learning approach in a scenario with $n = 10000$ agents, $m = 5$ targets, global rewards, and a grid of 150×150 cells. All strategy-specific parameters correspond to the standard values, except for some parameters in PHEROMONE (pheromone decay rate $\Delta = 0.7$, $\epsilon = 0.05$) and RAS (number of actions $N = 3$).

Figure 10.25 shows the results, and we can see that the PHEROMONE approach is able to reach a higher solution quality than the state-of-the-art Exchange Target Strategy (ETS). The SEED strategy shows a steadily increasing performance and reaches a good solution quality level, too. The remaining strategies, in particular the BASIC DSL approach without additional coordination techniques do not perform well.

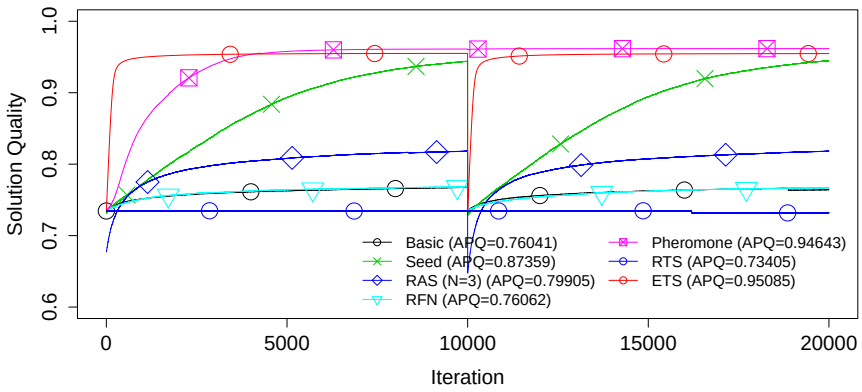


Fig. 10.25 Simulations with global reward for 10000 agents and 5 targets.

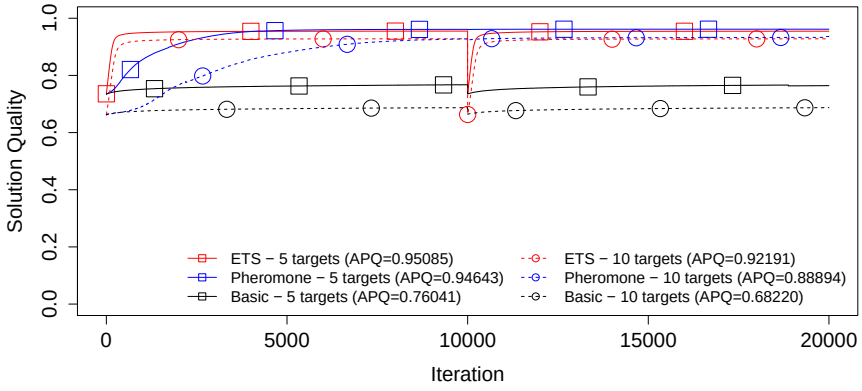
In order to understand this behavior, recall that the size of the joint action space is heavily increased from 3^{500} in the standard scenario to 5^{10^4} possible joint actions in this large setting. Hence, the algorithms have to search a tremendously large joint action space in order to find a high quality solution.

From its theoretical properties, we know that BASIC DSL would find an optimal solution given enough runtime. However, since only the ϵ-greedy action selection will "steer" the search process, the approach will quickly become impractical. As we have seen in previous experiments, too small exploration rates will lead to a slow performance increase, which will even be lower here due to the large joint action space. Too high exploration rates, however, would lead to an unstable learning environment and convergence to lower solution qualities. Assuming knowledge about the game structure, i.e. the duration of each stage game, one could consider to use techniques like the Boltzmann exploration (see e.g. [CB98]), that adjust the exploration rate ϵ over time. Since we do not have such knowledge in general cooperative sequential stage games, this approach will not be applied here. Instead, we rely on the proposed coordination strategies.

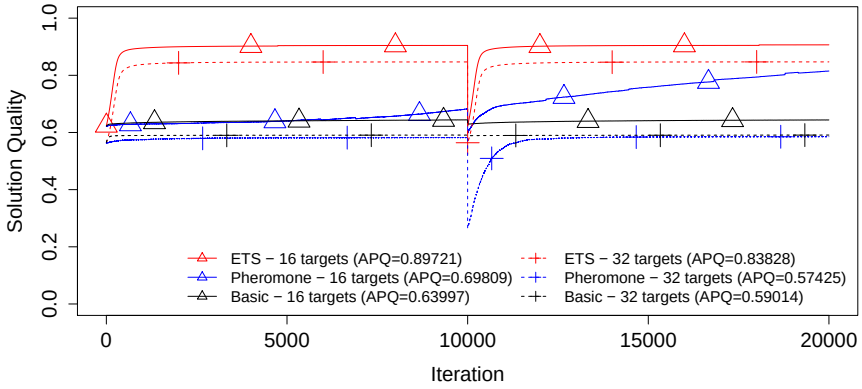
As the above results indicate, some coordination techniques are able to deal with the increased joint action space. However, as we will show in the next experiments visualized in Fig. 10.26, they also suffer from the general curse of dimensionality. We reuse the same 10^4 agent setting, but we increase the number of agent individual actions, by considering $m \in \{5, 10, 16, 32\}$ targets.

The comparison of DSL-based results with those achieved by ETS shows that their performance decreases with increasing numbers of targets, i.e. with larger joint action spaces. It is, however, worth noting, that PHEROMONE for $m = 5$ and $m = 10$ targets reaches a higher solution quality than ETS, and also shows an increasing solution quality for 16 targets. For 32, however, PHEROMONE is even worse than the basic approach. This behavior is rooted in the pheromone radius, the parameter values, and the scenario settings. For instance, if too many targets are too close to each other, their pheromone regions will probably overlap, which results in more or less random pheromone-based agent decisions.

In the end, we hence conclude that the general curse of dimensionality found in MARL is, unfortunately, also part of our approach. However, additional coordination



(a) Results for settings with $m = 5$ and $m = 10$ targets.



(b) Results for settings with $m = 16$ and $m = 32$ targets.

Fig. 10.26 Simulations with global reward for 10000 agents and $m \in \{5, 10, 16, 32\}$ targets.

techniques and proper parameter values can help to deal with larger settings, at least to a certain extent.

10.5.2 Hotspot Distributions

Up to now, we have only investigated uniformly at random positioned agents. In the next experiments, we investigate the standard scenario with 500 agents and three targets, but we use a Gaussian distribution in order to create agent positions around a "hotspot" centered in the environment. The target positions remain uniformly distributed. Figure 10.27 shows an exemplary setting.

The results for this setting are shown in Fig. 10.28, and indicate that the DSL-based approaches are able to reach the same or a higher quality level than ETS. In particular, even the basic approach achieves higher solutions qualities. The reason for this behavior is that the DSL-based approaches are not fixed to a certain DISTRIBUTION objective value as it is the case in ETS. Hence, the learning approach can better balance both objectives, and thus reaches a better overall solution.

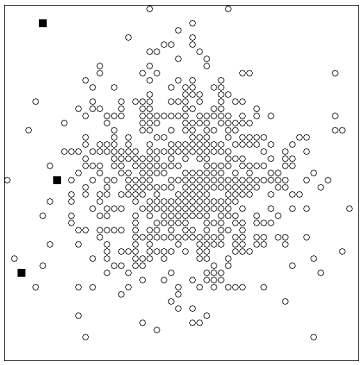


Fig. 10.27 Exemplary hotspot scenario: agents are visualized as circles, targets as black boxes.

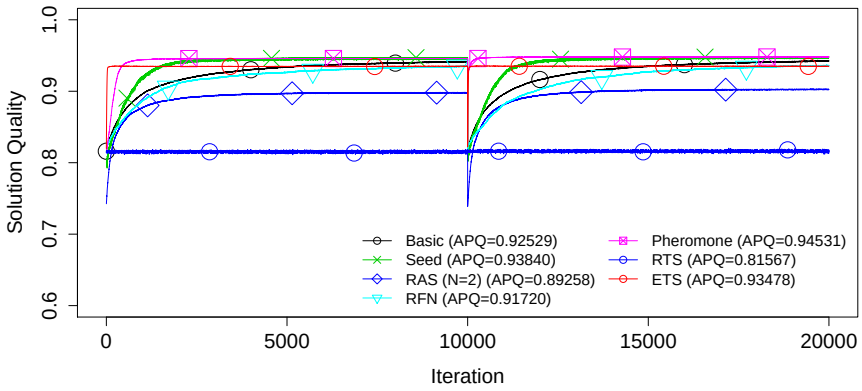


Fig. 10.28 Simulation results for three targets, globally transformed rewards, and 500 agents that are randomly positioned around the middle of the environment according to a Gaussian distribution.

One exception is the RAS strategy, which allows the agents to only select between the two nearest targets. Due to the spatial distribution, this restriction prevents the algorithm from optimizing the DISTRIBUTION objective.

According to these results, we can state that DSL is a flexible approach that benefits from not incorporating assumptions on "optimal" solutions for the IAPP, like "optimal" partition sizes, as in ETS, or "minimized distances", as in the RAS strategy.

10.5.3 Long Stage Games

In this section, we consider a setting with $n = 1000$ agents and $m = 16$ targets, a 50×50 grid, global rewards, and a single stage game with 10^6 iterations. We compare the behavior of BASIC DSL and PHEROMONE to ETS with respect to the speed of improvement. Figure 10.29 presents the results from which we can see that BASIC DSL steadily improves, but the improvements become less the longer the stage game lasts. In particular, even after 10^6 iterations, it still has not reached the

solution quality found by ETS. The time required for reaching high quality or optimal solutions hence clearly makes the approach impractical if no additional coordination is conducted. On the other hand, PHEROMONE shows faster improvement and is able find significantly better solutions than both, the state-of-the-art ETS approach and the basic learning approach.

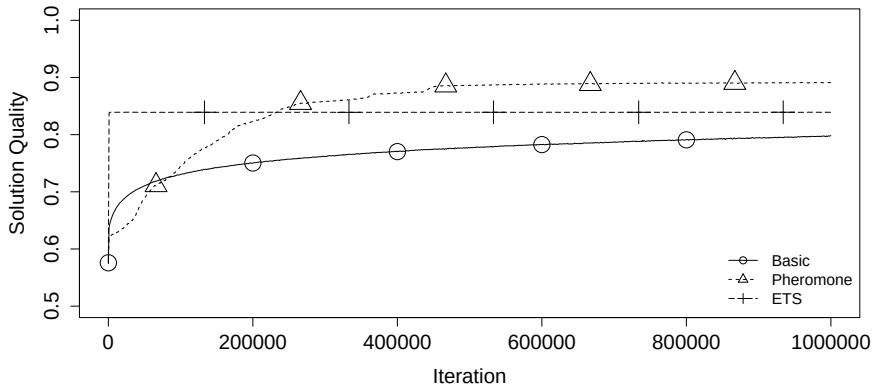


Fig. 10.29 Single stage game with 1000 agents, 5 targets, and 1.000.000 iterations.

10.5.4 Summary

The different settings investigated in this section can be summarized as follows:

- The curse of dimensionality also appears in the DSL approach. Hence, BASIC DSL is unable to find high quality solutions in too large joint action spaces efficiently. Coordination strategies like PHEROMONE or SEED are able to deal with larger joint action spaces than the basic approach, but in the end they still suffer from the mentioned curse of dimensionality.
- DSL-based approaches are able to deal with non-uniformly distributed agents in the sense, that they can find high quality solutions. In particular, the flexibility that comes from using a general reward-based model allows to successfully apply DSL without modifications to different scenarios, which might not hold for other special purpose IAPP algorithms like ETS.
- With increasing runtime, BASIC DSL is able to find better and better solutions. However the speed of improvement decreases. With the help of additional coordination techniques faster improvements can be achieved, such that the learning approach is able to find solutions that might be close to optimal given enough runtime.

10.6 Discussion

This chapter presented a detailed empirical analysis of the basic DSL approach and its coordination strategies in the context of our IAPP variant. Since we already provided summaries at the end of each section, we only provide a high level view on the main findings of this chapter:

- DSL is able to find high quality solutions in MAS with huge joint action spaces.
- The DSL-based approaches are applicable in various settings without adjustments, including settings with changing numbers of agents, changing target positions, different spatial distributions, as well as in settings with different agent to target ratios, or in highly dynamic job generation settings.
- Although direct communication could help to improve the performance, it is not necessary for finding high quality solutions in general.
- The used reward type influences the learning abilities of the agents. However, local rewards frequently lead to high quality solutions, despite the loss of theoretical convergence guarantees.
- Locally calculated rewards are comparable to learning in single agent MDPs with noisy rewards. Hence, the convergence results from these single agent settings, together with the inherent properties of our IAPP variant, can help to explain the good results obtained from locally calculated rewards.
- Although sequential stage games formally do not consider a state, the DSL-based approaches still suffer from the curse of dimensionality. However, this problem only applies to the joint action space and not to the state-joint-action space, as it would be the case in standard joint action learners for cooperative games. With the help of coordination strategies, the border of feasible scenarios can be shifted even further. This particularly holds for settings with large numbers of agents that have only small numbers of agent individual actions.
- With the help of (semi-)local rewards, DSL is shown to learn without a central feedback in several settings.

Besides these findings, it should also be noted, that globally engineered rewards are quite useful if agents seldom change positions. But they may lead to no learning at all if changes occur too frequently, e.g. in each iteration. In addition, proper scenario-dependent parameter values are important, as the performance of the proposed coordination strategies strongly depends on it.

Based on these results, we can conclude that DSL does not only work well in small problems involving just two agents (cf. Sect. 8.4), but it can successfully be applied in large multiagent systems. Due to the sequential stage game model and the engineered rewards, the agents are also able to adjust their strategies to changing settings without the need for a global state of the system. Given appropriate, probably problem-specific, coordination techniques that steer the action selection at an individual level, significant improvements with respect to the speed of convergence and the solution quality level can be realized. Also, such strategies enable learning in even larger joint action spaces. From our results in dynamic scenarios, we can further infer that learning in large dynamic MAS requires not just time, but also some sort of stability. By stability, we mainly mean that not too many agents should explore their environment by performing random actions.

In the end, it remains to state that all DSL-based approaches have only been evaluated in one particular problem, namely in our variant of the Iterative Agent

Partitioning Problem. Like any other problem, our IAPP variant itself possess certain properties that influence the behavior of the investigated strategies. Hence, different problems might lead to different behaviors of the strategies. This investigation, however, shall not be part of this thesis and is left for future work.

Chapter 11

Storage Medium-Based Distributed Stateless Learning

This chapter combines the Distributed Stateless Learning (DSL) approach with storage media placed in the environment of our instance of the Iterative Agent Partitioning Problem (IAPP). In addition, it evaluates the proposed approach empirically.

In detail, we first present a generic approach that combines DSL and storage media, and then we describe a concrete variant of the approach, that works with simple media. Based on information provided by the agents, the media are able to propose actions. Therefore, media simply count how frequently agents execute different actions. An agent that asks a medium for advice, for instance if the agent has moved to a new position, will get the most frequently used action as answer. We empirically evaluate the proposed approach and investigate different aspects.

The main contributions and results of this chapter are as follows:

- We present a generic approach that combines externally stored information with learning in large MAS.
- The approach is, to the best of our knowledge, the first one that uses transfer learning in a large multiagent system with hundreds of agents.
- Storage media can help to allow learning in dynamic and open multiagent systems, because they offer the ability to store and maintain knowledge at certain regions in an environment, even when agents move to other regions.
- The concrete variant of the approach is evaluated exhaustively:
  - It significantly speeds up learning compared to the BASIC DSL approach without storage media.
  - It allows agents to learn from global rewards in dynamic settings, which is not possible in DSL without storage media.
  - It allows transferring knowledge from smaller instances of the IAPP to larger ones.

This chapter is organized as follows. First, in Sect. 11.1, we present the generic approach as well as the concrete implementation. Furthermore, we provide a general discussion of storage media in MARL. Then, Sect. 11.2 describes the considered

simulation settings, provides an exhaustive empirical analysis, and gives a detailed summary on the obtained results. Finally, we conclude the chapter in Sect. 11.3.

11.1 Approach

In this section, we propose the generic approach that combines DSL with external storage media. Additionally, we present a concrete variant of that general approach, which will be investigated later in this chapter.

The generic approach essentially corresponds to the DSL approach with embedded storage media usage. Algorithm 9 shows the pseudocode of the media-based algorithm and highlights differences to the ordinary DSL algorithm. The main differences can be found at the end of each iteration, when *relevant* information are stored on a medium nearby (cf. Alg. 9, lines 21-23), and in using that external knowledge to select actions if a corresponding condition is true (lines 6-12), instead of following an ordinary action selection strategy.

Algorithm 9 Generic DSL with storage media (executed by each agent $i \in \mathcal{A}$)

Input: Cooperative SSG obtained via CSSG-transformation T
1: **procedure** DISTRIBUTEDSTATELESSLEARNING
2: $t \leftarrow 0$
3: $\forall a \in A_i : q_0^i(a) \leftarrow 0$ ▷ initialize local q-function
4: $\sigma_i \leftarrow$ choose arbitrary action from A_i ▷ initialize strategy
5: **while** $(t < t_{\max})$ **do**
6: Action $a_t^i \leftarrow$ NULL
7: **if** useMediumCondition **then**
8: Medium $M \leftarrow$ nearest storage medium
9: Action $a_M \leftarrow M.\text{getProposedAction}()$ ▷ get action proposed by the medium
10: **if** a_M is set **then**
11: $a_t^i \leftarrow a_M$
12: **if** $a_t^i \neq$ NULL **then**
13: $a_t^i \leftarrow$ Select action using standard methods (e.g. ε-greedy)
14: Execute action a_t^i ▷ leads to joint action u_t
15: Observe reward $\rho_t(u_t)$ ▷ reward of joint action u_t
16: $\mathsf{maxq}_t^i \leftarrow \max_{a \in A_i}\{q_t^i(a)\}$
17: $q_{t+1}^i(a) \leftarrow q_t^i(a), \forall a \in A_i \setminus \{a_t^i\}$
18: $q_{t+1}^i(a_t^i) \leftarrow \max\{q_t^i(a_t^i), \rho_t(u_t) + \gamma \cdot \mathsf{maxq}_t^i\}$
19: **if** $q_{t+1}^i(a_t^i) > \mathsf{maxq}_t^i$ **then**
20: $\sigma_i \leftarrow$ choose a_t^i ▷ update strategy
21: **if** storeOnMediumCondition **then**
22: Medium $M \leftarrow$ nearest storage medium
23: $M.\text{store}(t, \rho_t(u_t), a_t^i)$ ▷ store relevant information externally
24: $t \leftarrow t + 1$

Concrete implementations of this generic approach have to specify the two conditions, useMediumCondition and storeOnMediumCondition, and describe the used storage media. In particular, a medium might not store all values specified in the above pseudocode, or it might request different information. Also, the method that returns a proposed action (line 9) could be designed differently and return, e.g., a

probability distribution over possible actions. Furthermore, a concrete approach must also consider aspects like strategic placement of storage media or their numbers.

In this chapter, we will investigate the following concrete variant of this generic schema as a proof-of-concept approach. The media, which we consider, are called *Counting Media (CM)*. A $\mathrm{CountingMedium}$ only counts how often each action was executed by agents nearby and returns the most frequently executed action whenever $\mathrm{getProposedAction}()$ is called. Agents submit their played action to their nearest medium at the end of each iteration. Hence, the $\mathrm{storeOnMediumCondition}$ in line 21 of the algorithm is always TRUE. Concerning the second condition, $\mathrm{useMediumCondition}$, we decided to rely on storage media knowledge after movement. This means that after an agent has moved to a new position it will follow the most often played action according to its nearest storage medium. As general action selection procedure, we use the basic ϵ-greedy approach. Finally, the positioning and number of storage media has to be defined. Therefore, we use a grid structure and place one medium at the center of each grid cell. This is equal to an earlier placement approach presented in Ch. 12 (cf. Fig. 12.7, page 199).

Since this concrete approach combines DSL with $\mathrm{CountingMedia}$, we will refer to it as *Distributed Stateless Learning with Counting Media (*DSLCM*)* approach.

