Ant Colony Optimization for Control

Jelmer van Ast

Cover illustration: Alex Wild
Cover design: Kim van Wijk & Jelmer van Ast

Ant Colony Optimization for Control

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 14 september 2010 om 15:00 uur
door

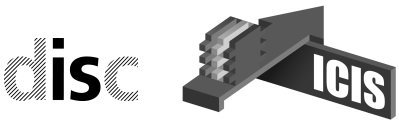
Jelmer Marinus VAN AST

elektrotechnisch ingenieur,
geboren te Oud-Beijerland.

Dit proefschrift is goedgekeurd door de promotoren:
Prof.dr. R. Babuška, M.Sc.
Prof.dr.ir. B. De Schutter

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. R. Babuška, M.Sc.	Technische Universiteit Delft, promotor
Prof.dr.ir. B. De Schutter	Technische Universiteit Delft, promotor
Prof.dr.ir. F.C.A. Groen	Universiteit van Amsterdam
Prof.dr.drs. L.J.M. Rothkrantz	Nederlandse Defensie Academie & Technische Universiteit Delft
Prof.dr.ir. H. Bersini	Université Libre de Bruxelles
Prof.dr.ir. A. van Keulen	Technische Universiteit Delft
Prof.dr.ir. G. van Straten	Wageningen University
Prof.dr.ir. J. Hellendoorn	Technische Universiteit Delft (reservelid)



This thesis has been completed in partial fulfillment of the requirements of the Dutch Institute for Systems and Control (DISC) for graduate studies. The research described in this thesis was financially supported by Senter, Ministry of Economic Affairs of the Netherlands within the BSIK-ICIS project "Self-Organizing Moving Agents" (grant no. BSIK03024).

Published and distributed by: Jelmer van Ast
E-mail: jelmer@vanast.info
Web: http://www.vanast.info

ISBN 978-90-9025609-2

Printed in the Netherlands

Acknowledgments

During the last four to five years, I have enjoyed advising M.Sc. students, assisting in lecturing courses, but first and foremost doing research with a great deal of independence. There are quite a few people to whom I owe a great deal of gratitude.

I would first and foremost like to thank Robert and Bart for their tireless support and guidance. Robert, it has been an honor and a pleasure working with you. Since the first lecture of yours that I attended in my third year, I knew that you would be the ideal mentor for me. Your enthusiasm, constructive criticism, and willingness to make time for me whenever I needed it, has been a great inspiration to me. You have been an example to me and helped define the way I try to interact with other people. Thank you for the trust you put in me when assisting you in your lectures. Bart, thank you so much for your very detailed feedback on almost anything I wrote. It is because of you that I have always tried to do the very same with the students I advised and papers I reviewed. Your great mathematical insight has helped me a lot in the theoretical part of my work and you have been a very kind and supportive listener during the tough moments of the last years.

Mom and Dad, thanks for your unconditional support and belief in me. Your constant faith that I will do well has backed me up a lot. Manja, being with you is always great fun. You have constantly supported me and I could not have wished for a nicer sister. Duncan, my dear friend, thank you for keeping me sane and creative. Discussing theatrical plots and writing plays has always been joyful and stress-relieving. Mathieu, thank you for the inspirational discussions we so frequently have. I look forward to working towards our shared vision.

I would like to thank my many colleagues, with whom it was a great pleasure to chat and have fun during lunch, coffee breaks, and after-work hours. I especially like to thank Diederick for the many moments we shared our thoughts and experiences, Kamana for the great meals we had together, Justin for the many insights you gave me on the most widespread issues, and Eric for the great fun it is with you around. Lucian and Zsófia, it was nice working in the same project. You always made time for me and helped me a lot with your experience.

I am grateful to my colleagues at the Interactive Collaborative Information Systems project who provided me with good feedback on my work. I also thank my committee members, Frans Groen, Léon Rothkrantz, Hugues Bersini, Fred van Keulen, Gerrit van Straten, Hans Hellendoorn, and of course Robert and Bart for the valuable feedback on my thesis and the good discussions we had about it.

To all my friends, family, and colleagues that I did not explicitly mention, thank you for the many ways in which I enjoyed your company and your support.

Jelmer van Ast
July 2010, Delft

Contents

Chapter 1

Introduction

This chapter introduces swarm intelligence, ant colony optimization, and the subject of this thesis: ant colony optimization for control. This thesis in particular focuses on control policy learning and the main achievement is the development of the ant colony learning algorithm. This chapter describes the contributions and also presents the outline of this thesis.

1.1 Swarm Intelligence and Ant Colony Optimization

Nature harbors a wide variety of intelligent species. Humans tend to believe that they embody the ultimate form of intelligence, but generally speaking they also agree that more creatures can be labeled *intelligent* to some extent. In their habitat, animals and humans share the ability to respond to inputs from their surroundings, explore their environment, memorize information, and learn from events. These abilities constitute what we will call intelligent behavior.

Scientists have long been trying to mimic this intelligent behavior, leading to the establishment of Artificial Intelligence (AI). This field has given rise to a variety of algorithms and hardware capable of behaving intelligently. In analogy to natural intelligence, this includes responding to sensory inputs, exploring different behaviors, storing information in memory, and even learning from the outcome of actions. The field of AI has become very broad, including research in symbolic reasoning, pattern recognition, and computer vision. Other research aims at mimicking processes fundamental to human intelligence, like neural networks (simulating brain functions), genetic algorithms (simulating evolution by natural selection), (fuzzy) expert systems (reasoning with linguistic terms), and reinforcement learning (simulating learning from experience).

There exists another form of natural intelligence, which is even much more present in our world. This form is called *swarm intelligence* and entails the intelligent behavior of groups of individuals that may in themselves have only a very limited intellectual capacity. A good example is the behavior of ant colonies. While individual ants have only very limited capabilities of sensing their environment, making decisions, and storing information, the colony as a whole is very capable in these respects. Seen from a distance, the colony almost acts as one organism searching its environment for food with its many sensors, storing information in its structure and in the chemical patterns inside and surrounding it. Changes in the environment

are observed by the colony and its behavior even adapts to such changes.

Like other forms of natural intelligence, swarm intelligence has also been subject to intense study by the AI community. Initially, biologists and computer scientists have teamed up to model the behavior of ants and other species known to demonstrate swarm intelligence. In the case of ants, most famous is the double bridge experiment (Deneubourg et al., 1990), in which ants forage for food and must choose between two branches of a bridge. This experiment has demonstrated that ants find shortest paths in a distributed manner by communicating through chemical trails, called pheromone trails. A shorter path results in a faster accumulation of pheromones, which biases other ants to choose that same path as well. This very simple form of positive feedback is the very basis of their success as a species.

After the publication of the results of the double bridge experiments, it did not take long before Dorigo (1992) realized that the problem of finding shortest paths, which ants had so elegantly solved, was related to many very difficult problems in computer science, namely those known as combinatorial optimization problems. Perhaps the most well known example of such a problem is the traveling salesman problem, for which the goal is to find the shortest tour in a graph, visiting all nodes exactly once and finishing in the starting node. The time and resources required to find the solution to such a problem are generally believed to grow exponentially for a linearly increasing number of nodes in the graph. The class of algorithms that Dorigo introduced is called Ant Colony Optimization (ACO) and has proven to be very successful in solving a wide variety of combinatorial optimization problems.

1.2 Ant Colony Learning

The basis of this thesis is the idea that the principles behind ACO can be applied to another class of problems, namely that of automatic control. We consider automatic control systems that control the state of a dynamic system to a certain reference value. The speed, accuracy, and robustness of a controller usually determine its performance. It can be very difficult to design a good controller, especially in the case that the dynamics of the system to be controlled are unknown, or uncertain. An established method to learn control policies in such cases is Reinforcement Learning (RL). In RL, the controller is usually called the *agent* and the system to be controlled is usually called the *environment*. By interacting with the environment and receiving rewards for its actions, the agent learns to effectively operate in the environment and to maximize the (discounted) sum of rewards. As an example, consider an agent that must find the shortest path to the exit in a maze. If the agent receives, for instance, a reward of minus one for each action that does not bring it to the exit directly and a reward of plus one for an action that does, maximizing the cumulated reward corresponds to mapping the states of the maze to the actions that take the agent to the exit as fast as possible. The mapping from states to actions is called the control policy. In this thesis, we develop novel algorithms for control policy learning based on the collective behavior of multiple agents. In the example of the maze, one can now imagine a collection of agents (called ants) that act in the maze at the same time, communicating their experiences by means of modifying pheromone levels. We call these algorithms Ant Colony Learning (ACL).

The main challenge in ACL is the representation of the state space. In the example of the maze, we implicitly assume that it consists of a finite number of discrete states, like squares on a chessboard. However, in most control problems, the state space is continuous. Still, in a computer program, the states must be represented in a discrete manner. In this thesis, we

develop two versions of ACL. In the first version, the continuous state space is simply partitioned with a finite number of bins, and a continuous-valued state can only be discretized to one bin. This version of ACL is called crisp ACL. It is a relatively straightforward extension of the ACL framework, but comes with the disadvantage that the crisp way of discretization introduces non-determinism in the learning problem, limiting the performance of the algorithm. Moreover, increasing the number of bins (to reduce the non-determinism) quickly increases the number of state-action pairs that must be sampled by the ants, and thus also increases the time and memory required to find the optimal control policy. The second version is called fuzzy ACL, which is a more elaborate extension of the ACL framework. Here, the state space is represented by membership functions, thereby preserving the continuity of the state values. This thesis introduces the ACL framework and both algorithms and investigates their performance both analytically and through computer experiments.

1.3 Research Focus and Contributions

The main purpose of the research presented in this thesis is to develop efficient algorithms for control policy learning, based on the swarm intelligence properties of ACO algorithms. Accordingly, we develop the ACL framework, in which the state-action space of a control system is sampled by a collection of ants. ACL allows for a parallel implementation of the ants, which has the potential to speed up the learning considerably.

The ACL framework requires the state-action space of the control system to be represented by a set of discrete values. However, most real-world control problems have state variables that are continuous-valued. Therefore, an important focus of this thesis is to develop ACL algorithms that effectively represent the state space of the control system with discrete variables, while sampling from the continuous state space of the system that must be controlled. In this thesis, we analyze the properties of the framework and the algorithms both theoretically and through a number of experimental studies. This thesis also presents a generalization of the ACL framework in order to put ACL in the context of other swarm intelligence techniques, such as Particle Swarm Optimization (PSO) and the control of swarms of moving agents.

The main contributions of this thesis are:

- The development and theoretical analysis of the ACL framework for control problems characterized by discrete state spaces.
- The extension of the ACL framework to control problems with continuous state spaces by the development of two algorithms, which are called crisp ACL and fuzzy ACL.
- The generalization of the ACL framework in order to relate it to other swarm intelligence techniques.

These contributions are further specified in the remainder of this section.

1.3.1 Ant Colony Learning Framework

The major contribution of Chapter 3 is the introduction of the ACL framework. It is based on the collective learning of control policies by a set of ants. We consider control policies as

mappings from states to actions, and thus as state-feedback controllers. The optimal policy controls the system from any initial state to a desired goal state, thereby minimizing a given cost function. The framework, as presented here, requires the control problem to have a discrete state space. A pheromone matrix is associated with all possible state-action pairs in the system, and at the start of the algorithm, its elements are initialized to a small positive value. In short, each ant is initialized in a certain state, chooses an action based on the pheromone matrix, and uses the action as an input to the system. The system then responds by changing its state, which is observed by the ant. This process continues until the ant either reaches the goal state, or is timed out. The state-action pairs visited form the solution, which is evaluated with respect to a cost function, based on which the pheromone matrix is updated. As better solutions lead to higher pheromone levels associated with the respective state-action pairs, these become more likely to be chosen by the ants again. Other contributions of this chapter involve a theoretical study of the behavior of the ACL algorithm, as well as an experimental study of the effect of various learning parameters on the performance of ACL. The ACL framework has first been published in (van Ast et al., 2008a) and has been subsequently refined in (van Ast et al., 2009b) and (van Ast et al., 2010a).

1.3.2 ACL in Continuous State Spaces

The main contribution of Chapter 4 is the development of two ACL algorithms for the control policy learning of continuous-state dynamic systems. The partitioning of the state space is a crucial aspect for ACL and the two ACL versions are presented with respect to this. In crisp ACL, the state space is partitioned using bins, such that each value of the state maps to exactly one bin. Fuzzy ACL, on the other hand, uses a partitioning of the state space with membership functions. In this case, each value of the state maps to the membership functions to a certain membership degree. Both ACL algorithms are extensions of the ACL framework, presented in Chapter 3. The other contribution of this chapter involves an experimental analysis of both crisp and fuzzy ACL by their application to the non-linear control problem of two-dimensional navigation with variable damping. Both algorithms are compared with respect to the learning speed, the quality of the learned policies, and the repetitiveness of the results over various runs of the algorithms. Crisp ACL has first appeared in (van Ast et al., 2008a) and has been refined in (van Ast et al., 2009b). Fuzzy ACL has been originally published in (van Ast et al., 2009a). The generalized pheromone update unifying crisp and fuzzy ACL has been introduced in (van Ast et al., 2010a) and has been refined in (van Ast et al., 2010b).

The contribution of Chapter 5 is the thorough experimental analysis of both ACL algorithms. We perform a number of experiments in which we study the behavior of ACL in various scenarios. The system used in all experiments is the inverted pendulum with limited input, for which the control goal is to swing it up from any initial state and to stabilize it in its unstable equilibrium. With the experiments, we study the influence of the global and local pheromone decay rates, the number of ants, and the density of the state space partitioning grid on the learning performance. Parts of this chapter have been published in (van Ast et al., 2009b) and in (van Ast et al., 2010b).

1.3.3 Generalization of the ACL Framework

ACL can be seen as being part of a much larger class of moving agent optimization and control algorithms. The contribution of Chapter 6 is the introduction of a general modeling framework for swarms of moving agents. It provides a better insight into the structure of swarm systems. The framework separately models the physical behavior of the swarm members and their decision making capabilities. This facilitates the integration of current swarm intelligence research, which focuses mainly on the physical behavior of the swarm members, with research on more sophisticated decision making in dynamic agent systems and AI. The proposed framework aims to integrate both fields to enable the development and analysis of more sophisticated swarm systems. It has been originally published in (van Ast et al., 2008b).

We relate ACL as well as two of the most established methods from the swarm community, namely Particle Swarm Optimization (PSO) and artificial potential functions for swarm aggregation to the proposed framework. Although ACL is a cooperative learning method, PSO is an optimization method, and swarm aggregation is a control problem, it is demonstrated how these methods can all be decomposed into similar elements and captured within one common framework. We have published a survey about particle swarms in optimization and control in (van Ast et al., 2008c). A general introduction to swarm intelligence has been published in (van Ast, 2010).

1.4 Thesis Outline

A graphical roadmap depicting the organization of this thesis is presented in Figure 1.1. The recommended order of reading is indicated with arrows. Chapter 6, presenting the generalization of the ACL framework, can be read separately from the chapters dealing with the implementation and experimental analysis of ACL.

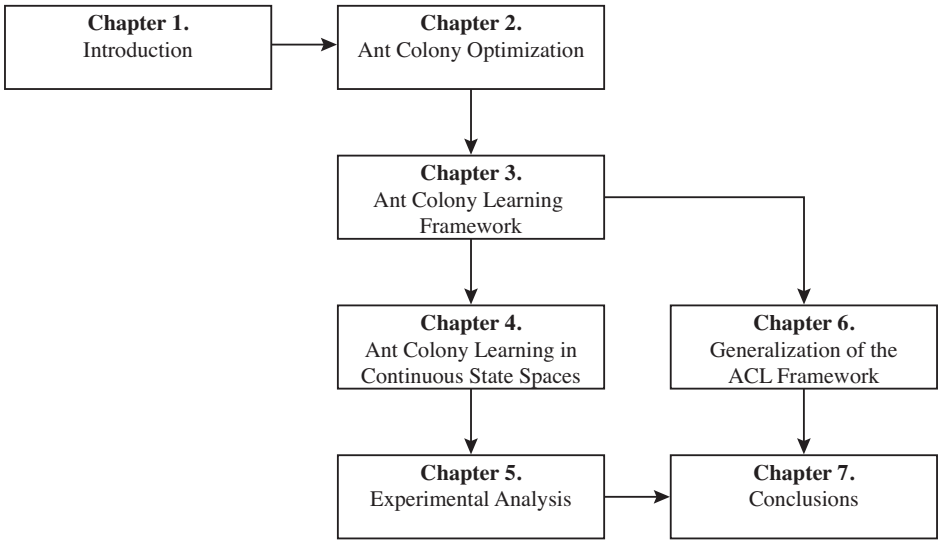


Figure 1.1: A roadmap of the thesis. The arrows indicate the recommended order of reading.

The main concepts of swarm intelligence and ant colony optimization are presented in Chapter 2. The first part of this chapter is a general introduction to swarm intelligence in an engineering setting. The second part focuses entirely on ACO. First, the behavior of real ants is discussed, as well as the modeling of this behavior by means of the double bridge experiments. Then, the ACO framework and the implementation of two important algorithms, the Ant System and the Ant Colony System are presented. This chapter provides the required background for understanding the remainder of the thesis. In Chapter 3, the ACL framework is presented. It defines the optimal control setting and discusses in what way the elements that constitute ACO algorithms need to be modified in order to be used in the state-action framework. A theoretical analysis of its convergence properties is presented and various experiments are carried out in order to study its performance. Chapter 4 presents ACL in the continuous domain by introducing the crisp and fuzzy ACL algorithms and studying their performance on a non-linear control problem with a continuous-valued state. In Chapter 5, a more thorough experimental analysis is carried out, comparing both algorithms for a variety of scenarios. The generalization of ACL to the moving agents framework is the subject of Chapter 6. Chapter 7 concludes this thesis and presents an overview of recommendations for future research.

Chapter 2

Ant Colony Optimization

This chapter provides the necessary background for the remaining chapters of this thesis. It presents a discussion about swarm intelligence, its occurrence in nature, and its application in engineering and computer science. The current chapter also presents an introduction to ant colony optimization and discusses its convergence properties and its applications.

2.1 Introduction

Since its introduction, in the PhD thesis of Dorigo (1992), the field of Ant Colony Optimization (ACO) has grown enormously. Especially with the publication of the Ant System (AS) by Dorigo et al. (1996) and the Ant Colony System (ACS) by Dorigo and Gambardella (1997), ACO has developed into one of the two most studied and applied swarm intelligence methods, alongside Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995). The ACO metaheuristic forms a class of nature-inspired optimization algorithms, particularly inspired by the mechanism for finding shortest paths between sources of food and the nest in colonies of ants. This mechanism is entirely based on the self-organization of relatively simple individuals. In this chapter, we introduce the basic concepts of swarm intelligence and self-organization in both a biological context and in an engineering context. In particular, we present a more detailed description of the behavior of ants in a colony, the famous double bridge experiment that has resulted in the first local model of ant decision making being able to reproduce global ant colony behavior, and finally ACO itself. The ACO framework, including the details on the AS, the ACS, and a convergence proof for a special class of ACO algorithms are especially important for the understanding of the subsequent chapters.

2.2 Swarm Intelligence

Swarm intelligence is an exciting and relatively new field within artificial intelligence and specifically studies intelligent behavior emerging from the collective behavior of a set of individuals. More specifically, researchers in the field of swarm intelligence study and develop algorithms and hardware (robots) based on the principles of this collective behavior. The emergent behavior is often regarded as intelligent, as it appears not to be directly encoded

by the developer. It is however a property of the swarm resulting from the interaction of its individual members. This section first discusses in more detail the original inspiration for swarm intelligence: natural swarms. We then informally define what constitutes a swarm and in which ways the principles behind it can be applied to engineering sciences. This is important in order to place the whole concept of ant-based algorithms, which will be discussed in the subsequent sections of this chapter, in the right perspective.

2.2.1 Collective Behavior in Nature

The main mechanism for swarm intelligence is that the collective behavior of the individuals in a swarm benefits the survival or reproduction of these individuals. Examples of swarms in nature are numerous and include flocks of birds, schools of fish, clouds of insects, herds of antelopes, groups of humans, and some say even the cells in a body. This latter example is not as strange as it might seem when we realize how on a larger scale, swarms seem to behave like a single organism.

It is important to realize that what constitutes a swarm is a subjective issue. Amongst several reasons, this is caused by the question of how many individuals it takes to get a swarm? For sure it takes more than one, but do two constitute swarm? Even with two individuals, some emergent global behavior may occur from the interactions of the two. But we tend to call something a swarm only if there are "a lot" of members. Like hundreds of birds, or thousands of flies. Furthermore, we tend to think of a swarm as containing "small" individuals. A cloud of mosquitoes clearly is a swarm. But how about humans? Walking from the train to the bus, one may not feel like being part of a swarm, but from the top of a tall building next to the station, one is likely to compare all these people on their way to their jobs to a swarm of ants. And how about stars in the sky? From some telescope images one may indeed look at them circling around the center of the milky-way in a swarm-like manner. But is this really a swarm?

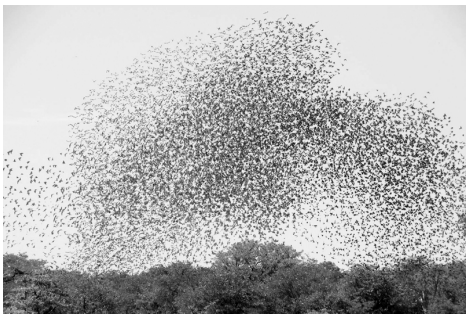
Let us aim for an informal definition, or classification of a swarm:

- There must be "enough" of them, but at least two or more.
- Size does not matter. A swarm of tiny individuals is just as much of a swarm, as a swarm of huge individuals.
- The individuals must be able to communicate, or at least influence each other, when being within a certain range of each other.
- There must be a certain level of cohesiveness: the communication topology of the individuals must be connected, such that information in principle can propagate through the swarm. If a portion of the swarm gets disconnected from the rest, we would rather regard it as two swarms.
- There must be some kind of movement: birds in tree tops are just a group of birds. Once they all fly up, triggered by a few that noticed some potential threat, we speak of them as a swarm.
- Each individual must be able to receive input in whatever form from the environment (including the others of its kind), and must be able to act. The mapping between the

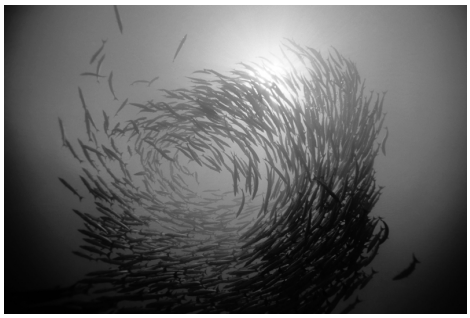
input and the action may be of any kind, but it is highly responsible for the characteristics (the behavior) of the swarm. For example, if it causes all individuals to spread out far away from each other, the swarm would not satisfy the cohesiveness requirement: it will dissolve. As an other example, if this mapping causes all individuals to stall and become unresponsive, there is not much of a swarm left either, as it violates the movement criterion.

Only if a set of individuals comply with all these requirements, we can call it a swarm.

Swarms *emerge* from the interactions of their members, so to speak. The behavior of the individuals is encoded in their DNA and possibly also in their memory, dictating the (possibly probabilistic) mapping of their inputs and current state to their actions. The behavior of the swarm is encoded in its members and in the interaction between them. On a micro scale, the swarm behaves chaotically: although the response of the individuals to inputs and their interactions might be deterministic, their exact motion in the swarm may be very different for tiny variations in their initial states. This chaotic behavior occurs even though the behavior of the individuals is very simple. Fortunately, studying the swarm on a micro scale is of limited interest to us, as will be argued below. What is much more interesting, and fascinating, to most people is that a swarm may often look like a single organism. The individuals seem to act as one. A flock of birds that fly up from the trees seems to steer through the sky, sometimes expanding in size, and sometimes shrinking to a small and condensed unity. Figure 2.1(a) shows this fascinating phenomenon.



(a) A large number of birds that collectively fly up from trees behave as a flock. © Alastair Rae. Reprinted under the Creative Commons license.



(b) A school of fish forms a cohesive swarm. © Tammy Peluso. Reprinted with permission.

Figure 2.1: Two examples of swarming in nature: a flock of birds and a school of fish.

These global characteristics of the swarm seem to exhibit much more order than the behavior on the scale of the individuals. While it would be nearly impossible to describe and predict the motion of the individuals in the swarm, doing the same on the macro scale is much more feasible. The magic of swarms lies in the simple set of rules dictating the behavior of each swarm member, inducing complex and seemingly disordered behavior from the interaction of the members on a micro scale, but producing understandable and ordered behavior of the swarm on a macro scale.

The level of the chaotic behavior on a micro scale differs for different types of swarms. For example, in a school of fish, the individual fish may move through the school, but from a

global perspective the swarm may be like a ball-shaped single organism moving through the water[1]. A v-shaped flock of birds, however, is ordered on a micro scale as well. The point is also that we are not overly interested in the exact behavior of the individuals in the swarm. Whether it is one particular member that make a sudden move, inducing a movement of the whole swarm to the left, or it is another one does not really matter. The global behavior is the same.

From the previous discussion it becomes clear that the key concepts in swarm intelligence are *emergent behavior* and *self-organization*. The apparent mismatch between the vagueness of these concepts and the exact areas like control engineering, computer science, and algorithm design to which the principles of swarm intelligence are applied causes a lot of dispute amongst researchers. These terms seem to imply that there is something literally magical about the process of achieving ordered behavior on a global level, from apparent disordered behavior on a local level. The word "seemingly" is important here, and pinpoints the exact reason for the dispute over the two mentioned terms. As the global behavior follows logically and deterministically from a well-defined set of local rules, there is nothing magical about this. However, understanding, describing, or predicting the exact behavior of the individuals is often very difficult, or at least computationally intensive. The resulting behavior is therefore often *regarded* as something magical. To call the behavior emergent just means that it resulted from a seemingly disordered process: it facilitates discussion.

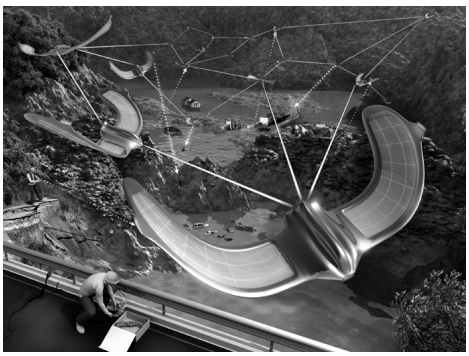
We have so far implicitly assumed that all individuals are the same, being in fact all copies of each other, only initialized differently. We call such swarms homogeneous. On the other hand, if one or more individuals are different from the rest, the swarm is called heterogeneous. Most swarms in nature are more or less homogeneous. However, with some types of insects, for instance with ants, the individuals may be different from each other in both physical characteristics, as well as in their function within the swarm. The soldier ants are much stronger than the workers (all female), and they are both different from the males, whose only task is to inseminate the queen. The workers may have different tasks, and also switch between these tasks; some are occupied with building the nest, some with nurturing the larvae, and some with foraging food. If a substantial source of food is found, other worker ants are recruited to switch to foraging. If the nest has been damaged, other workers may be recruited to help repair the nest. A colony of ants is thus a heterogeneous swarm, while a school of fish is generally a homogeneous swarm.

2.2.2 Principles of Swarm Intelligence in Engineering

Why is all of this of interest to us? The reason is that exactly the same principles of swarm intelligence in nature can be used in engineering swarms that can be of use to humans. The principles of natural swarms may be translated to engineered swarms in that a swarm consists of a *set of cooperating autonomous* individuals, called *agents*. These agents try to satisfy their own objectives. Through communication and coordination, cooperation with other agents leads to a better satisfaction of their individual objectives. Communication and interaction are often limited to a certain range, so that cooperation between agents only takes place locally. There is no supervisor or central controller and typically no hierarchical structure.

[1] With this behavior, the fish drive predators crazy, as a shark, for instance, does not know on which fish to focus for it is incapable of seeing the big picture. This behavior is however not effective as a defense to all predators: humans make good use of the swarming behavior of fish, catching whole schools at once with their nets.

Such engineered swarms will have certain advantages comparable to natural swarms, but in the engineering domain. These advantages relate to scalability, robustness, flexibility, and production costs. Scalability in this sense means that individuals may be added without requiring more computational power in the others. The key factor here is that individuals typically have a limited range of interaction, so that the introduction of a new individual only directly influences the behavior of a few others and not that of the whole swarm. Indirectly though, through the communication topology of the complete swarm, the new individual may influence all the others, but this can only be achieved through a series of interactions between others in the swarm. With respect to robustness, this limited range of interaction prevents the malfunctioning or removal of a single individual from influencing the complete swarm instantly. If well designed, the individuals within the range of the faulty one may compensate for its malfunctioning behavior, or replace it completely by taking over its functionality. The rest of the swarm will only be influenced in a much-reduced manner. Flexibility in the context of swarms means that the swarm may adapt to different circumstances. A well-designed robot swarm may in one situation spread out in order to explore an unknown territory, while in another situation when they face a wide gap that a single individual would not be able to cross, they may physically connect, helping each other to the other side of the gap (see also Figure 2.2(b)). The fourth main advantage of engineered swarms is that when such a swarm typically consists of many similar individuals, these may be produced in series, reducing the cost per unit. These costs may already be quite low, as individuals in a swarm are typically relatively simple and can sometimes even be built using cheap components, because of the robustness property described earlier. Figure 2.2 shows two examples of engineered swarms that are the state of the art in 2010.