11.1.1 Discussion

External storage media in the context of multiagent reinforcement learning can be considered from different perspectives:

Coordination technique: Since external storage media mainly influence the action selection process of agents, the above approach can be considered as an additional coordination technique. However, the generic approach also allows a combination of storage media and coordination techniques like RFN, RAS SEED or PS (cf. Ch. 9). Furthermore, storage media can also be used to implement the PHEROMONE strategy by providing a means for storing pheromones in an environment.

Learning object: With respect to the SEED approach, one may consider the storage media as permanent seed agents, which do not move but constantly learn using the DSL approach. However, by our definition of storage media, we want the media to be passive objects that focus on storing information, rather than executing a learning algorithm on their own. The same point of view also applies to the RFN strategy, where storage media could be considered as some kind of never moving agents that provide advices. Accordingly, taking storage media into account in a learning MAS is related to using an additional special kind of agents. Contrary to "real" agents, however, storage media have different properties concerning e.g. storage or computational capacities and mobility.

Transfer Learning: In Transfer Learning (TL), there exist one source task and one goal task. Agents learn to solve the usually simpler source task and use the gained knowledge as basis for learning in the related goal task (see Sect. 2.5 for more details on TL). Given an IAPP instance modeled as cooperative SSG, each stage game corresponds to a different task. By initially using knowledge from storage media, the agents essentially transfer learned knowledge from previous

tasks to a current task. Hence, the usage of storage media in the proposed approach can be considered as a kind of Transfer Learning, too.

In the following empirical analysis, we will investigate the proposed concrete implementation of the storage media-based approach to highlight similarities to the three above presented aspects.

11.2 Empirical Analysis

This section presents the empirical results for the storage media-based learning approach[1]. Sections 11.2.1–11.2.6 investigate and discuss different aspects of this approach and Sect. 11.2.7 provides a brief overview on all results.

Before we present the results, let us first briefly describe the considered scenarios. These settings have been chosen based on previous experiences. Furthermore, the problem is already complex enough on the one hand, but it does not require too much computational time on the other hand. In detail, we simulate the proposed concrete variant of the storage media-based approach for $k = 12000$ iterations using $m = 3$ randomly positioned targets and an environment with 50×50 cells. As usual, we use the discount factor $\gamma = 0.0$, ϵ-greedy action selection with $\epsilon = 0.01$, and a common globally transformed reward. We place one storage medium every tenth column and row in order to create a regular media grid and center it in the environment (see Fig. 11.7 for a visualization of the grid). For simplicity, we refer to this media placement as 10×10 grid. Agents can always interact with their nearest medium.

Contrary to previous experiments, we consider a setting that allows dynamically changing the number of agents during a simulation. The first phase, called *source task (st)*, lasts for $k_{\mathrm{st}} = 3000$ iterations and contains $n_{\mathrm{st}} \in \{100, 1000\}$ agents. After the source task is over, additional agents might be added to the system such that in total $n_{\mathrm{gt}} = 1000$ agents are part of the subsequent *goal task (gt)*. The goal task lasts $k - k_{\mathrm{st}} = 9000$ iterations. The terms source and goal task are used with reference to Transfer Learning. During the source task and during the first 3000 iterations of the goal task, no agent is allowed to move. Afterwards, agents either move randomly according to the $\mathrm{R}(3000)$ or $\mathrm{R}(5, 100)$ job generation pattern as described earlier in Sect. 10.1. Accordingly, the source task is a single cooperative stage game and the goal task a cooperative SSG with multiple stage games. Considering the entire game, i.e. source and goal task together, we have a cooperative SSG with multiple stage games and varying number of agents.

11.2.1 Storage Media as Coordination Technique

Before we investigate the approach in the source and goal task setting introduced before, we first compare BASIC DSL and the storage media-based approach in a simple cooperative stage game with three consecutive stage games, each lasting 3000 iterations. Figure 11.1 visualizes the results and shows that the storage media-based approach performs exactly as BASIC DSL in the first stage game if no initial storage

[1] The simulation tool is available for download [Kem12].

media knowledge from a source task is available. However, while BASIC DSL has to restart learning in consecutive stage games, the proposed approach is able to benefit from the knowledge on the storage media that was gained during the first stage game. Accordingly, the overall solution quality is higher for the proposed approach compared to the basic learning approach.

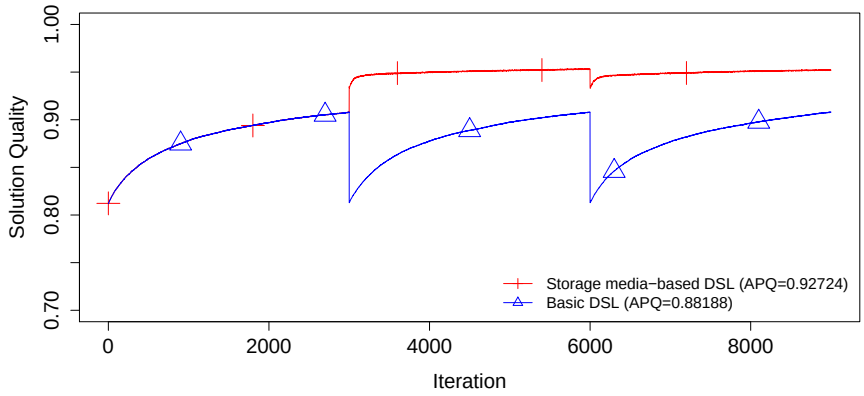


Fig. 11.1 Storage media-based approach versus BASIC DSL in a cooperative SSG with three stage games each lasting 3000 iterations, random jobs, 1000 agents, three targets, and $\epsilon = 0.01$.

Next consider Fig. 11.2, which contains the two curves from the previous figure and two settings with source tasks that use $n_{\text{st}} \in \{100, 1000\}$ agents. As the figure reveals, the storage media-based approach always leads to a better overall performance compared to the BASIC DSL approach.

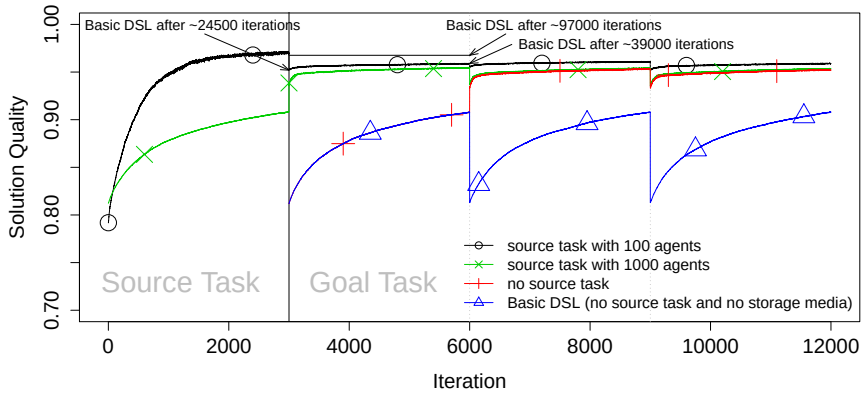


Fig. 11.2 Simulation results that show the influence of storage media compared to not using them.

In order to better classify the solution quality of the storage media-based approach, let us now focus on the first stage game of the goal task. Figure 11.3 shows the performance of BASIC DSL with $\epsilon = 0.01$ in that first stage game played for 97000 iterations. There, we can see that the storage media-based approach, using a source task with $n_{\text{st}} = 100$ agents, initially has a solution quality which BASIC DSL

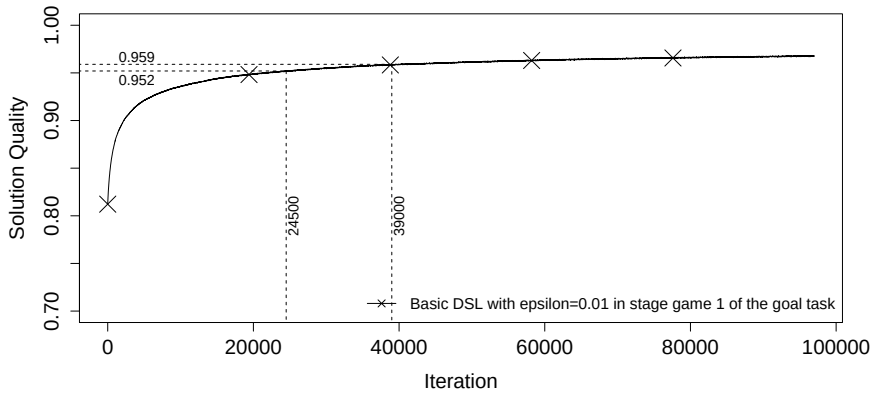


Fig. 11.3 Performance of BASIC DSL with $\epsilon = 0.01$ in the first stage game of the goal task shown in Fig. 11.2.

barely reaches after approx. 24500 iterations. Also, during only 3000 iterations, the storage media-based approach manages to reach the same solution quality as BASIC DSL after 14500 additional iterations. Hence, the storage media can significantly improve the convergence speed and coordinate the agents quickly towards high quality solutions which underlines the media's role as coordination technique.

Further influences of source tasks will be investigated in the next section.

11.2.2 Source Tasks

Figure 11.4 shows the development of solutions qualities given a source task with 100 agents and given no source task. The drop of the black curve at the beginning of the goal task follows from adding 900 agents. These additional agents initially follow the "strategy" learned by the 100 agents in the source task, simply by using the knowledge found on the storage media. Without source task, it obviously takes several iterations until agents manage to find better solutions (red curve). Accordingly, the differences between both curves in the first stage game of the goal task can be traced to the knowledge learned in the source task. In the following two stage games, the difference between both curves becomes smaller, but it still remains significant.

Next consider Fig. 11.5, in which the green curve shows the development of solution qualities if the source task also contains 1000 agents. The difference in the first stage game of the goal task to the setting without source task is slightly reduced compared to the source task with 100 agents (cf. also Fig. 11.6). For the two last stage games, no difference in solution qualities compared to the setting without source task is observable, i.e. the storage media propose the same targets after the first stage game of the goal task.

Interestingly, there is a small but significant difference between using a source task with 100 or 1000 agents as shown in Fig. 11.6. In particular, we can observe a small but not significant difference in the first stage game of the goal task, but a significant difference in the second and third stage game.

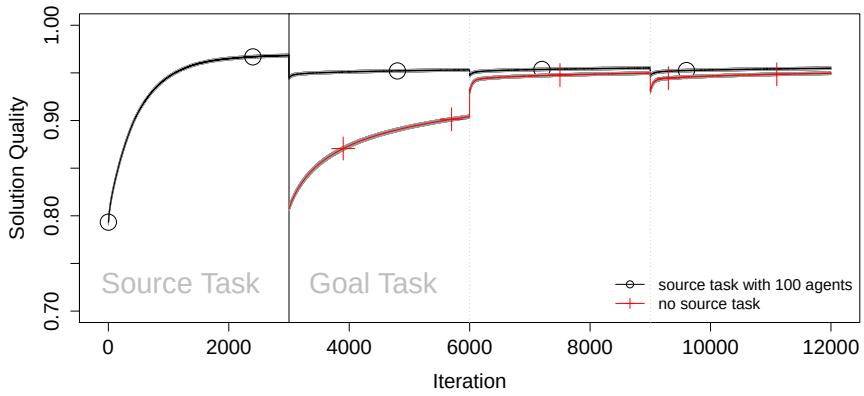


Fig. 11.4 Behavior of DSLCM given a source task with 100 agents or no source task and a goal task with 1000 agents. Average values over 1000 repetitions; gray shadows correspond to 95% confidence intervals.

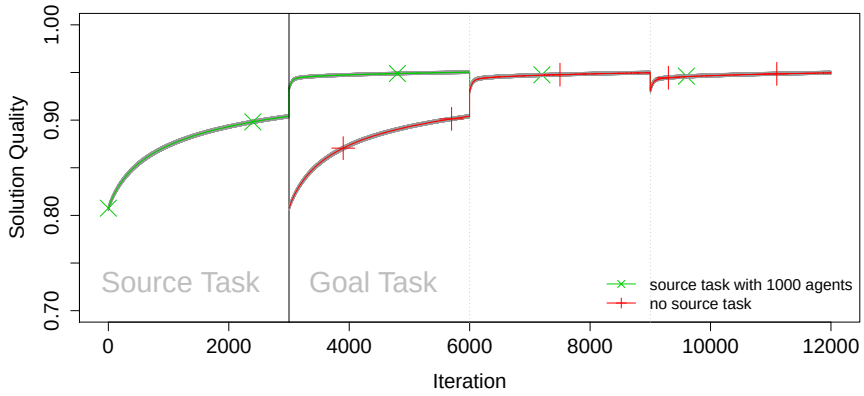


Fig. 11.5 Behavior of DSLCM given a source task with 1000 agents or no source task and a goal task with 1000 agents. Average values over 1000 repetitions; gray shadows correspond to 95% confidence intervals.

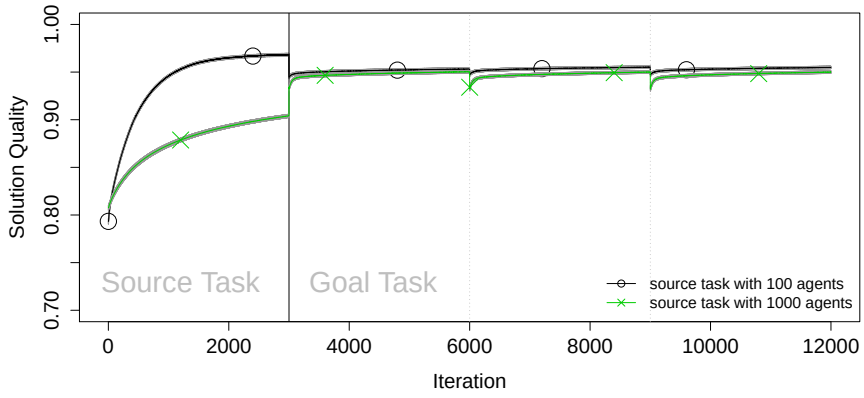
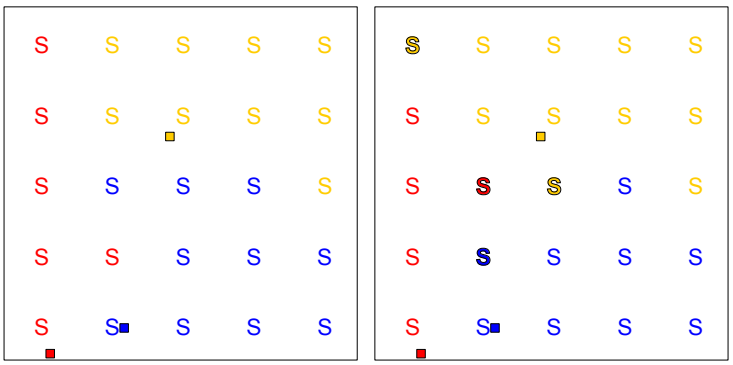


Fig. 11.6 Behavior of DSLCM give a source task with 100 resp. 1000 agents and a goal task with 1000 agents. Average values over 1000 repetitions; gray shadows correspond to 95% confidence intervals.

In order to understand why we observe a better goal task performance if a source task with less than 1000 agents is used, consider Fig. 11.7. It shows targets (colored boxes) and storage media ("S") at the end of the source task in a randomly chosen exemplary setting. The color of a storage medium corresponds to the color of the "learned" target. Comparing both subfigures, one observes differences in the proposed targets as highlighted in Fig. 11.7(b) by bold "S"-symbols. As we can see from the figure, if the source task contained 1000 agents, then only a small fraction of storage media propose the red target. Since agents will initially rely on the propositions of the media a lower DISTRIBUTION value can be expected compared to the 100 agent source task setting. Accordingly, the initial goal task performance depends on the performance in the source task since the learned behavior determines the initial storage media proposals. Assuming a higher solution quality in the source task with 1000 agents, we would obtain better initial target proposals from the storage media and thus a better performance in the goal task. This statement is supported by the third stage game of the goal task in Figs. 11.4 and 11.5. There, the initial solution quality is high, because the media already contain good proposals due to the higher solution qualities in the preceding stage games.



(a) $n_{\mathrm{st}} = 100$ (b) $n_{\mathrm{st}} = 1000$

Fig. 11.7 Storage media and their proposed target at the end of the source task.

11.2.3 Number of Media

As we have seen in the previous section, the media influence the solution quality by their target proposals. Figure 11.7 also shows that the media themselves "learn" to solve the IAPP in the sense that they build target regions like those discussed in Ch. 12. Contrary to *real* target regions, the regions created by the discussed storage medium-based learning approach cannot have an arbitrary form. Essentially, the storage media describe a discretization of target regions. Hence, the approximation quality depends on the number of media and, in general, also on their position. Besides the tradeoff between the number of storage media and the expected solution quality, one also has to consider the number of agents in a source task. This is important, because if there is a large number of media and a low number of agents,

then the probability of having media without target proposals increases. This follows because maybe not each medium is the nearest medium to any agent.

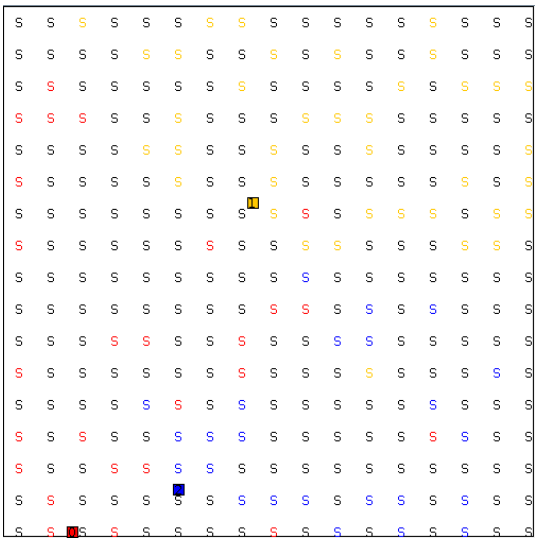


Fig. 11.8 289 storage media on a 3×3 grid at the end of a source task with 100 agents. Black storage media ("S") have no initial target proposal for the goal task.

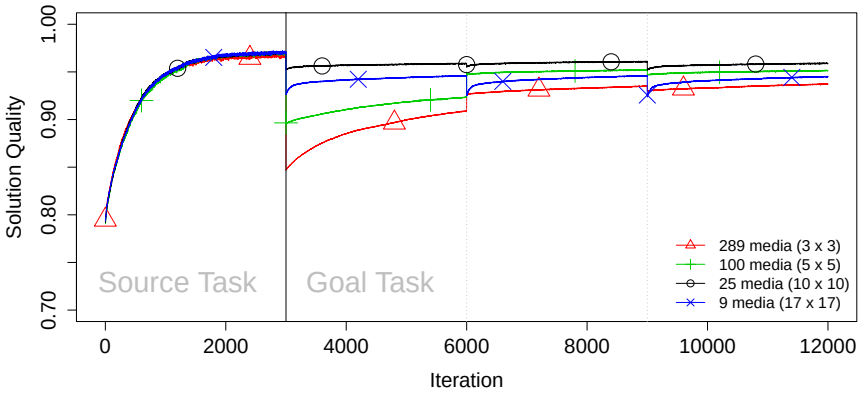


Fig. 11.9 Source task with 100 agents, goal task with 1000 agents and different media grids, where $x \times x$ denotes a grid with a medium every x-th row and every x-th column.

That statement is supported by the target proposals shown in Fig. 11.8 and the results provided in Fig. 11.9. For 289 media placed on a 3×3 grid, the initial solution quality of the goal task is significantly lower than for all other situations that use fewer media. With a decreasing number of media (e.g. 100 and 25 as in Fig. 11.9), the initial performance is increased. However, if not sufficiently many media are used, then the discretization becomes too rough and the solutions worsen again (e.g. 9 vs. 25 media in Fig. 11.9).

Another important factor for the storage media approach is the number of targets. Although we do not provide any simulation results for larger numbers of targets at this point, it can be expected that solutions become worse if the number of storage media and their placement does not allow to find well approximated target regions.

11.2.4 Influence of Source Task Solutions and Exploration

From previous experiments with BASIC DSL, we know that the development of solution qualities strongly depends on the choice of ϵ for the ϵ-greedy action selection (cf. Fig. 10.2 on page 146). Hence, we next consider the influence of $\epsilon \in \{0.001, 0.01, 0.1\}$ to the storage media-based approach. Figures 11.10 and 11.11 present the results for source tasks with 100 resp. 1000 agents.