(a) Artist impression of a swarm of Micro Air Vehicles (SMAVs) capable of autonomously establishing emergency wireless networks (SMAVNETs) between multiple ground-users in a disaster area. From: the SMAVNET project at EPFL, Lausanne, Switzerland. © LIS, EPFL. Reprinted with permission.

(b) Swarm-bots forming a chain to bridge a gap, which is too large for an individual to cross alone. The SWARM-BOT project develops hardware for testing and using the capability of self-assembling, self-organizing, and metamorphosis of robotic systems at ULB, Brussels, Belgium and EPFL, Lausanne, Switzerland. © LIS, EPFL. Reprinted with permission.

Figure 2.2: Two examples of projects aiming at engineering swarms: the SMAVNET project at EPFL, Switzerland and the SWARMBOT project at ULB, Belgium.

The main difficulty in engineering swarms is, of course, that we must design the individuals and the specifics of their interaction in such a way that the resulting behavior on the global

level is as desired. This is a challenging difficulty and it is the central question in all research on engineering swarm intelligence. It is however not the only difficulty. An important question is how to mathematically guarantee performance. How to test a swarm and measure its performance? How do delays in communication and errors propagate through a swarm, and how does this influence the behavior? If the swarm is to converge in some sense, how to make sure that it converges correctly and fast enough? And in case the swarm must not converge, how to find the right balance between not converging and staying cohesive? How do external influences (disturbances) affect the swarm? Some of these questions are also relevant to the subfield of swarm intelligence to which we have contributed, namely that of optimization and the learning of control policies.

There are a lot of opportunities for engineering swarms. One may think of a swarm of satellites, dynamically optimizing their coverage for observing the earth, for providing communication, or for global positioning. Other swarms of space vehicles could be deployed to explore unknown regions of space, or planets. On Earth, a swarm of rescue robots could be developed to rescue people in inaccessible places, such as in a tunnel filled with smoke. In traffic, cars might be understood as swarm members and the overall performance of the traffic network might be improved by self-organization of the vehicles. The engineering of swarms, however, does not have to be limited to the physical world. Various classes of swarm intelligence *algorithms* have been developed for solving various types of optimization problems. In these problems, the swarms operate in a virtual world, called the optimization or parameter space, in which their objective is to find the parameter values that optimize a certain cost function. The most prevalent swarm intelligence optimization methods are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). The latter is the basis of our work and is introduced in the subsequent sections. The former is discussed in more detail in Section 6.3.2.

2.3 Ant Colonies

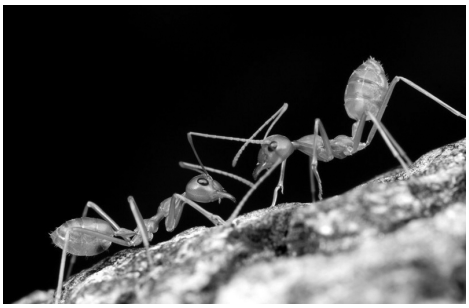
The kinds of swarms we are most interested in for our research are ant colonies. The self-organizing mechanisms at the foundation of the organization of each ant colony have enabled them to be such evolutionary successful species. They have been present on the earth for hundreds of millions of years and have covered most of the earth, except for Antarctica[2]. In this section, we briefly outline the main characteristics of ant colonies and the experiments that have resulted in the first models of ant behavior, with the purpose of showing the source of inspiration that has led to ACO algorithms. The facts outlined in Section 2.3.1 can be found in (Gordon, 1999) and the excellent web resource (Gordon, 2003).

2.3.1 Ants in Nature

An ant colony consists of a queen, male ants, and sterile female workers. A colony is founded when a virgin queen and a couple of males, all with wings, leave the nest in search for a new place to start a colony. The queen mates with the males, after which the males die and the queen digs herself into the soil where she will start laying eggs for the rest of her life, which may last for as long as 15 years. After the first eggs have turned into larvae, they will develop

[2]Despite its name ;-)

into pupae and finally into adult female workers. These workers will perform all tasks in and around the nest. For most of the 10 000 ant species in the world, these tasks can be subdivided into foraging, patrolling, nest maintenance, and midden work. Foraging is the most important task in the colony and consists of walking around in search for food and returning it to the nest entrance. Other ants will then take this food and bring it inside the nest, mostly for feeding the larvae. Patrolling ants are the first to leave the nest in the morning and determine the trails along which the foragers will start searching for food. They do this by walking away from the nest and leaving a trail of a chemical substance, called *pheromones*, on their way back. The sole making it back of these ants is the indication that it is safe to explore around that trail for food. Nest maintenance is the task of taking pieces of dirt from the nest to the nest entrance and piling it into what is known as a midden, and to maintain the internal structure of the nest. Midden workers take the midden from the nest entrance, mark it with some chemical characteristic for this colony, and lay it down somewhere else. These heaps of chemically marked refuse are marking the colonies territory and guiding ants belonging to the colony back to the nest entrance. Furthermore, some specialized foraging ants may act as soldier ants, attacking intruding ants from other colonies in order to protect the territory.



(a) *Oecophylla longinoda* workers encounter each other on a tree branch and by quickly touching each other with their antennae are informed about each others task. © Alex Wild. Reprinted with permission.

(b) Argentine ants, introduced by human commerce to California, attack a native *Pogonomyrmex* harvester ant. Native ants in many places around the world have disappeared in areas invaded by Argentine ants. © Alex Wild. Reprinted with permission.

Figure 2.3: Two images of ants illustrating some of their behavior described in this section.

About 25% of the workers is working outside the nest, while another 25% is taking care of the larvae inside the nest, and the remaining 50% is inside the nest and doing nothing. They seem to wait for something to do and act as a buffer between the ants working outside the nest and those feeding the larvae deep inside the nest. The buffer ants may be recruited to work outside once the need for that arises. Ants performing a certain task carry a certain task-specific odor from spending a lot of time close to each other. Ants touch each other when they meet, sensing each others odor, and by the rate of these encounters they are believed to be able to infer whether they should keep on doing the task they were doing, or change their task. In this way, ants are capable of switching tasks on the basis of a dynamic need for it. For instance, if a lot of food is found and being brought back to the nest, foraging ants will focus on that trail leaving a lower density of foraging ants at other locations, thereby inducing the tension in other ants to switch to foraging. It has turned out that not all switches are possible. All types of workers may switch to foraging, while foraging ants will never

switch to doing other jobs. Nest maintenance workers may switch to patrolling as well, while new nest maintenance ants can only be recruited from the buffer ants in the nest. In this way, foraging ants are the sink in this network of switching tasks and the buffer ants inside the nest act as a source of workers. As long as the queen is alive, she will keep on laying eggs from the first insemination, and the colony will keep on growing in size. The size of the colony is thereby a function of the age of the colony and reveals certain properties of the colony. For instance, younger colonies are more adaptive to sudden changes around their nest, but tend to focus on known good sources of food and will not explore much. On the other hand, older colonies seem to adapt less to changes in their immediate surroundings and appear to be more willing to take the risk of exploring more and further away from the nest.

The central aspect in ant colonies, which is especially interesting for the inspiration to develop ant-based algorithms, is that the interaction between ants is taking place in large numbers and that there is randomness involved in the interactions. The behavior of the individual ants as a result of this seems to be very chaotic and not very effective. Gordon (2003) mentions that she often feels the urge of helping the ants as they do not seem to be doing a good job of doing something useful. However, the resulting behavior on a global level is highly successful and predictable to a large extent. The behavior may not be perfect, but at least very good, and with the additional benefits of flexibility, scalability, and robustness.

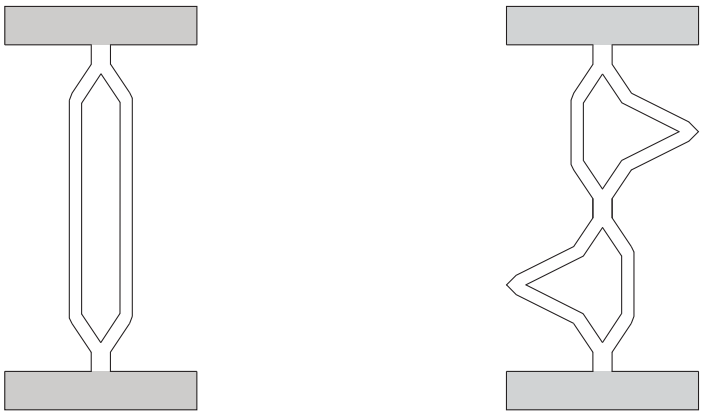
2.3.2 Double Bridge Experiment

Deneubourg et al. (1990) and Goss et al. (1989) have been the first to perform the famous double bridge experiments in order to model the self-organizing behavior of foraging ants. This model has later become the basis for the first ACO algorithms. We explain the double bridge experiments, the fitted model, and the most important results such that the foundations of ACO can be understood and the amazing self-organization principles in ant colonies can be appreciated even better.

The first double bridge experiment was carried out by Deneubourg et al. (1990) and involved the Argentine ant *Iridomyrmex humilis*, which was reclassified in the early 1990s to the genus *Linepithema*, thereby changing its scientific name to *Linepithema humile*. The workers of the Argentine ant are about 3 millimeters long with their queen being two to four times this length. Although being native to Northern Argentine, Uruguay, Paraguay, and Southern Brazil, they have spread over large parts of the world, mainly with Mediterranean climates, including the United States, Europe, and Japan, with the aid of humans. This makes them one of the most widespread invasive ant species (Tsutsui et al., 2001), often displacing most, or all native ants. As such, they are widely considered to be a pest. Unlike most other ant species, Argentine ants from distant locations still act non-aggressively when placed together. Amongst other findings, this has led to the belief that all Argentine ants in fact form one single mega-colony. According to Walker (2009): "Whenever ants from the main European and Californian super-colonies and those from the largest colony in Japan came into contact, they acted as if they were old friends. These ants rubbed antennae with one another and never became aggressive or tried to avoid one another. In short, they acted as if they all belonged to the same colony, despite living on different continents separated by vast oceans.".

What made the Argentine ant interesting for the double bridge experiments is that, unlike most other ant species, they deposit pheromones not only when returning with food, but also while advancing from the nest exploring new regions for finding food. In the setup discussed

in (Deneubourg et al., 1990), there are two contained arenas, separated from each other, but connected by a bridge. In one arena is the nest of Argentine ants, which can leave the arena only by the bridge, leading them to the only entrance to the other arena, in which food is located. The bridge, however has two branches, hence the name double bridge. A schematic of this double bridge setup is shown in Figure 2.4(a). Both branches are at an angle of 60°, such that ants reaching the arena are likely to move on forwards, rather than returning directly back on the other branch. The distance covered by the bridge was 15 cm. The width of each branch is 1 cm. The bridge is covered with white sand, such that restarting the experiment involves only replacing the sand with new (chemically unmarked) sand, and putting all ants back in the nest. During the experiment, the number of ants on each bridge was counted in 3-minute intervals.



(a) Double bridge with branches of equal size.

(b) Double bridge with branches of unequal size.

Figure 2.4: Schematic view of the setups used in the double bridge experiments from (Deneubourg et al., 1989, 1990; Goss et al., 1989).

It was observed that initially, both branches were chosen by an equal number of ants. However, as each ant leaves a pheromone trail continuously, and the level of pheromone intensity influences the other ants in their decision of choosing the right, or the left branch, a small difference between the pheromone intensities on the branches may result in the breaking of this symmetry. This positive feedback mechanism stimulates the ants to favor one of the branches over the other, such that rapidly the vast majority of the ants will choose the same branch. The hypothesis was that the decision making of the ants could be described by the following probability function:

$$P_A = \frac{(C + N_A)^\alpha}{(C + N_A)^\alpha + (C + N_B)^\alpha} = 1 - P_B, \tag{2.1}$$

where P_A and P_B are the probabilities that an ant will choose branch A and B respectively, and where N_A and N_B represent the number of ants that have passed branch A and B respectively after in total $N = N_A + N_B$ ants have crossed the bridge. The parameters C and

α must be chosen such that the model matches the observed behavior. Note that this probability function does not directly include the pheromone values, but depends on the number of crossings of a branch, which is assumed to be reflected by the pheromones. The evaporation of pheromones is not modeled. In the case of the Argentine ants, the mean lifetime of a pheromone is 30 minutes: much longer than it takes an ant to cross the bridge. The effect of pheromone evaporation was thus neglected in the experiments. Through Monte Carlo simulations on this model, the parameters α and C were fitted to the data, resulting in $\alpha \approx 2$ and $C \approx 20$. This was the first empirical proof that a simple probability rule, including only local measures, was the underlying mechanism behind the self-organizing behavior of foraging ants.

The second double bridge experiment, important for the eventual development of the first ACO algorithms, was published by Goss et al. (1989) and involved branches of different length. The setup was similar to the one from Deneubourg et al. (1989). The two branches were connected to either end of the gap under an angle of 60° such that there would be no preference of the ants for one of the branches by their initial angle. However, after a few centimeters, one of the branches proceeded at a larger angle, resulting in that branch being longer than the other. The setup is depicted in Figure 2.4(b). In fact, in the middle of the bridge, the ants choose again between two branches, and now the longer and shorter branch are swapped, such that possible preference of an ant of choosing left or right is canceled out. Initially, the ants were equally likely to choose the longer, or the shorted branches. However, the ants that had chosen the shorter branch reached the food sooner and on their way back, the pheromone concentration on the shorter branches was somewhat stronger than that on the longer branches. They were thus more likely to choose a shorter branch, further adding to the pheromone concentration on that branch. Quickly, the pheromone difference became considerable and effectively all ants kept choosing the shorter branches. Monte Carlo simulations using the same probability function as in (2.1) were capable of reproducing these results *in silico*. The self-organizing principles thus demonstrated that the global decision problem of finding the shortest path can be solved on the basis of local information and a large number of individuals. An additional observation was made when the shorter branches were absent from the start of the experiment and introduced only later, when the ants had already created a strong pheromone trail on the longer branches. The ants were found to be incapable of switching to the shorter bridges. The Monte Carlo simulations confirmed this behavior: with only depositing pheromones and without a significant evaporation, a resulting strong trail is practically irreversible.

In (Deneubourg et al., 1990) the same researchers went one step further by modeling the exploring and foraging behavior of ants in a two-dimensional arena by a sequence of binary decision problems similar to choosing one of two branches in the double bridge experiments. They found that with Argentine ants, the front of the exploring ants proceeds slowly, carefully moving forward and branching out through the arena. The trails that they leave behind enable other ants to make decisions more quickly, forming some sort of highways for ants. Once an ant has found some food, it returns to the nest over these same trails, thereby reinforcing the trails even more. The result is a pattern of branches, where successful exploration results in strong trails leading to these food-rich regions, and in further exploration around these regions. A second mechanism contributing to the increase in exploration and the reinforcement of pheromone trails is the recruitment of new workers from the nest by the ants returning to the nest and carrying food.

In most ant species, foraging ants do not deposit pheromones on their way out of the nest, or deposit much less pheromones. Also, the amount of pheromones deposited on the way back may depend on the quality, or amount of food that has been found, thereby reinforcing the trails leading to better food sources even more. Note that with only depositing pheromones on the way back to the nest, the self-organizing mechanism of finding shortest paths still works. This forms the basis for ACO.

2.4 ACO Metaheuristic

The ACO metaheuristic has been developed to solve combinatorial optimization problems (Dorigo and Blum, 2005). A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. The formal definition of ACO is thus a very general one. In this section we will define the ACO metaheuristic (or framework) in a somewhat less general way, already tailored in some sense to match the definition of our Ant Colony Learning (ACL) algorithm, which will be introduced in Chapter 3. First, however, we will define what constitutes a combinatorial optimization problem.

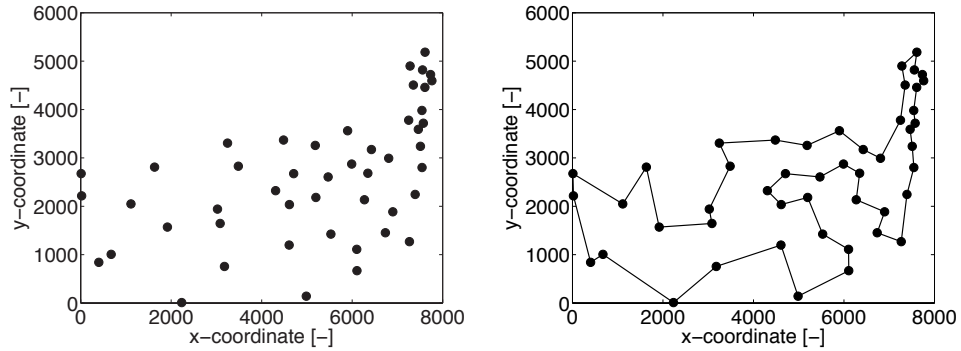
2.4.1 Combinatorial Optimization Problems

A combinatorial optimization problem can be represented as a tuple $P = \langle \mathcal{S}, F \rangle$, where $\mathcal{S}$ is the solution space with $s \in \mathcal{S}$ a specific candidate solution and where $F : \mathcal{S} \rightarrow (0, \infty)$ is a fitness function assigning strictly positive real values to candidate solutions, where higher values correspond to better solutions. When we need to indicate a specific solution, we index it with a subscript: s_i. The optimization problem is to find the solution $s^* \in \mathcal{S}$ that globally maximizes F. If there are more solutions that all maximize the fitness function, these are denoted by the set $\mathcal{S}^* \subseteq \mathcal{S}$. The solution s^* is called an optimal solution and $\mathcal{S}^*$ is called the set of optimal solutions. A solution is an ordered set of *solution components*, of which the optimal combination needs to be found, hence the name combinatorial optimization. If we need to indicate a specific solution component, of a certain solution, we index it with a superscript within parentheses: $s_i^{(j)}$ is the j^{th} solution component of solution i.

Note that this definition is somewhat different from the definition in for instance (Dorigo and Blum, 2005; Dorigo et al., 2006). It is simplified in order to avoid unnecessarily complex notation and to avoid conflicts with the notation we will use in ACL. In the definition used by Dorigo and Blum (2005), there is an additional set of constraints, constraining the so called *feasible* set of solutions to those solutions from the solution space that satisfy these constraints. Whenever we talk about a solution, we will imply that this is a feasible solution and that the solution space is defined for the specific problem at hand such that it only contains feasible solutions. Furthermore, Dorigo and Blum (2005) make an explicit distinction between a decision variable and its value. For the sake of simplicity, we do not make this distinction and it will be made clear from the context whether a solution s_i indicates the variable, or its value.

There are many combinatorial optimization problems, but the most well known is the Traveling Salesman Problem (TSP), which is also known as the Hamiltonian Path Problem. In this problem, there is a set of cities connected by roads of different lengths and the problem is to find the sequence of cities that takes the salesman to all cities, visiting each city exactly

once and bringing him back to the initial city with a minimum length of the tour. This problem is known to belong to the $\mathcal{NP}$-complete complexity class, essentially meaning that probably there does not exist an efficient algorithm that can solve the TSP. In other words, the worst-case time to find the optimal solution to the problem probably increases exponentially with the number of cities in the TSP. The word "probably" is used here, because this is only the case if $\mathcal{P} \neq \mathcal{NP}$, which is widely believed to be the case, but is still an open question in mathematics. A more precise explanation of $\mathcal{NP}$-completeness and the types of problems that have been proven to belong to this class can be found in the classic work of Garey and Johnson (1979). An example of a TSP is shown in Figure 2.5. The vertices in this example represent 48 capital cities of the USA and the optimal tour has a length of 10 628 km (Reinelt, 1991).



(a) The cities are the vertices of the construction graph.

(b) The optimal tour has a total length of 10 628 km.

Figure 2.5: Traveling salesman problem for 48 capital cities of the USA. The construction graph is the fully connected graph of cities.

The TSP is a benchmark for many optimization methods and is the central application for ACO, because of the widely available performance results of other (heuristic) algorithms for this problem, the strong resemblance to the shortest path finding behavior of ants, and because of the clear link to the construction graph, which is the main element of the ACO framework. An overview of papers presenting a comparison between ACO and other algorithms on the TSP and other problems is given in (Blum, 2005).

2.4.2 Framework for ACO Algorithms

In ACO, the combinatorial optimization problem is represented by a graph, called the *construction graph*, which consists of a set of vertices and a set of arcs connecting the vertices. For instance, in the case of the TSP the vertices denote the cities and the arcs represent the roads linking the cities. In the case of the control of dynamic systems, which will be further discussed in Chapter 3, the vertices represent the discrete states of the system and the arcs correspond to the system responding to an input in a particular state.

A particular solution found by an ant c is an ordered set of solution components, $s_c = (s_c^{(1)}, s_c^{(2)}, \ldots)$ and each solution component consists of a pair of vertices, say i and j, which are connected by the arc ij. The solution represents a path over the construction graph, where

"construction" refers to the ants *constructing* the solution incrementally by moving over the graph. An ant on the construction graph starts on an initial vertex and moves from vertex to vertex and adds the corresponding solution components to its *partial* solution $s_{\mathrm{p},c}$ until it reaches the terminal vertex and the partial solution becomes the (candidate) solution found by this ant. The way the terminal vertices are defined depends on the problem considered. For instance, in the TSP, the terminal vertex is equal to the ant's initial vertex, after having visited all other vertices exactly once. For the application to control problems, as considered in this thesis, the terminal vertex corresponds to the desired state of the system and is the same for all the ants. In order to make sure that the ants visit all vertices only once in the case of the TSP, an ant also adds the solution components to its private *tabu* list. When an ant must decide to which vertex it will move next, it can choose any vertex that is not in its tabu list.

Two variables are associated with an arc ij: a pheromone variable τ_{ij} and a heuristic variable η_{ij}. The pheromone variables (also simply called *pheromones*) represent the acquired knowledge about the optimal solutions over time and the heuristic variables (which will *not* be simply called heuristics in order to avoid confusion with the fact that ACO algorithms are heuristic algorithms) provide a priori information about the quality of the solution component, i.e., the quality of moving from vertex i to vertex j. In the case of the TSP, the heuristic variables typically represent the inverse of the distance between the respective pair of cities. In general, a heuristic variable represents a short-term quality measure of the solution component, while the task of the optimization problem is to acquire a concatenation of solution components that together form an optimal solution. A pheromone, on the other hand, encodes the measure of the long-term quality of concatenating the respective solution components.

2.4.3 Relation to Real Ants

The metaphorical ants in the ACO algorithm are simply called ants. Obviously being physically quite different from real ants, they show important similarities and differences in the following ways:

- ACO ants use a similar probability rule for their decision making (2.2) as the model (2.1) for decision making by real ants.
- Unlike real ants, ACO ants have a memory for the places (vertices) they have visited.
- Like most real ant species, ACO ants only drop pheromones after reaching the terminal vertex (food source), simulating walking back to the nest (initial vertex).
- Like in some ant species, the amount of pheromones dropped by the ACO ants is proportional to the quality of the solution (food source).

With these similarities and differences in mind, the ACO algorithms described in the next section can be related to the behavior of real ants.

2.5 ACO Algorithms

Many ACO algorithms have been developed, most of them only differing from the rest in a few minor points to make them perform better on a specific type of problems. This section

reviews the two most well known and most widely applied ACO algorithms, the *ant system* and the *ant colony system*. These algorithms stand at the basis for most of the algorithms later developed, as well as the ACL algorithm that is the main subject of this thesis. In order to understand the mechanisms of ACL, these algorithms must be well understood.

2.5.1 Ant System

The most basic ACO algorithm is called the Ant System (AS) (Dorigo et al., 1996) and works as follows: M ants are randomly distributed over the vertices of the construction graph. The heuristic variables η_{ij} may be set to encode prior knowledge of the problem by favoring the choice of some vertices over others. For each ant c, the partial solution $s_{\mathrm{p,c}}$ is initially empty and all pheromone variables are set to a small initial value $\tau_0 > 0$. We will call each move of the ants over the graph a *step*. In every step, each ant decides based on some probability distribution, which solution component (i, j) to add to its partial solution $s_{\mathrm{p},c}$. The probability $p_c\{j|i\}$ for an ant c on a vertex i to move to a vertex j within its feasible neighborhood $\mathcal{N}_{i,c}$ is defined as:

$$p_c\{j|i\} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in\mathcal{N}_{i,c}}\tau_{il}^{\alpha}\eta_{il}^{\beta}}, \quad \forall j \in \mathcal{N}_{i,c}, \tag{2.2}$$

with $\alpha \geq 1$ and $\beta \geq 1$ determining the relative importance of η_{ij} and τ_{ij} respectively. The feasible neighborhood $\mathcal{N}_{i,c}$ of an ant c on a vertex i is the set of vertices not yet visited (by ant c) that are connected to i and is dictated by the problem structure. By moving from vertex i to vertex j, ant c concatenates the associated solution component (i, j) to its partial solution $s_{\mathrm{p},c}$ until it reaches the terminal vertex and completes its candidate solution by storing $s_{\mathrm{p},c}$ as s_c. The ant is now said to have completed a *trial*.

After all ants have completed their trial, in the AS the candidate solutions of all ants are evaluated using the fitness function $F(s)$ and the resulting value is used to update the pheromone levels as follows:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{s\in\mathcal{S}_{\mathrm{upd}}} \Delta\tau_{ij}(s), \quad \forall(i,j), \tag{2.3}$$

with $\rho \in (0, 1)$ the evaporation rate and $\mathcal{S}_{\mathrm{upd}}$ the set of solutions that are eligible to be used for the pheromone update, which will be explained further on in this section. This update step is called the *global* pheromone update step, in contrast to the *local* pheromone update step that will be introduced in the ACS later on. The global pheromone update refers to the return of the ants to their nest, while leaving a pheromone trail. Different from real ants, in ACO all ants "wait" for each other before "walking back" together. The pheromone deposit $\Delta\tau_{ij}(s)$ is computed as:

$$\Delta\tau_{ij}(s) = \begin{cases} F(s), & \text{if } (i,j) \in s \\ 0 \quad , & \text{otherwise.} \end{cases} \tag{2.4}$$

The pheromone levels are a measure of how desirable it is to add the associated solution component to the partial solution. By pheromone evaporation, it can be avoided that the algorithm prematurely converges to suboptimal solutions. Note that in (2.3) the pheromone levels

on all vertices are evaporated and only those vertices that are associated with the solutions in the update set receive a pheromone deposit.

In the next trial, each ant repeats the previous steps, but now the pheromone levels have been updated and can be used to make better decisions about which vertex to move to. After a stopping criterion has been satisfied, for instance when a certain number of trials have passed, the values of τ_{ij} and η_{ij} on the graph encode the solution for all (i, j)-pairs. This final solution, in terms of the optimal vertex j to choose from a vertex i, can be extracted from the graph as follows:

$$j = \arg\max_l(\tau_{il}^{\alpha}\eta_{il}^{\beta}),$$

where ties are broken randomly in case the product $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ has the same maximum value for a given i and different values of j.

There exist various rules to construct $\mathcal{S}_{\mathrm{upd}}$, of which the most standard one is to use all the candidate solutions found in the trial. This update set is then called: $\mathcal{S}_{\mathrm{trial}}$[3]. This update rule is typical for the AS. However, other update rules have shown to outperform the AS update rule in various combinatorial optimization problems. Rather than using the complete set of candidate solutions from the last trial, either the best solution from the last trial, or the best solution since the initialization of the algorithm can be used. The former update rule is called *iteration-best* in the literature (which could be called *trial-best* in our terminology), and the latter is called *best-so-far*, or *global-best* in the literature (Dorigo and Blum, 2005). These methods result in a strong bias of the pheromone trail reinforcement towards solutions that have been proven to perform well. Additionally, they reduce the computational requirements of the algorithm. As the risk exists that the algorithm prematurely converges to suboptimal solutions, these methods are only superior to AS if extra measures are taken to prevent this, such as the local pheromone update rule, explained in Section 2.5.2. Two of the most successful ACO variants that implement the update rules mentioned above, are the Ant Colony System (ACS) (Dorigo and Gambardella, 1997) and the Max-Min Ant System (Stützle and Hoos, 2000). Because of its relation to ACL, we will explain the ACS next.

2.5.2 Ant Colony System

The ACS (Dorigo and Gambardella, 1997) is an extension to the AS and is one of the most successful and widely used ACO algorithms. There are some important differences between the AS and the ACS. First of all the ACS uses the global-best update rule in the global pheromone update step. This means that only the one solution that has been found since the start of the algorithm that has the highest fitness (i.e., that has a strictly higher fitness than the previous best solution), called s_{gb}, is used to update the pheromone variables at the end of the trial. This is a form of elitism in ACO algorithms that has shown to significantly speed up the convergence to the optimal solution. A second important difference is that the global pheromone update is only performed for the (i, j)-pairs that are an element of the global-best solution. This means that not all pheromone levels are evaporated, as with the AS, but only those that also receive a pheromone deposit. Furthermore, the pheromone deposit is weighted

[3]In ACO literature, this is usually called $\mathcal{S}_{\mathrm{iter}}$, where an iteration has the same meaning as what we call a trial. We prefer to use the word trial in order to stress the similarity between our ACO algorithm for optimal control and reinforcement learning (Sutton and Barto, 1998).

by ρ. As a result of this and the previous two differences, the global pheromone update rule is:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}(s_{\mathrm{gb}}), & \text{if } (i,j) \in s_{\mathrm{gb}} \\ \tau_{ij}, & \text{otherwise.} \end{cases} \tag{2.5}$$

An important element from the ACS algorithm that acts as a measure to avoid premature convergence to suboptimal solutions is the local pheromone update step, which occurs for each ant after each step within a trial and is defined as follows:

$$\tau_{ij} \leftarrow (1-\gamma)\tau_{ij} + \gamma\tau_0, \tag{2.6}$$

where $\gamma \in (0,1)$ is the local pheromone update rate, similar to ρ (the global pheromone update rule), ij is the index corresponding to the (i,j)-pair just added to the partial solution, and τ_0 is the initial value of the pheromone trail. The effect of (2.6) is that during the trial, the visited solution components are made less attractive for other ants to visit, in that way promoting the exploration of other, less frequently visited, solution components. The final important difference compared to the AS is that there is an explicit exploration-exploitation step with the selection of the next vertex j, where with a probability of ϵ, j is chosen as being the vertex with the highest value of $\tau_{ij}^{\alpha}\eta_{ij}^{\beta}$ (exploitation) and with the probability $1-\epsilon$, a random action is chosen according to (2.2) (exploration).

2.6 Convergence Results

The main results on proving convergence of certain ACO algorithms are published by Stützle and Dorigo (2002). In this work, the convergence of the $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$class of problems is proven. The $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$class consists of the ACO algorithms in which the global-best pheromone update rule is used and in which there is a lower bound on the pheromone levels $\tau_{\min}$. The algorithms ACS and Max-Min Ant System both belong to this class and as such the convergence of these algorithms has been proven. We present this proof quite extensively in this section, such that our convergence analysis of ACL in Section 3.4 can be compared to it.

The proof starts with Proposition 2.2 that says that in the $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$class of ACO algorithms the pheromone levels are bounded. By definition of the class, the lower bound is $\tau_{\min}$, which is usually equal to the initial value of the pheromone levels τ_0. This lower bound can always be enforced by replacing pheromone levels smaller than τ_0 by τ_0. In the following, we assume as AS-like algorithm, in which the lower bound on τ_0 is enforced in this way and the global-best update rule is used. It can be shown that the ACS and the Max-Min Ant System are special cases of this algorithm. Proposition 2.2 shows that these algorithms have a finite upper bound $\tau_{\max}$.

Assumption 2.1 *The initial pheromone level must be smaller than the fitness of any possible solution:*

$$\tau_0 < F(s), \quad \forall s$$

When using the global-best update rule, the value of τ_0 that satisfies this assumption can be determined by first generating the solutions by the ants (in the first trial) and then

determining the best solution and its corresponding fitness. In the future trials, the solution used in the update is never worse than this solution and τ_0 can be safely chosen just a little smaller than this fitness.

Proposition 2.2 *In* $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$*, the following holds for any* (i,j)*:*

$$\tau_{ij}(k) \leq \tau_{\max}, \quad \forall k, \text{ with } \tau_{\max} = \frac{1}{\rho}F(s^*),$$

where k *is the trial counter and* $F(s^*)$ *is the fitness of the optimal solution* s^**.*

Proof According to the global pheromone update rule in the AS algorithm (2.3)-(2.4), the update for the pheromone level on an edge $(i,j) \in s(\kappa)$, with $s(\kappa)$ the *only* solution used in the update and κ counting the number of trials in which (i,j) has been visited, is:

$$\tau_{ij}(\kappa+1) = (1-\rho)\tau_{ij}(\kappa) + F(s(\kappa)).$$

The solution of this linear difference equation can directly be written as:

$$\tau_{ij}(\kappa) = (1-\rho)^{\kappa}\tau_0 + \sum_{l=0}^{\kappa-1}(1-\rho)^l F(s(\kappa)). \tag{2.7}$$

Since $(1-\rho)^{\kappa}\tau_0 \leq (1-\rho)^{\kappa}F(s)$, for all s, and $\rho \in (0,1)$, we can derive that $\tau_{ij}(\kappa) \leq \sum_{l=0}^{\kappa}(1-\rho)^l F(s)$. Furthermore, since $(1-\rho)^l \geq 0, \forall l$, and $F(s) > 0, \forall s$, we know that τ_{ij} is non-decreasing with an upper bound of:

$$\tau_{ij}(\kappa) \leq \sum_{l=0}^{\kappa}(1-\rho)^l F(s) \leq \sum_{l=0}^{\infty}(1-\rho)^l F(s^*) = \frac{1}{\rho}F(s^*) = \tau_{\max}.$$

Since (i,j) pairs that are not visited in a particular trial k either are annealed or left unchanged, we have:

$$\tau_{ij}(k) \leq \tau_{\max}, \quad \forall k, \text{ with } \tau_{\max} = \frac{1}{\rho}F(s^*).$$

■

Now we know there exists a finite $\tau_{\max}$, Stützle and Dorigo (2002) show that the values of τ^*_{ij}, with $(i,j) \in s^*$, which are the pheromone levels that are part of the optimal solution for the $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$class of algorithms converge to $\tau_{\max}$:

Proposition 2.3 *In* $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$*, the pheromone levels along the optimal solution converge to* $\tau_{\max}$*:*

$$\lim_{k\to\infty} \tau^*_{ij}(k) = \tau_{\max} = \frac{1}{\rho}F(s^*), \quad \forall (i,j) \in s^*,$$

where s^* *is the optimal solution and* τ^*_{ij} *are the pheromone levels associated with the components in* s^**.*

Proof The proof is in analogy to Proposition 2.2, with the reasoning that because of using the global-best update rule, the (i, j) pairs belonging to the optimal solution keep on receiving the maximal amount of pheromone deposit $F(s^*)$, thereby converging to $\tau_{\max}$. ■

It must be noted that with the global pheromone update rule of the ACS from (2.4)-(2.5), the pheromone update is weighted by ρ. Therefore, with the ACS, the upper bound is $\tau_{\max} = F(s^*)$.

The convergence of the optimal pheromone levels to $\tau_{\max}$ does not prove the convergence of the algorithm, as the optimal solution must be found first. The following theorem from (Stützle and Dorigo, 2002) states that this will eventually happen.

Theorem 2.4 *Let $P^*(k)$ be the probability that the algorithm finds an optimal solution at least once within the first k trials. Then, for an arbitrary choice of $\epsilon \in (0, 1]$, there exist an integer K such that:*

$$P^*(k) \geq 1 - \epsilon, \quad \forall k \geq K$$

and asymptotically

$$\lim_{k \to \infty} P^*(k) = 1.$$

Proof Due to the pheromone trail limits $\tau_{\min}$ and $\tau_{\max}$, we can guarantee that any feasible choice for the next vertex on the construction graph is done with a probability $p_{\min} > 0$. A trivial bound for $p_{\min}$ can be given by considering the situation where the optimal transition has an associated pheromone level of $\tau_{\min}$, where all the other (non-optimal) transitions have an associated pheromone level of $\tau_{\max}$. This bound can be given as:

$$p_{\min} \geq \hat{p}_{\min} = \frac{\tau_{\min}^{\alpha}}{(N_C - 1)\tau_{\max}^{\alpha} + \tau_{\min}^{\alpha}},$$

where N_C is the maximum number of vertices that a vertex can be connected to. Then, any generic solution s', including any optimal solution $s^* \in \mathcal{S}^*$, can be generated with a probability $\hat{p} \geq \hat{p}_{\min}^n > 0$, where n is the finite maximum length of a sequence. Because it is enough that one ant finds an optimal solution, a lower bound for $P^*(k)$ is given by:

$$P^*(k) \geq 1 - (1 - \hat{p})^k.$$

For a given value of ϵ, we have:

$$\begin{aligned} & 1 - (1 - \hat{p})^K \geq 1 - \epsilon \\ \Rightarrow & (1 - \hat{p})^K \leq \epsilon \\ \Rightarrow & K \geq \log_{(1-\hat{p})} \epsilon, \end{aligned}$$

which means that K is a finite integer and that an optimal solution will be found in a finite number of trials. We now know that:

$$1 - (1 - \hat{p})^k \leq P^*(k) \leq 1, \quad \forall k.$$

Hence, since $\lim_{k \to \infty} (1 - (1 - \hat{p})^k) = 1$, we can conclude that $\lim_{k \to \infty} P^*(k)$ exists and equals 1. ■

Now the actual convergence of $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$ can be proven as follows (Stützle and Dorigo, 2002):

Theorem 2.5 *Let k^* be the trial when the first optimal solution has been found. Then the following holds:*

$$\tau_{ij}(k) > \tau_{mn}(k), \quad \forall (i,j) \in s^*, \quad \forall (m,n) \in \mathcal{A} \setminus s^*,$$

for all $k > k^* + k_0 = k^* + \lceil (1-\rho)/\rho \rceil$, where $\lceil x \rceil$ denotes rounding x to the nearest integer larger than or equal to x and where $\mathcal{A}$ is the set of all arcs of the construction graph, i.e., all possible pairs of two vertices. The backslash $\setminus$ denotes the set difference operator.

Proof After k_0 trials since the trial in which the first optimal solution was found (i.e., for $k > k^* + k_0$), the pheromone trail on the connections used in the optimal solution is larger than that on any feasible connection. In fact, due to the use of the global-best pheromone update rule, only connections belonging to s^* increase their pheromone trails, while the pheromone trails of all other connections remain the same. The value $k_0 = \lceil (1-\rho)/\rho \rceil$ is derived in (Stützle and Dorigo, 2002) and this derivation will not be repeated here. ■

2.7 Applications of ACO

The Ant System, which is the basic ACO algorithm, and its variants, have successfully been applied to various optimization problems, such as the traveling salesman problem (Dorigo and Stützle, 2004), load balancing (Sim and Sun, 2003), job shop scheduling (Huang and Yang, 2008; Alaykran et al., 2007), optimal path planning for mobile robots (Fan et al., 2003), and routing in telecommunication networks (Wang et al., 2009). An implementation of the ACO concept of pheromone trails for real robotic systems is described by Purnamadjaja and Russell (2005). A survey of industrial applications of ACO is presented by Fox et al. (2007). An early publication of ant-based control is (Schoonderwoerd et al., 1996). In this paper, the authors present a method for achieving load balancing in telecommunication networks, where calls are routed according to the pheromone distribution on the nodes. An overview of ACO and other metaheuristics to stochastic combinatorial optimization problems can be found in (Bianchi et al., 2006).

One of the first real applications of the ACO framework to optimization problems in continuous search spaces is described in (Socha and Blum, 2007) and (Socha and Dorigo, 2008). In (Socha and Blum, 2007), the application is the training of feed-forward neural networks for pattern classification, and their continuous version of ACO shows a performance comparable to gradient-based neural network training algorithms. An earlier application of the ant metaphor to continuous optimization appears in (Bilchev and Parmee, 1995) and more recent work like the Aggregation Pheromones System and the Differential Ant-Stigmergy Algorithm can be found in respectively (Tsutsui et al., 2005) and (Korosec et al., 2007).

2.8 Concluding Remarks

In this chapter, we have presented an introduction to swarm intelligence and in particular to ACO. We have discussed the principles of collective behavior that can lead to the emergence

of swarms. The individuals in the swarm locally sense the environment and make decisions that are aimed at optimizing their personal objectives. The collective behavior of all these individuals however provides benefits for the individuals, such that they seem to act as if they are aware of the global behavior. These principles of self-organization can be applied in engineering sciences, where the decentralized control of autonomous agents can result, if designed well, in collective behavior to complete tasks that could never have been completed by an individual alone. Furthermore, properties inherent to engineered swarms, such as robustness to the failure or removal of individuals, scalability, flexibility, and cost benefits associated to manufacturing large quantities of similar units, provide additional benefits of designing a swarm of small robots as opposed to one large robot. Within the field of optimization, the swarm intelligence algorithms of Particle Swarm Optimization and ACO have become very successful since their introduction in the early nineties of the twentieth century.

The development of ACO has been inspired by the foraging behavior of some species of ants. It is known that ants communicate the quality of found paths to sources of food by depositing a special chemical, called pheromones. The double bridge experiment has resulted in a mathematical model of the decision process of the ants. This model was the basis for the ACO framework that has become very popular for solving combinatorial optimization problems. In this chapter, we have presented the ACO framework, its application domain, and a convergence proof for a certain class of ACO algorithms.

The inspiration from natural ants, as well as the definition of the ACO framework, the description of the AS and the ACS algorithms, and the convergence proof provide the necessary background for the introduction of ACL in the next chapter.

Chapter 3

Ant Colony Learning Framework

This chapter introduces the ant colony learning framework. Ant colony learning is a multi-agent approach for learning control policies. The optimal control setting in which the framework is situated is presented as well as a theoretical analysis of its behavior. This chapter furthermore compares ant colony learning to reinforcement learning and presents an experimental study of its behavior using computer simulations.

3.1 Introduction

In the previous chapter, we have seen that the foraging behavior of ants in a colony has inspired researchers to develop algorithms for solving combinatorial optimization problems. The metaphorical ants turned out to be able to find shortest paths in the construction graph in which the problems are represented. The length of a path in this sense must be understood as a general performance function, that can represent anything and not only shortest routes in the physical meaning. In this chapter, we go one step further in expanding the ant colony metaphor to the field of control policies. We see a control policy as a mapping from observations to actions with the aim of controlling the state of a certain system to a reference. More specifically, we will consider state-feedback control policies, meaning that the observations represent the state of the system, possibly distorted by noise or some other form of uncertainty. Simply put, we will aim to develop a way in which the ants derive the control policy automatically by interacting with the system, i.e., they will learn the control policy. We call the resulting framework Ant Colony Learning (ACL). This name reflects its main purpose, namely that of learning using the ant colony metaphor.

In (Birattari et al., 2002), ACO is linked to optimal control for the first time. Although presenting a formal framework, called *ant programming*, no application, nor study of its performance is presented. After this paper, no more work has been published on the subject. Our method shows similarities with the most notable other learning paradigm: reinforcement learning, with in particular the Q-learning algorithm (Watkins and Dayan, 1992). However, the key difference is that the power of our ant colony-inspired method lies in the cooperation of the agents. By jointly sampling the state space and exchanging information through pheromones, they form a powerful swarm, capable of efficiently deriving the optimal control policy of the system at hand. Earlier work (Gambardella and Dorigo, 1995) introduced the

Ant-Q algorithm, which is the most notable other work relating ACO with Q-learning. However, Ant-Q has been developed for combinatorial optimization problems and not for optimal control problems. Moreover, this algorithm has not been used after the ACS has been shown to be superior.

In this chapter, we present the optimal control context in Section 3.2 and the ACL framework in Section 3.3. We will limit ourselves first to the control of dynamic systems with discrete state spaces, such that we can discuss and study the basics of the framework first, before continuing with more complex systems. This chapter is devoted to laying down a comprehensive foundation of ACL, by analytically studying the behavior of our method in Section 3.4. We also present a comparison of ACL with reinforcement learning methods in Section 3.5. In Section 3.6, we study the behavior of ACL and the effect of its parameters on the learning performance through experiments.

3.2 Optimal Control Setting

Before we can present the ACL algorithm, we must describe the context in which the algorithm is situated. As said in the introduction, ACL aims at automatically deriving optimal control policies from the interaction with the system to be controlled. This section discusses the notions of optimality, control policy, and system. We will limit the discussion to systems with a discrete state space. In Chapter 4, systems with continuous state spaces will be considered.

3.2.1 Optimal Policy Learning Problem

Assume that we have a nonlinear dynamic system, characterized by a discrete-valued state vector $\mathbf{q} = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}^{\mathrm{T}} \in \mathcal{Q}$, with $\mathcal{Q}$ having a finite number of elements. Also assume that the state can be controlled by an input, or action $\mathbf{u} \in \mathcal{U}$ that can only take a finite number of values and that the state can be measured at discrete time steps, with a sampling time T_s with t the discrete time index. The sampled system is denoted as:

$$\mathbf{q}(t+1) \sim \mathbf{p}(\mathbf{q}(t), \mathbf{u}(t)), \tag{3.1}$$

with $\mathbf{p}$ a probability distribution function over the state-action space. The optimal control problem we consider is to control the state of the system from any given initial state $\mathbf{q}(0) = \mathbf{q}_0$ to a desired goal state $\mathbf{q}(t) = \mathbf{q}_\mathrm{g}$ in at most $t \leq T$ steps and in an optimal way, where optimality is defined by minimizing a certain cost function. As an example, take the following quadratic cost function:

$$J(s) = J(\tilde{\mathbf{q}}, \tilde{\mathbf{u}}) = \sum_{t=0}^{T-1} \mathbf{e}^{\mathrm{T}}(t+1)\mathbf{Q}\mathbf{e}(t+1) + \mathbf{u}^{\mathrm{T}}(t)\mathbf{R}\mathbf{u}(t), \tag{3.2}$$

with s the solution found by a given ant, $\tilde{\mathbf{q}} = \mathbf{q}(1), \dots, \mathbf{q}(T)$ and $\tilde{\mathbf{u}} = \mathbf{u}(0), \dots, \mathbf{u}(T-1)$ respectively the sequences of states and actions in that solution, $\mathbf{e}(t+1) = \mathbf{q}(t+1) - \mathbf{q}_\mathrm{g}$ the error at time $t+1$, and $\mathbf{Q}$ and $\mathbf{R}$ positive definite matrices of appropriate dimensions. The problem is to find a nonlinear mapping from the state to the input that, when applied to the system in $\mathbf{q}_0$, results in a sequence of state-action pairs $(\mathbf{u}(0), \mathbf{q}(1))$, $(\mathbf{u}(1), \mathbf{q}(2))$, ...,

$(\mathbf{u}(T-1), \mathbf{q}(T))$ that minimizes this cost function. The resulting nonlinear mapping function $\mathbf{h}$ is called the control policy:

$$\mathbf{u}(t) = \mathbf{h}(\mathbf{q}(t)) \tag{3.3}$$

We make the assumption that $\mathbf{Q}$ and $\mathbf{R}$ are selected in such a way that $\mathbf{q}_\mathrm{g}$ can be reached in at most T time steps. Conversely, we can also assume that we can select a T that satisfies this requirement on the basis of predefined matrices $\mathbf{Q}$ and $\mathbf{R}$. The matrices $\mathbf{Q}$ and $\mathbf{R}$ balance the importance of speed versus the aggressiveness of the controller. This kind of cost function is frequently used in optimal control of linear systems, where the optimal controller minimizing the quadratic cost can be derived as a closed expression after solving the corresponding Riccati equation using the $\mathbf{Q}$ and $\mathbf{R}$ matrices and the matrices of the linear state space description (Åström and Wittenmark, 1990). In our case, we aim at finding control policies for non-linear systems, which in general cannot be derived analytically from the system description and the $\mathbf{Q}$ and $\mathbf{R}$ matrices, or which cannot be analytically derived because the system description is unknown and all we have is a black-box representing the input-output mapping of (3.1). Note that our method is not limited to the use of quadratic cost functions. The cost function must however satisfy the following requirement for any possible solution s:

$$0 < \frac{\tau_0}{\rho} \leq J_\mathrm{max}^{-1} \leq J^{-1}(s) \leq J_\mathrm{min}^{-1},$$

with $J_\mathrm{max} = \max_s J(s)$ and $J_\mathrm{min} = \min_s J(s)$ respectively the largest and smallest possible value of the cost function, τ_0 the initial value of the pheromone levels, and ρ the global pheromone decay rate, as will be explained in Section 3.3. Note that this requirement is not at all restrictive, since adding a constant $\frac{\tau_0}{\rho}$ to the cost function renders this requirement satisfied. Likewise, for a given cost function and ρ, τ_0 can always be chosen such that this requirement is satisfied. Also note that we can trivially extend the optimal control problem that we consider here to include a set of goal states, denoted by $\mathcal{Q}_\mathrm{g}$. In that case, we can include a virtual goal state to which all goal states $\mathbf{q}_\mathrm{g} \in \mathcal{Q}_\mathrm{g}$ lead with probability one and a cost of zero. The virtual goal state is then considered the single goal state in the ACL algorithm.

We will use the inverse of the cost function, since $J^{-1}(s)$ will be used to represent the pheromone deposit of an ant for a solution s. Lower costs will thus correspond to larger pheromone deposits and maximizing the pheromone levels along optimal solutions will correspond to minimizing the cost function.

3.2.2 Markov Decision Processes

The control policy we aim to find with ACL will be a state feedback controller (3.3). This is a reactive policy, meaning that it will define a mapping from states to actions without the need of storing the states (and actions) of previous time steps. This poses the requirement on the system that it can be described by a state-transition mapping for a discrete state $\mathbf{q}$ and an action (or input) $\mathbf{u}$ in discrete time like (3.1). In this case, the system is said to be a Markov Decision Process (MDP) and the probability distribution function $\mathbf{p}$ is said to be the Markov model of the system. Note that finding an optimal control policy for a deterministic MDP is equivalent to finding the optimal sequence of state-action pairs from any given initial state to a certain goal state. The problem of finding optimal combinations of states and actions is a

combinatorial optimization problem. When the state transitions are stochastic, like in (3.1), it is a stochastic combinatorial optimization problem. We define optimality of a control policy with stochastic state transitions as the mapping from states to actions that minimizes the expected value of the cost of the resulting sequence of state-action pairs. As ACO algorithms are especially applicable to (stochastic) combinatorial optimization problems, the application of ACO to deriving control policies is evident.

3.3 Ant Colony Learning for Optimal Control

In ACL, multiple ants operate in parallel in order to find the optimal control policy. In order to model this parallelism, we assume that each ant interacts with its own copy of the system, as depicted in Figure 3.1.

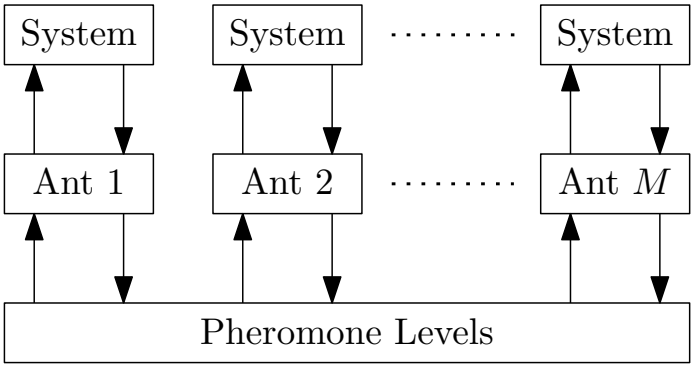


Figure 3.1: The general layout of Ant Colony Learning. Here, there are M ants, each of which interacts with its own copy of the system. The ants also read from and write to pheromone levels, which act as a common memory.

The general outline of ACL is then as follows. Let $\mathcal{C}$ denote the set of ants that have not yet found the goal. Initially, all M ants belong to $\mathcal{C}$ and are distributed randomly over the state space of the system. The pheromone levels associated with each state-action pair $(\mathbf{q}, \mathbf{u})$ are set to an initial value $\tau_{\mathbf{qu}}(0) = \tau_0$, with $0 < \tau_0 \leq \rho J_{\max}^{-1}$. In what is called a *trial*, all ants $c \in \mathcal{C}$ make interaction steps with their copy of the system. First, in its initial state, each ant decides based on the current pheromone levels associated with the available actions which action to perform, after which it applies this action to its own copy of the system. Each ant stores the state-action pair to its personal record, called the partial solution $s_{\mathrm{p},c}$ and the pheromone level at that state-action pair $\tau_{\mathbf{qu}}$ is annealed through the local pheromone update. Each copy of the system responds to the input by changing its state, which takes each ant to its new state. If this state is equal to the goal state $\mathbf{q}_{\mathrm{g}}$, the ant is removed from $\mathcal{C}$. The remaining ants repeat the process by choosing new actions until either all have reach the goal state, or the trial exceeds a predefined number of time steps T. All partial solutions are then added to the multiset $\mathcal{S}_{\mathrm{trial}}$, which is used in the global pheromone update step. It is a multiset, since it may contain the same solution multiple times. In this step, all solutions in the multiset are evaluated with respect to the cost function, and the state-action pairs contained in the solutions receive a pheromone update accordingly.

The following sections describe the main steps of the algorithm, i.e., the action selection, the local pheromone update, the global pheromone update, and the derivation of the control policy from the pheromone levels. It is important to understand that there is an inner loop and an outer loop in the algorithm, as can be clearly seen in Algorithm 3.1 on page 36. The inner loop contains the action selection and the local pheromone update, and the iterations are indexed by t. After at most T iterations, the outer loop takes over, in which the global pheromone update takes place. The iterations in the outer loop are indexed by k. In order to clearly distinguish the pheromone updates in the inner loop from the pheromone updates in the outer loop, the pheromone levels in the inner loop are labeled with the superscript "local": $\tau^{\text{local}}_{\mathbf{qu}}$. Before entering the inner loop, the current pheromone levels thus have to be copied to the local pheromone levels: $\tau^{\text{local}}_{\mathbf{qu}}(0) = \tau_{\mathbf{qu}}(k), \forall(\mathbf{q},\mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$. After the inner loop has been completed and the algorithm enters the outer loop again, the resulting local pheromone levels $\tau^{\text{local}}(T)$ are used for the global pheromone update. Note that although we make this distinction notationally, there is in fact just one set of pheromone levels from which the ants read and to which they write.

3.3.1 Action Selection

The action selection step takes place in the inner loop of the algorithm. In this step, each ant $c \in \mathcal{C}$ determines which action $\mathbf{u}_c \in \mathcal{U}_{\mathbf{q}_c}$ to apply to the system in a given state $\mathbf{q}_c$. With the AS, see (2.2), this is done based on both the pheromone levels and the heuristic variables. In this thesis, we will disregard the heuristic variables, assuming that no information about the quality of solution components is available a priori. This is implemented by setting all heuristic variables equal to one. It can be seen that η_{ij} disappears from (2.2) in this case. Without the heuristic variables, only the value of α remains as a tuning parameter, now in fact only determining the amount of exploration as higher values of α make the probability higher of choosing the action associated with the largest pheromone level. In fact, the balance between exploration and exploitation of the knowledge acquired is very important for the performance of any model-free learning algorithm. Various elaborate ways exist to effectively explore the state-action space of a system, such as directed exploration (Thrun and Möller, 1992) and dynamic exploration (van Ast and Babuška, 2006). In this thesis, however, we limit ourselves to the more straightforward and classical undirected exploration methods.

Note that since the action selection takes place within the inner loop of the algorithm, it is based on $\tau^{\text{local}}_{\mathbf{qu}}(t)$, with t the discrete time index of the current interaction step.

Boltzmann action selection

The AS action selection rule from (2.2) can be ported to the ACL framework as follows:

$$\mathbf{u}_c \sim \mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c\} = \frac{\left(\tau^{\text{local}}_{\mathbf{q}_c\mathbf{u}}(t)\right)^{\alpha}}{\sum_{\boldsymbol{\ell}\in\mathcal{U}_{\mathbf{q}_c}}\left(\tau^{\text{local}}_{\mathbf{q}_c\boldsymbol{\ell}}(t)\right)^{\alpha}}, \quad \mathbf{u} \in \mathcal{U}_{\mathbf{q}_c} \tag{3.4}$$

where $\mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c\}$ is the probability for an ant c to choose action $\mathbf{u}$ in state $\mathbf{q}_c$ and $\mathcal{U}_{\mathbf{q}_c}$ is the action set available to ant c in state $\mathbf{q}_c$. This action selection rule is called the *random proportional*, or *Boltzmann* action selection rule and the amount of exploration is implicit in the choice of α and the pheromone levels. From our experience, good values for α are $2-3$.

Typically, as the pheromone levels converge with the progress of the algorithm, the difference between the pheromone level associated with one particular action and those associated with the other actions increases, and the amount of exploration anneals accordingly.

ϵ-greedy action selection

In the ACS, the amount of exploration is kept constant, due to the inclusion of an explicit exploration probability ϵ. In the ACL framework, this is denoted as follows:

$$\mathbf{u}_c = \begin{cases} \underset{\boldsymbol{\ell} \in \mathcal{U}_{\mathbf{q}_c}}{\arg\max} \left(\tau^{\text{local}}_{\mathbf{q}_c \boldsymbol{\ell}}(t) \right) & \text{with probability } 1 - \epsilon \\ \text{random}(\mathcal{U}_{\mathbf{q}_c}) & \text{with probability } \epsilon, \end{cases} \tag{3.5}$$

where $\text{random}(\mathcal{U}_{\mathbf{q}_c})$ denotes the random selection of an action from the action set in state $\mathbf{q}_c$ from a uniform distribution. This action selection rule is called ϵ-greedy.