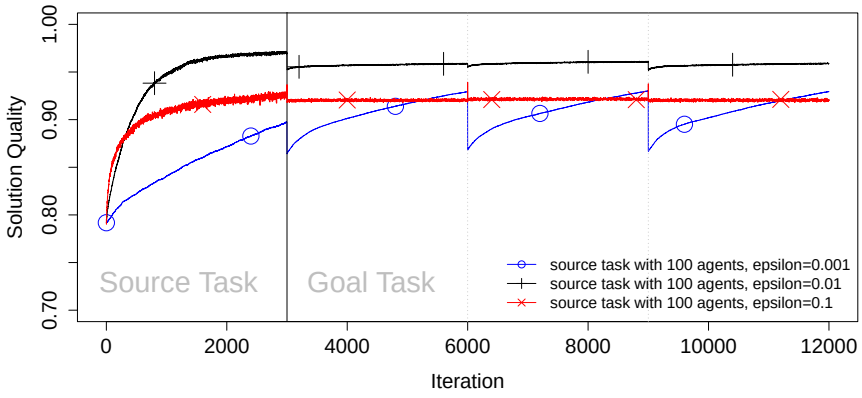


Fig. 11.10 Influence of ϵ and the solution quality in a source task with $n_{\mathrm{st}} = 100$ agents.

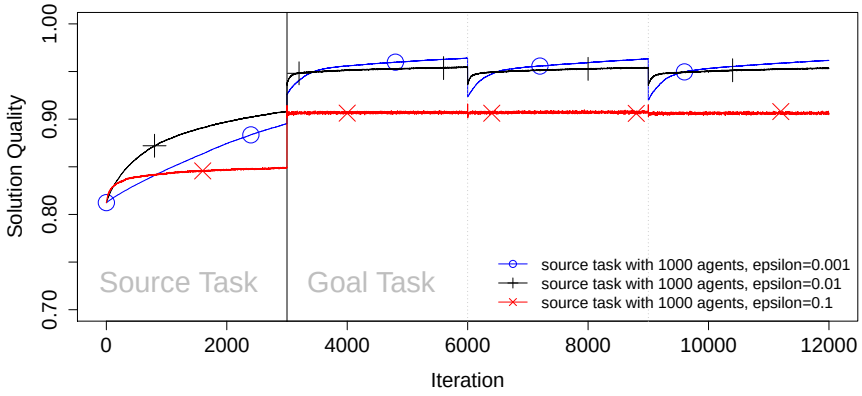


Fig. 11.11 Influence of ϵ and the solution quality in a source task with $n_{\mathrm{st}} = 1000$ agents.

Independent of the number of agents in the source task the figures show that the final performance in the source task determines the initial performance in the goal task, as well as the initial performance for the following stage games in the goal task. It is particularly noticeable, that for $\epsilon = 0.1$ a short increase in the solution quality can be observed when the knowledge of the media is used. However, the solution quality declines immediately as too many agents explore at the same time. Nevertheless, compared to the experiments with BASIC DSL and 500 agents in Sect. 10.2.1 Fig. 10.2, we observe a higher solution quality in the storage media-based approach, i.e. the influence of a comparatively high exploration rate in the proposed approach is smaller than in the basic learning approach.

In addition, these experiments indicate different performances in the goal task for $\epsilon = 0.001$ (blue graphs), although the final solution quality in the source task is comparable. The reason for this behavior can be explained by considering the target regions approximated by the storage media. For 1000 agents, the regions are mainly connected and reflect the target regions quite well. Contrary to that, in the 100 agent source task, no proper regions can be established due to very little exploration.

11.2.5 Dynamic Settings

Next, we investigate the approach in a goal task, in which agents initially work on jobs that last 3000 iterations. Afterwards, jobs with a random duration between 5 and 100 iterations located at random positions are assigned to the agents. Figure 11.12 summarizes the results.

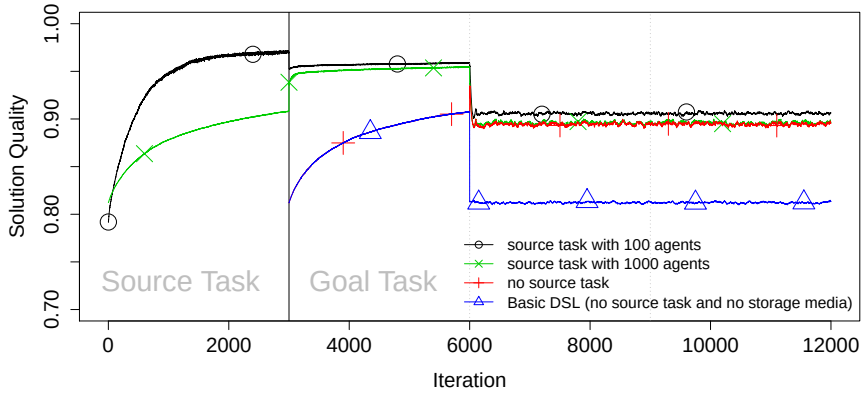


Fig. 11.12 Simulation results for a goal task with initially 3000 iteration long jobs followed by a more dynamic job generation using $\mathrm{R}(5, 100)$.

As the figure reveals, during the dynamic phase of the goal task storage media enable agents to find solutions that are significantly better compared to BASIC DSL without storage media. Furthermore, the agents benefit from better initial target proposals that are obtained if $n_{\mathrm{st}} = 100$ agents are used in the source task.

11.2.6 Locally Calculated Reward

Finally, this section investigates the storage media-based approach with locally calculated rewards as described in Sect. 9.2.1.2. Figures 11.13 and 11.14 present the results for different goal tasks, as described in the figures' captions.

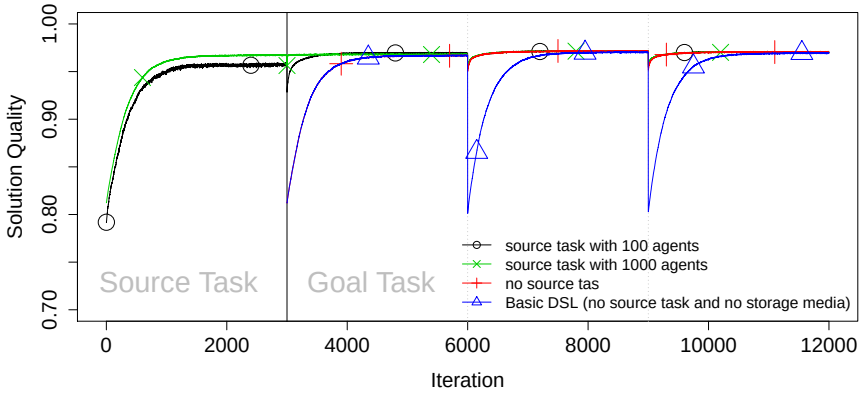


Fig. 11.13 Locally calculated rewards in a goal task with three stage games.

Figure 11.13 shows that a source task with $n_{\mathrm{st}} = 1000$ agents leads to the best results, followed by a source with 100 agents. In particular, by using storage media, there is no difference in the second and third stage game of the goal task, independent of a possible source task. Without storage media, the agents in all stage games achieve the same final solution quality, but they require much more time to do so. Accordingly, the benefits from using storage media under locally calculated rewards can be identified in faster convergence compared to BASIC DSL without storage media. Also, a simple source task with fewer agents than in the goal task seems to be a good tradeoff between improved performance and additional computational effort.

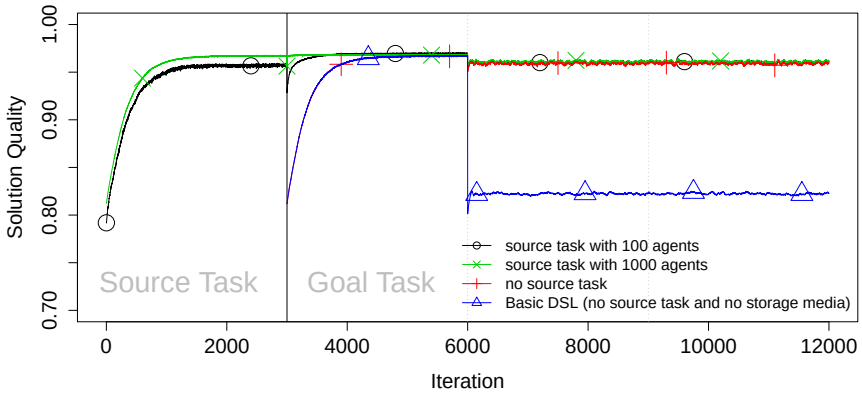


Fig. 11.14 Locally calculated rewards in a dynamic goal task with frequently changing agent positions.

In the end, reconsider the goal task that creates jobs according to $\mathrm{R}(5, 100)$ after the first "stable" stage game is over. The results for this setting are illustrated in Fig. 11.14 and basically are in line with the three stage game goal task discussed before. In detail, the solution qualities under local rewards do not differ in the dynamic phase if storage media are used. Contrary to BASIC DSL, the storage media-based approach is able to deal with the dynamic phase and finds high quality solutions by relying on the knowledge stored on the media.

11.2.7 Summary

The empirical findings presented in the preceding subsections considered various aspects of the proposed storage media-based learning approach. Although further experiments could be done, the most important characteristics have been considered. In detail, the main results can be summarized as follows:

- Agents can benefit from learned and externally stored knowledge, in particular, such knowledge enables the agents to quickly find high quality solutions.
- Using knowledge from a simplified source task significantly improves the overall performance of the learning MAS. Furthermore, agents that can base their learning on source task knowledge can improve the solution quality faster than agents that learn without the help of such knowledge.
- If the source and the goal task do not differ in the number of agents, then agents benefit only in the first stage game. In consecutive stage games no significant difference is observable.
- Even though storage media do not actively execute a learning algorithm, the statistic data that they gather from the learning agents enables them to approximate target regions. However, the quality of these approximations depends on the number of media, their positioning in the environment, as well as on the number of agents.
- The performance of agents in the goal task is influenced by the solution quality obtained in the source task because this determines the media's initial target proposals.
- The storage media-based approach is able to find better solutions under high exploration rates than BASIC DSL.
- In dynamic settings, in which agents change positions in (almost) each iteration, the proposed approach still finds high quality solutions ($\mathsf{APQ} \approx 0.9$) using a global reward, which does not hold for BASIC DSL that only reaches an average partitioning quality of $\mathsf{APQ} \approx 0.8$.
- Under locally calculated rewards, the benefit of storage media in a goal task with three stage games is limited to an improved convergence speed. It does not lead to a higher final solution quality. In highly dynamic settings with constantly moving agents, however, the storage media-based approach with local rewards achieves an APQ of approximately 0.95, whereas BASIC DSL without media only achieves approximately 0.82.

11.3 Conclusion

This chapter presented a novel generic approach towards multiagent reinforcement learning in combination with knowledge stored outside of the learning agents. A concrete implementation of that approach was analyzed empirically and shown to possess several beneficial properties concerning convergence speed, speed of learning, the achieved solution qualities, and the behavior under dynamic settings. Since the preceding Sect. 11.2.7 already provides a summary on all these results, we won't repeat them. Instead, we discuss the obtained insights from a broader point of view.

First of all, consider the approach from the perspective of Transfer Learning. Since TL in MARL is a relatively new research area, only few related papers have been published so far. However, from these papers, e.g. [BPV12] or [VDHN11], one can deduce that one difference between source and goal task in multiagent settings can be found in the number of agents involved. This is also what distinguishes our simplified source task with 100 agents from the goal task with 1000 agents. Since the aforementioned works only consider small systems with two or three agents, our approach—to the best of our knowledge—is the first one that considers transfer of knowledge in large MAS with hundreds of agents. Contrary to [BPV12] and [VDHN11], our storage media-based approach also does neither transfer Q-values nor strategies from source to target task, but it directly influences the action selection by transferring statistical knowledge from the source task. In general, and also taking the insights from the region-based heuristics in Ch. 12 into account, we can state that storage media and knowledge transfer work well in our setting due to an inherent property of the considered IAPP instance. The property we refer to is that of *target regions* as investigated in Ch. 12. Given that agents are able to learn this underlying structure, the resulting knowledge can be stored on media such that, independent of actual agent positions and, to some extent, also independent of the number of agents, high quality solutions are likely to be found when the media's knowledge is used.

Accordingly, using storage media in the context of learning multiagent systems can generally be considered as a coordination technique, too. In particular, the media can be useful in dynamic systems with mobile agents or in open agent systems where the number of agents changes over time. The key property is its ability to maintain knowledge in an environment even if agents "disappear".

Chapter 12

Region-Based Heuristics

In this chapter, we present some region-based heuristics for the IAPP instance considered in this work.

The basic idea of the developed algorithms stems from an interesting observation on high quality solutions for a fixed iteration of the IAPP. As illustrated in Fig. 12.1, such solutions show connected regions of agents that select exactly the same target. From this observation, we claim that such connected regions can be used as basis for finding good solutions in all iterations of an IAPP instance, if the mobile agents are always placed according to a uniform distribution. The basic idea is, to calculate such regions and to let each agent choose the target belonging to the region the agent is located in.

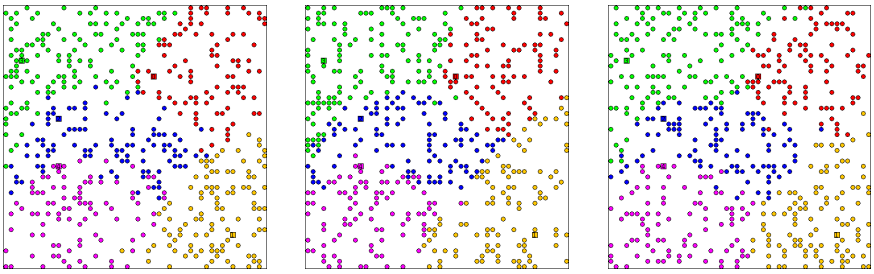


Fig. 12.1 High quality solutions for three IAPP iterations with same target but different agent positions ($m = 5, n = 500$).

The remainder of this section is organized as follows. First, we define the regions and formalize the above claim as an hypothesis. Then, we investigate the error between using regions and optimal solutions in a two target IAPP setting. Afterwards, we develop a local heuristic based on the Exchange Target Strategy (ETS) [Goe07] and analyze the number of messages it sends. All the aforementioned parts have already been published in [KKB11d], and as an extended version in a technical report [KKB10b]. Hereafter, we will use the latter for reference purposes and basically follow its structure. Finally, we also investigate Voronoi Diagrams as a special region type.

12.1 Target-Regions

First of all, we need to define the regions that are claimed to provide a good basis for target selection:

Definition 28 (Target-Regions [KKB11d]). A *Target-Region* $\mathsf{TR}(T)$ for any target T in a *full* grid environment is defined by a set of cells C_T. The set consists of target T's cell and all cells of agents that, in an *optimal solution*, are assigned to T.

In a *full grid environment* all cells are occupied by agents or targets. Contrary to a full grid environment, an environment with at least one free cell is called *sparse*. One issue with the above definition is that the regions are based on optimal solutions, and as mentioned earlier, calculating such optimal solutions is assumed to be at least $\mathcal{NP}$-hard for general settings. Thus, we will have to use approximated Target-Regions.

12.1.1 Properties

Before we investigate the expected solution qualities for TR-based algorithms, we briefly state some useful properties. Due to Prop. 2 on page 68 we know that there is more than one optimal solution for a fixed positioning in general. This leads to the following property:

Property 6 ([KKB10b]). *There can be different Target-Regions for the same full grid environment.*

Since each agent and target in our environment occupies one cell and because optimal solutions according to Prop. 3 can have different partition sizes, Prop. 7 follows immediately.

Property 7 ([KKB10b]). *Target-Regions do not necessarily occupy the same area.*

Next, consider Fig. 12.2, which allows us to state Prop. 8.

Property 8 ([KKB10b]). *Target-Regions may be unconnected.*

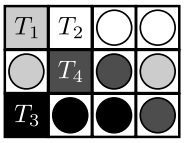


Fig. 12.2 Optimal solution ($f =\sim 0.9347$) with unconnected Target-Regions.

Finally, we can deduce Prop. 9 from Prop. 5 in conjunction with the Target-Regions shown in Fig. 12.3, which illustrates the setting used in the proof of Prop. 5.

Property 9. *Target-Regions used in a sparse environment, that resulted in an optimal solution at iteration t, are not guaranteed to produce optimal partitionings in iteration $t+1$, if at least one agent has moved by at most one step.*

We will refer to these Target-Region properties later on.

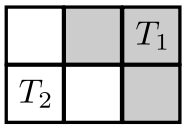


Fig. 12.3 Target-Regions.

12.1.2 Solution Quality

According to Def. 28, Target-Regions provide optimal solutions for all iterations in IAPP instances that have a full environment. This follows because the solution quality (cf. (5.2)) does not distinguish between different agents as it considers only their distances and target assignments. Hypothesis 1 deals with the more interesting question about the development of the *expected* solution qualities in sparse environments. It is particularly noticeable, that the hypothesis deals with expectations, since Target-Regions by Prop. 9 are not guaranteed to lead to optimal solutions in the IAPP with moving agents.

Hypothesis 1 ([KKB10b]) *Consider the IAPP where agents create partitionings based on Target-Regions. Then the resulting average partitioning quality according to (5.1) is expected to be*

1. *high if agents are distributed uniformly, and*
2. *low if agents are distributed according to a normal distribution.*

The solution quality further depends on the ratios between the number of agents, targets, and the size of the environment.

In order to verify the expected high solution qualities in case of uniformly distributed agents, we conducted experiments that are described in detail in [KKB10b]. We used the TTO strategy (cf. Sect. 2.2 or [Goe07, Sect. 3.2.2]) to calculate optimal solutions for a set of different scenarios. Afterwards, we compared the results based on the pre-calculated Target-Regions with those obtained by using the TTO approach in every iteration. As metric we considered the average error, i.e. the average difference over all iterations between the optimal solutions and those obtained by Target-Regions. The investigated settings all involve two targets and different numbers of agents as well as environment dimensions. We had to restrict these investigations to the two target case, as there is no known algorithm for settings with more targets. Figure 12.4 visualizes the results with 95% confidence intervals, whereas each value is averaged over 100 repetitions.

The visualized simulation results also support the second statement of the hypothesis. To further clarify why solutions in settings with normally distributed agents are expected to result in worse solutions consider Fig. 12.5. There, the solution quality obviously is expected to be low since all agents are placed within a small part of the environment determined by the normal distribution. Accordingly, neither the DISTANCE nor the DISTRIBUTION objective values will be high. Extending this example by considering a position in the middle of the environment as well as a *large* standard deviation, however, leads to another conclusion. In such a setting the normal distribution may roughly approximate a uniform distribution. This could result in better solutions as agents may also benefit from Target-Regions.

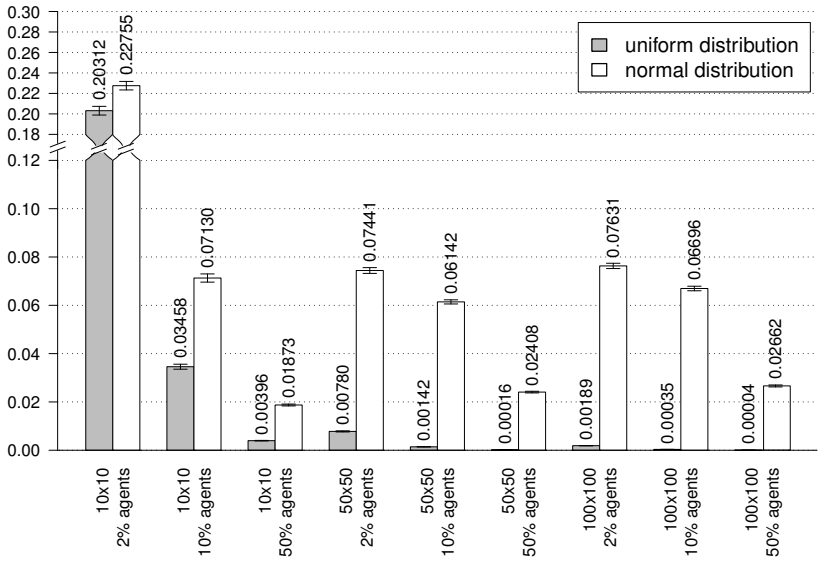


Fig. 12.4 Average error TTO vs. TTO-Target-Regions (from [KKB10b]).