Max-Boltzmann action selection

A combination of the two mentioned rules also exists:

$$\mathbf{u}_c \begin{cases} \sim \mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c\} = \frac{\left(\tau^{\text{local}}_{\mathbf{q}_c \mathbf{u}}(t)\right)^{\alpha}}{\sum_{\boldsymbol{\ell} \in \mathcal{U}_{\mathbf{q}_c}} \left(\tau^{\text{local}}_{\mathbf{q}_c \boldsymbol{\ell}}(t)\right)^{\alpha}}, \quad \mathbf{u} \in \mathcal{U}_{\mathbf{q}_c} & \text{with probability } 1 - \epsilon \\ = \text{random}(\mathcal{U}_{\mathbf{q}_c}) & \text{with probability } \epsilon, \end{cases} \tag{3.6}$$

which is called the *Max-Boltzmann* action selection rule.

3.3.2 Local Pheromone Update

The pheromones are initialized equally for all $(\mathbf{q}, \mathbf{u})$-pairs and set to a small positive value τ_0. During each trial, all ants construct their solutions in parallel by interacting with the system until they either have reached the goal state, or the trial exceeds a certain pre-specified number of steps T. After every step, each ant $c \in \mathcal{C}$ performs a local pheromone update for the $(\mathbf{q}_c, \mathbf{u}_c)$-pair just visited, equal to (2.6), but in the setting of ACL:

$$\tau^{\text{local}}_{\mathbf{q}_c \mathbf{u}_c}(t+1) = (1-\gamma)\tau^{\text{local}}_{\mathbf{q}_c \mathbf{u}_c}(t) + \gamma \tau_0, \tag{3.7}$$

with $\gamma \in [0, 1)$ the local pheromone decay rate and typically chosen to be very small. The range of γ is derived in Section 3.4.10. The effect of the local pheromone update is that pheromone levels along visited state-action pairs are decreased towards τ_0. The value of γ should not be chosen too large, as this would mean that the information stored in a pheromone level is lost whenever an ant visits the respective state-action pair. The purpose of the local pheromone update is to stimulate exploration of the state-action space, by making it less attractive for an ant to choose the same action in a certain state as its predecessor. In particular, when the pheromone levels for multiple actions in a given state are all equally large, and larger than other pheromone levels associated with that state, the local pheromone update makes it more likely for several ants visiting that state, to choose different actions. It is a standard step in most modern ACO algorithms (Dorigo and Stützle, 2004).

After the local pheromone update, it is determined if the ants have reached the goal, or if they have timed-out. The ants that have reached the goal are removed from $\mathcal{C}$. When the set $\mathcal{C}$ is empty, or when $t = T$ (whichever happens first), the algorithm continues with the outer loop. All (partial) solutions found are added to the multiset $\mathcal{S}_{\text{trial}}$ and the global pheromone update is performed.

3.3.3 Global Pheromone Update

Let us assume that the trial is completed after T time steps. The pheromone levels are then updated according to the following global pheromone update step:

$$\tau_{\mathbf{qu}}(k+1) = (1-\rho)\tau_{\mathbf{qu}}^{\text{local}}(T) + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}(k): \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \tag{3.8}$$

$$\forall (\mathbf{q}, \mathbf{u}) : \exists s \in \mathcal{S}_{\text{trial}}(k) : (\mathbf{q}, \mathbf{u}) \in s, \tag{3.9}$$

with $\mathcal{S}_{\text{trial}}$ the multiset of all candidate solutions found in the trial, k the trial counter, and $\rho \in (0, 1]$ the global pheromone decay rate. The range of ρ is derived in Section 3.4.10. With the best performing ACO algorithms, as discussed in Section 2.5.1, the global pheromone update is performed for only the solution that is currently the best one found. This solution is called the global-best solution. However, for an optimal control problem, it is not possible to only use the global-best solution in the pheromone update. All ants would have to be initialized to the same state, as starting from states that require less time and less effort to reach the goal would always result in a better global-best solution. Ultimately, initializing an ant exactly in the goal state would be the best possible solution and no other solution, starting from more interesting states would get the opportunity to update the pheromones in the global pheromone update phase. In order to find a control policy from *any* initial state to the goal state, the global-best update rule cannot be used. By using all solutions of all ants in (3.9), the resulting algorithm does allow for random initialization of the ants over the state space and is therefore used in ACL. This type of update rule is comparable to the AS update rule (2.3)-(2.4), with the important difference that only the pheromone levels are evaporated that are associated with the elements in $\mathcal{S}_{\text{trial}}$. The pheromone deposit is equal to $J^{-1}(s) = J^{-1}(\tilde{\mathbf{q}}, \tilde{\mathbf{u}})$, the inverse of the cost function over the sequence of quantized state-action pairs in s according to (3.2). Note that minimizing the cost, corresponds to maximizing the pheromone levels marking the optimal solution.

3.3.4 Control Policy

The control policy can be extracted from the pheromone levels as follows:

$$\mathbf{u} = \mathbf{h}(\mathbf{q}) = \arg\max_{\boldsymbol{\ell} \in \mathcal{U}_{\mathbf{q}}} (\tau_{\mathbf{q}\boldsymbol{\ell}}), \tag{3.10}$$

in which ties are broken randomly. This equations states that the control policy assigns the action to a given state that maximizes the associated pheromone levels.

3.3.5 ACL Algorithm

In Table 3.1, the three most important ACO algorithms and our ACL algorithm are compared based on how they define which solutions are used in the global pheromone update rule, and which global and which local pheromone update rule they use.

Table 3.1: An overview of the characteristics of popular ACO algorithms and ACL.

	AS	ACS	Max-Min AS	ACL
Update (multi)set $\mathcal{S}_{\mathrm{upd}}$	$\mathcal{S}_{\mathrm{trial}}$	$\{s_{\mathrm{gb}}\}$ (also $\{s_{\mathrm{ib}}\}$)	$\{s_{\mathrm{ib}}\}$ (also $\{s_{\mathrm{gb}}\}$)	$\mathcal{S}_{\mathrm{trial}}$
Local pheromone upd.	no	(2.6)	no	(3.7)
Global pheromone upd.	(2.3) - (2.4)	(2.4) - (2.5)	(2.4) - (2.5)	(3.9)

This table shows that ACL uses the same update set as the AS. Elitism introduced in ACS and the Max-Min AS has resulted in outperforming the AS in most combinatorial optimization problems. While these ACO variants are now the methods of choice for current applications of ACO to combinatorial optimization problems, the application of ACL to control policy learning prevents their forms of elitism from being used. Like ACS, the local pheromone update rule is used in ACL, but formulated within in the state-action framework. The effect of the local pheromone update rule will be studied in Section 3.6 and the sections on simulation experiments in the subsequent chapters of this thesis. The global pheromone update rule of ACL is comparable to the ones used in ACS and the Max-Min AS.

The complete algorithm is given in Algorithm 3.1. For the ease of notation, the time indices of $\mathbf{q}$ and $\mathbf{u}$ when used in the subscripts of the pheromone levels are omitted. Also, the distinction between τ^{local} and τ is not made, since no confusion about the order of the updates can arise here. The assignment $\mathbf{q}_c \leftarrow \mathrm{random}(\mathcal{Q})$ in Step 7 selects for ant c a random initial state $\mathbf{q}_c$ from the state space $\mathcal{Q}$ with a uniform probability distribution. In Table 3.2, the ACL parameters are presented.

It is interesting to note that exploration of the state-action space is induced by several elements of the algorithm. First and foremost, the action selection rule contains either an explicit exploration probability (ϵ, in the case of ϵ-greedy action selection), an implicit exploration probability (for larger values of α, state-action pairs with a larger pheromone level are more likely to be selected, in the case of Boltzmann action selection), or a combination of both (in the case of Max-Boltzmann action selection). The local pheromone update rule furthermore decays the pheromone levels of recently selected state-action pairs, stimulating ants to visit other state-action pairs. This can be called a secondary form of exploration, as the actual exploration is still stemming from the action selection rule. This holds true even more for the choice of the initial value of the pheromone levels, τ_0. When chosen very close to zero, a small pheromone deposit in the global pheromone update rule already causes a large difference between pheromone levels, thereby favoring these state-action pairs relatively strongly when using a Boltzmann action selection rule. For larger values of τ_0, the pheromone deposits will cause relatively smaller differences between pheromone levels, thereby maintaining a higher exploration rate when using a Boltzmann action selection rule. The interplay between these parameters and their effect on the exploration behavior has never been studied well. In most application of ACO, standard values for τ_0 and γ are chosen, while tuning the value of ϵ and α to get satisfying performance.

Table 3.2: Overview of the parameters of the ACL algorithm.

	Parameter	Domain	Meaning
Inputs	$\mathcal{Q}$	-	discrete state space
	$\mathcal{U}$	-	discrete action space
	$\mathbf{p}$	-	generative model
Parameters	M	≥ 1	number of ants
	τ_0	> 0	initial pheromone level
	γ	$[0, 1)$	local pheromone decay rate
	ρ	$(0, 1]$	global pheromone decay rate
	α	> 0	exponent of action selection rule
	T	≥ 1	maximal number of discrete time steps
	K	≥ 1	maximal number of trials
Variables	t, k	$\leq T, \leq K$	time index of the system, trial counter
	c	$1, 2, \ldots, M$	an ant
	$\mathcal{C}$	-	set of ants that have not found the goal yet
	$\mathbf{q}, \mathbf{q}_c$	$\mathcal{Q}$	discrete state (observed by ant c)
	$\mathbf{u}, \mathbf{u}_c$	$\mathcal{U}$	discrete action (chosen by ant c)
	$s_{\mathrm{p},c}$	-	partial solution of ant c
	$\mathcal{S}_{\mathrm{trial}}$	-	multiset of solutions
Output	$\tau_{\mathbf{qu}}$	-	pheromone levels for a given $(\mathbf{q}, \mathbf{u})$-pair

3.4 Convergence Analysis

In this section we present a theoretical study on the convergence of ACL. Convergence analysis of ACL is different from the convergence analysis of the $\mathrm{ACO}_{\mathrm{gb},\tau_{\min}}$ class of ACO algorithms, as reviewed in Section 2.6. That convergence proof mainly relied on the global-best update rule, with which the pheromones are only updated if they belong to the best solution found so far. With ACL, all solutions found in a trial are used to update the pheromone levels at the end of that trial. We will see that this severely complicates the convergence analysis. Convergence is a property required for any algorithm in the sense that it must at least theoretically be guaranteed that the algorithm will produce the desired solution. Practically, it may be different. The theoretical analysis may require certain assumptions on the implementation of the algorithm that cannot be met in practice. Or the actual convergence for a real problem may require a huge number of iterations. In Section 3.6, we present an experimental evaluation of the performance of ACL. We also study the behavior of the important ACL parameters, relating them to the theoretical findings in this section.

3.4.1 Definition of Convergence

With ACL we aim to automatically derive a control policy that will bring the system from any initial state to a predetermined desired (goal) state in an optimal fashion. The control policy is a mapping from the state space to the action space. In discrete time, the control policy will lead to a sequence of state-action pairs starting in a given initial state and terminating with the action that brings the state of the system to the desired value. A sequence of state-action

Algorithm 3.1 The Ant Colony Learning algorithm.

Input: $\mathcal{Q}, \mathcal{U}, \mathbf{p}, M, \tau_0, \gamma, \rho, \alpha, T, K$

Initialize the algorithm:
$k \leftarrow 0; \tau_{\mathbf{qu}}(0) \leftarrow \tau_0, \quad \forall (\mathbf{q}, \mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$
repeat
  Initialize the trial:
  $t \leftarrow 0; \mathcal{S}_{\text{trial}} \leftarrow \emptyset; \mathcal{C} \leftarrow \{1, 2, \ldots, M\}$
  for all ants $c \in \mathcal{C}$ in parallel **do**
    Initialize the partial solution:
    $s_{\text{p},c} \leftarrow \emptyset$
    Initialize the state of the system:
    $\mathbf{q}_c(0) \leftarrow \text{random}(\mathcal{Q})$
    repeat
      Select action:
      $\mathbf{u}_c(t) \sim \mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c(t)\} = \frac{\tau^{\alpha}_{\mathbf{q}_c\mathbf{u}}}{\sum_{\boldsymbol{\ell} \in \mathcal{U}_{\mathbf{q}_c}} \tau^{\alpha}_{\mathbf{q}_c\boldsymbol{\ell}}}, \quad \mathbf{u} \in \mathcal{U}_{\mathbf{q}_c}$
      Update partial solution:
      $s_{\text{p},c} \leftarrow s_{\text{p},c} \cup \{(\mathbf{q}_c(t), \mathbf{u}_c(t))\}$
      Apply action to system:
      $\mathbf{q}_c(t+1) \sim \mathbf{p}(\mathbf{q}_c(t), \mathbf{u}_c(t))$
      Perform the local pheromone update:
      $\tau_{\mathbf{q}_c\mathbf{u}_c} \leftarrow (1-\gamma)\tau_{\mathbf{q}_c\mathbf{u}_c} + \gamma\tau_0$
      if $\mathbf{q}_c(t+1) = \mathbf{q}_{\text{g}}$ **then**
        If an ant reaches the goal, its trial terminates:
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$
      end if
      $t \leftarrow t+1$
    until $t = T$ **or** $\mathcal{C} = \emptyset$
    Add the solutions found to the solution multiset:
    $\mathcal{S}_{\text{trial}} \leftarrow \mathcal{S}_{\text{trial}} \cup \{s_{\text{p},c}\}, \quad \forall c \in \{1, 2, \ldots, M\}$
  end for
  Perform the global pheromone update:
  $\tau_{\mathbf{qu}} \leftarrow (1-\rho)\tau_{\mathbf{qu}} + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}: \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \quad \forall (\mathbf{q}, \mathbf{u}) : \exists s \in \mathcal{S}_{\text{trial}} : (\mathbf{q}, \mathbf{u}) \in s$
  $k \leftarrow k+1$
until $k = K$

Output: $\tau_{\mathbf{qu}}, \quad \forall (\mathbf{q}, \mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$

pairs is called a solution. A predefined cost function evaluates solutions and a solution is then called optimal if it has the lowest cost compared to the cost of all possible solution trajectories from the initial state to the goal state. A policy is called optimal if for all states in the state space the solutions are optimal.

We will say that ACL converges, if in the limit, when the number of trials in the algorithm approaches infinity, the resulting control policy becomes and stays equal to the optimal control policy for the considered system, state and action space partitioning, goal state, and

cost function. Note that for the convergence analysis in this section we consider the ACL algorithm as presented in this chapter, i.e., in the setting of discrete state spaces.

3.4.2 Assumptions

For the convergence analysis in this chapter, we require the following assumptions on the ACL algorithm:

1. For any initial state $\mathbf{q}_0$ there is exactly one optimal solution to the goal state $\mathbf{q}_\mathrm{g}$.
2. No loops are allowed in the solutions generated by the ants.
3. The state space is discrete and all state transitions are deterministic.

For some control problems, it may be that for certain initial states there are multiple solutions leading to the goal state that are all optimal. In such cases the first assumption can be satisfied by adding a small gradient to the objective function. With respect to the second assumption, as ACL contains a local pheromone update step, it is not sufficient to remove possible loops after the ants have converged. In that case, the local pheromone update would already have been performed twice or more for the pheromone variables associated with the state-action pairs on the crossings of the loops. Also the pheromone levels on the state-action pairs of the removed loop would have received a local pheromone update once or more incorrectly. The correct way of ensuring that the second assumption is satisfied, is by only allowing an ant to choose an action from a given state if it has not chosen this action from this state before in the current trial, thus by maintaining a tabu-list per state and per ant. The last assumption is merely a repetition of the context in which we consider the algorithm in this chapter. It requires the state space to be originally discrete (such as with grid search problems) without noise in the state transitions and state measurements.

3.4.3 Total Pheromone Update

There are two types of pheromone update in the algorithm: the *local pheromone update* and the *global pheromone update*. Summarizing Section 3.3, the local pheromone update is applied within the inner loop of the algorithm to $\tau^{\mathrm{local}}_{\mathbf{qu}}$ directly after an ant visits $(\mathbf{q}, \mathbf{u})$ and is defined as:

$$\tau^{\mathrm{local}}_{\mathbf{qu}}(t+1) = (1-\gamma)\tau^{\mathrm{local}}_{\mathbf{qu}}(t) + \gamma\tau_0, \tag{3.11}$$

with $\gamma \in [0, 1)$ the local pheromone decay rate and τ_0 the initial value of the pheromone levels. At the start of a trial k, the current pheromone levels are copied to the local pheromone levels, $\tau^{\mathrm{local}}_{\mathbf{qu}}(0) = \tau_{\mathbf{qu}}(k)$ for all $(\mathbf{q}, \mathbf{u})$-pairs. In the first trial, when $\tau^{\mathrm{local}}_{\mathbf{qu}}(0) = \tau_{\mathbf{qu}}(0) = \tau_0$ for all $(\mathbf{q}, \mathbf{u})$-pairs, the local pheromone update has no effect. Only after some pheromone variables have received a pheromone deposit from the global pheromone update, these pheromone levels can become larger than τ_0. Let us assume that the trial is ended when $t = T$ and that in that trial $M_{\mathbf{qu}}(k)$ ants have visited the $(\mathbf{q}, \mathbf{u})$-pair. With (4.17) it is shown that the pheromone levels have then been updated according to:

$$\tau^{\mathrm{local}}_{\mathbf{qu}}(T) = (1-\gamma)^{M_{\mathbf{qu}}(k)}(\tau^{\mathrm{local}}_{\mathbf{qu}}(0) - \tau_0) + \tau_0 \tag{3.12}$$

$$= (1-\gamma)^{M_{\mathbf{qu}}(k)}(\tau_{\mathbf{qu}}(k) - \tau_0) + \tau_0. \tag{3.13}$$

The global pheromone update is applied to $\tau^{\text{local}}_{\mathbf{qu}}(T)$ at the end of a trial, if $(\mathbf{q}, \mathbf{u})$ is an element of the solution of one or more ants. The global pheromone update is then defined as follows:

$$\tau_{\mathbf{qu}}(k+1) = (1-\rho)\tau^{\text{local}}_{\mathbf{qu}}(T) + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}(k): \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \tag{3.14}$$

$$\forall (\mathbf{q}, \mathbf{u}) : \exists s \in \mathcal{S}_{\text{trial}}(k) : (\mathbf{q}, \mathbf{u}) \in s, \tag{3.15}$$

with k the trial counter, $\rho \in (0, 1]$ the global pheromone decay rate, $\mathcal{S}_{\text{trial}}(k)$ the multiset of solutions found in the current trial, and $J^{-1}(s)$ the inverse of the cost of the solution s. We also call $J^{-1}(s)$ the pheromone deposit based on s.

We can aggregate (3.13) and (3.15) to get an expression from the pheromone update at the end of a trial as follows:

$$\begin{aligned} \tau_{\mathbf{qu}}(k+1) = (1-\rho) \left\{ (1-\gamma)^{M_{\mathbf{qu}}(k)} (\tau_{\mathbf{qu}}(k) - \tau_0) + \tau_0 \right\} \\ + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}(k): \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \quad M_{\mathbf{qu}}(k) > 0, \end{aligned} \tag{3.16}$$

$$\tau_{\mathbf{qu}}(k+1) = \tau_{\mathbf{qu}}(k), \qquad \text{otherwise}, \tag{3.17}$$

with $M_{\mathbf{qu}}(k)$ the number of ants that have visited $(\mathbf{q}, \mathbf{u})$ during trial k. From (3.16)-(3.17) we can derive two important conclusions:

1. If a $(\mathbf{q}, \mathbf{u})$ *is not* visited by any of the ants in a given trial, the pheromone level $\tau_{\mathbf{qu}}$ will not be updated in that trial.

2. If a $(\mathbf{q}, \mathbf{u})$ *is* visited by one or more ants in a given trial, the pheromone level $\tau_{\mathbf{qu}}$ will be updated in that trial, and may increase or decrease in value depending on the pheromone deposits of the ants and the values of γ and ρ.

By introducing κ as the counter of the number of trials in which the pheromone level of a particular state-action pair $(\mathbf{q}, \mathbf{u})$ receives a global pheromone update, we can write (3.16) and (3.17) as follows:

$$\begin{aligned} \tau^{\text{upd}}_{\mathbf{qu}}(\kappa+1) = (1-\rho) \left\{ (1-\gamma)^{M_{\mathbf{qu}}(\kappa)} (\tau^{\text{upd}}_{\mathbf{qu}}(\kappa) - \tau_0) + \tau_0 \right\} \\ + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}(\kappa); \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \end{aligned} \tag{3.18}$$

in which the superscript "upd" is used to avoid confusion between pheromone updates indexed with k and those with κ. With this expression, we have effectively got rid of (3.17). Next, we show that since the probability of any state-action pair being visited by at least one ant in any given trial is bounded from below by a non-zero probability, we are allowed to use (3.18), instead of the joint equations (3.16)-(3.17) in the remainder of the convergence analysis in this section.

3.4.4 Lower Bound on the State-Action Selection Probability

The analysis in this section focuses on the pheromone levels associated with a general state $\mathbf{q}$. In $\mathbf{q}$ the possible actions are the elements of the set $\mathcal{U}_{\mathbf{q}} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$. An action will be selected according to the ϵ-greedy action selection rule (3.5), where the currently optimal action is chosen with a probability $1 - \epsilon$, while with a probability ϵ the action is randomly drawn from a uniform distribution over the action set $\mathcal{U}_{\mathbf{q}}$. The parameter ϵ is called the exploration probability and its value determines the trade-off known as the exploration-exploitation dilemma. From this, we can state the following proposition:

Proposition 3.1 (Lower bound on the probability of choosing an action) *The probability of a given action $\mathbf{u}_i$ being selected in any state $\mathbf{q}$ in any trial using the ϵ-greedy action selection rule is at least $\frac{\epsilon}{N}$ provided that $\mathbf{q}$ is visited by at least one ant.*

Proof In the worst case, an action $\mathbf{u}_i$ is only chosen through exploration and only by one ant. With ϵ the exploration probability and N number of possible actions, the probability that any given action is chosen is thus at least $\frac{\epsilon}{N}$. ■

In other words, $\mathbf{u}_i$ will be chosen on average at least every $\frac{N}{\epsilon}$ trials. It is important to note that this average is independent of the number of trials, unlike for instance the random proportional action selection rule from (3.4), where the probability of choosing an action depends on the pheromone levels, which may change from trial to trial. However, in Section 3.4.10, we will show that also with that action selection rule, each action in any given state will be chosen at least once every finite number of trials.

It is important, though, that each state is visited frequently enough. Since the ants are randomly initialized over the state space with a uniform distribution at the start of each trial, we can state the following proposition:

Proposition 3.2 (Lower bound on the probability of visiting a state) *The probability of a given state $\mathbf{q}$ being visited in a given trial is at least $\frac{1}{|\mathcal{Q}|}$, with $|\mathcal{Q}|$ the number of states in the state space $\mathcal{Q}$.*

Proof In the worst case, a state $\mathbf{q}$ is only selected during the initialization of a trial and only by one ant. With $|\mathcal{Q}|$ the number of states in the state space $\mathcal{Q}$, the probability that any given state is selected is thus at least $\frac{1}{|\mathcal{Q}|}$. ■

From these two propositions, the following directly follows:

Corollary 3.3 (Lower bound on the probability of a given state-action pair being selected) *Each state-action pair is selected in a given trial with a probability of at least $\frac{\epsilon}{N|\mathcal{Q}|}$.*

Proof Since the probability of choosing an action and visiting a state are independent from each other, the probability of selecting a state-action pair in any given trial is at least $\frac{\epsilon}{N|\mathcal{Q}|}$.

Because of this corollary, each state-action pair will be visited on average at least every $\frac{N|\mathcal{Q}|}{\epsilon}$ trials, which is constant during the execution of the algorithm. When $k \to \infty$, the results that will be derived for (3.18) will thus also apply to (3.16) - (3.17).

3.4.5 Bounds on the Pheromone Levels

We will proceed our analysis by deriving lower and upper bounds for the pheromone levels. Starting with the lower bound, we prove the following proposition:

Proposition 3.4 (Lower bound on pheromone levels) *If τ_0 is chosen such that $0 < \tau_0 \leq J_{\max}^{-1}\rho$, with $J_{\max}$ the largest possible value of the performance function, the lower bound on the pheromone levels is τ_0.*

Proof By induction, we can show that a pheromone level can never become smaller than τ_0 when using the pheromone update expression from (3.18):

$$\begin{aligned}
\tau_{\mathbf{qu}}^{\mathrm{upd}}(0) &= \tau_0 \\
\tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa+1) &= (1-\rho)\left\{(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}(\tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa)-\tau_0)+\tau_0\right\} + \rho \sum_{\substack{s\in\mathcal{S}_{\mathrm{trial}}(\kappa):\\ (\mathbf{q},\mathbf{u})\in s}} J^{-1}(s) \\
&\geq (1-\rho)\left\{\underbrace{(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}(\tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa)-\tau_0)}_{\geq 0 \quad \text{by induction}}+\tau_0\right\} + \rho M_{\mathbf{qu}}(\kappa) J_{\max}^{-1} \\
&\geq (1-\rho)\tau_0 + \rho J_{\max}^{-1} \\
&\geq (1-\rho)\tau_0 + \tau_0 \geq \tau_0,
\end{aligned} \tag{3.19}$$

which corresponds to the case when only one ant visits this state-action pair and finds the worst possible solution when the associated pheromone level was already at its lowest possible value. Here we have used that $J(s) \leq J_{\max}$ and thus that $J^{-1}(s) \geq J_{\max}^{-1}$, for all s. ■

This means that ACL belongs to the class of ACO algorithms that have a lower bound on the pheromone levels, $\mathrm{ACO}_{\tau_{\min}}$, with $\tau_{\min} = \tau_0$. In order to derive the upper bound on the pheromone levels, we use the following two lemmas:

Lemma 3.5 *Given the following first-order scalar difference equation:*

$$y(k+1) = ay(k) + b, \tag{3.20}$$

with $a \in (-1, 1)$, $b \in \mathbb{R}$, and an arbitrary initial point $y(0)$, the final value of the sequence $y(k)$ is:

$$\lim_{k\to\infty} y(k) = \frac{b}{1-a}. \tag{3.21}$$

Proof This follows trivially from the final value theorem of the z-transform (Åström and Wittenmark, 1990). ■

Lemma 3.6 *Given the following first-order scalar difference equation:*

$$y(k+1) = ay(k) + b, \tag{3.22}$$

with $a \in [0, 1)$, $b \in \mathbb{R}$, and an initial point $y(0)$. If $y(0) \leq \frac{b}{1-a}$, then $y(k)$ is non-decreasing.

Proof Using:

$$y(k) = a^k y(0) + (a^{k-1} + \ldots + a + 1)b$$
$$y(k+1) = a^{k+1} y(0) + (a^k + a^{k-1} + \ldots + a + 1)b$$

we can derive the following expression for the difference between two consecutive time steps of this difference equation:

$$y(k+1) - y(k) = (a^{k+1} - a^k)y(0) + a^k b = a^k(b + ay(0) - y(0)).$$

Since $a \in [0, 1)$, we have $a^k \geq 0$. So, in order to make $y(k)$ non-decreasing, we need that: $(b + ay(0) - y(0)) \geq 0$, which is equal to requiring that $y(0) \leq \frac{b}{1-a}$. ■

Using these two lemmas, we can prove the following proposition:

Proposition 3.7 (Upper bound on pheromone levels) *For any $(\mathbf{q}, \mathbf{u})$-pair, the pheromone levels are bounded from above:*

$$\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) \leq \frac{\beta_{\text{upper}} + \rho M J_{\min}^{-1}}{1 - \alpha_{\text{upper}}}, \tag{3.23}$$

with

$$\alpha_{\text{upper}} = (1-\rho)(1-\gamma)$$
$$\beta_{\text{upper}} = (1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right]$$

and with $J_{\min} = \min_s J(s)$ the smallest possible value of the performance function.

Proof Considering a given $(\mathbf{q}, \mathbf{u})$-pair, let us rewrite (3.18) as follows:

$$\tau_{\mathbf{qu}}^{\text{upd}}(\kappa+1) = (1-\rho)(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + (1-\rho)\left[(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}(-\tau_0) + \tau_0\right] + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}(\kappa): \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s). \tag{3.24}$$

This equation is of the form:

$$\tau_{\mathbf{qu}}^{\text{upd}}(\kappa+1) = \alpha(\kappa)\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta(\kappa) + \delta(\kappa). \tag{3.25}$$

Let us now introduce $\theta_{\mathbf{qu}}(\kappa)$, which satisfies the following difference equation:

$$\theta_{\mathbf{qu}}(\kappa+1) = \alpha_{\text{upper}}\theta_{\mathbf{qu}}(\kappa) + \beta_{\text{upper}} + \delta_{\text{upper}}, \tag{3.26}$$

in which α_{upper}, β_{upper}, and δ_{upper} are upper bounds of $\alpha(\kappa)$, $\beta(\kappa)$, and $\delta(\kappa)$ respectively:

$$\alpha(\kappa) \leq \alpha_{\text{upper}} = (1-\rho)(1-\gamma)$$
$$\beta(\kappa) \leq \beta_{\text{upper}} = (1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right]$$
$$\delta(\kappa) \leq \delta_{\text{upper}} = \rho M J_{\min}^{-1},$$

The upper bound for $\alpha(\kappa)$ is obtained by taking $M_{\mathbf{qu}}(\kappa) = 1$ for all κ, while the upper bounds for $\beta(\kappa)$ and $\delta(\kappa)$ are obtained for $M_{\mathbf{qu}}(\kappa) = M$ for all κ. We take the same initial values for $\tau_{\mathbf{qu}}^{\text{upd}}$ and $\theta_{\mathbf{qu}}$, so $\theta_{\mathbf{qu}}(0) = \tau_0$. Using Lemma 3.6, we can now show that $\theta_{\mathbf{qu}}(\kappa)$ is a non-decreasing function of κ. We immediately see that $\alpha_{\text{upper}} \geq 0$ and we must show that:

$$\begin{aligned}\frac{\beta_{\text{upper}} + \delta_{\text{upper}}}{1 - \alpha_{\text{upper}}} &= \frac{(1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right] + \rho M J_{\text{min}}^{-1}}{1 - (1-\rho)(1-\gamma)} \geq \tau_0 = \theta_{\mathbf{qu}}(0) \\ &\Rightarrow (1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right] + \rho M J_{\text{min}}^{-1} - \tau_0 + (1-\rho)(1-\gamma)\tau_0 \geq 0.\end{aligned}$$

Recalling the ranges for ρ, viz. $(0, 1]$ and for γ, viz. $[0, 1)$, we can show this as follows:

$$\underbrace{(1-\rho)\left[\underbrace{(1-\gamma)^M(-\tau_0) + \tau_0}_{\geq 0}\right]}_{\geq 0} + \underbrace{\underbrace{\rho M J_{\text{min}}^{-1}}_{\geq \tau_0} - \tau_0}_{\geq 0} + \underbrace{(1-\rho)(1-\gamma)\tau_0}_{\geq 0} \geq 0.$$

Moreover, by induction it follows that $\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) \leq \theta_{\mathbf{qu}}(\kappa)$:

$$\begin{aligned}\tau_{\mathbf{qu}}^{\text{upd}}(0) &= \theta_{\mathbf{qu}}(0) \\ \tau_{\mathbf{qu}}^{\text{upd}}(\kappa+1) &= \alpha(\kappa)\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta(\kappa) + \delta(\kappa) \\ &\leq \alpha_{\text{upper}}\theta_{\mathbf{qu}}(\kappa) + \beta_{\text{upper}} + \delta_{\text{upper}} = \theta_{\mathbf{qu}}(\kappa+1).\end{aligned}$$

We can use Lemma 3.5 to get to $\lim_{\kappa\to\infty} \theta_{\mathbf{qu}}(\kappa) = \dfrac{\beta_{\text{upper}} + \delta_{\text{upper}}}{1 - \alpha_{\text{upper}}}$. Since we have shown that $\theta_{\mathbf{qu}}(\kappa)$ is non-decreasing, we have now arrived at the conclusion that:

$$\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) \leq \theta_{\mathbf{qu}}(\kappa) \leq \frac{\beta_{\text{upper}} + \rho M J_{\text{min}}^{-1}}{1 - \alpha_{\text{upper}}}.$$

■

Note that the upper bound that we have derived is only tight for $M = 1$.

3.4.6 Bounds on the Expected Value of the Pheromone Levels

At this point in the analysis of the behavior of the pheromone levels, we have found an expression for the total pheromone update, viz. (3.18), and we have found a lower and an upper bound for the pheromone levels. These bounds are useful as they give us information about the range in which the pheromone levels reside. Moreover, the bounds give the relations between the global and local pheromone decay rates ρ and γ, the number of ants M, and the initial value of the pheromone levels τ_0. However, two shortcomings of these bounds prevent us from drawing conclusions about convergence of the algorithm:

1. Our upper bound is not tight for $M > 1$.

2. Our upper bound is the same for all pheromone levels.

Especially the second issue prevents us from analyzing whether and when the pheromone levels associated with the optimal state-action pairs become larger than the pheromone levels associated with suboptimal state-action pairs. In this section, our aim is to find expressions for the behavior of individual pheromone levels. However, two factors in the algorithm in particular complicate such an analysis:

1. The total pheromone update from (3.18) contains $M_{\mathbf{qu}}(\kappa)$, the number of ants that have visited the $(\mathbf{q}, \mathbf{u})$-pair in trial κ, which is dependent on many uncertain factors, such as the other pheromone levels and the exploration probability.

2. The total pheromone update of $\tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa)$ depends on the pheromone deposits $J^{-1}(s)$ from all solutions found in trial κ. The update of a particular pheromone level thus depends on all state-action pairs prior to $(\mathbf{q}, \mathbf{u})$ and all state-action pairs following $(\mathbf{q}, \mathbf{u})$, which also depends on many uncertain factors, such as the other pheromone levels and the exploration probability.

There are too many uncertainties involved in the algorithm in order to find a closed expression of the final pheromone levels. This is inherent to learning algorithms that contain random variables, such as exploration, and in which the *credit assignment*, such as the distribution of rewards in reinforcement learning, or the distribution of the pheromone deposits in ACL, depends on a series of decision variables.

In the following, we eliminate the uncertainty arising from exploration by looking at the expected value of the pheromone levels. We eliminate the uncertainty arising from the state-action pairs prior to $(\mathbf{q}, \mathbf{u})$ and the state-action pairs following $(\mathbf{q}, \mathbf{u})$ by considering the situation depicted in Figure 3.2.

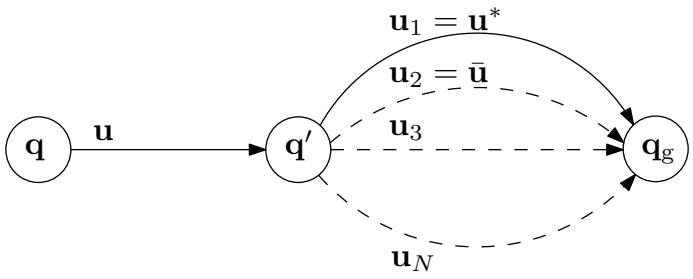


Figure 3.2: From a state $\mathbf{q}$, the action $\mathbf{u}$ will take the system to another state $\mathbf{q}'$, from which there are N possible actions. The action $\mathbf{u}^*$ in this state will bring the system to the goal state optimally. The action $\bar{\mathbf{u}}$ does so sub optimally, but still with a lower cost than the other actions. The other actions do so with decreasing optimality.

Here, we regard the $M_{\mathbf{qu}}(\kappa)$ ants to start in state $\mathbf{q}$ and all choose the action $\mathbf{u}$. All ants are taken to $\mathbf{q}'$ after which they can choose between N possible actions. The action $\mathbf{u}_1 = \mathbf{u}^*$ takes the ants to the goal state immediately and with the lowest cost compared to the other available actions. It is thus considered to be the optimal action. The action $\mathbf{u}_2 = \bar{\mathbf{u}}$ is the second-best action. It takes the ants to the goal state with a higher cost compared to $\mathbf{u}_1$, but with a lower cost compared to all the other possible actions. Let us also assume that the inverse of the cost (i.e., the pheromone deposit) resulting from all possible actions is ranked

as follows:

$$J^{-1}_{\min,\mathbf{q}'} = J^{-1}(\mathbf{q}', \mathbf{u}_1 = \mathbf{u}^*) > J^{-1}(\mathbf{q}', \mathbf{u}_2 = \bar{\mathbf{u}}) = J^{-1}_{\text{second},\mathbf{q}'} \\ > J^{-1}(\mathbf{q}', \mathbf{u}_3) = J^{-1}_{\text{third},\mathbf{q}'} > \ldots > J^{-1}(\mathbf{q}', \mathbf{u}_N) = J^{-1}_{\max,\mathbf{q}'}$$

Here $J^{-1}_{\mathbf{q}'}$ is a short-hand expression for the cost to result from an action chosen in $\mathbf{q}'$ and possible other state-action pairs following $\mathbf{q}'$. The subscripts "min", "second", etc. then denote respectively the lowest, or next to lowest possible cost to resulting in this manner. By definition, we thus have $J^{-1}(\mathbf{q}', \mathbf{u}_1 = \mathbf{u}^*) = J^{-1}_{\min,\mathbf{q}'}$ and $J^{-1}(\mathbf{q}', \mathbf{u}_N) = J^{-1}_{\max,\mathbf{q}'}$.

We will analyze the behavior of the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$. Since the cost resulting from the state-action pair $(\mathbf{q}, \mathbf{u})$ is independent from the action chosen in $\mathbf{q}'$, any constant cost for $J^{-1}(\mathbf{q}, \mathbf{u})$ will do for our analysis. Without the loss of generality and for the sake of simplicity, we take $J^{-1}(\mathbf{q}, \mathbf{u}) = 0$, although formally this is impossible, since $J^{-1}(\mathbf{q}, \mathbf{u}) \geq J^{-1}_{\max} > 0$ for any $(\mathbf{q}, \mathbf{u})$-pair. Note that it thus also holds that $J^{-1}_{\min,\mathbf{q}} = J^{-1}_{\min,\mathbf{q}'}$, $J^{-1}_{\text{second},\mathbf{q}} = J^{-1}_{\text{second},\mathbf{q}'}$, etc. We assume in this section that the optimal action $\mathbf{u}^*$ from $\mathbf{q}'$ is currently also associated with the highest pheromone level and is thus also designated to be optimal. During learning, this does not have to be the case, since other actions may be associated with higher pheromone levels and $\mathbf{u}^*$ is thus not yet known to be the optimal action. This situation will be further discussed in Section 3.4.9.

Computing the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$ involves taking the expected value of the number of ants $M_{\mathbf{qu}}(\kappa)$ in the exponent, which severely complicates deriving an analytical expression for the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$. We must thus shift our aim once again by choosing to derive upper and lower bounds on the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$ instead. At this point, it is the best we can achieve.

Proposition 3.8 (Upper bound on the expected value of pheromone levels) *For any state-action pair $(\mathbf{q}, \mathbf{u})$, the expected value of the pheromone levels is bounded from above:*

$$E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq \frac{\beta_{\text{upper}} + \rho M J^{-1}_{\text{exp},\mathbf{q}}}{1 - \alpha_{\text{upper}}}, \tag{3.27}$$

with

$$\alpha_{\text{upper}} = (1-\rho)(1-\gamma), \\ \beta_{\text{upper}} = (1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right], \\ J^{-1}_{\text{exp},\mathbf{q}} = (1-\epsilon)J^{-1}_{\min,\mathbf{q}} + \epsilon J^{-1}_{\text{avg},\mathbf{q}},$$

and $J^{-1}_{\text{avg},\mathbf{q}}$ the inverse of the average cost expected to result when moving from $\mathbf{q}$ to the goal.

Proof When choosing $\mathbf{u}^*$, the pheromone level $\tau^{\text{upd}}_{\mathbf{qu}}$ is increased the most when:

$$\tau^{\text{upd}}_{\mathbf{qu}}(\kappa+1) = \underbrace{(1-\rho)(1-\gamma)}_{\alpha_{\text{upper}}}\tau^{\text{upd}}_{\mathbf{qu}}(\kappa) + \underbrace{(1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right]}_{\beta_{\text{upper}}} + \rho M J^{-1}_{\min,\mathbf{q}}.$$

When choosing $\bar{\mathbf{u}}$, the pheromone level $\tau^{\text{upd}}_{\mathbf{qu}}$ is increased the most when:

$$\tau^{\text{upd}}_{\mathbf{qu}}(\kappa+1) = \underbrace{(1-\rho)(1-\gamma)}_{\alpha_{\text{upper}}}\tau^{\text{upd}}_{\mathbf{qu}}(\kappa) + \underbrace{(1-\rho)\left[(1-\gamma)^M(-\tau_0) + \tau_0\right]}_{\beta_{\text{upper}}} + \rho M J^{-1}_{\text{second},\mathbf{q}}.$$

The values for α_{upper} and β_{upper} have been derived in the proof of Proposition 3.7. Also recall the ranges for ρ, viz. $(0, 1]$ and for γ, viz. $[0, 1)$. The largest increase of $\tau_{\mathbf{qu}}^{\text{upd}}$ for the other actions follows in a similar manner. The probability that the optimal action $\mathbf{u}^*$ is chosen is:

$$p(\mathbf{u}^*) = 1 - \epsilon + \frac{\epsilon}{N} = 1 - \left(\frac{N-1}{N}\right)\epsilon,$$

namely the probability of not exploring plus the probability of selecting that action while exploring (which is uniformly distributed). The probability of choosing any of the other actions is:

$$p(\mathbf{u}_i) = \frac{\epsilon}{N},$$

for $\mathbf{u}_i \neq \mathbf{u}^*$. The expected value of $\tau_{\mathbf{qu}}^{\text{upd}}$ when increasing the most can now be computed as follows:

$$\begin{aligned}
E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa+1)] = &\left[1 - \left(\frac{N-1}{N}\right)\epsilon\right](\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \rho M J_{\text{min},\mathbf{q}}^{-1}) \\
&+ \left[\frac{\epsilon}{N}\right](\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \rho M J_{\text{second},\mathbf{q}}^{-1}) \\
&\vdots \\
&+ \left[\frac{\epsilon}{N}\right](\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \rho M J_{\text{max},\mathbf{q}}^{-1}) \\
=&\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \left[1 - \left(\frac{N-1}{N}\right)\epsilon\right]\rho M J_{\text{min},\mathbf{q}}^{-1} \\
&+ \underbrace{\left[\frac{\epsilon}{N}\right]\rho M J_{\text{second},\mathbf{q}}^{-1} + \ldots + \left[\frac{\epsilon}{N}\right]\rho M J_{\text{max},\mathbf{q}}^{-1}}_{N-1 \text{ terms}} \\
=&\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + [1-\epsilon]\rho M J_{\text{min},\mathbf{q}}^{-1} \\
&+ \left[\frac{\epsilon}{N}\right]\rho M(J_{\text{min},\mathbf{q}}^{-1} + J_{\text{second},\mathbf{q}}^{-1} + \ldots + J_{\text{max},\mathbf{q}}^{-1}) \\
=&\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \rho M[(1-\epsilon)J_{\text{min},\mathbf{q}}^{-1} + \epsilon J_{\text{avg},\mathbf{q}}^{-1}] \\
=&\alpha_{\text{upper}}\tau_{\mathbf{qu}}^{\text{upd}}(\kappa) + \beta_{\text{upper}} + \rho M J_{\text{exp},\mathbf{q}}^{-1},
\end{aligned}$$

where $J_{\text{exp},\mathbf{q}}^{-1} = (1-\epsilon)J_{\text{min},\mathbf{q}}^{-1} + \epsilon J_{\text{avg},\mathbf{q}}^{-1}$ is the expected pheromone deposit on $\tau_{\mathbf{qu}}^{\text{upd}}$. Since $E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)] \geq E[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)]$, for all κ, the following difference equation describes the evolution of the upper bound of the expected value of $\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)$:

$$E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa+1)] = \alpha_{\text{upper}}E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)] + \beta_{\text{upper}} + \rho M J_{\text{exp},\mathbf{q}}^{-1}.$$

Since we can show that $E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(0)] = \tau_0 \leq \frac{\beta_{\text{upper}} + \rho M J_{\text{exp},\mathbf{q}}^{-1}}{1-\alpha_{\text{upper}}}$, we know from Lemma 3.6 that $E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)]$ is non-decreasing. Using Lemma 3.5, we can now conclude that:

$$E[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)] \leq E_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)] \leq \frac{\beta_{\text{upper}} + \rho M J_{\text{exp},\mathbf{q}}^{-1}}{1-\alpha_{\text{upper}}}.$$

■

Note that $J^{-1}_{\text{avg},\mathbf{q}}$ is generally not known, although it might be possible to estimate it. For the convergence analysis in Section 3.4.7 and in Section 3.4.8, it is not necessary to know the exact value of $J^{-1}_{\text{avg},\mathbf{q}}$. Similar to the upper bound, we can derive a lower bound on the expected value of pheromone levels.

Proposition 3.9 (Lower bound on the limit of the expected value of pheromone levels) *For any state-action pair $(\mathbf{q}, \mathbf{u})$, in the limit for $\kappa \to \infty$, the expected value of the pheromone levels is bounded from below:*

$$\lim_{\kappa\to\infty} E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \geq \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}}, \tag{3.28}$$

with

$$\begin{aligned}
\alpha_{\text{lower}} &= (1-\rho)(1-\gamma)^M, \\
\beta_{\text{lower}} &= (1-\rho)\left[(1-\gamma)(-\tau_0)+\tau_0\right], \\
J^{-1}_{\text{exp},\mathbf{q}} &= (1-\epsilon)J^{-1}_{\text{min},\mathbf{q}} + \epsilon J^{-1}_{\text{avg},\mathbf{q}}.
\end{aligned}$$

Proof When choosing $\mathbf{u}^*$, $\tau^{\text{upd}}_{\mathbf{qu}}$ is increased the least when:

$$\tau^{\text{upd}}_{\mathbf{qu}}(\kappa+1) = \underbrace{(1-\rho)(1-\gamma)^M}_{\alpha_{\text{lower}}}\tau^{\text{upd}}_{\mathbf{qu}}(\kappa) + \underbrace{(1-\rho)\left[(1-\gamma)(-\tau_0)+\tau_0\right]}_{\beta_{\text{lower}}} + \rho J^{-1}_{\text{min},\mathbf{q}},$$

since $\alpha(\kappa) = (1-\rho)(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}$ is the smallest for $M_{\mathbf{qu}}(\kappa) = M$ for all κ and $\beta(\kappa) = (1-\rho)\left[(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}(-\tau_0)+\tau_0\right]$ is the smallest for $M_{\mathbf{qu}}(\kappa) = 1$ for all κ, recalling the ranges for ρ, viz. $(0,1]$ and for γ, viz. $[0,1)$. The smallest increase of $\tau^{\text{upd}}_{\mathbf{qu}}$ for the other actions follows in a similar manner. Following similar steps as in the proof of Proposition 3.8, we can derive the following difference equation describing the evolution of the lower bound of the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$:

$$E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa+1)] = \alpha_{\text{lower}} E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] + \beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}},$$

with $E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$, for all κ. Since we can show that $E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(0)] = \tau_0 \leq \frac{\beta_{\text{lower}}+\rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}}$, we know from Lemma 3.6 that $E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ is non-decreasing. Using Lemma 3.5, we can now conclude that:

$$E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}}$$

and that in the limit for $\kappa \to \infty$:

$$\lim_{\kappa\to\infty} E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \geq \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}}.$$

■

The expected value of the pheromone levels lies between these bounds, $E_{\text{lower}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq E_{\text{upper}}[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$, for all κ. In the limit for $\kappa \to \infty$, the expected value of the pheromone levels lies between the derived upper and lower bounds:

$$\frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}} \leq \lim_{\kappa\to\infty} E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] \leq \frac{\beta_{\text{upper}} + \rho M J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{upper}}}.$$

The expressions for $\alpha(\kappa)$, α_{upper}, α_{lower}, $\beta(\kappa)$, β_{upper}, and β_{lower} are presented in Table 3.3. This can be a useful reference when reading the discussion in the subsequent sections.

Table 3.3: The expressions for $\alpha(\kappa)$ and $\beta(\kappa)$ and their upper and lower bounds.

$\alpha(\kappa)$ and its upper and lower bound	$\beta(\kappa)$ and its upper and lower bound
$\alpha(\kappa) = (1-\rho)(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}$	$\beta(\kappa) = (1-\rho)\left[(1-\gamma)^{M_{\mathbf{qu}}(\kappa)}(-\tau_0) + \tau_0\right]$
$\alpha_{\text{upper}} = (1-\rho)(1-\gamma)$	$\beta_{\text{upper}} = (1-\rho)\left[(1-\gamma)^{M}(-\tau_0) + \tau_0\right]$
$\alpha_{\text{lower}} = (1-\rho)(1-\gamma)^{M}$	$\beta_{\text{lower}} = (1-\rho)\left[(1-\gamma)(-\tau_0) + \tau_0\right] = \tau_0\gamma(1-\rho)$

3.4.7 Convergence of the Expected Value of the Pheromone Levels

In the previous section, we have derived upper and lower bounds of the expected value of pheromone levels. The difference between the upper and lower bound for $\kappa \to \infty$ is a measure of the certainty we have about the expected value of the pheromone levels; it tells us to what extent the expected value of the pheromone levels converges. We will call this difference $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = \frac{\beta_{\text{upper}} + \rho M J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{upper}}} - \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}}}{1-\alpha_{\text{lower}}}. \tag{3.29}$$

Figure 3.3 illustrates the evolution of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ and its upper and lower bound. The expected value of the pheromone level may vary between these bounds, depending on the number of ants visiting $(\mathbf{q}, \mathbf{u})$ and the exploration. The bounds are however non-decreasing and converge to the limits as derived in Proposition 3.8 and Proposition 3.9. The difference between the upper and lower bound converges to $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$.

From (3.29) and Table 3.3 we know that $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ depends on the ACL parameters ρ, γ, M, τ_0, ϵ, and the cost function J. In this section, we will study the value of $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ for various settings of these parameters. This enables us to analyze the behavior of the expected value of the pheromone levels in relation to the ACL parameters.

Only one ant: $M = 1$

In this case, $\beta_{\text{upper}} = \beta_{\text{lower}}$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}}$ and thus:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = 0.$$

This means that when there is only one ant, the expected value of a pheromone level is equal to its upper and lower bound. For the case of only one ant, the expected values of the pheromone levels thus converge. The value to which $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ converges is:

$$\lim_{\kappa\to\infty} E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] = \frac{\tau_0\gamma(1-\rho) + \rho J^{-1}_{\text{exp},q}}{1-(1-\rho)(1-\gamma)}.$$

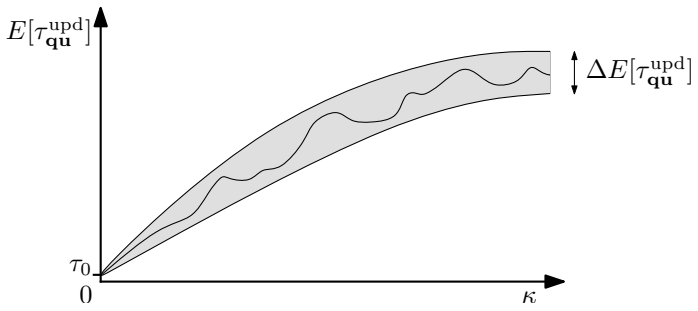


Figure 3.3: Illustration of a possible evolution of the expected value of the pheromone level $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$. The upper and lower bound of this expected value converge. The difference between the converged upper and lower bound, $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ is a measure of the amount of certainty about the expected value of $\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)$ for $\kappa \to \infty$. Note that although κ is a discrete time index, this plot shows a continuous evolution of the expected value of the pheromone levels for the sake of clarity.

When the exploration rate ϵ is decayed after $\tau_{\mathbf{q}'\mathbf{u}^*}$ has become larger than any other pheromone level associated with $\mathbf{q}'$ and any possible action in that state, $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ even converges to the true optimum:

$$\lim_{\kappa\to\infty} E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)] = \frac{\tau_0\gamma(1-\rho) + \rho J^{-1}_{\text{min},q}}{1-(1-\rho)(1-\gamma)}.$$

Recalling the upper bound from (3.23) that we have derived in Proposition 3.7, we can see that $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ thus converges to the true upper bound on the pheromone levels associated with $\mathbf{q}$. We thus conclude that not only $E[\tau^{\text{upd}}_{\mathbf{qu}}]$, but also $\tau^{\text{upd}}_{\mathbf{qu}}$ itself converges to the true optimum.

Annealed exploration: $\epsilon \to 0$

Like just mentioned in the previous paragraph, if the exploration rate is annealed after $\tau_{\mathbf{q}'\mathbf{u}^*}$ has become larger than the pheromone levels associated with the other actions and state $\mathbf{q}'$, it holds that $J^{-1}_{\text{exp},\mathbf{q}} = J^{-1}_{\text{min},\mathbf{q}}$ and thus:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = \frac{\beta_{\text{upper}} + \rho M J^{-1}_{\text{min},\mathbf{q}}}{1-\alpha_{\text{upper}}} - \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{min},\mathbf{q}}}{1-\alpha_{\text{lower}}}.$$

With an annealed exploration rate an $M > 1$, it thus still holds that $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) \geq 0$ and thus that the expected value of the pheromone levels does not necessarily converge.

No local pheromone update: $\gamma = 0$

With the local pheromone decay rate set to zero, we have $\beta_{\text{upper}} = \beta_{\text{lower}} = 0$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 1-\rho$ and thus:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = (M-1)J^{-1}_{\text{exp},\mathbf{q}}.$$

The difference between the upper and lower bound of the expected value of the pheromone levels thus increases proportional to the number of ants and is independent of the global pheromone decay rate. With $\gamma = 0$, the local pheromone update rule is effectively disabled, and the pheromone levels are completely determined by the global pheromone update rule. We will later see that the effect on $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ is the same as when $\rho = 1$. Even when the exploration is annealed and $J^{-1}_{\text{exp},\mathbf{q}} = J^{-1}_{\min,\mathbf{q}}$, $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ does not necessarily converge.

Maximal local pheromone update: $\gamma \to 1$

When the local pheromone decay rate is increased to 1, it follows that $\beta_{\text{upper}} = \beta_{\text{lower}} = (1-\rho)\tau_0$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 0$ and thus:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = \rho(M-1)J^{-1}_{\text{exp},\mathbf{q}}.$$

This differs by a factor ρ from the case of $\gamma = 0$. With a local pheromone decay rate of one, the pheromone levels of visited state-action pairs are effectively reset to τ_0, meaning that with the global pheromone update rule, the pheromone deposit almost completely determines the pheromone levels. Aside from the factor ρ, this has the same effect on the difference between the upper and lower bounds on the expected value of the pheromone levels as with $\gamma = 0$ and as with $\rho = 1$. Likewise, even when the exploration will be annealed, $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ does not necessarily converge.

No global pheromone update: $\rho \to 0$

When the global pheromone decay rate is decreased to 0, we find that $\beta_{\text{upper}} = \tau_0(1-(1-\gamma)^M)$, $\beta_{\text{lower}} = \tau_0\gamma$, $\alpha_{\text{upper}} = 1-\gamma$, $\alpha_{\text{lower}} = (1-\gamma)^M$, and thus:

$$\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = \frac{\tau_0}{\gamma}\left(\frac{(1-(1-\gamma)^M)^2 - \gamma^2}{1-(1-\gamma)^M}\right).$$

For $M = 1$, this reduces to $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = 0$, as we have already seen above. For $M > 1$, we can show that $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) > 0$ as follows:

$$\begin{aligned}
&(1-\gamma)^M < 1-\gamma \\
\Rightarrow &1-(1-\gamma)^M > \gamma \\
\Rightarrow &(1-(1-\gamma)^M)^2 > \gamma^2 \\
\Rightarrow &(1-(1-\gamma)^M)^2 - \gamma^2 > 0 \\
\Rightarrow &\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) > 0.
\end{aligned}$$

For $M \to \infty$, the expression reduces to $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) = \frac{\tau_0}{\gamma}(1-\gamma^2)$, which for small values of γ means that the difference between the upper and lower bounds of the expected value of pheromone levels scales almost inversely proportional with γ. A global pheromone decay rate of zero means that the pheromone levels hardly receive any pheromone deposit, regardless of the cost of the solutions found. For a given γ and annealed ρ, $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ thus does not necessarily converge, but at least $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ is bounded independently of M. When γ would approach 1, then $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}]) \to 0$ and $E[\tau^{\text{upd}}_{\mathbf{qu}}]$ converges. However, since in that case

$\beta_{\text{upper}} = \beta_{\text{lower}} = \tau_0$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 0$, it can be seen that $E[\tau_{\mathbf{qu}}^{\text{upd}}]$ converges to τ_0. Since this means that the expected value of the pheromone levels as well as the pheromone levels themselves all stay at their initial value, convergence in this sense is of course useless.

Maximal global pheromone update: $\rho = 1$

When the global pheromone decay rate is set to one, which means that the pheromone levels at the end of each trial are completely determined by the pheromone deposit in that trial, we have $\beta_{\text{upper}} = \beta_{\text{lower}} = 0$, $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 0$, and thus:

$$\Delta(E[\tau_{\mathbf{qu}}^{\text{upd}}]) = (M-1)J_{\text{exp},\mathbf{q}}^{-1},$$

which is equal to the case when $\gamma = 0$. This is logical as in this case, the effect of the local pheromone update rule is completely nullified at the end of the trial, when the global pheromone update takes place. Likewise, even when the exploration is annealed, $E[\tau_{\mathbf{qu}}^{\text{upd}}]$ does not necessarily converge.