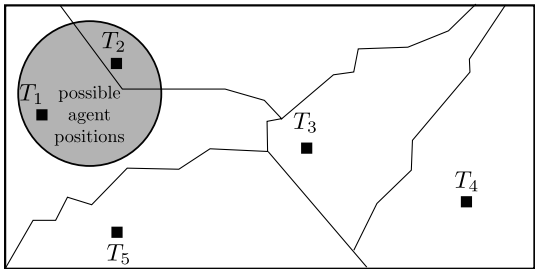


Fig. 12.5 Schematic Target-Regions with normally distributed agents.

Based on this empirical evaluation of Hypothesis 1, we conclude that Target-Regions are a proper means to find high quality solutions for the IAPP if agents are uniformly distributed. In the next section, we will describe an approach that approximates Target-Regions for settings with $m \geq 3$ targets.

12.2 Approximation of Target-Regions

The *Exchange Target Strategy (ETS)* (cf. Sect. 2.2) is known for calculating high quality solutions in static settings [Goe07]. However, it requires $\mathcal{O}(\mathsf{k}n)$ messages in each iteration, where k denotes the number of neighbors of an agent and n the total number of agents. Since the solutions produced by ETS also show clear regions, we will use it to approximate Target-Regions.

Definition 29 (ETS-Target-Region [KKB10b]). An *ETS-Target-Region ETS-TR*(T) for any target T in a full grid environment is defined by a set of cells that consists of

target T's cell and all cells whose agents are assigned to T after the Exchange Target Strategy has converged.

Definition 29 defines the approximated Target-Regions in a vague sense, as it depends on the convergence of ETS. In practice, we mean by convergence that no significant improvement can be observed after some x iterations of the ETS algorithm.

In the remainder of this section, we continue by showing that ETS-Target-Regions lead to high quality solutions in the two target case. Additionally, we provide a general formula to estimate the expected solution quality (Sect. 12.2.1). Afterwards, we propose a storage media-based algorithm to approximate ETS-Target-Regions in sparse environments (Sect. 12.2.2).

12.2.1 Quality of ETS-Target-Regions

To investigate the expected solution quality of ETS-Target-Regions, we construct the regions for a particular setting offline. Therefore, we consider a full environment and execute the ETS algorithm for x iterations, where x must be sufficiently large to ensure convergence in the considered problem setting. After x iterations, the agents mark their position with their current target assignment. In a second step, we place the desired number of agents in that environment and let them choose their target based on the available ETS-Target-Region information.

In [KKB10b], we simulated the ETS-based approach in the same settings as in the previous section. The regions where evolved over $x = 2000$ ETS iterations, as this always has led to convergence in preliminary experiments. Figure 12.6 shows the resulting average error and 95% confidence intervals of the ETS-based approach compared to the optimal algorithm for two targets.

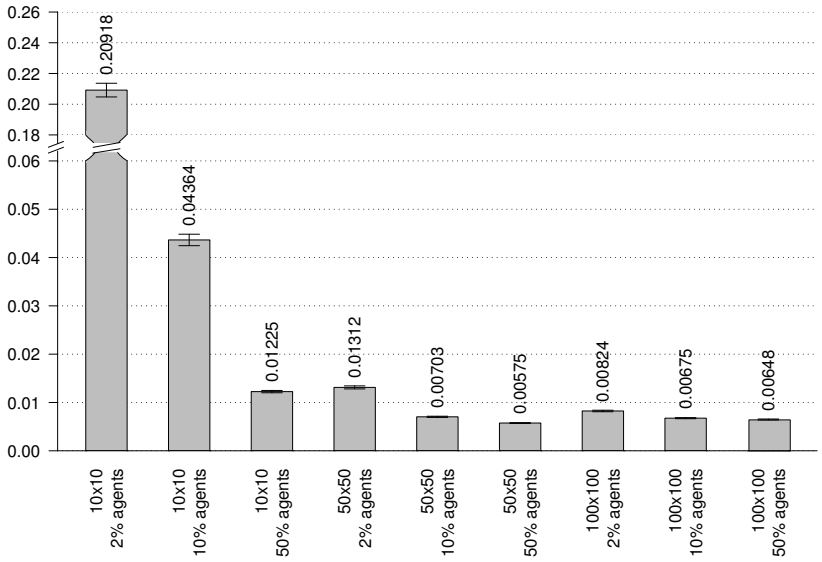


Fig. 12.6 Average error TTO vs. ETS-Target-Regions (from [KKB10b]).

A comparison of these results to those for Target-Regions (see Fig. 12.4), reveals slightly higher error rates. Nevertheless, given proper ratios for the number of agents in an environment, the average error is around 1% or less. Thus, we conclude that ETS-Target-Regions constitute a good means to approximate Target-Regions. Further empirical results involving more than two targets will be presented in Sect. 12.2.2.

Based on [KKB10b], let us describe a formula to estimate the average partitioning quality that can be expected when using ETS-Target-Regions in settings with uniformly distributed agents. The idea of the estimation is to combine an expectation value for the DISTRIBUTION objective with the DISTANCE objective value obtained in the last iteration of the ETS-Target-Region creation procedure.

In order to calculate the expected DISTRIBUTION value—and although we know from Prop. 7 that this assumption does not hold for general settings—we assume equal sized ETS-Target-Regions. In detail, in an environment $\mathcal{E}$ that contains $c = |\mathcal{E}| - |\mathcal{T}|$ possible agent positions, each region's size is assumed to be $\frac{c}{m}$. Furthermore, let $H_m(n)$ describe all possible partitionings of n non-distinguishable agents to m distinguishable targets. Formally, define $H_m(n) = \{h \mid \sum_{j\in\{1,\dots,m\}} h[j] = n\}$ to be the set that contains all tuples h with m non-negative integer values whose components sum to n. With the help of a discrete random variable X over all possible agent positionings, a discrete uniform distribution $Pr(X)$ over X, and a function $d : X \to [0, 1]$ that gives the DISTRIBUTION value for a positioning as in evaluation function (5.2), we can calculate the expected value $E[d(X)]$ as follows:

$$\begin{aligned} E[d(X)] &= \sum_j d(x_j) Pr(x_j) \\ &= \sum_{h\in H_m(n)} \left(\frac{\prod_{i=1}^{m} h[i]}{\left(\frac{n}{m}\right)^m} \cdot \frac{\prod_{i=1}^{m} \binom{\frac{c}{m}}{h[i]}}{\binom{c}{n}} \right) \end{aligned} \tag{12.1}$$

Given further information about region sizes, the approach above can also be used to estimate DISTRIBUTION objective values for other Target-Region approximations.

In contrast to the DISTRIBUTION estimation, a general formula for determining the expected DISTANCE value may not exist, as the value strongly depends on the positions of agents and targets. Thus, the estimation formula uses the DISTANCE objective value $\mathrm{do}_{\mathrm{last}}$ obtained in the last iteration of the ETS-Target-Region creation process.

Equation (12.2) combines both estimates to calculate the overall expected average solution quality over all iterations in an IAPP instance.

$$\widehat{\mathsf{APQ}} = \alpha \cdot E[d(X)] + \beta \cdot \mathrm{do}_{\mathrm{last}} \tag{12.2}$$

We verified these formulas empirically for different settings. Table 12.1 shows the results for the DISTRIBUTION estimation. From the low percentage difference, we conclude a high precision in the investigated experiments. The overall estimated solution quality, as presented in Tab. 12.2, also looks promising. Further details on the experiments can be found in in [KKB10b].

Table 12.1 Average distribution objective value (distr.) vs. expectation value $E[d(X)]$ (taken from [KKB10b]).

Grid	2% Agents			10% Agents			50% Agents		
Size	distr.	$E[d(X)]$	diff. (%)	distr.	$E[d(X)]$	diff. (%)	distr.	$E[d(X)]$	diff. (%)
10x10	0.51510	0.50515	1.932	0.90836	0.90928	-0.101	0.98989	0.99010	-0.021
50x50	0.98040	0.98039	0.001	0.99644	0.99640	0.004	0.99961	0.99960	0.001
100x100	0.99508	0.99510	-0.002	0.99910	0.99910	0.000	0.99990	0.99990	0.000

Table 12.2 Overall estimated solution quality $\widehat{\mathsf{APQ}}$ for n agents and m targets in a 50×50 grid environment (taken from [KKB10b]).

m	n	$E[d(X)]$	$\mathrm{do}_{\mathrm{last}}$	$\widehat{\mathsf{APQ}}$	APQ
2	250	0.99640	0.99582	0.99611	0.99616
3	250	0.98922	0.99282	0.99102	0.98914
4	200	0.97261	0.97453	0.97357	0.97093
4	40	0.85884	0.97453	0.91669	0.92136

12.2.2 Storage Media-Based ETS-Target-Region Approximation

Since ETS-Target-Regions have been shown to provide high quality solutions in the previous section, we next describe a storage media-based algorithm for sparse environments. The basic idea of the *Sparse-ETS-Target-Region (S-ETS-TR)* approach [KKB10b] is to execute the ETS approach in a first phase for $j_{\max}$ iterations. Afterwards, the resulting target assignment of each agent and the agent's position are stored on a storage medium nearby. At the end of phase one, each medium has a detailed statistics on how frequently which target was selected at which position. In the second phase, the agents stop to execute the Exchange Target Strategy, but request a target assignment from a nearby medium. Since there are probably unevaluated positions, a simple classification technique is used to find a proper assignment from surrounding, evaluated positions.

Algorithms 10 and 11 describe the S-ETS-TR approach in detail. The classification approach is presented in Alg. 12[1].

We empirically investigated the S-ETS-TR approach in a 51×51 grid environment with 9 regularly aligned storage media and different numbers of agents and targets. Agents consider a 8-neighborhood and use a flip probability of 0.1 in ETS. Jobs have a randomly chosen duration from 20 to 30 iterations and both phases of S-ETS-TR last 500 iterations. Figure 12.7 illustrates this setting and provides an example for the development of the ETS-Target-Region approximations by the S-ETS-TR approach. The simulation results shown in Tab. 12.3 as well as in Figs. 12.8 and 12.9 are averaged over 100 repetitions. The average partitioning qualities refer to the second phase of the approaches, only. The misclassification rate considers all but the unclassified cells and denotes the percentage of differently classified cells in S-ETS-TR compared to ETS-TR.

[1] Note that the classification process is realized through the media (cf. Alg. 12). Contrary to artifacts in the A&A meta-model (cf. Sect. 6.1.1), however, storage media should not be able to realize such computational functionalities. Nevertheless, for the sake of simplicity in our simulations, we decided to not implement the classification within the agents, e.g. by requesting required information from and storing the result on the media.

Algorithm 10 ([KKB10b]) Executed by each agent $i \in \mathcal{A}$

```
1: procedure SPARSE-ETS-TARGET-REGIONS
2:     M ← nearest storage medium
3:     if current iteration < j_max then                          ▷ Phase 1
4:         T ← target assignment of an ETS iteration
5:         M.STORE-INFO(T, p(i))        ▷ Store target and position information on medium
6:     else                                                       ▷ Phase 2
7:         T ← m.CLASSIFY-POSITION(p(i))
8:     if T is set then
9:         assign agent to T
10:    else
11:        keep last target assignment
```

Algorithm 11 ([KKB10b]) Executed on storage media.

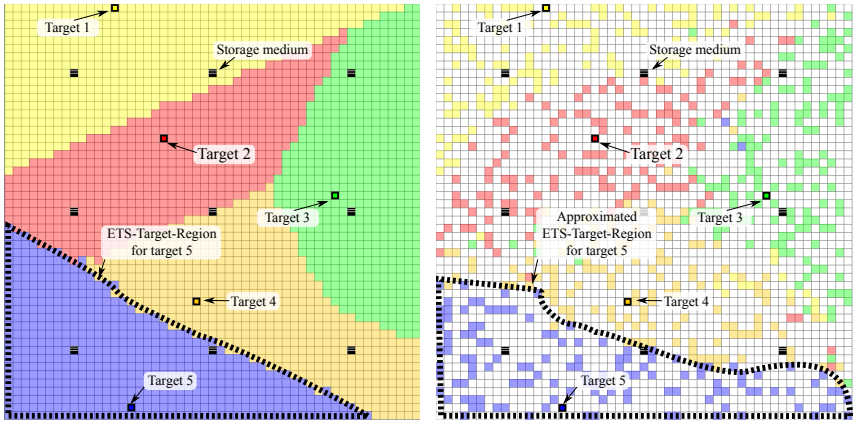
```
1: procedure STORE-INFO(Target T, Position p)
2:     add T to known targets T_known
3:     rp ← map p to relative position used in internal data structures (CA and TA)
4:     CA(T)[rp.x][rp.y] += 1                  ▷ Increase counter for T at relative position
5:     T_max ← arg max_{T' ∈ T_known} {CA(T')[rp.x][rp.y]}
                          ▷ Determine most frequently selected target T_max at rp
6:     TA[rp.x][rp.y] ← T_max   ▷ Store T_max in target assignment array at relative position
```

Algorithm 12 ([KKB10b]) Executed on storage media.

```
1: procedure CLASSIFY-POSITION(Position p)
2:     rp ← map p to relative position used in internal data structure TA
3:     if TA[rp.x][rp.y] is classified then
4:         return TA[rp.x][rp.y]      ▷ Return most frequently selected target for that position
5:     N_rp ← Moore neighborhood of rp in TA
6:     determine set of targets T_max that are selected at most in N_rp
7:     if no element in N_rp is classified then
8:         return null
9:     else
10:        T ← select random target from T_max
11:        store T in TA[rp.x][rp.y]
12:        return T
```

Table 12.3 Simulation results for S-ETS-TR ([KKB10b]).

m	n	Misclassification rate (%)	Confidence interval	Unclassified cells (%)	Confidence interval
2	100	5.376	[5.064,5.687]	19.881	[19.770,19.992]
2	500	6.363	[5.835,6.890]	0.008	[0.005,0.011]
5	100	11.743	[11.358,12.128]	19.714	[19.597,19.832]
5	500	13.947	[13.291,14.603]	0.007	[0.004,0.010]
16	100	21.861	[21.437,22.284]	19.607	[19.504,19.710]
16	500	22.832	[22.159,23.505]	0.010	[0.007,0.014]



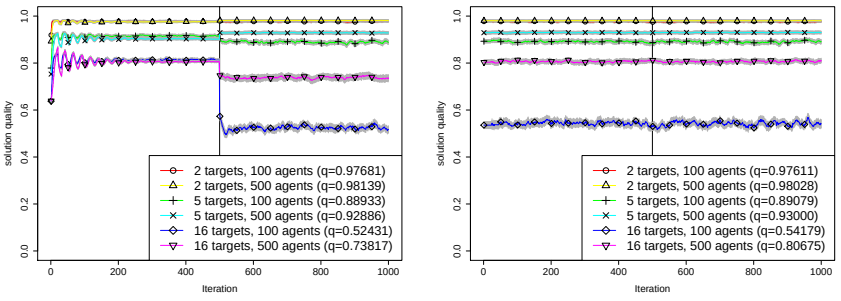
(a) ETS-Target-Regions that have evolved after 2000 iterations in a full environment.

(b) 50 S-ETS-TR iterations.

(c) 500 S-ETS-TR iterations.

(d) 1000 S-ETS-TR iterations

Fig. 12.7 ETS-Target-Regions (a) and the development of their approximations through the S-ETS-TR approach (b-d). Cell colors correspond to the target of a region. ([KKB10b])



(a) S-ETS-TR

(b) ETS-TR

Fig. 12.8 Average partitioning qualities with 95% confidence intervals as gray shadows. The vertical line at iteration 500 marks the transition to the second phase. ([KKB10b])

First of all, the results show that average partitioning qualities obtained in S-ETS-TR are very close to those obtained using ETS-Target-Regions in four out of six settings (cf. Fig. 12.8). Differences can be observed in the settings for $m = 16$ targets (cf. Fig. 12.9). We argue that these differences are rooted in the agent to target ratio as well as in the higher misclassification rates. However, it is unclear why the error is more than twice as large for $n = 500$ compared to $n = 100$ agents. We conjecture that this issue can be traced back to the properties of the evaluation function. Note that misclassification of cells mainly occurs at region boundaries. Since a higher number of targets also leads to more regions boundaries, the increasing misclassification rates in Tab. 12.3 hence are not surprising.

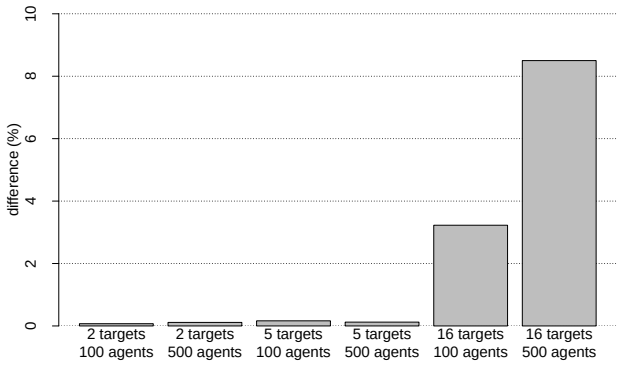


Fig. 12.9 Difference of S-ETS-TR APQ from ETS-TR APQ in percentage terms (based on [KKB10b, Tab. 3]).

12.2.2.1 Analysis

Based on the analysis provided in [KKB10b], we next analyze the S-ETS-TR approach regarding the cost objective in terms of messages. The number of messages per agent in the pure Exchange Target Strategy (without storage media interaction) per iteration is bounded from above by $2\mathsf{k} + 2$, where k is the size of the agent's neighborhood[2]. This follows because an agent sends a request to and receives a response from each neighbor. In addition, at most two messages are required to actually initiate a target exchange.

Since every agent stores information on a medium in each iteration, the number of messages in the first phase of S-ETS-TR over all n agents is $j_{\max} \cdot n \cdot (2\mathsf{k} + 2 + 1)$, where $j_{\max}$ is the number of iterations in the first phase. In the second phase, the agents only interact with the media by sending a request and receiving a response message. Hence, the number of messages for the second phase is bounded by $(\imath - j_{\max}) \cdot 2n$, where $\imath$ denotes the total number of iterations. Accordingly, the upper bound of messages in S-ETS-TR is given by

[2] Contrary to [KKB10b], where we made no clear statement, we here assume that an agent is not part of its own neighborhood. Hence, we explicitly have to consider two messages for initiating the actual target exchange.

$$j_{\max} \cdot n \cdot (2\mathsf{k}+3) + (\imath - j_{\max}) \cdot 2n.$$

Since the pure ETS approach sends at most $\imath n 2(\mathsf{k}+1)$ messages during $\imath$ iterations, it can be shown by basic transformations that S-ETS-TR becomes more cost effective than ETS if $\imath > j_{\max} \cdot \frac{\mathsf{k}+\frac{1}{2}}{\mathsf{k}}$ holds. For the experiments investigated in the previous section, the storage medium based approach hence is more cost efficient in terms of messages than the basic Exchange Target Strategy.

12.3 Voronoi Regions

The decomposition of a metric space into regions using Voronoi diagrams (VD) has been studied exhaustively. A Voronoi diagram for a set of sites S in the Euclidean space is defined by a set of Voronoi regions (VR) for each site $s \in S$. These regions are defined in such a way that each point $p \in \mathrm{VR}(s)$ is closer to s than to any other $s' \in S \setminus \{s\}$. A special variant of Voronoi Diagrams is generated from sites that also are the centroids, i.e. centers of mass of their corresponding regions according to a given density function ρ. In this work, we assume a constant density function $\rho = 1$. In literature, these variants are often called Centroidal Voronoi Tessellations (CVT) [DFG99] or diagrams (CVD). For further information on Voronoi diagrams please consult [DFG99], [Aur91], or [LWL$^+$09].

In the context of the IAPP, we may use the target positions as generating sites for Voronoi diagrams in order to approximate Target-Regions. By definition, Voronoi regions will result in an optimal solution value for the distance objective. However, no general statement on the distribution quality can be made, except for the worst-case which is a distribution value of zero. This worst case can happen since region sizes could differ significantly, which might leads to situations where the probability that at least one agent is located within each region approaches zero.

Figure 12.10 shows Voronoi regions that approximate Target-Regions in two different scenarios. Obviously, in a *full* environment for the setting shown on the left hand side of the figure, the number of agents assigned to target T_2 and T_4 will not fulfill the equal distribution objective. However, the Centroidal Voronoi diagram on the right hand side of Fig. 12.10 calculates exact Target-Regions and, hence, finds an optimal solution in a full environment. In general, Voronoi diagrams

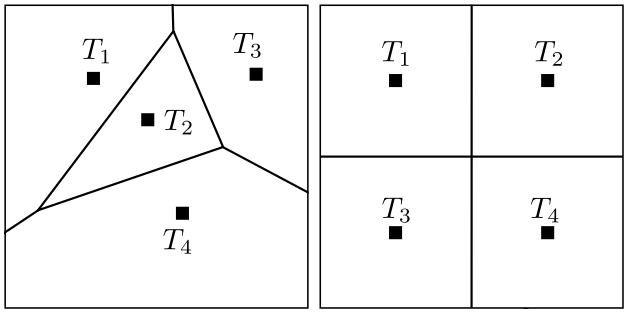


Fig. 12.10 Voronoi diagrams.

describe Target-Regions exactly whenever the considered scenario has the following properties:

- The environment can be divided in m equally sized rectangles R_j.
- Each target $T_j \in \mathcal{T}$ is located at the center of a region R_j.
- The environment is *full*, i.e. all positions are occupied by agents or targets.

Note that all Voronoi diagrams in these settings are CVTs, i.e. targets are located on the centroid of their corresponding Target-Regions. In general, however, (Centroidal) Voronoi diagrams are unable to calculate Target-Regions as stated in the Lemma 10.

Lemma 10. *Voronoi diagrams as well as Centroidal Voronoi diagrams can not guarantee to construct optimal partitionings.*

Proof. By definition, any Voronoi region for a site s contains those points that are closer, according to the Euclidean metric, to s than to any other generating site $s' \in S \setminus \{s\}$. It follows, that VR are always connected polygons. However, we know from Prop. 8 that Target-Regions must not necessarily be connected. Obviously, the lemma follows. □

Next, we consider a special IAPP variant in which only the number of targets is given but not their position. We investigate this variant using Centroidal Voronoi diagrams and investigate their performance if used as approximation technique. First of all, there exist several approaches to calculate such CVTs; a frequently used approach is the iterative method of Lloyd (see e.g. [DFG99]). It starts with a (random) initial set of $|S|$ sites for which a Voronoi Diagram (VD) is calculated. Then, the centroids of the resulting Voronoi regions are determined and used as new sites for the next iteration. The algorithm repeatedly calculates VDs and new sites until it terminates due to some convergence criterion. Voronoi diagrams in two dimensions can be calculated efficiently, e.g. in time $\mathcal{O}(|S| \log |S|)$ using Fortune's sweepline approach [For87]. An introduction to Voronoi Diagrams and a survey on fundamental techniques can be found in [GO04, Ch. 23], and [Aur91], for instance.

Inspired by Lloyd's iterative approach, Voronoi regions in our special IAPP variant are evolved for a given number of targets. Targets then are placed at the resulting generating sites of the CVT. To investigate the solution qualities that can be expected from this approach, we consider the following hypothesis:

Hypothesis 2 *Let* $\mathrm{CVT}(\mathcal{T})$ *be a centroidal Voronoi diagram for a large number of targets* $m \to \infty$ *in a grid environment. If uniformly at random distributed agents in each iteration of the IAPP select targets based on the Voronoi regions of* $\mathrm{CVT}(\mathcal{T})$*, then the average partitioning quality is expected to be*

$$\widehat{\mathsf{APQ}} = \alpha \cdot E[d(X)] + \beta \cdot 1, \tag{12.3}$$

where $E[d(X)]$ *is calculated as in (12.1) described in Sect. 12.2.1.*

Proof. (informal) Gersho [Ger79] conjectured that Voronoi regions in an optimal CVT are *asymptotically congruent* and their geometric form depends on the dimension of the point set. According to [LWL+09], this conjecture remains open for dimensions ≥ 3. However, for our particular 2-dimensional case, the conjecture holds and regions for $m \to \infty$ sites converge to regular hexagons (see e.g. [Gru04] for more details and references to proofs). Hence, in our 2-dimensional grid environment, CVTs for $m \to \infty$ sites result in Voronoi regions that have approximately equally

sized areas. As this satisfies the assumption made for calculating the estimated distribution objective value $E[d(X)]$ in Sect. 12.2.1, the first part of (12.3) follows. In the end, we obtain the hypothesis since Voronoi diagrams from their very nature provide optimal solutions for the distance objective, too. □

At this point, it is important to note that CVTs in general are not unique [DFG99]. Figure 12.11 shows an exemplary CVT in a 500×500 grid environment with 128 targets after 120 Lloyd iterations. As one can see, the region sizes and shapes differ between the border and the inner regions. This difference stems from the rectangular shape of the environment and is not taken into account in the estimation above. Accordingly, the formula should be used with care or the distribution of agents has to be restricted to inner regions in order to obtain a good estimation.

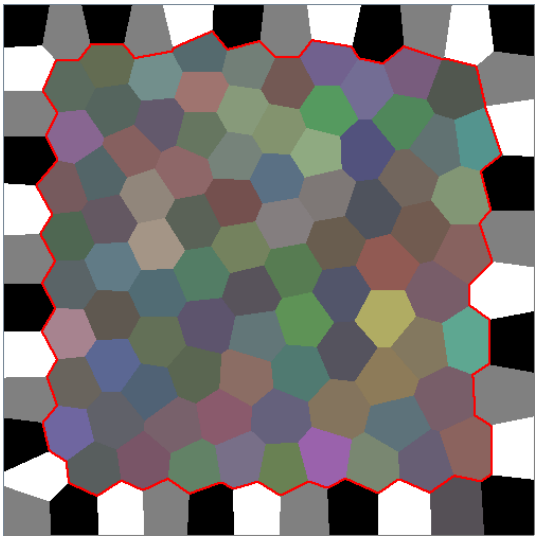


Fig. 12.11 Centroidal Voronoi Tesselation. Black, gray, and white regions are border regions.

Finally, note that (12.3) can also be used to estimate the solution quality in setting with fewer targets, if the targets are aligned as shown on the right side of Fig. 12.10 and described thereafter.

12.4 Discussion

This chapter introduced the concept of target regions to solve the IAPP. We developed this concept stepwise by first comparing it empirically with optimal solutions for settings with two targets. Then, we extended it to settings with more targets using a more or less central approach to approximate target regions. As the empirical results attributed high potential to these approximations, we continued with a distributed approach that approximates regions using the concept of storage media as introduced in Ch. 6. Finally, we also investigated Voronoi diagrams in the context of the IAPP.

It was shown for IAPP settings with two targets that target regions are able to calculate near optimal partitionings in each iteration if agents are distributed

uniformly at random. We also argued, and empirically underlined, that solution qualities are expected to be worse if agents are distributed according to a Gaussian distribution. The approximated regions, called ETS-Target-Regions, were empirically shown to solve the two target instances with an error of approximately 1% or less for appropriate settings. We deduced and evaluated a formula to estimate the expected solution quality of ETS-Target-Regions if agents are distributed uniformly. Here, the simulations showed a very good estimation quality in the investigated settings. The simulations of the storage media-based approximation of ETS-Target-Regions produced solutions that were at the same level than those of the centrally calculated ETS-Target-Regions in most cases. We also proved the conditions under which the approximation technique using storage media is more efficient than the ordinary ETS approach in terms of number of messages. In detail, the media-based algorithm becomes more efficient the longer the media are used by the agents to select a target. In the end, we briefly showed that Voronoi diagrams are not suitable for solving the IAPP in general, but they can be used for some specific settings that fulfill some assumptions on the target positions and/or number.

Part IV

Conclusion

Chapter 13

Conclusions and Future Work

Finally, this chapter briefly summarizes the main conclusions and research contributions of this thesis. We also provide an outlook on future work.

13.1 Conclusions

In this thesis, we considered multiagent reinforcement learning and coordination in large cooperative systems. In particular, we focused on a special subclass of games in which agents are confronted with a sequence of stateless games, whereas each game is played repeatedly for a certain but unknown timespan.

We defined a formal game class for the aforementioned settings, named (cooperative) sequential stage games (SSG). Based on so-called engineered rewards, i.e. reward values that in some sense encode states, we developed a multiagent reinforcement learning approach called Distributed Stateless Learning (DSL). This approach was theoretically analyzed and proven to learn (near-)optimal joint strategies for each stage game contained in the sequence of cooperative sequential stage games. DSL is based on the concept of engineered rewards and a special variant of the Distributed Q-Learning algorithm of Lauer and Riedmiller [LR00]. We exhaustively evaluated the approach in the context of a simple cooperative sequential stage game as well as in the Iterative Agent Partitioning Problem (IAPP), which is a multi-objective optimization problem in the form of a multiagent system. The results of these experiments underlined the proven convergence properties.

Furthermore, we developed several coordination strategies that can improve the speed of learning in large systems composed of several hundreds or thousands of agents. We empirically investigated these strategies in various settings of the IAPP. In particular, we showed that DSL, with the help of these additional coordination techniques, can learn faster and also manages to find high quality solutions in frequently changing settings.

Given that agents may employ sensors with limited accuracies, we examined how learning under noised reward perceptions can be realized. In particular, we first showed that deterministic Q-Learning is unable to learn under noised rewards. To address this negative result, we proposed to adopt the optimistic update rule used

in Distributed Q-Learning. Using this update rule, we proved that learning optimal policies is possible in single agent settings with noised rewards. Additionally, we examined so-called noise robust rewards, i.e. reward values that are so distant to each other that they remain distinguishable under noise. Although this kind of rewards seems to be promising, we proved it to be useless in settings with a single agent. However, we showed that the DSL approach under agent-individual noised reward perceptions can learn optimal solutions if rewards are noise robust and the game contains exactly one optimal joint action. We argued that the approach can also be extended to enable learning in general cooperative settings with multiple optimal joint strategies, if the noise robust rewards have a special structure. Furthermore, we proved that DSL learns (near-)optimal joint strategies if all agents obtain the same noised reward values. Accordingly, learning from noised rewards is possible in our approach.

As general coordination concept, we introduced storage media that are located in the environment. Agents can use them to maintain (learned) knowledge externally, such that all agents next to a medium can benefit from its stored knowledge, even if the agent that gathered the original data has become unreachable. We combined these media with the DSL approach and investigated it in the context of the IAPP. The results indicated that storage media enable agents to reuse learned knowledge from simpler settings to improve learning in more complex, resp. larger problems.

Since we focused our empirical evaluations on a particular variant of the IAPP, we also provided general insights into that problem. Furthermore, we presented an approach for the IAPP that combines a communication intensive state-of-the-art algorithm with the concept of storage media. The results indicate that media can help to significantly reduce the communication overhead of that approach. In addition, we identified that high quality solutions of the IAPP show a special structure in the form of regions which belong to targets.

In short, the main contributions of this thesis hence can be summarized as follows:

- We considered learning and coordination in multiagent problems that change stepwise and remain stationary in between these steps.
- A formal model for such sequential stage games (SSG) was developed.
- We presented the concept of engineered rewards which, in a sense, encode state information.
- Based on such encoded information, we elaborated a distributed and efficient multiagent reinforcement learning algorithm for cooperative SSGs.
- We provided a careful theoretical analysis of the approach that includes convergence results. Additionally, we investigated its behavior under noisy reward perceptions, i.e. under stochastic rewards.
- Several coordination techniques to improve convergence speed were proposed.
- In the context of a large multi-objective optimization problem, we provided an exhaustive empirical analysis of the approach and the coordination strategies, and showed their ability to learn under various dynamic settings, including, e.g., frequent changes and different reward calculation methods. Also more general insights into that problem were presented.
- We showed how externally stored knowledge improves learning in large systems.
- Learning under noisy rewards in deterministic Q-Learning was examined.

Finally, we can conclude that multiagent reinforcement learning and coordination in scenarios with sequentially changing settings is a challenging problem, but nevertheless it seems to be feasible even in large systems.

13.2 Future Work

As we have seen in the previous section, several interesting issues have been investigated in this thesis. However, there still are various aspects and possible approaches that are worth to be considered in future work. We want to point out some of them briefly in the following:

- We considered approaches for a sequence of situations that can be modeled as stateless games. In many problems, however, situations are not stateless, but they can only be described sufficiently by stochastic games, or other even more complex models. Hence, it would be interesting to investigate approaches that address learning in sequences of such games.
- Also, we only considered cooperative SSGs. Thus, the proposed DSL approach is not able to deal with competitive or mixed SSGs. Since several applications have to deal with selfish and rational agents, such non-cooperative settings in general are an interesting research area, too. Accordingly, learning in sequences of such games should also be investigated in future work.
- We concentrated our empirical investigations on a particular variant of the IAPP. In future work, one could consider different variants, including even more dynamically changing scenarios with respect to the number of agents or targets, as well as settings where targets request different numbers of agents. Additionally, the consideration of other more complex objectives could lead to interesting insights.
- In general, it would be interesting to consider the proposed approaches and concepts in completely different problem domains, which can also be modeled as cooperative sequential stage games.

Part V

Appendix

Appendix A

Medium Coverage

This appendix covers the details on the storage media placement problem discussed in Sect. 6.3. In detail, we consider an environment with a single rectangular obstacle that prohibits communication between agents located at different sides of the obstacle. From a single agent's perspective, we investigate where a storage medium should be placed in order to allow indirect communication via the medium with as many agents as possible that are located on the other side of the obstacle.

We start by developing a formula to calculate the coverage of a medium in Sect. A.1. Section A.2 then proves a position that leads to the maximum coverage under the assumptions stated below.

Since this placement problem depends on several parameters, e.g. the communication radii of agents and media, as well as the dimensions of the obstacle, we restrict our investigations to settings that can be described by the following assumptions. Fig. A.2 illustrates such a setting.

Assumptions 3 *The investigated setting includes one agent i with communication radius r_i, one rectangular obstacle* so*, and a storage medium M with an interaction radius of r_M. Furthermore, we assume:*

- *The considered agent i is located at* $\mathsf{pos}(i) = (0, 0)$.
- *The sensor obstacle*
  1. *is placed such that its bottom right edge is at position (b, a), with $a, b > 0$,*
  2. *has height $h > 0$,*
  3. *has a length $> r_i + b$, such that we only have to consider the case in which the medium is placed on the right hand side of the obstacle.*
- *To simplify the calculations and to reduce the number of (geometric) cases that have to be considered, the medium's communication radius has to be*

$$r_M \leq \min \left\{ \frac{1}{2} r_i, \left| \sqrt{(a+h)^2 + \left(\frac{b(a+h)}{a} \right)^2} \right| \right\}.$$

The first part of the minimum operator allows us to neglect cases where agents are located outside of the considered agent's communication radius. The second element follows from point $\left(\left(\frac{b(a+h)}{a} \right), a + h \right)$, which is the intersection of the

prolonged upper obstacle edge and a visibility line (ℓ, cf. Fig. A.2) that determines the area shadowed by the obstacle.

A.1 Computation of Medium Coverage

First of all, we are interested in the area that is *shadowed* by a sensor obstacle for a particular agent i. By *shadowed* we mean that direct communication between agent i and any another agent is not possible due to the obstacle. Figure A.1 visualizes the shadowed region as light gray area for agent i.

The shadowed area $\mathrm{A}_s^{i,\mathsf{so}}$ for agent i with respect to obstacle so is defined by a visibility line ℓ that connects the agent with the bottom right edge of the obstacle and its intersection with the communication range of the agent. The latter is defined by the agent's position $\mathsf{pos}(i)$ and its communication radius r_i. Under assumptions A3, this area can be calculated using (A.1), where the variables are those shown in Fig. A.1 and $\mathrm{A}\,(\mathsf{A}), \mathrm{A}\,(\mathsf{B}), \mathrm{A}\,(\mathsf{C}), \mathrm{A}\,(\mathsf{D}))$ denote the areas shown on the right hand side of that figure. Note that the areas can be calculated using basic geometric formula which we won't discuss in detail.

$$\begin{aligned}\mathrm{A}_s^{i,\mathsf{so}} &= \mathrm{A}\,(\mathsf{A}) + \mathrm{A}\,(\mathsf{B}) - (\mathrm{A}\,(\mathsf{D}) - \mathrm{A}\,(\mathsf{C})) \\ &= \left(r_i^2 \left(\frac{\pi\alpha}{180^\circ} - \sin\alpha\cos\alpha\right)\right) + \left(\frac{h \cdot \frac{h}{\tan\beta}}{2}\right) \\ &\quad - \left[\left(\frac{\pi \cdot r_i^2 \cdot \gamma}{360^\circ}\right) - \left(\frac{1}{2} \cdot \frac{r_i}{\sin\zeta} \cdot \sin\gamma \cdot r_i \cdot \sin\epsilon\right)\right] \end{aligned} \tag{A.1}$$

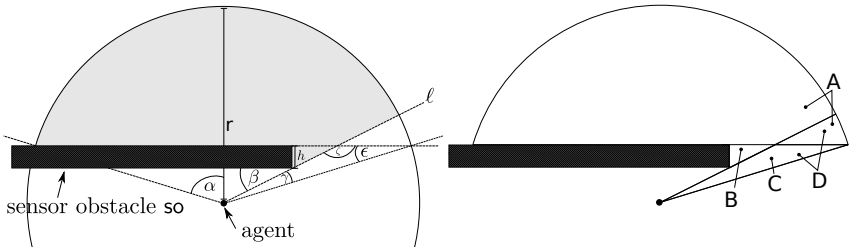


Fig. A.1 Area that is shadowed by the obstacle.

Before we can define the coverage of a medium, we have to specify the influence of the sensor obstacle to the communication of agents. We assume that communication between agents at two positions P and Q is possible if and only if the line connecting P and Q does not intersect with the obstacle and if the Euclidean distance between the two positions is lower or equal to the agents' communication radius r_i. Formally, this is modeled by using

$$\boldsymbol{\delta}_{\mathsf{so}}(P,Q) = \begin{cases} \infty & , PQ \cap \mathsf{so} \neq \emptyset \\ \|P - Q\|_2 & , \text{otherwise} \end{cases} \tag{A.2}$$

as distance that takes obstacle so into account. Hence, agents located at P and Q can communicate if $\boldsymbol{\delta}_{\mathsf{so}}(P,Q) \leq r_i$ holds.

Next, we can define the *coverage* of a medium, which basically corresponds to the percentage of the shadowed region than can be reached via the medium.

Definition 30 (Coverage). Let M be a storage medium located at position (x_M, y_M) with communication radius r_M, and let $\mathrm{A}_s^{i,\mathsf{so}}$ denote the shadowed region for agent i with respect to senor obstacle so. Furthermore, let $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$ denote the area that is defined over all positions $P \in \mathrm{A}_s^{i,\mathsf{so}}$ for which $\boldsymbol{\delta}_{\mathsf{so}}(P,(x_M,y_M)) \leq r_M$ holds, i.e. the part of $\mathrm{A}_s^{i,\mathsf{so}}$ where an agent can be placed such that it would be able to interact with the medium at (x_M, y_M).

The coverage $C_M^{i,\mathsf{so}}$ of medium M with respect to obstacle so for agent i then is defined by:

$$C_M^{i,\mathsf{so}} = \begin{cases} \frac{\mathrm{A}_c^{i,\mathsf{so}}(x_M,y_M)}{\mathrm{A}_s^{i,\mathsf{so}}}, & \text{if } \boldsymbol{\delta}_{\mathsf{so}}(\mathsf{pos}(i),(x_M,y_M)) \leq r_M \\ 0, & \text{otherwise} \end{cases} \tag{A.3}$$

where $\boldsymbol{\delta}_{\mathsf{so}}$ denotes the distance function (A.2).

According to this definition, a positive non-zero coverage implies that agent i is able to indirectly communicate through medium M with any other agent i' located in $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$.