These observations characterize the influence of M, ϵ, γ, and ρ on the expected value of the pheromone levels. Most importantly, they tell us that for $M = 1$, the expected value of the pheromone levels as well as the pheromone levels themselves converge. Moreover, for larger values of M, the expected value of the pheromone levels is increasingly less predictable and thus does not necessarily converge. It is important however to realize that convergence of $E[\tau_{\mathbf{qu}}^{\text{upd}}]$ does not necessarily imply convergence to the optimal policy. Moreover, non-convergence of $E[\tau_{\mathbf{qu}}^{\text{upd}}]$ does not rule out convergence of the policy. However, using the expressions for the upper and lower bounds on the expected value of the pheromone levels, we can analyze the behavior of the policy.

3.4.8 Convergence of the Policy

In Section 3.4.6, it has been assumed that the state-action pair with the highest pheromone level indeed corresponds to the optimal action $\mathbf{u}^*$, leading to an inverse cost $J^{-1}(\mathbf{q}', \mathbf{u}^*) = J_{\text{min},\mathbf{q}'}^{-1}$. However, during learning, it may very well be the case that any other action is associated with the highest pheromone level. Let us assume that in fact the (second-best) action $\bar{\mathbf{u}}$ that leads to a pheromone deposit $J^{-1}(\mathbf{q}', \bar{\mathbf{u}}) = J_{\text{second},\mathbf{q}'}^{-1} < J_{\text{min},\mathbf{q}'}^{-1}$ currently has the highest pheromone level and is thus designated to be optimal, while in fact it is not. In order to study what will happen in that case, we need the lower bound on the expected value of the pheromone level from (3.28) in Proposition 3.9, in which it was assumed that $\mathbf{u}^*$ was already associated with the highest pheromone level. We also need the upper bound on the expected value of the pheromone level from (3.27) in Proposition 3.8, but now for the case when $\bar{\mathbf{u}}$ is assigned the highest pheromone level. This will be denoted by a bar above E_{upper}:

$$\bar{E}_{\text{upper}}[\tau_{\mathbf{qu}}^{\text{upd}}(\kappa)] \leq \frac{\beta_{\text{upper}} + \rho M \overline{J_{\text{exp},\mathbf{q}}^{-1}}}{1 - \alpha_{\text{upper}}}, \tag{3.30}$$

with $\overline{J_{\text{exp},\mathbf{q}}^{-1}} = (1-\epsilon)J_{\text{second},\mathbf{q}}^{-1} + \epsilon J_{\text{avg},\mathbf{q}}^{-1} < J_{\text{exp},\mathbf{q}}^{-1}$. Note that since we have taken $J^{-1}(\mathbf{q}, \mathbf{u}) = 0$, it holds that $J_{\text{exp},\mathbf{q}}^{-1} = J_{\text{exp},\mathbf{q}'}^{-1}$ and $\overline{J_{\text{exp},\mathbf{q}}^{-1}} = \overline{J_{\text{exp},\mathbf{q}'}^{-1}}$.

We will now look at the difference between this upper and the lower bound from (3.28). This difference will be called $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{exp},\mathbf{q}'}}{1-\alpha_{\text{lower}}} - \frac{\beta_{\text{upper}} + \rho M \overline{J^{-1}_{\text{exp},\mathbf{q}'}}}{1-\alpha_{\text{upper}}}. \tag{3.31}$$

Figure 3.4 illustrates a possible evolution of $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}$. When this value is negative, the region of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ when $\tau_{\mathbf{q}'\mathbf{u}^*} > \tau_{\mathbf{q}'\bar{\mathbf{u}}}$ and the region of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ when $\tau_{\mathbf{q}'\mathbf{u}^*} < \tau_{\mathbf{q}'\bar{\mathbf{u}}}$ overlap. This means that it may not be visible in the value of $\tau^{\text{upd}}_{\mathbf{qu}}$ that the action $\mathbf{u}^*$ has become the optimal action in the state $\mathbf{q}'$. Even more importantly, it means that the policy in state $\mathbf{q}'$ does not necessarily converge.

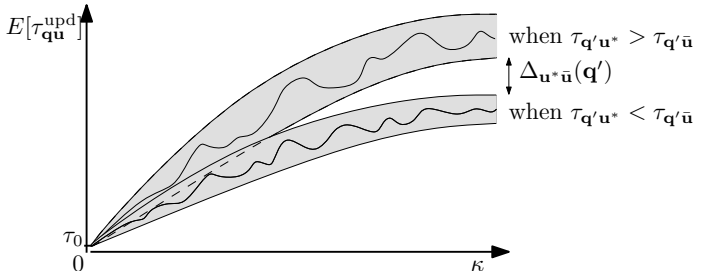


Figure 3.4: Illustration of a possible evolution of the expected value of the pheromone level $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ for the case when $\tau_{\mathbf{q}'\mathbf{u}^*} > \tau_{\mathbf{q}'\bar{\mathbf{u}}}$ and for the case when $\tau_{\mathbf{q}'\mathbf{u}^*} < \tau_{\mathbf{q}'\bar{\mathbf{u}}}$. The difference between the upper bound of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ when $\tau_{\mathbf{q}'\mathbf{u}^*} < \tau_{\mathbf{q}'\bar{\mathbf{u}}}$ and the lower bound of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$ when $\tau_{\mathbf{q}'\mathbf{u}^*} > \tau_{\mathbf{q}'\bar{\mathbf{u}}}$ converges to $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$. This value is an indicator for the convergence of the policy in the state $\mathbf{q}'$. When it is positive, the policy in $\mathbf{q}'$ converges. When it is negative, the policy in $\mathbf{q}'$ does not necessarily converge, but may still do so. Note that although κ is a discrete time index, this plot shows a continuous evolution of the pheromone levels in relation to the ACL parameters.

Like with $\Delta(E[\tau^{\text{upd}}_{\mathbf{qu}}])$ in (3.29), the expression of $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ in (3.31) depends on the ACL parameters ρ, γ, M, τ_0, ϵ, and the cost function J. In this section, we will study the value of $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ for various settings of these parameters. With this, we can analyze the behavior of the policy in $\mathbf{q}'$ in relation to the ACL parameters. Note that we discuss the behavior of the policy in $\mathbf{q}'$ and not the behavior of the policy in $\mathbf{q}$. Since there is only one possible action in $\mathbf{q}$, the policy in that state is trivial and does not change during the course of the algorithm.

Only one ant: $M = 1$

In this case, $\beta_{\text{upper}} = \beta_{\text{lower}}$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}}$ and thus:

$$\begin{aligned}\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') &= \frac{\rho}{1-(1-\rho)(1-\gamma)}(J^{-1}_{\text{exp},\mathbf{q}'} - \overline{J^{-1}_{\text{exp},\mathbf{q}'}}) \\ &= \frac{\rho(1-\epsilon)}{1-(1-\rho)(1-\gamma)}(J^{-1}_{\text{min},\mathbf{q}'} - J^{-1}_{\text{second},\mathbf{q}'}) \geq 0.\end{aligned}$$

This means that when there is only one ant, the difference between the expected value of the pheromone levels associated with the optimal action and the first suboptimal action is larger than zero, when $\epsilon \neq 1$. The expected policy in $\mathbf{q}'$ will thus converge to the optimal policy, even when the exploration is not annealed. As seen in Section 3.4.7, when considering the scenario of having only 1 ant, the policy in $\mathbf{q}'$ is only guaranteed to converged when the exploration rate is annealed "properly".

Annealed exploration: $\epsilon \to 0$

When the exploration rate is annealed "properly", i.e., after $\tau_{\mathbf{q}'\mathbf{u}^*}$ has outgrown the pheromone levels associated with the other actions and state $\mathbf{q}'$, it holds that $J^{-1}_{\text{exp},\mathbf{q}'} = J^{-1}_{\text{min},\mathbf{q}'}$ and $\overline{J^{-1}_{\text{exp},\mathbf{q}'}} = J^{-1}_{\text{second},\mathbf{q}'}$ thus:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = \frac{\beta_{\text{lower}} + \rho J^{-1}_{\text{min},\mathbf{q}'}}{1-\alpha_{\text{lower}}} - \frac{\beta_{\text{upper}} + \rho M J^{-1}_{\text{second},\mathbf{q}'}}{1-\alpha_{\text{upper}}}.$$

For some $M > 1$, depending on the difference between $J^{-1}_{\text{min},\mathbf{q}'}$ and $J^{-1}_{\text{second},\mathbf{q}'}$, $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ becomes smaller than zero. The policy in $\mathbf{q}'$ thus does not necessarily converge.

No local pheromone update: $\gamma = 0$

With the local pheromone decay rate set to zero, we have $\beta_{\text{upper}} = \beta_{\text{lower}} = 0$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 1 - \rho$ and thus:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = J^{-1}_{\text{exp},\mathbf{q}'} - M\overline{J^{-1}_{\text{exp},\mathbf{q}'}}.$$

We have $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') < 0$ for

$$M > \frac{(1-\epsilon)J^{-1}_{\text{min},\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}{(1-\epsilon)J^{-1}_{\text{second},\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}. \tag{3.32}$$

Like the previous considered scenario, with annealed exploration, this means that depending on the difference between $J^{-1}_{\text{min},\mathbf{q}'}$ and $J^{-1}_{\text{second},\mathbf{q}'}$ there exists an $M > 1$ for which the policy in $\mathbf{q}'$ does not necessarily converge.

Maximal local pheromone update: $\gamma \to 1$

When the local pheromone decay rate is increased to 1, it follows that $\beta_{\text{upper}} = \beta_{\text{lower}} = (1-\rho)\tau_0$ and $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 0$ and thus:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = \rho J^{-1}_{\text{exp},\mathbf{q}'} - \rho M\overline{J^{-1}_{\text{exp},\mathbf{q}'}}.$$

Like with $\gamma = 0$, we have for $M > \dfrac{(1-\epsilon)J^{-1}_{\text{min},\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}{(1-\epsilon)J^{-1}_{\text{second},\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}$, it holds that $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') < 0$. As explained in Section 3.4.6, with a local pheromone decay rate of almost one, the pheromone levels of visited state-action pairs are almost completely reset to τ_0, meaning that with the global pheromone update rule, the pheromone deposit almost completely determines

the pheromone levels. This has the same effect on $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ as with $\gamma = 0$ and as with $\rho = 1$. Likewise, even when the exploration will be annealed and depending on the difference between $J^{-1}_{\min,\mathbf{q}'}$ and $J^{-1}_{\text{second},\mathbf{q}'}$, there exists an $M > 1$ for which the policy in $\mathbf{q}'$ does not necessarily converge.

No global pheromone update: $\rho \to 0$

When the global pheromone decay rate is decreased to approach 0, we find that $\beta_{\text{upper}} = \tau_0(1-(1-\gamma)^M)$, $\beta_{\text{lower}} = \tau_0\gamma$, $\alpha_{\text{upper}} = 1-\gamma$, $\alpha_{\text{lower}} = (1-\gamma)^M$, and thus:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = \frac{\tau_0}{\gamma}\left(\frac{\gamma^2 - (1-(1-\gamma)^M)^2}{1-(1-\gamma)^M}\right).$$

For $M = 1$, this reduces to $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = 0$ and for $M > 1$ we can see that $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') < 0$. For $M \to \infty$, the expression reduces to $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = -\frac{\tau_0}{\gamma}(1-\gamma^2)$, which for small values of γ means that $-\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ scales almost inversely proportional with γ. A global pheromone decay rate of almost zero means that the pheromone levels hardly receive any pheromone deposit, regardless of the cost of the solutions found. For a given γ and annealed ρ, $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}')$ is bounded independent of M. When γ would approach 1, then $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') \to 0$, meaning that all pheromone levels will converge to the same value, namely to τ_0. Since this means that the expected value of the pheromone levels as well as the pheromone levels themselves all stay at their initial value, convergence in this sense is of course useless.

Maximal global pheromone update: $\rho = 1$

When the global pheromone decay rate is set to one, which means that the pheromone levels at the end of each trial are completely determined by the pheromone deposit in that trial, we have $\beta_{\text{upper}} = \beta_{\text{lower}} = 0$, $\alpha_{\text{upper}} = \alpha_{\text{lower}} = 0$, and thus:

$$\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') = J^{-1}_{\text{exp},\mathbf{q}'} - M\overline{J^{-1}_{\text{exp},\mathbf{q}'}},$$

so for $M > \dfrac{(1-\epsilon)J^{-1}_{\min,\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}{(1-\epsilon)J^{-1}_{\text{second},\mathbf{q}'} + \epsilon J^{-1}_{\text{avg},\mathbf{q}'}}$, we have $\Delta_{\mathbf{u}^*\bar{\mathbf{u}}}(\mathbf{q}') < 0$, which is equal to the case when $\gamma = 0$. This is logical as in this case, the effect of the local pheromone update rule is completely nullified at the end of the trial, when the global pheromone update takes place. Likewise, even when the exploration will be annealed and depending on the difference between $J^{-1}_{\min,\mathbf{q}'}$ and $J^{-1}_{\text{second},\mathbf{q}'}$, there exists an $M > 1$ for which the policy in $\mathbf{q}'$ does not necessarily converge.

In this section, we have learned about the convergence of the policy in a state $\mathbf{q}'$, as depicted in Figure 3.2. We have shown that the expected policy converges to the optimal policy for $M = 1$. For an increased number of ants, convergence of the expected policy is only guaranteed for the scenarios when $\gamma = 0$, $\gamma \to 1$, or $\rho = 1$ when the difference between the inverse of the cost associated with the optimal action and the inverse of the cost associated with the other possible actions is larger than some quantity, as expressed in (3.32). For other cases, the policy in $\mathbf{q}'$ is not formally guaranteed to converge.

3.4.9 Convergence of the Policy for the Complete State Space

With the analysis in the previous subsections, we have characterized to some extent the behavior of the pheromone levels. Because of the unpredictability of the number of ants that visit a particular state, the best we could do was to derive upper and lower bounds on the expected value of pheromone levels. For $M = 1$ the expected value of a pheromone level $\tau_{\mathbf{qu}}$ converges as the number of visits of $(\mathbf{q}, \mathbf{u})$ increases. For $M > 1$, the expected value of a pheromone level is less predictable, but the size of the range in which it can vary does converge to a constant value larger than zero. In particular, we have studied the situation from Figure 3.2. There, the pheromone level $\tau_{\mathbf{qu}}$ was dependent on the action chosen in the state $\mathbf{q}'$ further "downstream" towards the goal. The goal would be reached after choosing the optimal action $\mathbf{u}^*$ in $\mathbf{q}'$. We have drawn similar conclusions on the convergence of the policy in $\mathbf{q}'$. For $M = 1$ convergence of the expected policy to the optimal policy has been proven. However, for $M > 1$, it was shown that depending on the cost function, the expected policy in $\mathbf{q}'$ does not necessarily converge. Although we have not been able to prove nor disprove convergence for $M > 1$, we will show with the experiments in Section 3.6 and in both Chapter 4 and Chapter 5 that multiple ants speed up the learning of control policies and that ACL converges to good control policies in most cases.

In order to draw conclusions about the possible convergence of the policy over the complete state space, we reason as follows. Consider that all ants are initialized in $\mathbf{q}'$, but that the number of ants may vary from trial to trial. This state is just prior to the goal state when choosing the optimal action $\mathbf{u}^*$. Any ant that chooses $\mathbf{u}^*$ thus contributes the maximal possible pheromone update $J^{-1}_{\min,\mathbf{q}'}$ to the pheromone update. As, according to Proposition 3.1, the probability of choosing any possible action $\mathbf{u}_i$ is non-zero, there will be a time at which $\tau^{\text{upd}}_{\mathbf{q}'\mathbf{u}^*}$ is larger than any other pheromone level associated with $\mathbf{q}'$. Then, $\tau^{\text{upd}}_{\mathbf{qu}}$ will behave according to the discussion in Section 3.4.7 and as illustrated in Figure 3.3. Now, consider that there are in fact multiple actions to choose from in $\mathbf{q}$, but that the action that leads to $\mathbf{q}'$ is the optimal action. If we now look at what happens to the policy in $\mathbf{q}$, the same reasoning will apply to that state as to $\mathbf{q}'$. In this way, we can propagate back the reasoning to all states in the state space: the pheromone levels will rise and fall depending on the number of ants visiting these states, the amount of exploration, the parameter values, and the optimality of the pheromone levels at states closer to the goal. On average, these pheromone levels will only converge if $M = 1$, but also for $M > 1$ there will be a tendency towards higher pheromone levels associated with the optimal actions. Likewise, the expected policy over the complete state space will converge to the optimal policy when $M = 1$, but we have not been able to prove that for $M > 1$ the expected policy will converge to the optimal policy. Neither have we been able to prove the opposite, namely that the expected policy will *not* converge to the optimal policy.

Note that also the state-action pairs visited before coming to a state $\mathbf{q}$ affect the amount of pheromones deposited on $\tau_{\mathbf{qu}}$, but that its effect is only an offset to the behavior of $E[\tau^{\text{upd}}_{\mathbf{qu}}(\kappa)]$. It does therefore not affect the reasoning above.

3.4.10 Remarks

Based on the convergence analysis, we are able to give some extra remarks about the ACL algorithm.

Action selection rule

The convergence analysis still holds when using a different action-selection rule, like (3.4). The reason is that because of the existence of an upper bound as well as a lower bound on the pheromone levels $\tau_0 \leq \tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa) \leq \frac{\beta_{\mathrm{upper}} + \rho M J_{\mathrm{min}}^{-1}}{1 - \alpha_{\mathrm{upper}}}, \forall \kappa$, with $\alpha_{\mathrm{upper}} = (1-\rho)(1-\gamma)$ and $\beta_{\mathrm{upper}} = (1-\rho)[(1-\gamma)^M(-\tau_0) + \tau_0]$, the probability of choosing any action is always nonzero and bounded from below with a fixed lower bound. The average number of trials after which any action has been chosen is thus bounded, which is an essential requirement for the convergence analysis for $\tau_{\mathbf{qu}}^{\mathrm{upd}}(\kappa)$ to apply to $\tau_{\mathbf{qu}}(k)$.

The effect of ϵ

The value of the exploration rate ϵ in (3.5) is important for the speed of convergence. If ϵ is small (meaning low exploration) and $\mathbf{u}^*$ becomes designated to be the optimal action, $\tau_{\mathbf{qu}^*}$ is likely to be quickly reinforced. However, before $\mathbf{u}^*$ becomes the best action, it takes longer before it is selected frequently enough to make $\tau_{\mathbf{qu}^*}$ the largest. For large values of ϵ, the opposite is true. For the value of the exponent α in (3.4) a similar reasoning for its influence on the convergence speed applies.

3.5 Related Methods

ACL is an algorithm for the learning of optimal control policies by interacting with the system at hand, much like Reinforcement Learning (RL) methods do. In RL, there is typically a single agent that receives a reward after each interaction step with the system and its task is to find the optimal control policy that leads to a maximal cumulative reward. Formally, the RL agent aims at maximizing the discounted return, which is defined as the discounted sum of immediate rewards:

$$R(t) = \sum_{i=0}^{T} \gamma_{\mathrm{rl}}^i r(t+i+1), \tag{3.33}$$

with γ_{rl} the discount factor. The return at time t consists of future immediate rewards, which are of course unknown to the agent. During the course of the learning, the agent therefore aims at maximizing its *expected* return by following a certain policy. This is called the value function:

$$V^{\mathbf{h}}(\mathbf{q}) = E^{\mathbf{h}}\{R(t) \mid \mathbf{q}(t) = \mathbf{q}\}, \tag{3.34}$$

which means that the value function in a state $\mathbf{q}$ by following a policy $\mathbf{h}$, is the value of the (discounted) return that is expected to be obtained when starting in that state and following that policy. In a similar way, the state-action value function, or Q-function, is defined as the expected return of taking an action $\mathbf{u}$ in a state $\mathbf{q}$ and following the policy $\mathbf{h}$:

$$Q^{\mathbf{h}}(\mathbf{q}, \mathbf{u}) = E^{\mathbf{h}}\{R(t) \mid \mathbf{q}(t) = \mathbf{q}, \mathbf{u}(t) = \mathbf{u}\}. \tag{3.35}$$

The optimal policy $\mathbf{h}^*$ now is defined as the policy that maximizes these value functions:

$$V^*(\mathbf{q}) = \max_{\mathbf{h}} V^{\mathbf{h}}(\mathbf{q}), \tag{3.36}$$

$$Q^*(\mathbf{q}, \mathbf{u}) = \max_{\mathbf{h}} Q^{\mathbf{h}}(\mathbf{q}, \mathbf{u}). \tag{3.37}$$

RL requires the system to be controlled to be an MDP and techniques for finding the optimal control policy can be divided into the classes *model-free* and *model-based*. In the following two subsections, the most important algorithms from both classes will be described after which in Section 3.5.3, ACL is compared to both methods.

3.5.1 Dynamic Programming

Solution techniques for an MDP that use an exact model of the environment are known as dynamic programming. Dynamic programming is based on the *Bellman equations*, which give recursive relations of the value functions. For state-values this corresponds to

$$\begin{aligned} V^{\mathbf{h}}(\mathbf{q}) &= E^{\mathbf{h}} \left\{ \sum_{i=0}^{\infty} \gamma_{\mathrm{rl}}^i r(t+i+1) \mid \mathbf{q}(t) = \mathbf{q} \right\} \\ &= E^{\mathbf{h}} \left\{ r(t+1) + \gamma_{\mathrm{rl}} \sum_{i=0}^{\infty} \gamma_{\mathrm{rl}}^i r(t+i+2) \mid \mathbf{q}(t) = \mathbf{q}' \right\} \\ &= \sum_{\mathbf{u} \in \mathcal{U}_{\mathbf{q}}} \mathbf{p}^{\mathbf{h}}(\mathbf{u} \mid \mathbf{q}) \sum_{\mathbf{q}' \in \mathcal{Q}} \mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} \left[\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} + \gamma_{\mathrm{rl}} E^{\mathbf{h}} \left\{ \sum_{i=0}^{\infty} \gamma_{\mathrm{rl}}^i r(t+i+2) \mid \mathbf{q}(t) = \mathbf{q}' \right\} \right] \\ &= \sum_{\mathbf{u} \in \mathcal{U}_{\mathbf{q}}} \mathbf{p}^{\mathbf{h}}(\mathbf{u} \mid \mathbf{q}) \sum_{\mathbf{q}' \in \mathcal{Q}} \mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} \left[\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} + \gamma_{\mathrm{rl}} V^{\mathbf{h}}(\mathbf{q}') \right], \end{aligned} \tag{3.38}$$

with $\mathbf{p}^{\mathbf{h}}(\mathbf{u} \mid \mathbf{q})$ the probability of performing action $\mathbf{u}$ in state $\mathbf{q}$ while following policy $\mathbf{h}$, and $\mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$ and $\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$ the state-transition probability function and the expected value of the next immediate reward, respectively.

With (3.38) substituted in (3.36) an expression called the *Bellman optimality equation* can be derived (Sutton and Barto, 1998) for state values as:

$$V^*(\mathbf{q}) = \max_{\mathbf{u}} \sum_{\mathbf{q}' \in \mathcal{Q}} \mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} \left[\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} + \gamma_{\mathrm{rl}} V^*(\mathbf{q}') \right], \tag{3.39}$$

where $\mathbf{q}'$ denotes the next state and for state-action values as:

$$Q^*(\mathbf{q}, \mathbf{u}) = \sum_{\mathbf{q}' \in \mathcal{Q}} \mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} \left[\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} + \gamma_{\mathrm{rl}} \max_{\mathbf{u}' \in \mathcal{U}_{\mathbf{q}}} Q^*(\mathbf{q}', \mathbf{u}') \right]. \tag{3.40}$$

One can solve these equations when the dynamics of the environment ($\mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$ and $\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$) are known. This corresponds to knowing the world model a priori. When either $V^*(\mathbf{q})$ or $Q^*(\mathbf{q}, \mathbf{u})$ has been found, the optimal policy simply consists in taking the action $\mathbf{u}$ in each state $\mathbf{q}$ encountered for which $Q^*(\mathbf{q}, \mathbf{u})$ is the highest or results in the highest expected direct reward plus discounted state value $V^*(\mathbf{q})$. The beauty is that a one-step-ahead search yields

the long-term optimal actions, since $V^*(\mathbf{q}')$ incorporates the expected long-term reward in a local and immediate quantity.

Value Iteration (Sutton and Barto, 1998) uses the Bellman optimality equation from (3.39) as an update rule:

$$V_{i+1}(\mathbf{q}) = \max_{\mathbf{u}} E\left\{r(t+1) + \gamma_{\mathrm{rl}} V_i(\mathbf{q}(t+1)) \mid \mathbf{q}(t) = \mathbf{q}, \mathbf{u}(t) = \mathbf{u}\right\} \tag{3.41}$$

$$= \max_{\mathbf{u}} \sum_{\mathbf{q}' \in \mathcal{Q}} \mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} \left[\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'} + \gamma V_i(\mathbf{q}')\right]. \tag{3.42}$$

These iterations must be performed over all states and the algorithm is stopped when the change in value function is smaller than some specific small positive value.

3.5.2 Q-Learning

In temporal difference learning methods (Sutton and Barto, 1998), the agent processes the immediate rewards it receives at each time step, thereby learning from each action without the need of a model of the system. A learning rate (α_{rl}) determines the importance of the new estimate of the value function over the old estimate. The simplest temporal difference update rule is as follows:

$$V(\mathbf{q}(t)) \leftarrow V(\mathbf{q}(t)) + \alpha_{\mathrm{rl}} \left[r(t+1) + \gamma_{\mathrm{rl}} V(\mathbf{q}(t+1)) - V(\mathbf{q}(t))\right], \tag{3.43}$$

where the term between the square brackets is called the temporal difference error. An algorithm that learns Q-values is Watkins' Q-learning (Watkins and Dayan, 1992). This algorithm is a so called *off-policy* temporal difference control algorithm as it learns the action-values that are not necessarily on the policy that it is following. In its simplest form, *1-step Q-learning* is as follows:

$$Q(\mathbf{q}(t), \mathbf{u}(t)) \leftarrow Q(\mathbf{q}(t), \mathbf{u}(t)) + \alpha_{\mathrm{rl}} \left[r(t+1) + \gamma_{\mathrm{rl}} \max_{\mathbf{u} \in \mathcal{U}_{\mathbf{q}}} Q(\mathbf{q}(t+1), \mathbf{u}) - Q(\mathbf{q}(t), \mathbf{u}(t))\right]. \tag{3.44}$$

Under certain conditions on the state space sampling and the learning rate, V and Q will converge in the limit to the state value function and state-action value function belonging to the optimal policy respectively. With (3.44), $Q(\mathbf{q}(t), \mathbf{u}(t))$ is only updated after receiving the new reward $r(t+1)$. At the next time step, the agent is in the state $\mathbf{q}(t+1)$ and when it performs an action for which it receives an immediate reward $r(t+2)$, this reward in turn is only used to update $Q(\mathbf{q}(t+1), \mathbf{u}(t+1))$. It would be reasonable to also update $Q(\mathbf{q}(t), \mathbf{u}(t))$, since this state-action pair was also responsible for receiving the reward two time steps later. Wisely, states further away in the past should be credited less than states that occurred more recently. How the credit for a reward should best be distributed is called *the credit assignment problem*. A well known method in RL is to use *eligibility traces* (Sutton and Barto, 1998). In Q-learning, such eligibility traces are associated with each state-action pair and indicate for the state-action pairs how eligible they are to be updated with a particular reward. For the off-policy nature of Q-learning, incorporating eligibility traces in Q-learning needs special attention. State-action pairs are only eligible for change when they are followed by greedy actions. Eligibility traces, thus only work up to the moment when an explorative

action is taken. Whenever this happens, the eligibility trace is reset to zero. The algorithm, known as $Q(\lambda)$-learning, is given as:

$$Q(\mathbf{q}, \mathbf{u}) \leftarrow Q(\mathbf{q}, \mathbf{u}) + \alpha_{\mathrm{rl}} \delta e(\mathbf{q}, \mathbf{u}), \quad \forall (\mathbf{q}, \mathbf{u}), \tag{3.45}$$

where

$$\delta = r(t+1) + \gamma_{\mathrm{rl}} \max_{\mathbf{u}' \in \mathcal{U}_{\mathbf{q}}} Q(\mathbf{q}(t+1), \mathbf{u}') - Q(\mathbf{q}(t), \mathbf{u}(t)), \tag{3.46}$$

and where $e(\mathbf{q}, \mathbf{u})$ is the eligibility trace for the respective state-action pair. According to Sutton and Barto (1998), the development of $Q(\lambda)$-learning has been one of the most important breakthroughs in RL.

3.5.3 Relation to ACL

The reinforcement learning algorithms presented above show strong similarities with the procedure of ACL. Both RL and ACL are capable of learning optimal control policies. The temporal-difference update in RL involves the immediate reward obtained after performing an interaction step with the system. In the case of Q-learning, the reward is used to update the state-action value function, which reflects the expected long-term reward (or return) for choosing a particular action in a particular state. The pheromone levels in ACL have a similar function, as they also reflect the long-term quality of choosing a certain action in a certain state. Without eligibility traces, the main difference between the updating in Q-learning and ACL is that in Q-learning, the Q-value for a state-action pair is updated using the immediate reward and the Q-values in the next state, meaning that immediate rewards need to propagate through the state-action space. In ACL, the pheromones are updated with pheromone deposits, which are functions of the solutions found. In that way, the quality of a solution immediately affects all state-action pairs that contributed to it. With eligibility traces, this difference becomes smaller, as an eligibility trace distributes the immediate rewards (to some extent) to the other state-action pairs leading to that immediate reward. In RL, the control objective needs to be translated in a reward function, defining the immediate reward for each state-action pair. This makes it difficult, or even impossible, to define objective functions that take into account performance criteria concerning the behavior of the controlled system, such as rise-time, settling-time, and overshoot. Since in ACL the pheromone deposits are based on complete solutions found, taken into account such performance criteria is possible and much more straightforward.

Aside from the updating, the main difference between RL and ACL is, of course, that instead of a single agent, in ACL there are multiple agents (ants), all interacting with the system in parallel and all contributing to the update of the value function (the set of pheromone levels). Because of the parallelism of the ants, ACL could be called a multi-agent learning algorithm. However, this must not be confused with multi-agent reinforcement learning, which stands for the setting in which multiple reinforcement learning agents all act in a single environment according to their own objective and in order for each of them to behave optimally, they need to take into account the behavior of the other agents in the environment. Note that in ACL, the ants do not need to consider, or even notice, the existence of the other ants. All they do is benefit from each other's findings.

There exist parallel implementations of reinforcement learning. In (Laurent and Piat, 2001), an application of RL to a block-pushing problem is considered. With multiple blocks

the complexity of the problem is large. In order to deal with this complexity, an architecture is used which realizes several learning processes at the same time. Based on an camera image of the state, several RL processes run in parallel, all using the same action-value function for all the objects. A global Q-value function is composed by taking the maximum over the value functions resulting from the parallel processes and this Q-value function is used to control the system and generate new samples. This implementation results in a speed-up of the learning and can be used in on-line learning. It is different from ACL in that there is only one agent that interacts with the system, using the global Q-value function. In (Grounds and Kudenko, 2007), a parallel implementation of SARSA(λ) is presented, with value functions represented using linear function approximators. SARSA(λ) is very similar to Q(λ)-learning and the reader is referred to Sutton and Barto (1998) for more information about this RL technique. Each agent learns independently based on a separate simulation of the single-agent problem. The agents periodically exchange information extracted from the weights of their function approximators. This accelerates convergence towards the optimal policy. The architecture is implemented on real hardware with real resource constraints, in which the main limitation is in the communication cost. It is comparable to ACL, in the sense that the agents do not share an environment in which they must coordinate their actions, or compete for rewards. The agents learn the value function for the whole state space. The difference between this architecture and ACL lies in the fact that all agents maintain their own, private value function, and only once in a while communicate the weights of the value functions. The weights are merged distributively based on all weights and visit counts of all agents. Kushida et al. (2007) present a comparative study of parallel RL implemented on a cluster of personal computers. With the *state division learning method*, the Q-value function is split into subsets and each personal computer in the cluster works on a separate subset. The *prioritized field learning method* is similar to this technique, but here the fields (subsets) have a priority rating, which depends dynamically on the amount of changes in the Q-values. If there are more changes, the field is considered to be more important and thus receives a higher priority. The third parallel RL method compared in this study is *parallel fuzzy Q-learning*, in which the environment is also divided in subsets and each personal computer in the cluster learns the fuzzy rules of each subset. The reader is referred to (Kushida et al., 2007) for details on the comparative study. The parallel RL methods compared in this study are very different from ACL, because of the splitting of the environment in subsets. In ACL, all agents learn by using the complete environment (state-space).

Both RL and ACL require the dynamical system to be an MDP, but neither requires the model to be known. As such, both RL and ACL can be categorized as model-free learning algorithms, since the agent does (the ants do) not have direct access to the model of the system and must learn the control policy just by interacting with it. Note that as the ants interact with the system in parallel, the implementation of this algorithm to a real system requires multiple copies of this system, one for each ant. However, when a (black-box) model of the physical system is available, the parallelism can take place in software. In that case, the learning happens off-line and after convergence, the resulting control policy can be applied to the real system. In principle, the ants could also interact with one physical system, but this would require either a serial implementation in which each ant performs a trial and the global pheromone update takes place after the last ant has completed its trial, or a serial implementation in which each ant only performs a single interaction step, after which the state of the system must be reinitialized for the next ant. In the first case, the local pheromone

update will lose most of its use, while in the second case, the repetitive reinitializations will cause a strong increase of learning time and heavy load on the system. In either case, the usefulness of ACL will be strongly compromised in that case.

3.6 Experiments: Grid Search

In Section 3.4 we have analytically studied the behavior of ACL and, in particular, discussed the effect of two of its key parameters, namely the local and global pheromone decay rates. An theoretical study on the convergence of ACL does, however, not say much about the performance of the algorithm. Therefore, we present an experimental evaluation of ACL on two problems that have by nature a discrete state space: grid search in single dimension and in two dimensions. In these problems, the optimal control policy can easily be seen by visual inspection and the focus can be on the effect of a single parameter on the behavior of the algorithm. In all experiments, the performance is recorded over the trials and each experiment is carried out 30 times, such that we can present plots with an average and max-min performance. This gives us information about the repeatability of the experiments and the representativeness of the results. Before we will present the results, the following section first discusses the various performance measures that we use.

3.6.1 Performance Measures

The most important performance measure, which will be presented for all experiments, is the *cost of the policy*. It is a measure for the current quality of the learned policy. After each trial in an experiment (i.e. after all ants having completed their tours, or having been terminated prematurely, and the pheromone levels have been updated), the system is simulated using the current policy for a set of initial states and a certain simulation time. Simulating the system using the current policy can be implemented using the ACL algorithm, but without updating the pheromone levels ($\gamma = \rho = 0$) and with instead of the action selection rule, the current policy (no exploration). The cost of the resulting solutions are logged and averaged over the number of solutions. The resulting value is a measure of the cost of the policy:

$$J^{\mathbf{h}} = \frac{1}{|\mathcal{S}_{\text{sim}}|} \sum_{s \in \mathcal{S}_{\text{sim}}} J(s), \tag{3.47}$$

with $\mathcal{S}_{\text{sim}}$ containing the solutions resulting from the simulation of the policy $\mathbf{h}$ using a fixed set of initial states. The value of $J^{\mathbf{h}}$ is determined after each trial and is plotted. The plot thus shows the evolution of the policy cost over the trials.

There are several other performance measures that we will use only for some of the experiments, in order to study and discuss specific aspects of the algorithm. One such other performance measure is the *cost of the ants*. This cost is different from $J^{\mathbf{h}}$, because ants continually explore the state space, which may or may not lead to a lower cost of the found solutions. At the end of each trial, the cost of the tour of each ant is logged and averaged over the number of ants. The resulting value is a measure of the cost of the ants in that respective trial. The plot will show the evolution of this measure over the trials and comparing it with the plot showing the evolution of $J^{\mathbf{h}}$ gives an idea of the amount of exploration.

The *policy variation* is a measure from trial to trial, indicating the fraction of states for which the policy has changed. This number, by itself does not tell anything about the quality of the policy, as a policy variation of zero may still correspond with a low-performance policy. However, if $J^{\mathbf{h}}$ does not converge, the policy variation can be used to see the fraction of the state space for which it is seemingly hard for the policy to converge. At the same time, if $J^{\mathbf{h}}$ has converged while there is still some policy variation, this indicates that there are several policies with an equally optimal performance.

For each state, there is a certain number of actions to choose from. If the current policy indicates that a certain action is the best, it does not say how much better it is compared to the second-best action. If it is only a little better, a small update of the pheromone level for that other action in this state, may switch the policy for this state. In that case, the policy is considered to be very *sensitive*. The sensitivity of the policy in a certain state is defined as follows:

$$\begin{aligned}
\mathbf{h}_1(\mathbf{q}) &= \arg\max_{\mathbf{u}\in\mathcal{U}_\mathbf{q}} (\tau_{\mathbf{qu}}), \\
\mathbf{h}_2(\mathbf{q}) &= \arg\max_{\mathbf{u}\in\mathcal{U}_\mathbf{q}\setminus\{\mathbf{h}_1(\mathbf{q})\}} (\tau_{\mathbf{qu}}), \\
\varphi(\mathbf{q}) &= \left(\frac{\tau_{\mathbf{q}\mathbf{h}_2(\mathbf{q})}}{\tau_{\mathbf{q}\mathbf{h}_1(\mathbf{q})}}\right)^{\alpha},
\end{aligned} \tag{3.48}$$

where α is the same as the α from the Boltzmann action selection rule (3.4). The sensitivity of the policy is the average sensitivity over all states and is a measure in the range of $[0, 1]$. Please note that this performance measure should be interpreted with care, because if the difference between the largest and the second-largest pheromone level increases, while at the same time all pheromone levels increase by approximately the same amount, the relative difference between these two pheromone levels may decrease, which is thus observed as an increased sensitivity. Nevertheless, when all pheromone levels increase and the sensitivity decreases, this does mean that the susceptibility to changes in the policy due to small changes in pheromone levels decreases.

3.6.2 1-D Grid Search

The 1-dimensional grid search problem is a very simple problem, in which there are 10 discrete states in a row. The only possible actions for an ant are to move west, or to move east and the goal state is at the far east. The system is deterministic.

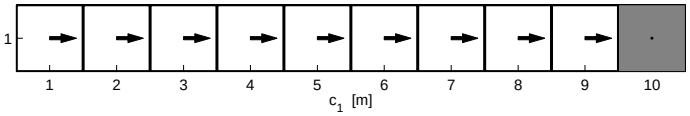
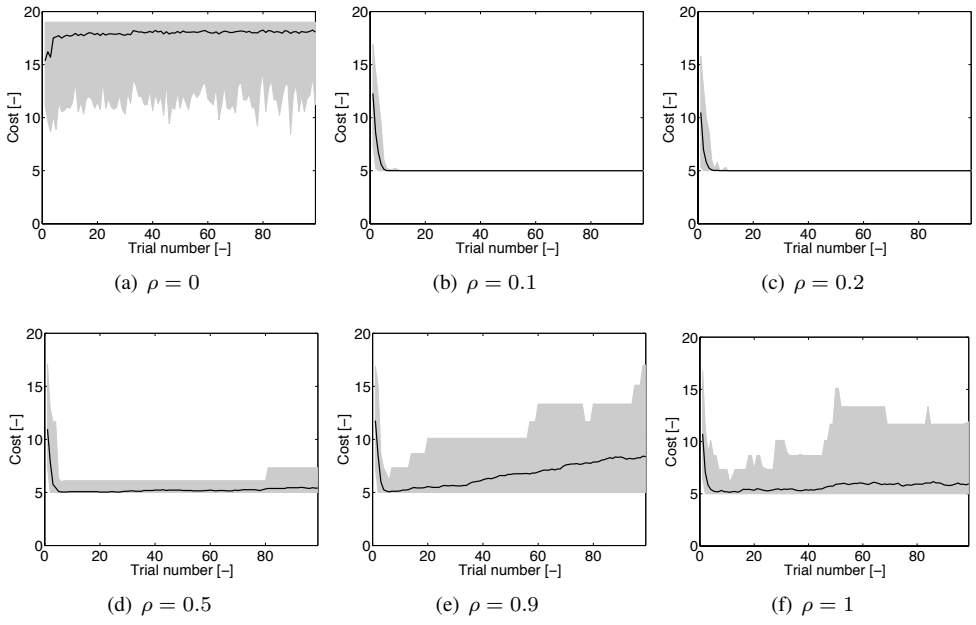


Figure 3.5: Optimal control policy for the 1-dimensional grid search problem.

Figure 3.5 shows the layout and the optimal control policy. The number of steps needed to reach the goal is taken as the cost. The cost of the optimal control policy averaged over all initial states, $J^{\mathbf{h}^*}$, is thus 5.

3.6.3 Varying Global Pheromone Decay Rate

In these experiments, the local pheromone decay rate is set to $\gamma = 0$, effectively eliminating the local pheromone update rule. This is done in order to study the influence of the global pheromone decay rate, ρ, in the absence of γ. The number of ants is $M = 9$. We experiment with different values of ρ from the set $\rho \in \{0, 0.1, 0.2, 0.5, 0.9, 1\}$. The results are presented in Figure 3.6.



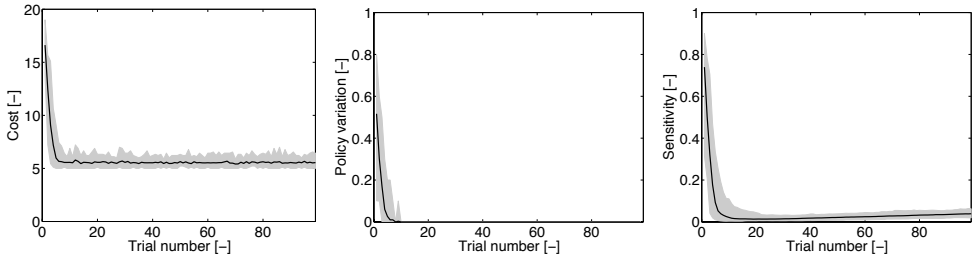
(a) $\rho = 0$ (b) $\rho = 0.1$ (c) $\rho = 0.2$

(d) $\rho = 0.5$ (e) $\rho = 0.9$ (f) $\rho = 1$

Figure 3.6: Cost of the policy ($J^{\mathbf{h}}$) for $\gamma = 0$ and varying ρ. The black line in each plot is the average policy performance over 30 experiments and the gray area represents the max-min range of the policy performance for these experiments.

It can be seen that taking $\rho = 0$ will not lead to convergence. This is obvious, as $\rho = 0$ means that the pheromone update does not change the pheromone levels at all, and the pheromone levels will retain their initial value. The slight increase of $J^{\mathbf{h}}$ possibly indicates that on average a pheromone matrix with all pheromone levels equal to τ_0 performs better than with random pheromone levels. On the other hand, for large ρ, the policy, although converging at first, will eventually not settle at one specific (optimal) policy. This is because for larger ρ, the policy is increasingly influenced by the pheromone update in a trial, which may be not optimal due to exploration. Clearly, the best choice for ρ is a small value larger than zero, e.g. 0.1. This result also confirms the allowed range for ρ as derived in Section 3.4.10, namely $\rho \in (0, 1]$.

Figure 3.7 shows plots of the other performance measures for the setting $\rho = 0.1$, $\gamma = 0$. It indeed shows that the exploration of the ants on average results in a slightly higher cost than optimal and that exploration continues, even while the policy has converged. It can also be seen from the policy variation and sensitivity that the optimal policy quickly becomes clearly dominant over other possible policies.

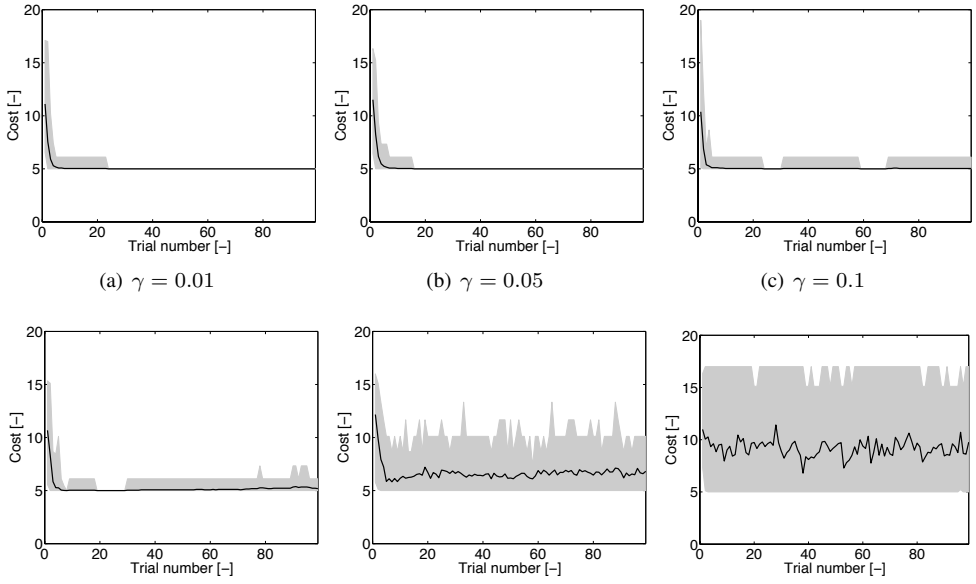


(a) Performance of the ants (b) Policy variation (c) Sensitivity of the policy

Figure 3.7: Other performance measures for $\rho = 0.1$ and $\gamma = 0$. The black line is the average over the 30 experiments and the gray area represents the max-min range.

3.6.4 Varying Local Pheromone Decay Rate

The value $\rho = 0.1$ is now used to study the influence of γ. We experiment with different values of γ from the set $\gamma \in \{0.01, 0.05, 0.1, 0.2, 0.5, 0.9\}$. Figure 3.8 show the results.



(a) $\gamma = 0.01$ (b) $\gamma = 0.05$ (c) $\gamma = 0.1$

(d) $\gamma = 0.2$ (e) $\gamma = 0.5$ (f) $\gamma = 0.9$

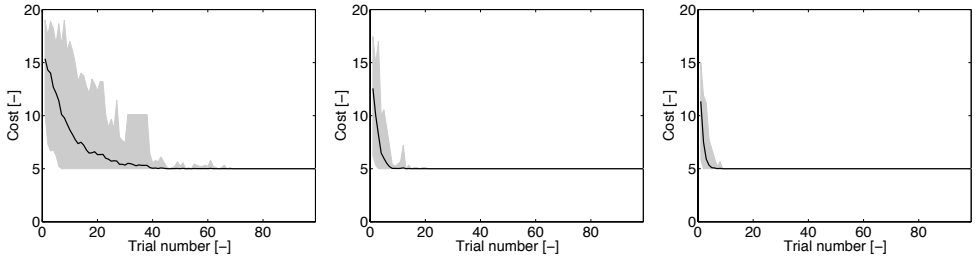
Figure 3.8: Cost of the policy ($J^{\mathbf{h}}$) for $\rho = 0.1$ and varying γ. The black line is the average over the 30 experiments and the gray area represents the max-min range.

For larger γ, the pheromone levels of visited $(\mathbf{q}, \mathbf{u})$-pairs are decayed more, until for $\gamma = 1$, these pheromone levels are reset to their initial value at each visit. This effect on the convergence of the policy can clearly be seen in Figure 3.8. Again, $M = 9$ ants are used and each experiment is carried out 30 times. It turns out that, for this problem, it is best to take $\gamma = 0$ (i.e. Figure 3.6(b)), corresponding to not use the local pheromone update step at

all. The local pheromone update step may serve a better purpose for larger state spaces, or in problems with noise, as it stimulates exploration (see Section 3.3.2).

3.6.5 Varying Number of Ants

The last experiments on the 1-dimensional grid search illustrate the effect of the number of ants on the learning performance. The results are depicted in Figure 3.9 and show $J^{\mathbf{h}}$ over the course of the trials for 1, 5, and 9 ants.



(a) $M = 1$ (b) $M = 5$ (c) $M = 9$

Figure 3.9: Cost of the policy for $\rho = 0.1$, $\gamma = 0$, and a varying number of ants. The black line is the average over the 30 experiments and the gray area represents the max-min range.

The results show that for an increasing number of ants, the learning time decreases indicating that the ants cooperate and jointly speed-up the learning. It also shows that it is not necessary to have a 100% coverage of the states with ants, which in this case of only 9 states corresponds to 9 ants, in order to obtain the optimal policy. Increasing the number of ants reduces the learning time in terms of number of trials.

3.6.6 2-D Grid Search

This problem is similar to the 1-D grid search problem, but consists of a two-dimensional discrete world and a set of obstacles (see Figure 3.10).

The obstacles reduce the problem of having multiple equally optimal control policies, which would be the case in multi-dimensional state spaces without any obstacles. There are four actions available to the ants: north, south, east, and west. The goal is the state in the center of the state space. Just as with the 1-D grid search the number of steps to the goal serves as the cost.

3.6.7 Varying State-Transition Probability

In the first experiments we vary the state-transition probability. A state-transition probability of, e.g., $p = 0.9$ means that there is a probability of 0.1 that the state transition is not carried out and that the ant thus observes that its action lead to staying in the same state. We want to know how well ACL can deal with stochasticity, because this will be a key issue in Chapter 4, when dealing with the quantization of continuous state spaces.

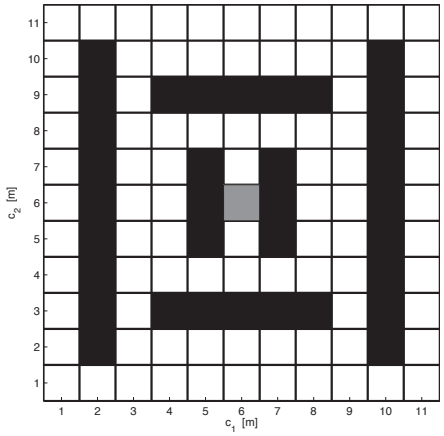
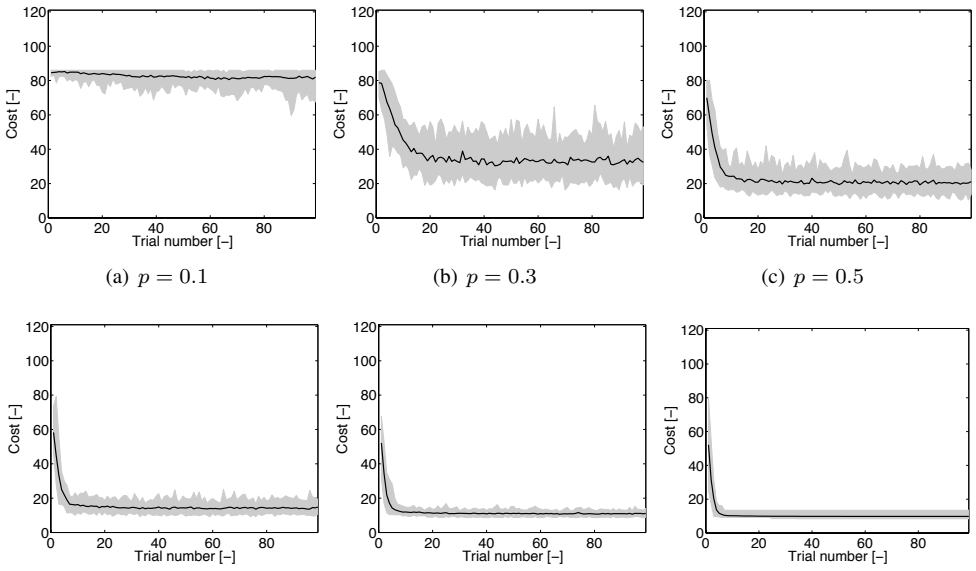


Figure 3.10: Illustration of the discrete world from these experiments. There are 11 by 11 states, of which some are blocked by obstacles (black). The goal state is in the center (gray).



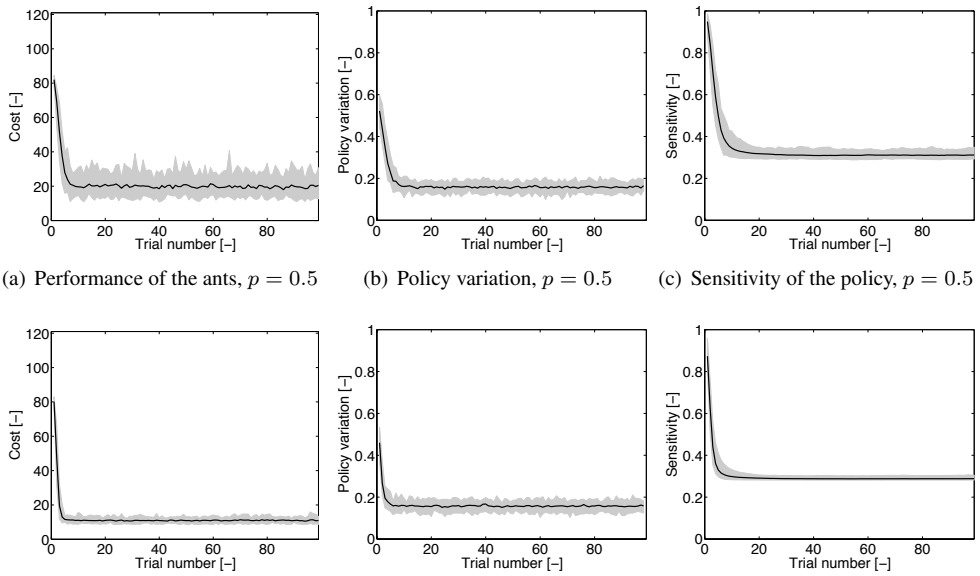
(a) $p = 0.1$ (b) $p = 0.3$ (c) $p = 0.5$

(d) $p = 0.7$ (e) $p = 0.9$ (f) $p = 1$

Figure 3.11: Cost of the policy ($J^{\mathbf{h}}$) for $\rho = 0.1$, $\gamma = 0$, $M = 86$ (= 100% coverage), and varying transition probability p. The black line is the average over the 30 experiments and the gray area represents the max-min range.

The results from Figure 3.11 show that although the convergence becomes stronger and more certain for higher state-transition probabilities, the average cost of the policy ($J^{\mathbf{h}}$) for this problem will converge to about $\frac{1}{p}$ times the average number of steps to the goal from any given initial state, which is about 10 for this 11×11 size problem with obstacles. In

Figure 3.12, the other performance measures are considered for the cases of $p = 0.5$ and $p = 0.9$. Interestingly, these cases demonstrate comparable performance indicating that ACL is quite robust to state-transition stochasticity. Finally Figure 3.13 shows the control policies in the cases of these two state-transition probabilities. Clearly, these control policies are very similar and are near optimal.



(a) Performance of the ants, $p = 0.5$ (b) Policy variation, $p = 0.5$ (c) Sensitivity of the policy, $p = 0.5$

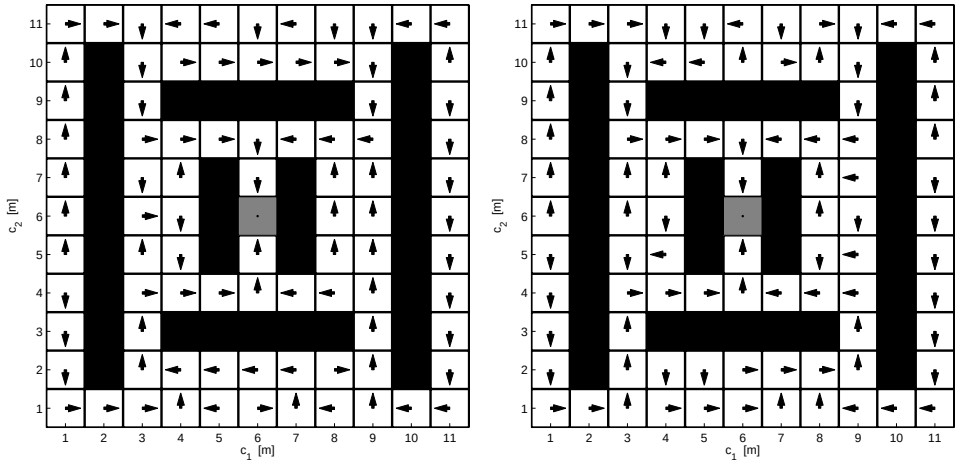
(d) Performance of the ants, $p = 0.9$ (e) Policy variation, $p = 0.9$ (f) Sensitivity of the policy, $p = 0.9$

Figure 3.12: Other performance measures, for $\rho = 0.1$, $\gamma = 0$, $M = 86$ (= 100% coverage), and varying state transition probability p. The black line is the average over the 30 experiments and the gray area represents the max-min range.

3.6.8 2-D Grid Search: Varying State Space Size

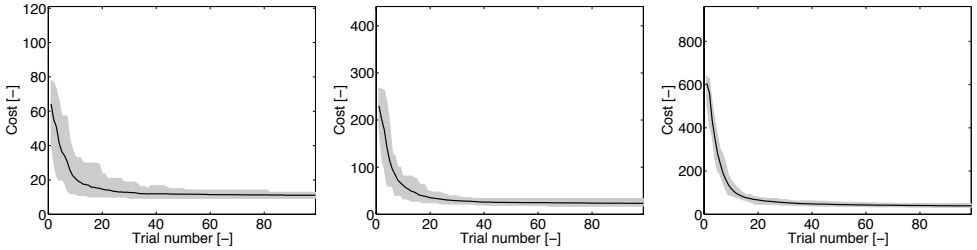
The final experiments on the 2-D grid search problem involves varying the size of the state space and the number of ants covering it. We consider grids of sizes 11×11, 21×21, and 31×31, including the obstacles, corresponding to respectively 86, 278, and 640 states. We evaluate the cases where there is a 20% and 100% coverage by ants. The state transition probability is again 1 and the pheromone decay rates are as before, namely $\rho = 0.1$ and $\gamma = 0$.

The results in Figure 3.14 show that the convergence speed increases with the number of ants, but that even with only 20% coverage, the convergence is reasonably fast. The speed of convergence in the number of trials is not too much related to the problem size, but rather to the number of ants.