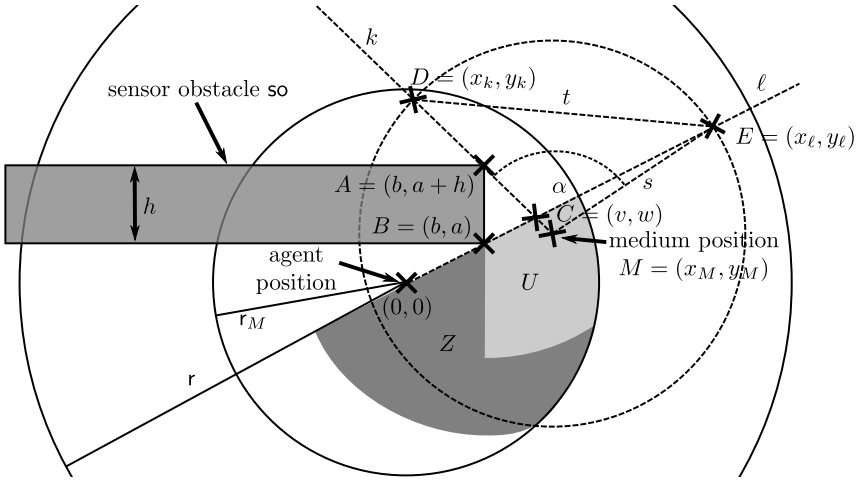


Fig. A.2 Setting considered in this section.

Due to the geometric properties of the medium placement problem, calculating the covered area $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$ involves several cases. In the remainder of this section, we will thus restrict our investigations to a special case that is required for proving the optimal position of a medium that ensures maximum coverage (Theorem 10). All other cases can easily be constructed.

In the following paragraphs, we present a formula that can be applied if the medium is placed within the light gray region U as visualized in Fig. A.2. Formally, U can be defined over all positions (x_M, y_M) of a medium that satisfy the following requirements:

1. $x_M^2 + y_M^2 \leq r_m^2$, to ensure communication with agent i.
2. $(x_M - b)^2 + (y_M - a - h)^2 \leq r_m^2$, to ensure that the triangle ΔABC is part of the calculation, and thus $v > b$ follows for point $C = (v, w)$.
3. $y_M \leq \frac{a}{b} x_M$, to ensure that (x_M, y_M) is below the visibility line ℓ.
4. $x_M > b$, i.e. the medium is placed on the right hand side of the right border of the obstacle.

In the following, let $|PQ| = \|P - Q\|_2$ be the Euclidean distance between two positions P and Q. Consider Fig. A.2 and let $C(x_M, y_M)$ be a function that calculates point $C = (v, w)$, $D(x_M, y_M)$ a function for point $D = (x_k, y_k)$, and $E(x_M, y_M)$ a function for point $E = (x_\ell, y_\ell)$. For a better readability, we will skip the arguments and simply write C, D, or E if we refer to the corresponding functions. We overload M to denote not just the medium but also the medium's position with $M = (x_M, y_M)$. We will use z and n to refer to the help functions $z(x_M, y_M)$ and $n(x_M, y_M)$ defined below in (A.5) and (A.7).

Then, we can deduce from Fig. A.2 that the covered area of a medium placed at $(x_M, y_M) \in U$ under the assumptions above is determined by a combination of three areas. This results in the following formula for $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$:

$$\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M) = \mathrm{A}_{\mathrm{sector}}(x_M, y_M) - \mathrm{A}_{\Delta CEM}(x_M, y_M) + \mathrm{A}_{\Delta ABC}(x_M, y_M) \tag{A.4}$$

where $\mathrm{A}_{\mathrm{sector}}$ is the area of the sector defined by the position of the medium, radius r_M, and angle α. The involved areas can be computed using the following set of equations:

$$z(x_M, y_M) = \frac{1}{2}(h + |AC| + |BC|) \tag{A.5}$$

$$\mathrm{A}_{\Delta ABC}(x_M, y_M) = \sqrt{z(z-h)(z-|AC|)(z-|BC|)} \tag{A.6}$$

$$n(x_M, y_M) = \frac{1}{2}(r + |CE| + |CM|) \tag{A.7}$$

$$\mathrm{A}_{\Delta CEM}(x_M, y_M) = \sqrt{n(n-r)(n-|CE|)(n-|CM|)} \tag{A.8}$$

$$\mathrm{A}_{\mathrm{sector}}(x_M, y_M) = \frac{r_M^2 \cdot \pi \cdot \alpha}{360^\circ} \tag{A.9}$$

In these formulas, we use the elements shown in Fig. A.2, Heron's formula, and further basic geometric equations, which are all based on the agent's position $(0, 0)$, the medium interaction radius r_M, and properties of the obstacle.

Figure A.3 illustrates the area coverage according to (A.4) for varying medium positions (x_M, y_M) in an example with $a = 1, b = 2, h = 2, r_M = 5$. In this example, the maximum coverage $C_{\max}$ is obtained if and only if the medium is placed at position $M = (\sqrt{20}, \sqrt{5})$.

A.2 Optimal Coverage

After we determined a way to calculate the coverage of a medium, the question about the optimal coverage achievable by a single medium will be the focus in this section. In particular, we search for the position of the medium that maximizes the coverage.

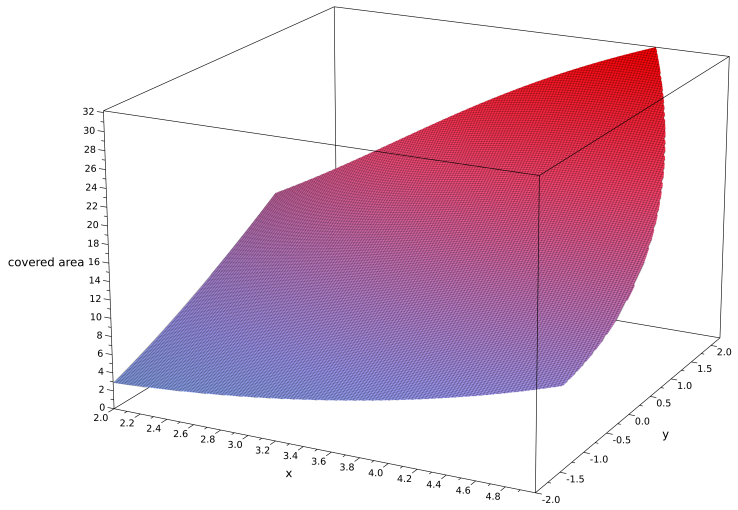


Fig. A.3 Covered area for different medium positions in the given exemplary settings.

Again, consider Fig. A.2 which contains all elements used in this section. The line ℓ is defined by $y = \frac{a}{b}x$ and the medium's communication circle centered at the agent's position $\mathsf{pos}(i) = (0, 0)$ is defined by $(x-0)^2 + (y-0)^2 = r_M^2$. Note that this circle denotes the general area where medium and agent could interact without obstacle. Accordingly, an interesting intersection point of ℓ and the medium interaction circle is given by $(x_{\text{opt.}}, y_{\text{opt.}})$, having

$$x_{\text{opt.}} = \sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}}$$

and

$$y_{\text{opt.}} = \frac{a}{b}\sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}}.$$

In the remainder of this section, we prove that the coverage of a medium is maximized if it is placed at $(x_{\text{opt.}}, y_{\text{opt.}})$. Therefore, we concentrate on all possible positions in region U as defined before. The basic steps of the proof are:

1. Show that the lowest gradient of line s and the maximum negative gradient of line k are achieved if $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$ (cf. Lemma 11 and 12).
2. Accordingly, angle α, which is defined by those two lines, is maximal for $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$ (see Lemma 13).
3. Prove that the area of triangle ΔABC is maximal for $C(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$ (see Lemma 14).
4. Show that the coverage $C_M^{i,\mathsf{so}}$ is maximal for $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$ (see Lemma 15)

We continue with stating and proving the aforementioned lemmas.

Lemma 11. *Let $s(x_M, y_M)$ be a line defined by the positions $(x_M, y_M) \in U$ and (x_l, y_l) as shown in Fig. A.2. Under the mentioned assumptions, the gradient of s is cannot be smaller than the gradient of ℓ, i.e. $g_s(x_M, y_M) \geq g_\ell = \frac{a}{b}$.*

Proof. To proof this lemma, we transform the inequality as follows:

$$\begin{aligned} g_s(x_M, y_M) &\geq g_\ell \\ \frac{y_l - y_M}{x_l - x_M} &\geq \frac{a}{b} \\ y_l - y_M &\geq \frac{a}{b}(x_l - x_M) \\ y_l &\geq \frac{a}{b}(x_l - x_M) + y_M. \end{aligned}$$

By substitution of $y_l = \frac{a}{b}x_l$, we then obtain:

$$\begin{aligned} \frac{a}{b}x_l &\geq \frac{a}{b}(x_l - x_M) + y_M \\ \frac{a}{b}x_l &\geq \frac{a}{b}x_l - \frac{a}{b}x_M + y_M \\ 0 &\geq -\frac{a}{b}x_M + y_M \\ \frac{a}{b}x_M &\geq y_M. \end{aligned}$$

Since $y_M \leq \frac{a}{b}x_M$ holds as a precondition, $g_s(x_M, y_M) \geq g_\ell$ follows. Thus, the gradient of $s(x_M, y_M)$ is greater than or equal to the gradient of ℓ. □

Since $(x_l, y_l) \in \ell$ and $(x_l, y_l) \in s(x_M, y_M)$, an immediate conclusion from Lemma 11 is:

Corollary 5. *The gradient for all lines $s(x_M, y_M)$ defined over $(x_M, y_M) \in U$ and $(x_l, y_l) \in \ell$ is minimal if and only if $y_M = \frac{a}{b}x_M$.*

Lemma 12. *Let line $k(x_M, y_M)$ be defined by the positions $(b, a+h)$ and $(x_M, y_M) \in U$ as shown in Fig. A.2. The gradient $g_k(x_M, y_M) = \frac{y_M - a - h}{x_M - b}$ of line $k(x_M, y_M)$ is maximal for $x_M = x_{\text{opt.}}$ and $y_M = y_{\text{opt.}}$.*

Proof. Based on the two positions $(b, a + h)$ and (x_M, y_M), we can describe line $k(x_M, y_M)$ by

$$y = \frac{y_M - a - h}{x_M - b}x + y_M - \frac{y_M - a - h}{x_M - b}x_M$$

Let (v, w) be the intersection of $k(x_M, y_M)$ and ℓ, using $y = \frac{a}{b}x$ for line ℓ. In this proof, it is then sufficient to investigate line $k'(v)$ defined by $(b, a + h)$ and $(v, w) = (v, \frac{a}{b}v)$ as representative for all possible lines $k(x_M, y_M)$.

In brief, the idea of the proof is as follows. First we show that the gradient function $g_{k'}(v)$ of line $k'(v)$ is strictly monotonically increasing. From this monotonicity it then follows that the maximal gradient can be found at the upper end of v's domain.

Since $w = \frac{a}{b}v$, the gradient $g_{k'}(v)$ of line $k'(v)$ is given by

$$g_{k'}(v) = \frac{\frac{a}{b}v - a - h}{v - b},$$

having $v \in (b, x_{\text{opt.}}]$ according to our assumptions, i.e. since $(x_M, y_M) \in U$. To proof that $g_{k'}(v)$ is strictly monotonically increasing, we have to show that $g_{k'}(v_1) < g_{k'}(v_2)$ for $v_1 < v_2$:

$$
\begin{aligned}
g_{k'}(v_1) &< g_{k'}(v_2) \\
\frac{\frac{a}{b}v_1 - a - h}{v_1 - b} &< \frac{\frac{a}{b}v_2 - a - h}{v_2 - b} \\
\left(\frac{a}{b}v_1 - a - h\right)(v_2 - b) &< \left(\frac{a}{b}v_2 - a - h\right)(v_1 - b) \\
\frac{a}{b}v_1v_2 - av_2 - hv_2 - \frac{a}{b}v_1b + ab + bh &< \frac{a}{b}v_1v_2 - av_1 - hv_1 - \frac{a}{b}v_2b + ab + bh \\
-hv_2 &< -hv_1 \\
v_2 &> v_1
\end{aligned}
$$

Since $v_1 < v_2$ holds as a precondition, the gradient function $g_{k'}(v)$ is strongly monotonically increasing. Accordingly, the maximal gradient is found at the end of v's domain. Hence, line $k^\star$ defined over $(b, a+h)$ and $(x_{\text{opt.}}, y_{\text{opt.}})$ has the largest possible gradient. Note that the gradient is negative, as $v > b$ holds by precondition.

As line $k(x_M, y_M)$ only intersects ℓ at $(x_{\text{opt.}}, y_{\text{opt.}})$ if $(v, w) = (x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$, the lemma follows. □

Lemma 13. *If the medium is positioned at $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}}) \in U$, then angle α between lines $k(x_M, y_M)$ and $s(x_M, y_M)$ as shown in Fig. A.2 is maximal*

Proof. We prove this statement by contradiction. Let us assume that α is not maximized under the given preconditions at position $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$. As the angle is defined by the two lines intersecting at (x_M, y_M), α depends on the gradients of both lines. In Lemma 12 we showed that the largest negative gradient $g_k(x_M, y_M)$ of line $k(x_M, y_M)$ is obtained if $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$. From Corollary 5 we know that $g_s(x_M, y_M)$ is minimal if $y_M = \frac{a}{b}x_M$ and, thus, especially also for $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$. As $(x_{\text{opt.}}, y_{\text{opt.}})$ maximizes $g_k(x_M, y_M)$ and minimizes $g_s(x_M, y_M)$, no larger angle α is possible. This contradicts the assumption, the lemma follows. □

Lemma 14. *The area of triangle ΔABC defined by $A = (b, a), B = (b, a+h), C = (v, w)$ is maximized for $(v, w) = (x_{\text{opt.}}, y_{\text{opt.}})$.*

Proof. It is well known that the area of a triangle given a base Δb and a height Δh can be computed by $\frac{\Delta b \Delta h}{2}$. For triangle ΔABC, as visualized in Fig. A.2, let AB be a base with length $|AB| = h$. The corresponding height is $v - b$. Hence, the area $\mathrm{A}_{\Delta ABC}(v)$ of triangle ΔABC depending on v can be computed by

$$\mathrm{A}_{\Delta ABC}(v) = \frac{h \cdot (v - b)}{2}.$$

Obviously, for any $v_1 < v_2, v_1, v_2 \in (b, x_{\text{opt.}}]$ it follows that $\mathrm{A}_{\Delta ABC}(v_1) < \mathrm{A}_{\Delta ABC}(v_2)$. Accordingly, $\mathrm{A}_{\Delta ABC}(v)$ is strongly monotonically increasing and thus the triangle's area is maximal for $v = x_{\text{opt.}}$. Since $w = \frac{a}{b}v$ the lemma follows. □

Finally, we show that the coverage of a medium is maximal at position $(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}})$ among all possible positions $(x_M, y_M) \in U$:

Lemma 15. *Under the mentioned assumptions, and given a specific agent i and a specific sensor obstacle* so*, we can use (A.4) to calculate the coverage $C_M^{i,\mathsf{so}}$ for any medium M placed at a position $(x_M, y_M) \in U$. The maximum coverage of M occurs for*

$$(x_M, y_M) = (x_{\text{opt.}}, y_{\text{opt.}}) = \left(\sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}}, \frac{a}{b} \sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}} \right),$$

where r_M is the medium's communication radius and (b, a) the bottom right edge position of obstacle so*.*

Proof. For all considered positions $(x_M, y_M) \in U$, the coverage according to Def. 30 is given by

$$C_M^{i,\mathsf{so}} = \frac{\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)}{\mathrm{A}_s^{i,\mathsf{so}}}.$$

Since the area $\mathrm{A}_s^{i,\mathsf{so}}$ shadowed by the obstacle is fixed for a given agent and a given sensor obstacle, we only have to investigate $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$. Recall that the latter is calculated by

$$\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M) = \mathrm{A}_{\text{sector}}(x_M, y_M) - \mathrm{A}_{\Delta CEM}(x_M, y_M) + \mathrm{A}_{\Delta ABC}(x_M, y_M).$$

To maximize the covered area, we thus have to maximize $\mathrm{A}_{\text{sector}}(x_M, y_M)$ and $\mathrm{A}_{\Delta ABC}(x_M, y_M)$, and at the same time minimize $\mathrm{A}_{\Delta CEM}(x_M, y_M)$. Since

$$\mathrm{A}_{\text{sector}}(x_M, y_M) = \frac{r_M^2 \cdot \pi \cdot \alpha}{360^\circ},$$

the area of the sector is maximal if α is maximized under the given constraints. Due to Lemma 13, we know that the angle is maximal for $x_M = x_{\text{opt.}}$ and $y_M = y_{\text{opt.}}$. According to Lemma 14, $\mathrm{A}_{\Delta ABC}(x_M, y_M)$ is also maximal for $x_M = x_{\text{opt.}}$ and $y_M = y_{\text{opt.}}$. Finally, it remains to show that $\mathrm{A}_{\Delta CEM}(x_M, y_M)$ is minimal at the same position. Assume $M = (x_{\text{opt.}}, y_{\text{opt.}})$, then $C = M$ and therefore $\mathrm{A}_{\Delta CEM}(x_M, y_M) = 0$. The lemma follows. □

With the help of Lemma 15, we proved the position in region U that leads to the maximum coverage. To show that no other position outside of U can lead to a higher coverage, we again have to take a closer look at Fig. A.2. Obviously, except for region Z, any other position will result in a coverage of zero. Hence, it is left to show that the coverage for any position in region Z is smaller than the maximum found above. We do not intend to provide the required proof in this work, as it immediate follows from the tedious additional geometric cases that have to be considered for determining $\mathrm{A}_c^{i,\mathsf{so}}(x_M, y_M)$. Finally, we can conclude the following theorem:

Theorem 10. *Given an agent at position $(0, 0)$ and a sensor obstacle* so *with its bottom right position at (b, a). Under the assumptions mentioned before, a medium M with communication radius r_M achieves the maximum possible coverage if it is placed at position*

$$\left(\sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}}, \frac{a}{b} \sqrt{\frac{r_M^2}{1 + \left(\frac{a}{b}\right)^2}} \right)$$

Appendix B

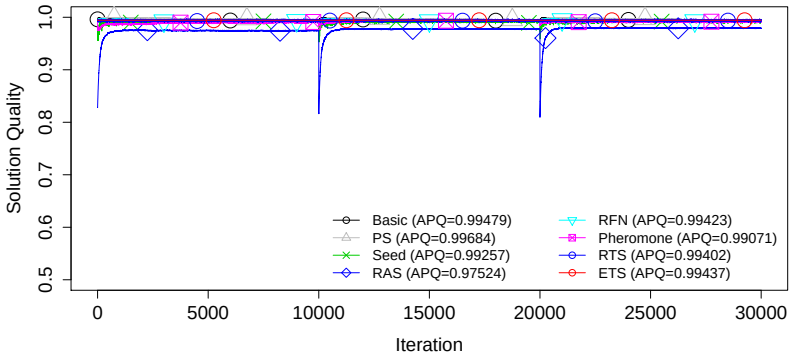
Simulation Results

This appendix presents some additional simulation details on the performance of the different reward calculation concepts investigated in Sect. 10.4. In particular, we present the respective DISTRIBUTION and DISTANCE objective values.

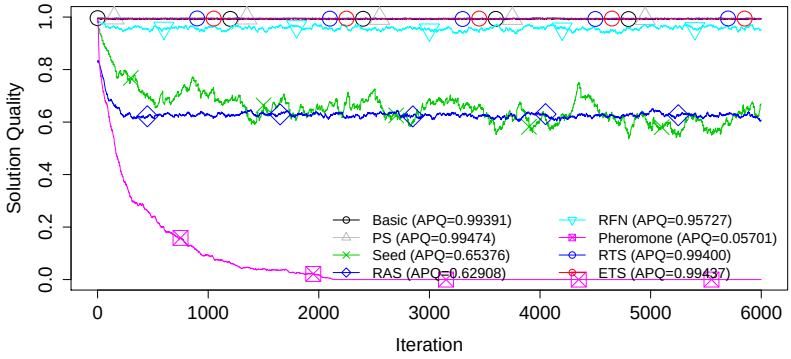
The rest of this page is intentionally left blank.

B.1 Simulation Details for Global Reward Settings

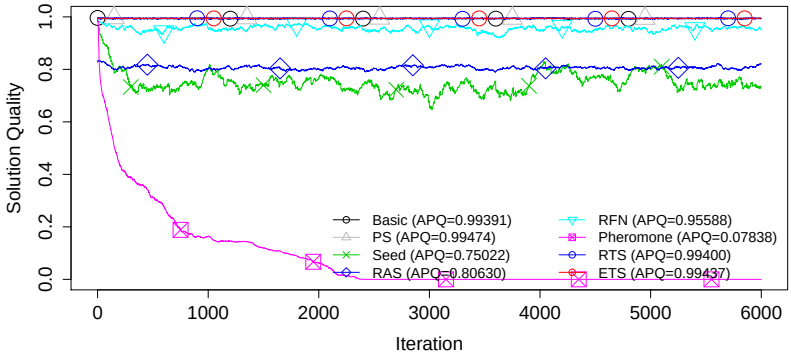
This section presents the development of the DISTRIBUTION and the DISTANCE objective values that belong to Fig. 10.19 on page 162.



(a) Job generation setting $\mathrm{R}(10000)$.

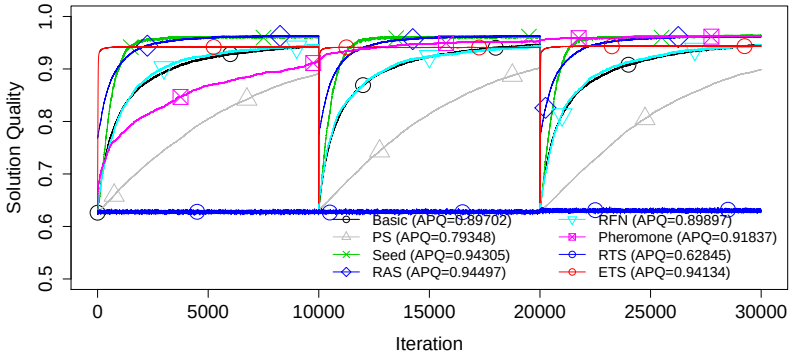


(b) Job generation setting $\mathrm{R}(5, 100)$.

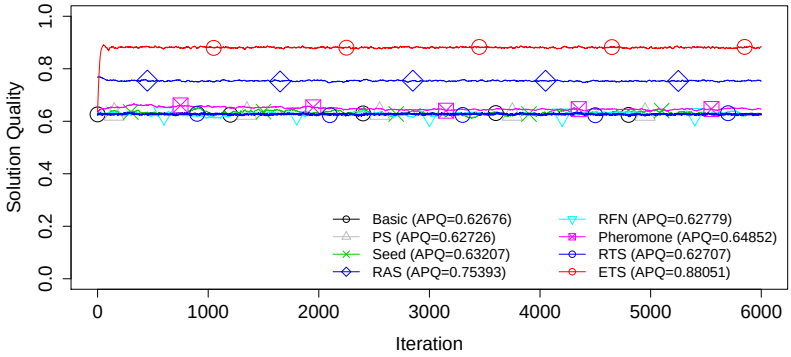


(c) Job generation setting $\mathrm{C}(5, 100)$.

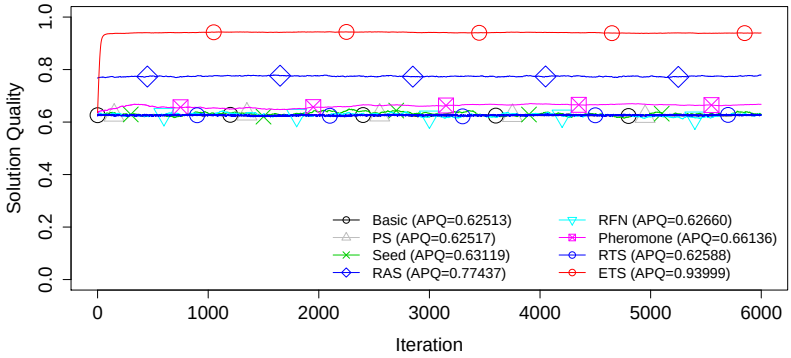
Fig. B.1 Development of the DISTRIBUTION objective value for the different strategies given globally transformed rewards.



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.

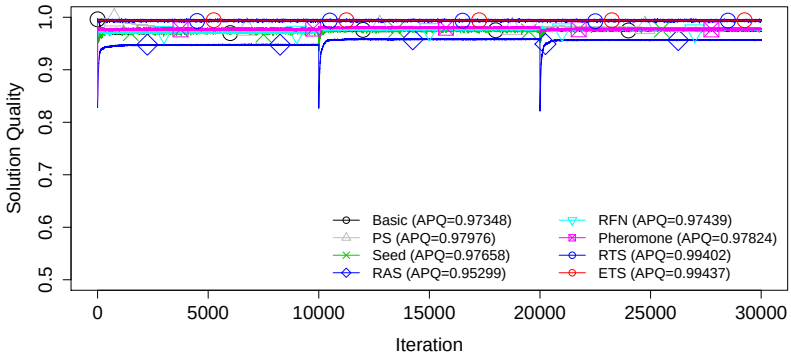


(c) Job generation setting $\mathrm{C}(5, 100)$.

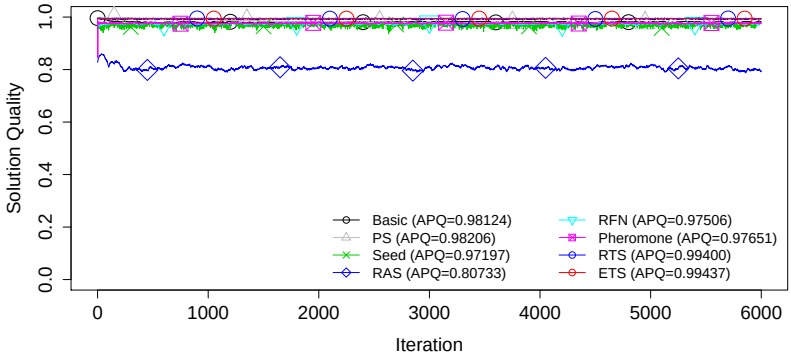
Fig. B.2 Development of the DISTANCE objective value for the different strategies given globally transformed rewards.

B.2 Simulation Details for Local Reward Settings

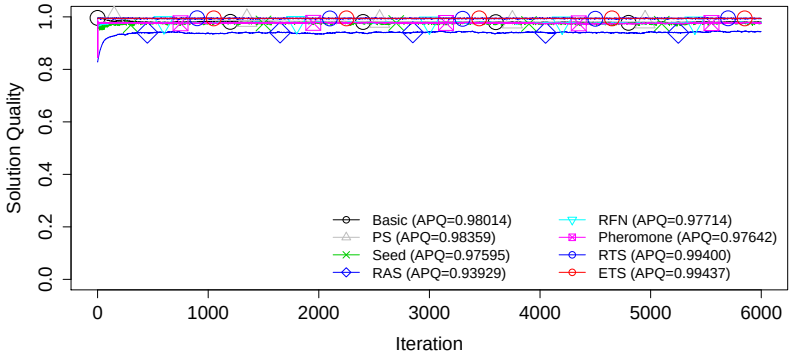
This section presents the development of the DISTRIBUTION and the DISTANCE objective values that belong to Fig. 10.21 on page 165.



(a) Job generation setting $\mathrm{R}(10000)$.

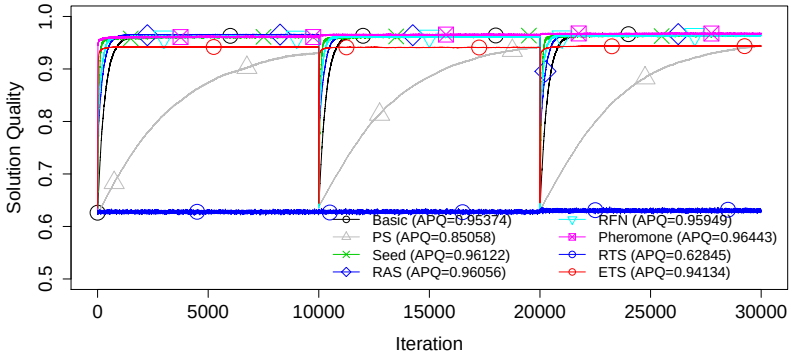


(b) Job generation setting $\mathrm{R}(5, 100)$.

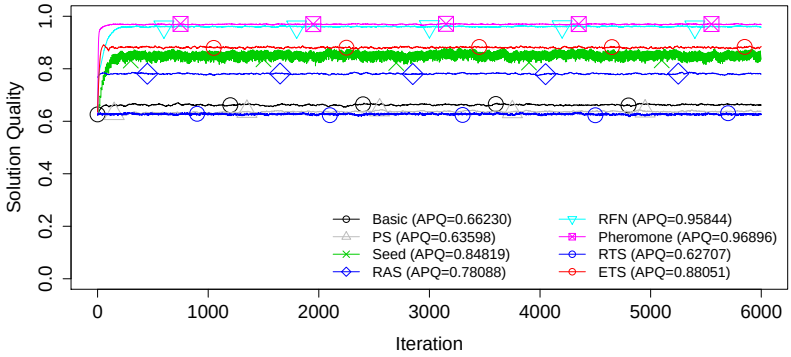


(c) Job generation setting $\mathrm{C}(5, 100)$.

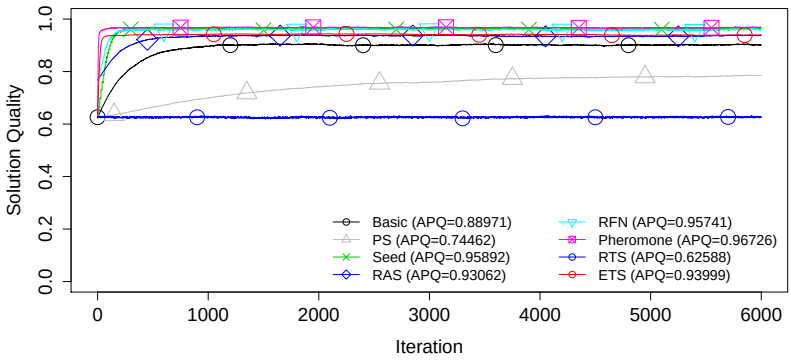
Fig. B.3 Development of the DISTRIBUTION objective value for the different strategies given local rewards.



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.

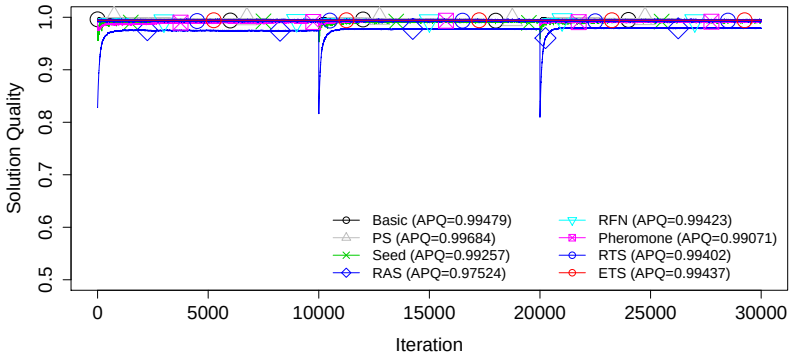


(c) Job generation setting $\mathrm{C}(5, 100)$.

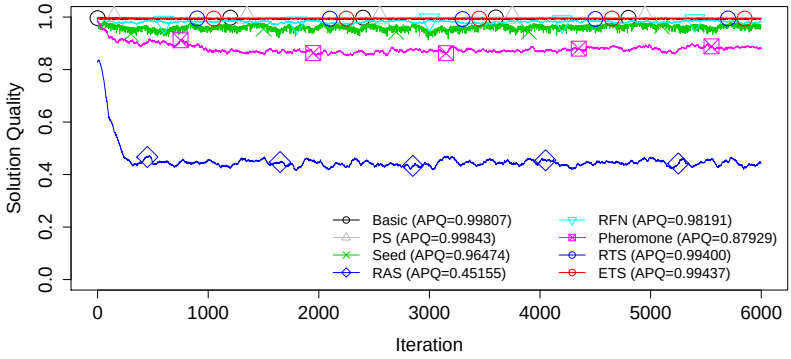
Fig. B.4 Development of the DISTANCE objective value for the different strategies given local rewards.

B.3 Simulation Details for Locally Transformed Global Rewards

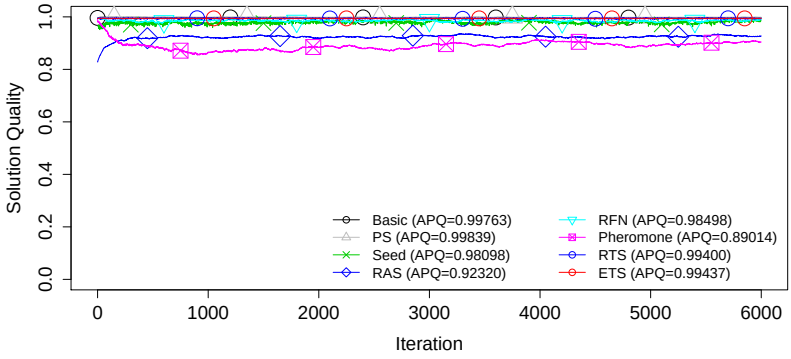
This section presents the development of the DISTRIBUTION and the DISTANCE objective values that belong to Fig. 10.22 on page 167.



(a) Job generation setting $\mathrm{R}(10000)$.

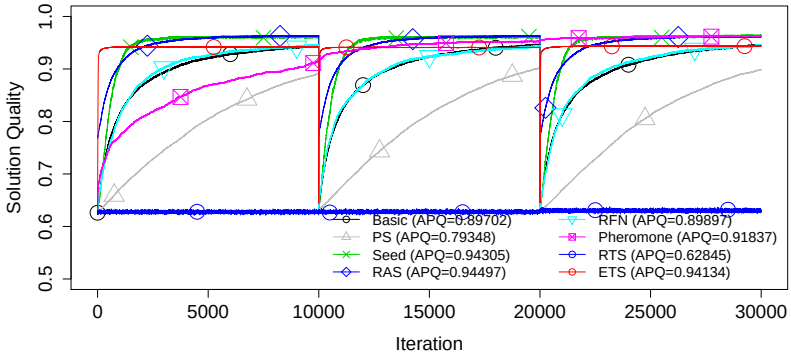


(b) Job generation setting $\mathrm{R}(5, 100)$.

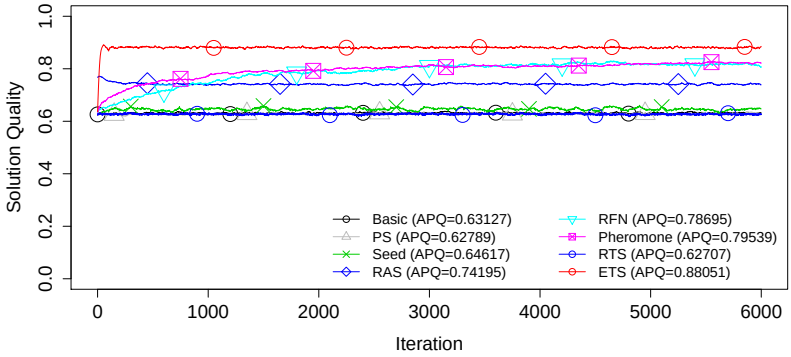


(c) Job generation setting $\mathrm{C}(5, 100)$.

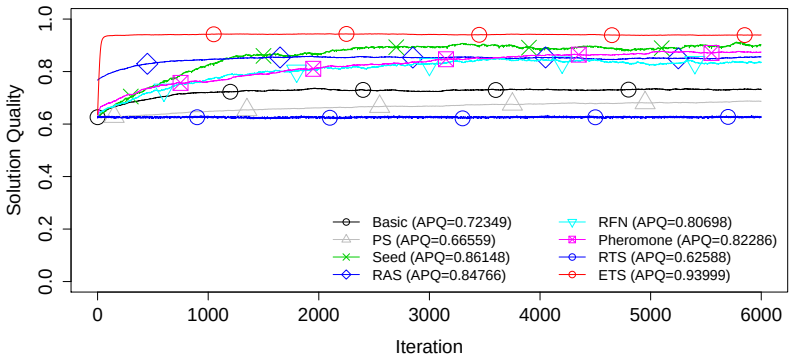
Fig. B.5 Development of the DISTRIBUTION objective value for the different strategies given locally transformed global rewards.



(a) Job generation setting $\mathrm{R}(10000)$.



(b) Job generation setting $\mathrm{R}(5, 100)$.



(c) Job generation setting $\mathrm{C}(5, 100)$.

Fig. B.6 Development of the DISTANCE objective value for the different strategies given locally transformed global rewards.

List of Notations

α	Learning rate	31
a	Simple action	23
A	Set of simple actions	26
$\mathcal{A}$	The set of agents	10
A_i	Set of actions of agent i	34
C	Abbreviation for the circle job generation strategy	144
δ	State transition function	26
$\boldsymbol{\delta}$	Distance function	64
d	Duration of a job	62
$d_{\max}$	Maximum duration of a job	62
$d_{\min}$	Minimum duration of a job	62
$\mathcal{E}$	The environment	62
f	Evaluation function for partitionings	62
γ	Discount factor in Q-Learning based approaches	26
Γ	A (multi state) game	34
g	Game transition function for sequential stage games	49
G	Static (stateless) game	34
$\mathcal{G}$	A set of games	49
i	An agent	10
$\mathcal{J}$	Set of jobs	62
k	Total number of iterations	62
k	Number of neighbors of an agent	62
m	Total number of targets, i.e. $m = \lvert\mathcal{T}\rvert$	62
M	A Markov Decision Process	26
M	Storage medium	78
$\mathcal{M}$	Set of storage media	78
n	Total number of agents, i.e. $n = \lvert\mathcal{A}\rvert$	62
$\mathcal{N}$	A neighborhood	62
$\boldsymbol{o}$	Joint observation	40
$\boldsymbol{\Omega}$	Set of joint observations	40
π	A policy	26
$\boldsymbol{p}$	Partitioning function that maps each agent to a target	62
p	A position	62
$q^\star$	Optimal local q-value (usually referring to a particular agent i)	118
$\widehat{Q}$	Estimation of a Q-value	30
$Q^\star$	Optimal Q-value	27
ρ	Reward value or reward function	26
$\widetilde{\rho}$	A noisy reward or noisy reward function	93
$\widetilde{\rho}^{\max}$	Maximal positive noised reward value	94

r_i	Communication radius of an agent i	10
r_M	Communication radius of a storage medium	78
R	Abbreviation for the random job generation strategy	144
R	A return	25
σ	A strategy	34
$\boldsymbol{\sigma}$	Joint strategy	35
Σ	Set of strategies	35
$\boldsymbol{\Sigma}$	Set of joint strategies	109
s	A state	23
$\mathcal{S}$	Set of states	26
$\boldsymbol{S_{A,T}}$	Set of possible partitionings	62
$\mathcal{S}_{\mathcal{A},\mathcal{T}}$	A concrete partitioning described by subsets	62
θ	Noise rate	93
t	Time index	23
t	Transformation function for two static games	106
T	A single target	62
T	Transformation function for cooperative sequential stage games	107
$\mathcal{T}$	Set of targets	62
u	Joint action	38
U	Set of joint actions	34

Abbreviations

ACO	Ant Colony Optimization	17
APQ	Average Partitioning Quality	63
BASIC	Synonym for Distributed Stateless Learning with the ordinary ϵ-greedy action selection procedure	145
CCSG	Cooperative Common Static Games	106
CSG	Common Static Games	49
CSSG	Cooperative Sequential Stage Sames	49
Dec-POMDP	Decentralized Partially Observable Markov Decision Process	40
DISTANCE	The distance objective part of the partitioning quality	64
DISTRIBUTION	The distribution objective part of the partitioning quality	64
DSL	Distributed Stateless Learning	106
DSLCM	Distributed Stateless Learning with Counting Media	179
DQ	Distributed Q-Learning	38
ERBMDP	Engineered Reward-Based Markov Decision Process	52
ETS	Exchange Target Strategy	14
IAPP	Iterative Agent Partitioning Problem	62
IDS	ID-Dependent Target Strategy	148
LTG	Locally Transformed Global Reward	142
MARL	Multiagent Reinforcement Learning	33
MAS	Multiagent System	11
MDP	Markov Decision Process	26
NCSSG	Noisy Cooperative Sequential Stage Same	117
NE	Nash equilibrium	35
PHEROMONE	Pheromone-Based Strategy	130
POMDP	Partially Observable Markov Decision Process	32
POSG	Partially Observable Stochastic Game	51
PS	Previous-Strategy	135
PSO	Particle Swarm Optimization	18
RAS	Reduced-Action-Set Strategy	133
RFN	Random-From-Neighborhood Strategy	135
RL	Reinforcement Learning	23
RTS	Random Target Selection Strategy	144
SEED	Seed-Based-Strategy	134
SG	Stochastic Games	34
SSG	Sequential Stage Games	49
TL	Transfer Learning	18
TTO	Two Target Optimal Algorithm	15

References

[Abe03] D. Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes. Technical report, National ICT Australia, 2003.

[AL08] S. Abdallah and V. R. Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *J. Artif. Intell. Res. (JAIR)*, 33:521–549, 2008.

[ATTW11] H. B. Ammar, M. E. Taylor, K. Tuyls, and G. Weiss. Reinforcement learning transfer using a sparse coded inter-task mapping. In *9th European Workshop on Multiagent Systems (EUMAS'11)*, 2011.

[Aur91] Franz Aurenhammer. Voronoi diagrams — a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[AVAS04] N. Abe, N. K. Verma, C. Apté, and R. Schroko. Cross channel optimized marketing by reinforcement learning. In *Proc. of the 10th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD'04)*, pages 767–772. ACM, 2004.

[BAHZ09] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein. Policy iteration for decentralized control of markov decision processes. *J. Artif. Intell. Res.*, 34:89–132, 2009.

[BBDS08] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.

[BBDS10] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in Multi-Agent Systems and Applications - 1*, volume 310 of *Studies in Computational Intelligence*, pages 183–221. Springer, 2010.

[BdCL08] M. Babes, E. M. de Cote, and M. L. Littman. Social reward shaping in the prisoner's dilemma. In *Proc. of the 7th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 1389–1392. IFAAMAS, 2008.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[BEDSB11] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Approximate reinforcement learning: An overview. In *Proc. of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL'11)*, pages 1–8. IEEE, 2011.

[BGIZ02] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[BHJ+07] P. V. Bahl, M. T. Hajiaghayi, K. Jain, S. V. Mirrokni, L. Qiu, and A. Saberi. Cell breathing in wireless lans: Algorithms and evaluation. *IEEE Trans. Mob. Comput.*, 6(2):164–178, 2007.

[BHZ04] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the 19th National Conf. on Artificial Intelligence (AAAI'04)*, pages 709–715. AAAI / MIT Press, 2004.

[BKB11] M. Baumann and H. Kleine Büning. State aggregation by growing neural gas for reinforcement learning in continuous state spaces. In *Intl. Conf. on Machine Learning and Applications (ICMLA'11)*, pages 430–435. IEEE, 2011.

[Bou99] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proc. of the 16th Intl. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 478–485. Morgan Kaufmann, 1999.

[Bow04] M. H. Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pages 209–216. MIT Press, 2004.

[BPV12] G. Boutsioukis, I. Partalas, and I. Vlahavas. Transfer learning in multi-agent reinforcement learning domains. In *Recent Advances in Reinforcement Learning (EWRL'11), Revised Selected Papers*, volume 7188 of *LNCS*, pages 249–260. Springer, 2012.

[BTS10] S. Barrett, M. E. Taylor, and P. Stone. Transfer learning for reinforcement learning on a physical robot. In *Proc. of the Adaptive Learning Agents Workshop @ AAMAS 2010*, pages 24–29, 2010.

[Buş08] L. Buşoniu. *Reinforcement Learning in Continuous State and Action Spaces*. PhD thesis, Delft University of Technology, 2008.

[BV00] M. Bowling and M. Veloso. An analysis of stochastic game theory for multiagent reinforcement learning. Technical Report CMU-CS-00-165, Carnegie Mellon University, 2000.

[CB98] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. of the 15th National Conf. on Artificial Intelligence (AAAI'98)*, pages 746–752. AAAI, 1998.

[CB03] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: a bayesian approach. In *Proc. of the 2nd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'03)*, pages 709–716. ACM, 2003.

[Cha04] H. S. Chang. An ant system based exploration-exploitation for reinforcement learning. In *Proc. of the IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume 4, pages 3805–3810. IEEE, 2004.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[Cor03] Daniel D. Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Intl. Lisp Conf.*, 2003.

[DFDCG09] F. Ducatelle, A. Förster, G. Di Caro, and L.M. Gambardella. New task allocation methods for robotic swarms. In *9th IEEE/RAS Conf. on Autonomous Robot Systems and Competitions*, 2009.

[DFG99] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[DG97] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolutionary Computation*, 1(1):53–66, 1997.

[DJBS89] R. Dodhiawala, V. Jagannathan, L. Baum, and T. Skillman. The first workshop on blackboard systems. *AI Magazine*, 10(1):77–80, 1989.

[DK11] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proc. of the 10th Intl. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'11)*, pages 225–232. IFAAMAS, 2011.

[Doh03] K. Dohmen. *Improved Bonferroni Inequalities Via Abstract Tubes: Inequalities and Identities of Inclusion-Exclusion Type*, volume 1826 of *Lecture Notes in Mathematics*. Springer, 2003.

[DS04] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT Press, 2004.

[DZKS06] M.B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proc. of the IEEE*, 94(7):1257–1270, 2006.

[DZMB02] M. Dorigo, M. Zlochin, N. Meuleau, and M. Birattari. Updating aco pheromones using stochastic gradient ascent and cross-entropy methods. In *Applications of Evolutionary Computing (EvoWorkshops'02)*, volume 2279 of *LNCS*, pages 21–30. Springer, 2002.

[EDM03] E. Even-Dar and Y. Mansour. Learning rates for q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

[EM88] R. Engelmore and T. Morgan. *Blackboard Systems*. Addison-Wesley, 1988.

[Eng05] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[ENT11] M. Elango, S. Nachiappan, and M. K. Tiwari. Balancing task allocation in multi-robot systems using k-means clustering and auction based mechanisms. *Expert Systems with Applications*, 38(6):6486–6491, 2011.

[Fer99] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[For87] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.

[Fri95] B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7 (NIPS'94)*, pages 625–632. MIT Press, 1995.

[Ger79] A. Gersho. Asymptotically optimal block quantization. *IEEE Trans. on Information Theory*, 25(4):373–380, 1979.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman And Company, 1979.

[GK08] M. Grzes and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Proc. of the 4th IEEE Intl. Conf. on Intelligent Systems (IS'08)*, volume 2, pages 22–29. IEEE, 2008.

[GKBPW05] A. Goebels, H. Kleine Büning, S. Priesterjahn, and A. Weimer. Towards online partitioning of agent sets based on local information. In *Proc. of the IASTED Intl. Conf. on Parallel and Distributed Computing and Networks (PDCN'05)*, pages 674–679. IASTED/ACTA Press, 2005.

[GLP02] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proc. of the 19th Intl. Conf. on Machine Learning (ICML'02)*, pages 227–234. Morgan Kaufmann, 2002.

[GM04] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *Intl. J. of Robotics Research*, 23(9):939–954, 2004.

[GO04] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 2nd edition, 2004.

[Goe07] A. Goebels. *Agent Coordination Mechanisms for Solving a Partitioning Task*. Logos, 2007.

[Gra59] P.-P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–80, 1959.

[Gru04] P. M. Gruber. Optimum quantization and its applications. *Advances in Mathematics*, 186(2):456–497, 2004.

[GW99] C. Gaskett and A. Wettergreen, D.and Zelinsky. Q-learning in continuous state and action spaces. In *Australian Joint Conference on Artificial Intelligence (AUS-AI'99)*, volume 1747 of *LNCS*, pages 417–428. Springer, 1999.

[GWP05] A. Goebels, A. Weimer, and S. Priesterjahn. Using cellular automata with evolutionary learned rules to solve the online partitioning problem. In *Proc. of the IEEE Congress on Evolutionary Computation (CEC'05)*, pages 837–843. IEEE, 2005.

[HBZ04] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the 19th National Conf. on Artifical Intelligence (AAAI'04*, pages 709–715. AAAI Press, 2004.

[HCB00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. of the 33rd Hawaii Intl. Conf. on System Sciences (HICSS'00)*. IEEE, 2000.

[HM99] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artif. Life*, 5(2):173–202, 1999.

[HMLIR07] V. Heidrich-Meisner, M. Lauer, C. Igel, and M. Riedmiller. Reinforcement learning in a nutshell. In *Proc. of 15th European Symposium on Artificial Neural Networks (ESANN'07)*, pages 277–288. d-side publications, 2007.

[HTP+06] P. J. 't Hoen, K. Tuyls, L. Panait, S. Luke, and J. La Poutré. An overview of cooperative and competitive multiagent learning. In *Learning and Adaption in Multi-Agent Systems (LAMAS'05)*, volume 3898 of *LNCS*, pages 1–46. Springer, 2006.

[HW05] C. M. Henein and T. White. Agent-based modelling of forces in crowds. In *Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation (MABS'04)*, volume 3415 of *LNCS*, pages 173–184. Springer, 2005.

[JGKvS04] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proc. of the 5th ACM/IFIP/USENIX Intl. Conf. on Middleware*, volume 3231 of *LNCS*, pages 79–98. Springer, 2004.

[JJS94] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.*, 6(6):1185–1201, 1994.

[JS02] G. Judd and P. Steenkiste. Fixing 802.11 access point selection. *SIGCOMM Comput. Commun. Rev.*, 32(3):31, 2002.

[JVG+07] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3), 2007.

[Kal10] P. Kallien. Evolution regelbasierter Wissensobjekte in Multiagentensystemen. Bachelor's thesis, University of Paderborn, 2010.

[KBHR08] R. Kitio, O. Boissier, J. F. Hübner, and A. Ricci. Organisational artifacts and agents for open multi-agent organisations: "giving the power back to the agents". In *Proc. of the Intl. Workshops on Coordination, Organizations, Institutions, and Norms in Agent Systems III (COIN'07), Revised Selected Papers*, volume 4870 of *LNCS*, pages 171–186. Springer, 2008.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 482–491. IEEE, 2003.

[KE95] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. of IEEE Intl. Conf. on Neural Networks*, pages 1942–1948. IEEE, 1995.

[Kem08] Thomas Kemmerich. Algorithms for the general online partitioning problem. Master's thesis, University of Paderborn, 2008.

[Kem10] T. Kemmerich. Influence of communication graph structures on pheromone-based approaches in the context of a partitioning task problem. In *Proc. of the Intl. Workshops on Coordination, Organizations, Institutions, and Norms in Agent Systems V (COIN @ IJCAI'09), Revised Selected Papers*, volume 6069 of *LNAI*, pages 257–272. Springer, 2010.

[Kem12] T. Kemmerich. Simulation tool. `http://www.cs.upb.de/fileadmin/Informatik/AG-Kleine-Buening/files/Publikationen/kemmerich-thesis-simulation-tool.tgz`, 2012.

[KK02] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proc. of the 18th National Conf. on Artificial Intelligence (AAAI'02)*, pages 326–331. AAAI / MIT Press, 2002.

[KK04] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In *Proc. of the 3rd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'04)*, volume 3, pages 1258–1259. IEEE, 2004.

[KKB10a] T. Kemmerich and H. Kleine Büning. External coordination media in capacity-constrained multiagent systems. In *Proc. of the IEEE/WIC/ACM Intl. Joint Conf. on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, pages 109–116. IEEE, 2010.

[KKB10b] T. Kemmerich and H. Kleine Büning. Region-based heuristics for an iterative partitioning problem in multiagent systems (extended version). Technical Report TR-RI-10-320, University of Paderborn, 2010.

[KKB11a] T. Kemmerich and H. Kleine Büning. A convergent multiagent reinforcement learning approach for a subclass of cooperative stochastic games. In *Proc. of the Adaptive Learning Agents Workshop @ AAMAS 2011*, pages 75–82. IFAAMAS, 2011.

[KKB11b] T. Kemmerich and H. Kleine Büning. Coordination in large multiagent reinforcement learning problems. In *Proc. of the IEEE/WIC/ACM Intl. Joint Conf. on Web Intelligence and Intelligent Agent Technology (WI-IAT'11)*, pages 236–239. IEEE, 2011.

[KKB11c] T. Kemmerich and H. Kleine Büning. On the power of global reward signals in reinforcement learning. In *Proc. of the 9th German Conf. on Multiagent System Technologies (MATES' 11)*, volume 6973 of *LNAI*, pages 53–64. Springer, 2011.

[KKB11d] T. Kemmerich and H. Kleine Büning. Region-based heuristics for an iterative partitioning problem in multiagent systems. In *Proc. of the 3rd Intl. Conf. on Agents and Artificial Intelligence (ICAART'11)*, volume 2, pages 200–205. SciTePress, 2011.

[KKB12] T. Kemmerich and H. Kleine Büning. A convergent multiagent reinforcement learning approach for a subclass of cooperative stochastic games. In *Proc. of the Intl. Workshop on Adaptive and Learning Agents (ALA @ AAMAS'11), Revised Selected Papers*, volume 7113 of *LNAI*, pages 37–53. Springer, 2012.

[KKN06] G. Kasbekar, J. Kuri, and P. Nuggehalli. Online association policies in IEEE 802.11 WLANs. In *Proc. of the 4th Intl. Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt'06)*, pages 11–20. IEEE, 2006.

[KLC98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *J. Artif. Intell.*, 101(1-2):99–134, 1998.

[Kli10] D. Klippenstein. Lokales Abschätzen globaler Systemzustände in Multi-Agenten Systemen. Bachelor's thesis, University of Paderborn, 2010.

[KLM96] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285, 1996.

[KS04] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA'04)*, pages 2619–2624. IEEE, 2004.

[KvS07] A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7, 2007.

[LBEK11] T. Lettmann, M. Baumann, M. Eberling, and T. Kemmerich. Modeling agents and agent systems. In *Trans. on Computational Collective Intelligence V*, volume 6910 of *LNCS*, pages 157–181. Springer, 2011.

[LH06] N. Li and J. C. Hou. A scalable, power-efficient broadcast algorithm for wireless networks. *Wireless Networks*, 12(4):495–509, 2006.

[Lit01] M. L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[Lov91] W. Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[LR00] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proc. of the 17th Intl. Conf. on Machine Learning (ICML'00)*, pages 535–542. Morgan Kaufmann, 2000.

[LR04] M. Lauer and M. Riedmiller. Reinforcement learning for stochastic cooperative multi-agent systems. In *Proc. of the 3rd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'04)*, volume 3, pages 1516–1517. IEEE, 2004.

[LWL+09] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang. On centroidal voronoi tessellation — energy smoothness and fast computation. *ACM Trans. on Graphics*, 28(4), 2009.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[MLFP08] L. Matignon, G. J. Laurent, and Le Fort-Piat. A study of fmq heuristic in cooperative multi-agent games. In *Proc. of the Workshop on Multi-Agent Sequential Decision Making in Uncertain Multi-Agent Domains (MSDM @ AAMAS'08)*, pages 77–91, 2008.

[Mon82] G. E. Monahan. A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Nas51] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[NCC+06] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall. Improved access point selection. In *Proc. 4th Intl. Conf. on Mobile systems, applications and services (MobiSys'06)*, pages 233–245. ACM, 2006.

[NCD+04] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Proc. of the 9th Intl. Symposium on Experimental Robotics (ISER'04)*, pages 363–372. Springer, 2004.

[NHR99] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. of the 16th Intl. Conf. on Machine Learning (ICML'99)*, pages 278–287. Morgan Kaufmann, 1999.

[OAVBFR09] E. M. Ould-Ahmed-Vall, D. M. Blough, B. H. Ferri, and G. F. Riley. Distributed global id assignment for wireless sensor networks. *Ad Hoc Networks*, 7(6):1194–1216, 2009.

[ORV+04] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proc. of the 3rd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'04)*, volume 1, pages 286–293. IEEE, 2004.

[ORV08] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[OW12] M. Otterlo and M. Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 3–42. Springer, 2012.

[PHDL07] X. Pan, C. S. Han, K. Dauber, and K. H. Law. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *AI Soc.*, 22(2):113–132, 2007.

[PL05] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[PR99] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100. IEEE, 1999.

[PR09] M. Piunti and A. Ricci. Cognitive use of artifacts: Exploiting relevant information residing in MAS environments. In *Proc. of the 1st First Intl. Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS'08), Revised Selected Papers*, volume 5605 of *LNCS*, pages 114–129. Springer, 2009.

[PT87] C. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, 1987.

[PT09] S. Proper and P. Tadepalli. Multiagent transfer learning via assignment-based decomposition. In *Proc. of the Intl. Conf. on Machine Learning and Applications (ICMLA'09)*, pages 345–350. IEEE, 2009.

[PTT10] M. Ponsen, M. E. Taylor, and K. Tuyls. Abstraction and generalization in reinforcement learning: A summary and framework. In *Proc. of the Intl. Workshop on Adaptive and Learning Agents (ALA @ AAMAS'09), Revised Selected Papers*, volume 5924 of *LNCS*, pages 1–32. Springer, 2010.

[RA98] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proc. of the 15th Intl. Conf. on Machine Learning (ICML'98)*, pages 463–471. Morgan Kaufmann, 1998.

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[RPV11] A. Ricci, M. Piunti, and M. Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[SA11] T. C. Service and J. A. Adams. Coalition formation for task allocation: theory and algorithms. *Autonomous Agents and Multi-Agent Systems*, 22(2):225–248, 2011.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[See73] Robert T. Seeley. *Calculus of One & Several Variables*. Scott, Foresman and Company, 2nd edition, 1973.

[Sha53] L. S. Shapley. Stochastic games. *Proc. of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[SK98] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artif. Intell.*, 101(1-2):165–200, 1998.

[SLB09] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[SMY+04] N. Sato, F. Matsuno, T. Yamasaki, T. Kamegawa, N. Shiroma, and H. Igarashi. Cooperative task execution by a multiple robot team and its operators in search and rescue operations. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS'04)*, volume 2, pages 1083–1088. IEEE, 2004.

[Spa12] M. T. J. Spaan. Partially observable markov decision processes. In *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 387–414. Springer, 2012.

[SPG07] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[Swa10] Daniel Swars. Reinforcement Learning in kooperativen Multiagentensystemen. Master's thesis, University of Paderborn, 2010.

[SZ08] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[Sze10] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2010.

[Tes95] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[TS09] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[TS10] L. Torrey and J. W. Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[Tsi94] J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, 1994.

[TSWM10] L. Torrey, J. W. Shavlik, T. Walker, and R. Maclin. Transfer learning via advice taking. In *Advances in Machine Learning I*, volume 262 of *Studies in Computational Intelligence*, pages 147–170. Springer, 2010.

[TW11] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks*. Prentice Hall, 5th edition, 2011.

[VDHN11] P. Vrancx, Y.-M. De Hauwere, and A. Nowé. Transfer learning for multi-agent coordination. In *Proc. 3rd Intl. Conf. on Agents and Artificial Intelligence (ICAART'11)*, volume 2, pages 263–272. SciTePress, 2011.

[vH12] H. van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 207–251. Springer, 2012.

[VKT+09] P. Varakantham, J.-Y. Kwak, M. E. Taylor, J. Marecki, P. Scerri, and M. Tambe. Exploiting coordination locales in distributed pomdps via social model shaping. In *Proc. of the 19th Intl. Conf. on Automated Planning and Scheduling (ICAPS'09)*. AAAI, 2009.

[Vla07] N. Vlassis. *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2007.

[Vos91] M. D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. on Software Engineering*, 17(9):972–975, 1991.

[Wat89] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

[WD92] C. J. C. H. Watkins and P. Dayan. Technical note q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[WdS06] Y. Wang and C. W. de Silva. Multi-robot box-pushing: Single-agent q-learning vs. team q-learning. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS'06)*, pages 3694–3699. IEEE, 2006.

[Weh08] B. Wehe. Verfahren zur Partitionierung von Multi-Agentensystemen in Szenarien mit Hindernissen. Master's thesis, University of Paderborn, 2008.

[Wei99] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Wie03] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *J. Artif. Intell. Res.*, 19(1):205–208, 2003.

[Woo09] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2nd edition, 2009.

[WS03] X. Wang and T. Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Advances in Neural Information Processing Systems 15 (NIPS'02)*, pages 1571–1578. MIT Press, 2003.

[YG04] E. Yang and D. Gu. Multiagent reinforcement learning for multi-robot systems: A survey. Technical Report CSM–404, University of Essex, 2004.

[YT07] D. Yagan and C.-K. Tham. Coordinated reinforcement learning for decentralized optimal control. In *Proc. of IEEE Intl. Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL'07)*, pages 296–302. IEEE, 2007.

[Zim80] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *IEEE Trans. on Communications*, 28(4):425–432, 1980.

[ZZL09] X. Zheng, T. Zhong, and M. Liu. Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment*, 44(3):437–445, 2009.