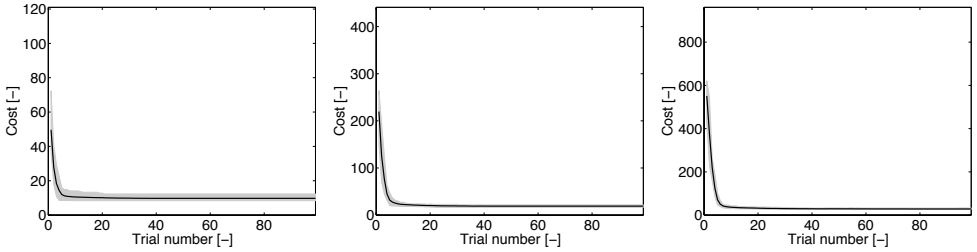


(a) $p = 0.5$ (b) $p = 0.9$

Figure 3.13: Control policy, for $\rho = 0.1$, $\gamma = 0$, $M = 86$ (= 100% coverage), and varying state transition probability p.



(a) 11×11 maze with 20% state space coverage by ants (b) 21×21 maze with 20% state space coverage by ants (c) 31×31 maze with 20% state space coverage by ants



(d) 11×11 maze with 100% state space coverage by ants (e) 21×21 maze with 100% state space coverage by ants (f) 31×31 maze with 20% state space coverage by ants

Figure 3.14: Cost of the policy ($J^{\mathbf{h}}$) for various sizes of the problem and the percentage of the state space covered by the ants for $\rho = 0.1$, $\gamma = 0$, and a state transition probability of 1. The black line is the average over the 30 experiments and the gray area represents the max-min range.

3.7 Concluding Remarks

In this chapter, we have introduced the ACL framework in the setting of discrete state spaces. ACL is a multi-agent approach for learning control policies. The control policy is a state-feedback controller and optimality is defined by a cost function that must be designed before applying the algorithm. The interaction of the ants with the system consists for each ant in choosing an action in the current state of the system and applying this action to the system, after which the system moves to a new state and the ant observes this state. After a sequence of state-action pairs, the ant may arrive in the goal state. After all ants have either reached this state, or have timed out, their obtained solutions are evaluated over the cost function and according to these costs, the amount of pheromone deposit is determined for all state-action pairs visited by the ants. After this procedure, the ants are reinitialized over the state space, but the pheromone levels now contain more information about which actions have a better chance of leading to the goal state in an optimal fashion.

We have theoretically analyzed the convergence properties of ACL for the case of a discrete state space with noiseless state transitions. We have derived upper and lower bounds for the pheromone levels and for the expected value of the pheromone levels. We have related these bounds to the convergence of the policy. We have shown that the expected policy converges to the optimal policy for $M = 1$. For an increased number of ants, convergence of the expected policy is only guaranteed for the scenarios when $\gamma = 0$, $\gamma \to 1$, or $\rho = 1$ when the difference between the inverse of the cost associated with the optimal action and the inverse of the cost associated with the other possible actions is larger than some quantity, derived in this chapter. For other cases, the policy is not formally guaranteed to converge.

We have also performed experiments with ACL on one- and two-dimensional grid search problem in various situations. For these particular problems, we have derived good values for the global and local pheromone decay rates and validated the theoretical statements that we have made about these values. Furthermore, we have studied the effect of the number of ants and the size of the state space on the performance of ACL. We concluded that in these experiments, the speed of convergence is proportional to the number of ants used, but that taking the number of ants equal to approximately 20% of the number of states in the system results already in quite fast convergence. The size of the problem turned out not to effect the learning speed much in the sense of the number of trials. Another experiment studied the effect of state-transition stochasticity on the learning performance. The results for these experiments showed that ACL was capable of finding optimal control policies even in the presence of strong noise levels. This result is important as in the next chapter, we will present ACL in the case of continuous state spaces, for which the quantization of the state space introduces state-transition non-determinism.

Chapter 4

Ant Colony Learning in Continuous State Spaces

This chapter presents the generalization of the ACL framework for continuous state spaces. It discusses the partitioning of the state space with crisp bins and fuzzy membership functions and compares their effect on the performance of ACL.

4.1 Introduction

In the previous chapter, we have introduced the ACL framework. It is a novel control policy learning methodology, in which a collection of agents, called ants, jointly interacts with the system at hand in order to find the optimal mapping between states and actions. Through the interaction by pheromones, the ants are guided by each other's experience towards better control policies. We have analyzed the convergence properties for the case of a discrete state space and noiseless observations and deterministic state transitions. We have also performed a series of experiments demonstrating the quick convergence and good performance of ACL, even for the case of non-deterministic state-transitions. These experiments indicate that ACL is much more widely applicable than to only discrete, noiseless dynamical systems.

In this chapter, we will therefore move an important step forward by presenting and discussing ACL in the context of control policy learning for non-linear systems with a continuous state space. In order to represent the continuous state space as a discrete set of variables, we present two methodologies. The first one is presented in Section 4.2 and involves the partitioning of the state space into a set of bins, enabling direct application of the ACL algorithm presented in Section 3.3, but introducing non-deterministic state transitions. The second methodology is more elegant as it involves the partitioning of the state space using fuzzy membership functions, thereby retaining the continuity of the state space, while still having only a finite set of variables for the algorithm. This methodology is presented in Section 4.3 and establishes a generalization of the concept of pheromones and the local and global pheromone update rules in order to integrate fuzzy interpolation in the ACL framework. In Section 4.4 we present a theoretical analysis of the generalized ACL framework and Section 4.5 presents an example application of both ACL methodologies to a continuous-state dynamic system. We compare the performance of the algorithms to the baseline performance

of fuzzy Q-iteration, a well known model-based reinforcement learning method for continuous domains.

4.2 ACL with Crisp State Space Partitioning

For a system with a continuous state space, learning algorithms like ACL can only be applied if the state space is discretized. The most straightforward way to do this is to partition the state space into a finite number of bins, such that each state value is assigned to exactly one bin. These bins can be enumerated and used as the discrete states. This section discusses first the non-determinism that is introduced by discretizing the state space in this way and then the resulting ACL algorithm.

4.2.1 State Space Partitioning

Assume a general discrete-time, continuous-state system represented by the following dynamic model:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{4.1}$$

where $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^{\mathrm{T}} \in \mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state vector with $\mathcal{X}$ the state space and n the order of the system. Furthermore, $\mathbf{u} \in \mathcal{U}$ is the discrete input and t is the discrete time index. For a continuous-time dynamic system, we assume that it can be sampled with a sufficiently short sampling time T_s to get the discrete-time representation (4.1).

In order to apply ACL, the state $\mathbf{x}$ must be discretized into a finite number of bins to get the discrete state $\mathbf{q}$. We assume that the complete state space $\mathcal{X}$ is partitioned with such bins, and that each continuous-valued state $\mathbf{x}$ in this space falls in exactly one bin. For each dimension, the discrete state space $\mathcal{Q}$ represents the centers of these bins and when discretizing $\mathbf{x}$, the nearest element of $\mathcal{Q}$ in a Euclidean sense counts as its discrete representation. This discretization function will be denoted as:

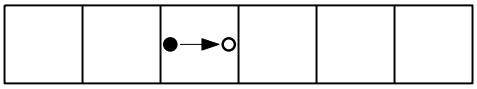
$$\mathbf{q} \leftarrow \mathrm{discretize}(\mathbf{x}, \mathcal{Q}). \tag{4.2}$$

Like in Section 3.2.1, $\mathbf{q} = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}^{\mathrm{T}} \in \mathcal{Q}$ is denoted as a vector, with $\mathcal{Q}$ a finite and countable set of discrete states. Since $\mathbf{q}$ contains discrete values only, all possible states can be enumerated and considered to be elements of the set $\mathcal{Q}$. The number of discretization bins, or possible values of the discrete state, is denoted by $|\mathcal{Q}|$. More bins means that $\mathbf{x}$ can be represented with a greater accuracy, but also that storing its discrete representation requires more memory. Likewise, more bins also means that the CPU time needed to search for the nearest bin of a state $\mathbf{x}$ increases.

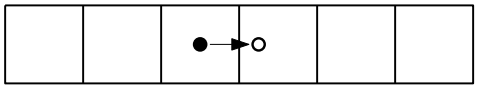
Non-determinism introduced by discretization

Depending on the number of the partitioning bins and their distribution over the state space, portions of the state space will be represented by the same discrete state. As a result of this, although the system in the continuous state space is deterministic, its discrete representation is non-deterministic. The reason why is illustrated in Figure 4.1. Both Figure 4.1(a) and Figure 4.1(b) show the same system, but with different continuous-valued states that are

represented by the same discrete state. When the same action is applied to both systems, their continuous-valued states change in a similar way, but the effect in the discrete domain is different. With the first system, the discrete state has not changed, while the discrete state of the second system has.



(a) A system with a continuous-valued state makes a transition to a new state, as a result of a certain action. However, in the discretized domain, no state transition has occurred.



(b) The same system from (a) with a different continuous-valued state responding to the same action results in a similar continuous state transition. In the discretized domain, the effect is a transition from the same discrete state as in (a) to another state.

Figure 4.1: An illustration of non-determinism introduced to the discrete state transitions if the underlying system is continuous. In both cases, the initial state is discretized to the same bin, the same action is applied, but the resulting discrete state is different.

One could say that applying an input to the system that is in a particular discrete state results in the system to move to a next discrete state with some probability. We can model the non-deterministic discrete state transitions by considering the continuous model to be cast as a discrete stochastic *automaton*. An automaton is defined by the triple $\Sigma = (\mathcal{Q}, \mathcal{U}, \phi)$, with $\mathcal{Q}$ a finite or countable set of discrete states, $\mathcal{U}$ a finite or countable set of discrete inputs, and $\phi : \mathcal{Q} \times \mathcal{U} \times \mathcal{Q} \rightarrow [0, 1]$ a state transition function. Given a discrete state vector $\mathbf{q} \in \mathcal{Q}$ and a discrete input $\mathbf{u} \in \mathcal{U}$, the (Markovian) state transition function ϕ defines the probability of this state transition, $\phi(\mathbf{q}, \mathbf{u}, \mathbf{q}')$, making the automaton stochastic. The probabilities over all states $\mathbf{q}'$ must sum up to one for each state-action pair $(\mathbf{q}, \mathbf{u})$. An example of a stochastic automaton is given in Figure 4.2. In this figure, it is clear that, e.g., applying an action $u = 1$ to the system in $q = 1$ can move the system to a next state that is either $q' = 1$ with a probability of 0.2, or $q' = 2$ with a probability of 0.8.

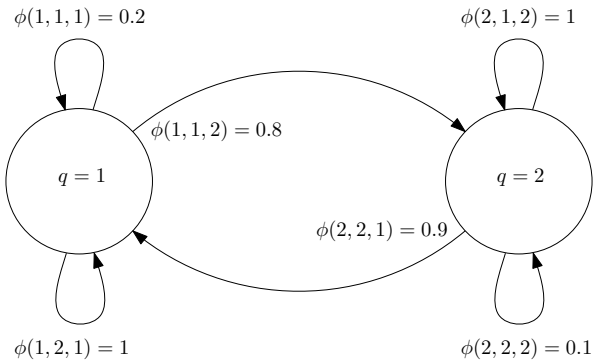


Figure 4.2: An example of a stochastic automaton, where $\phi(\mathbf{q}, \mathbf{u}, \mathbf{q}')$ represents the transition probability from a state q to a state q' given an action u.

The probability distribution function determining the transition probabilities reflects the system dynamics and the set of possible control actions is reflected in the structure of the

automaton. The probability distribution function could be estimated from simulations of the system over a fine grid of pairs of initial states and inputs, but for the application of ACL this is not necessary. The algorithm can directly interact with the continuous state dynamics of the system, as will be described in Section 4.2.2. It is important to realize that the ants only see the effects of their actions in the discrete domain and that they are blind to the underlying continuous system. To them, the system behaves as a stochastic automaton, without being aware of the actual state transition probabilities.

The effect of $|\mathcal{Q}|$ and T_s on the non-determinism

When we have a continuous-time dynamic system on which we wish to apply ACL, we need to sample it with a sampling time T_s in order to get the discrete-time dynamics of (4.1). This sampling time needs to be short enough in order to capture well the dynamics of the system. For a given state partitioning, represented by the set $\mathcal{Q}$, faster sampling means that it is less likely for a discrete state transition to occur. In other words, from t to $t+1$, it is less likely that the ants notice the effect in terms of the discrete state. Likewise, for a given sampling time, a finer partitioning results in a larger probability that a transition from a different continuous-valued state also results in a change of the discrete-valued state. We have seen though, that a finer partitioning means a higher load on the memory and CPU. For these reasons, there is a trade-off to be made between more states on one hand to capture the state dynamics and limit the non-determinism introduced, and fewer states on the other hand to keep the algorithm reasonably fast. A finer partitioning is thus never desirable, but sometimes necessary.

4.2.2 Crisp ACL Algorithm

We call the algorithm crisp ACL, in order to stress the nature of the state space partitioning. Its outline is very similar to the algorithm introduced in Section 3.3. Before running the algorithm, the number of discretization levels and their distribution over the state space must be chosen, i.e., $\mathcal{Q}$. At the start of every trial, each ant is initialized with a random continuous-valued state of the system. This state is then discretized using the set of bins defined by $\mathcal{Q}$. Each ant chooses its action according to one of the action selection rules from Section 3.3.1, applies this action to its copy of the system, and adds the state-action pair to its partial solution. The ant also performs a local pheromone update step in order to stimulate exploration by other ants. According to the system dynamics, the ant observes at the next step the next state to which it has moved and repeats the process by choosing a new action, until it reaches the discrete state marked as the goal and terminates its trial. After all ants have terminated the trial, all found solutions are added to the multiset $\mathcal{S}_\mathrm{trial}$. These solutions are evaluated over the cost function and a global pheromone update step updates the pheromone levels accordingly.

Like in Section 3.3, we explicitly distinguish between the steps in the inner loop and the steps in the outer loop of the algorithm. In the inner loop, the iterations are indexed by t, while in the outer loop, the iterations are indexed by k. In order to understand the timing of the pheromone updates unambiguously, the pheromone variables in the inner loop receive the superscript "local": $\tau_{\mathbf{qu}}^{\mathrm{local}}$. Before starting the inner loop, the current pheromone levels are copied to the local pheromone levels: $\tau_{\mathbf{qu}}^{\mathrm{local}}(0) = \tau_{\mathbf{qu}}(k)$ for all state-action pairs. The first step in the inner loop is the selection of the action.

Action Selection

In the action selection step, all ants that have not yet reached the goal, or have not yet timed-out, belong to the set $\mathcal{C}$ and select an action to apply to the system. The action selection may be each one of the three types discussed in Section 3.3.1. The default choice is the random proportional action selection rule, which is repeated here for the sake of clarity:

$$\mathbf{u}_c \sim \mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c\} = \frac{\left(\tau^{\text{local}}_{\mathbf{q}_c\mathbf{u}}(t)\right)^\alpha}{\sum_{\boldsymbol{\ell}\in\mathcal{U}_{\mathbf{q}_c}}\left(\tau^{\text{local}}_{\mathbf{q}_c\boldsymbol{\ell}}(t)\right)^\alpha}, \quad \mathbf{u}\in\mathcal{U}_{\mathbf{q}_c}, \tag{4.3}$$

where $\mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c\}$ is the probability for an ant c to choose action $\mathbf{u}$ in state $\mathbf{q}_c$ and $\mathcal{U}_{\mathbf{q}_c}$ is the action set available to ant c in state $\mathbf{q}_c$. The amount of exploration is implicit in the choice of α and the pheromone levels $\tau_{\mathbf{q}_c\mathbf{u}}$, $\mathbf{u}\in\mathcal{U}_{\mathbf{q}_c}$. As the pheromone levels converge, the amount of exploration is automatically reduced. The amount of exploration can be further reduced by annealing the value of α with the progress of the algorithm.

Local Pheromone Update

The local pheromone update is done in the same way as in Section 3.3.2:

$$\tau^{\text{local}}_{\mathbf{q}_c\mathbf{u}_c}(t+1) = (1-\gamma)\tau^{\text{local}}_{\mathbf{q}_c\mathbf{u}_c}(t) + \gamma\tau_0, \tag{4.4}$$

with $\gamma\in[0,1)$ the local pheromone decay rate and τ_0 the initial value of the pheromone levels. After the local pheromone update, all ants that have reached the goal are removed from the set $\mathcal{C}$. When this set is empty, or when the inner loop has timed-out (i.e., when $t=T$), the algorithm continues with the global pheromone step in the outer loop.

Global Pheromone Update

After having completed the trial, all found solutions are added to the multiset $\mathcal{S}_{\text{trial}}$ and the pheromone levels are updated according to the same global pheromone update step as in Section 3.3.3. Let us assume that this happens when $t=T$:

$$\begin{aligned}\tau_{\mathbf{qu}}(k+1) =& (1-\rho)\tau^{\text{local}}_{\mathbf{qu}}(T) + \rho \sum_{\substack{s\in\mathcal{S}_{\text{trial}}(k):\\ (\mathbf{q},\mathbf{u})\in s}} J^{-1}(s), \\ &\forall(\mathbf{q},\mathbf{u}) : \exists s\in\mathcal{S}_{\text{trial}}(k) : (\mathbf{q},\mathbf{u})\in s,\end{aligned} \tag{4.5}$$

with $\rho\in(0,1]$ the global pheromone decay rate. The algorithm then continues for an incremented k at the start of the outer loop until the maximal number of trials have taken place (i.e., when $k=K$).

Control Policy

The control policy can be extracted from the pheromone levels as follows:

$$\mathbf{u} = \mathbf{h}(\mathbf{q}) = \arg\max_{\boldsymbol{\ell}\in\mathcal{U}_{\mathbf{q}}} (\tau_{\mathbf{q}\boldsymbol{\ell}}), \tag{4.6}$$

Algorithm 4.1 The crisp ACL algorithm.

Input: $\mathcal{Q}, \mathcal{U}, \mathcal{X}, \mathbf{f}, M, \tau_0, \gamma, \rho, \alpha, T, K$

1: Initialize the algorithm:
$k \leftarrow 0; \tau_{\mathbf{qu}} \leftarrow \tau_0, \quad \forall (\mathbf{q}, \mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$
2: **repeat**
3: Initialize the trial:
$t \leftarrow 0; \mathcal{S}_{\text{trial}} \leftarrow \emptyset; \mathcal{C} \leftarrow \{1, 2, \ldots, M\}$
4: **for all** ants $c \in \mathcal{C}$ in parallel **do**
5: Initialize the partial solution:
$s_{\text{p},c} \leftarrow \emptyset$
6: Initialize the state of the system:
$\mathbf{x}_c(0) \leftarrow \text{random}(\mathcal{X})$
7: **repeat**
8: Discretize the state:
$\mathbf{q}_c(t) \leftarrow \text{discretize}(\mathbf{x}_c(t), \mathcal{Q})$
9: Select action:
$\mathbf{u}_c(t) \sim \mathbf{p}_c\{\mathbf{u}|\mathbf{q}_c(t)\} = \dfrac{\tau^{\alpha}_{\mathbf{q}_c\mathbf{u}}}{\sum_{\boldsymbol{\ell} \in \mathcal{U}_{\mathbf{q}_c}} \tau^{\alpha}_{\mathbf{q}_c\boldsymbol{\ell}}}, \quad \mathbf{u} \in \mathcal{U}_{\mathbf{q}_c}$
10: Update partial solution:
$s_{\text{p},c} \leftarrow s_{\text{p},c} \cup \{(\mathbf{q}_c(t), \mathbf{u}_c(t))\}$
11: Apply action to system:
$\mathbf{x}_c(t+1) \leftarrow \mathbf{f}(\mathbf{x}_c(t), \mathbf{u}_c(t))$
12: Perform the local pheromone update:
$\tau_{\mathbf{q}_c\mathbf{u}_c} \leftarrow (1-\gamma)\tau_{\mathbf{q}_c\mathbf{u}_c} + \gamma\tau_0$
13: **if** $\text{discretize}(\mathbf{x}_c(t+1), \mathcal{Q}) = \mathbf{q}_{\text{g}}$ **then**
14: If an ant reaches the goal, its trial terminates:
$\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$
15: **end if**
16: $t \leftarrow t + 1$
17: **until** $t = T$ **or** $\mathcal{C} = \emptyset$
18: Add the solutions found to the solution multiset:
$\mathcal{S}_{\text{trial}} \leftarrow \mathcal{S}_{\text{trial}} \cup \{s_{\text{p},c}\}, \quad \forall c \in \{1, 2, \ldots, M\}$
19: **end for**
20: Perform the global pheromone update:
$\tau_{\mathbf{qu}} \leftarrow (1-\rho)\tau_{\mathbf{qu}} + \rho \sum_{\substack{s \in \mathcal{S}_{\text{trial}}: \\ (\mathbf{q},\mathbf{u}) \in s}} J^{-1}(s), \quad \forall (\mathbf{q}, \mathbf{u}) : \exists s \in \mathcal{S}_{\text{trial}} : (\mathbf{q}, \mathbf{u}) \in s$
21: $k \leftarrow k + 1$
22: **until** $k = K$

Output: $\tau_{\mathbf{qu}}, \quad \forall (\mathbf{q}, \mathbf{u}) \in \mathcal{Q} \times \mathcal{U}$

in which ties are broken randomly. This equation means that the control policy assigns the action to a given state that maximizes the associated pheromone levels.

The complete algorithm is given in Algorithm 4.1. In this algorithm, the assignment $\mathbf{x}_c \leftarrow \text{random}(\mathcal{X})$ in Step 6 selects for ant c a random state $\mathbf{x}_c$ from the state space $\mathcal{X}$ with a uniform probability distribution. Although the domain $\mathcal{X}$ is listed as an input, in fact the

input is any discrete representation of this domain compatible with the $\mathrm{random}(\mathcal{X})$ function. The remaining parameters of the algorithm are the same as in Table 3.2.

4.3 ACL with Fuzzy State Space Partitioning

A problem with crisp ACL is that the number of bins required to accurately capture the dynamics of the original system may become very large even for simple systems with only a few state variables. Moreover, the time complexity of ACL grows exponentially with the number of bins, making the algorithm infeasible for realistic systems. In particular, note that for systems with fast dynamics in certain regions of the state space, the sampling time needs to be chosen smaller in order to capture the dynamics in these regions accurately. In other regions of the state space where the dynamics of the system are slower, the faster sampling requires a denser discretization, increasing the number of bins. All together, without much prior knowledge of the system dynamics, both the sampling time and the bin size need to be small enough, resulting in a rapid explosion of the number of bins.

A better alternative may be to approximate the state space by a parameterized function approximator, or more specifically, by a set of basis functions. In that case, there is still a finite number of parameters, but this number can typically be chosen to be much smaller compared to using crisp discretization. Furthermore, the mapping between the original continuous state space and the function used in the approximation is bijective, preventing the artificial introduction of noise, like it is the case in crisp ACL. There are various kinds of basis functions, such as radial or fuzzy basis functions. In this chapter, we have chosen to use fuzzy basis functions because it is the most straightforward way of interpolating between discrete states. For this reason, the ACL algorithm utilizing the fuzzy partitioning of the state space is called fuzzy ACL. In Section 7.3 it is explained that other types of basis functions may be used as well.

4.3.1 State Space Partitioning

With fuzzy approximation, the domain of each state variable is partitioned using membership functions. We define the membership functions for the state variables to be triangular-shaped, such that the membership degrees for any value of the state on the domain always sum up to one. Only the centers of the membership functions then have to be stored, as they completely define the fuzzy partitioning. An example of such a fuzzy partitioning is given in Figure 4.3.

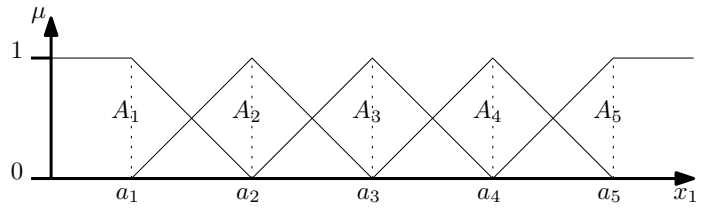


Figure 4.3: Membership functions $A_1, \ldots, A_5$, with centers $a_1, \ldots, a_5$ on an infinite domain.

Let A_i denote the membership functions for the state variable x_1, with a_i their centers for $i = 1, \ldots, N_A$, with N_A the number of membership functions for x_1. Similarly for x_2,

denote the membership functions by B_i, with b_i their centers for $i = 1, \ldots, N_B$, with N_B the number of membership functions for x_2. Likewise, the membership functions can be defined for the other state variables in $\mathbf{x}$, but for the sake of notation, the discussion in this chapter limits the number to two, without loss of generality. Note that, e.g., in the application in Section 4.5, the number of state variables is four. At a discrete time step t, the membership degrees of a specific value of the state to A_i and B_i are denoted by $\mu_{A_i}(x_1(t))$ and $\mu_{B_i}(x_2(t))$ respectively. The membership degree of x_1 to A_i can be computed as follows:

$$\mu_{A_i}(x_1) = \begin{cases} \max\left(0, \min\left(1, \frac{a_2 - x_1}{a_2 - a_1}\right)\right) & \text{if } i = 1 \\ \max\left(0, \min\left(\frac{x_1 - a_{i-1}}{a_i - a_{i-1}}, \frac{a_{i+1} - x_1}{a_{i+1} - a_i}\right)\right) & \text{if } i = 2, \ldots, N_A - 1 \\ \max\left(0, \min\left(\frac{x_1 - a_{N_A - 1}}{a_{N_A} - a_{N_A - 1}}, 1\right)\right) & \text{if } i = N_A \end{cases} \tag{4.7}$$

The degree of fulfillment is computed by multiplying the two membership degrees:

$$\beta_{ij}(\mathbf{x}(t)) = \mu_{A_i}(x_1(t)) \cdot \mu_{B_j}(x_2(t)).$$

Let the vector of all degrees of fulfillment for a certain state at time t be denoted by:

$$\begin{aligned} \boldsymbol{\beta}(\mathbf{x}(t)) = [&\beta_{11}(\mathbf{x}(t)) \quad \beta_{12}(\mathbf{x}(t)) \quad \ldots \quad \beta_{1N_B}(\mathbf{x}(t)) \\ &\beta_{21}(\mathbf{x}(t)) \quad \beta_{22}(\mathbf{x}(t)) \quad \ldots \quad \beta_{2N_B}(\mathbf{x}(t)) \\ &\ldots \quad \beta_{N_A N_B}(\mathbf{x}(t))]^{\mathrm{T}}, \end{aligned} \tag{4.8}$$

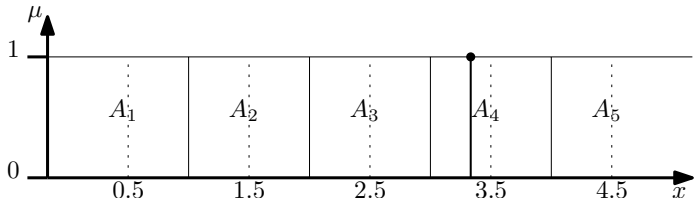
which is a vector containing the elements $0 \leq \beta_{ij} \leq 1$ for all combinations of i and j, and which elements sum up to one. In order to illustrate what this vector looks like, consider the example from Figure 4.4. In this example, a one-dimensional state $x = 3.2$ is partitioned with crisp bins and fuzzy membership functions of which the centers are the same.

Each element of this vector will be associated to a vertex in the construction graph of fuzzy ACL and thus also to an element from the set $\mathcal{Q}$, now representing the centers of the membership functions. The operator to partition a state $\mathbf{x}$ using the fuzzy membership functions defined by $\mathcal{Q}$ will be denoted as:

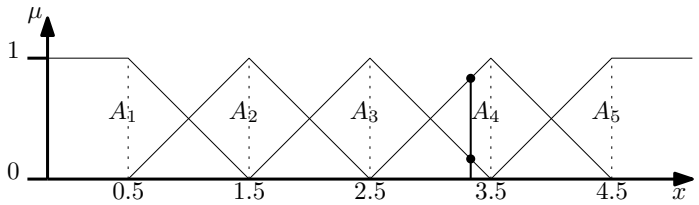
$$\boldsymbol{\beta}(\mathbf{x}) \leftarrow \text{fuzzify}(\mathbf{x}, \mathcal{Q}). \tag{4.9}$$

Note that the discretization operator from (4.2) is in fact a special case of (4.9), as also illustrated in Figure 4.4. With respect to memory requirements, the crisp representation using $\boldsymbol{\beta}(\mathbf{x})$ always has exactly one non-zero element, which is then by definition equal to one. With pair-wise overlapping normalized membership functions, like the ones shown in Figure 4.3, the fuzzy representation using $\boldsymbol{\beta}(\mathbf{x})$ has at most 2^d non-zero elements, with d the dimension of the state space. When $\boldsymbol{\beta}(\mathbf{x})$ is stored as a sparse data structure, this means that the memory requirements when using fuzzy partitioning scales exponentially with the number of dimensions. It also means that the memory requirements are independent of the number of membership functions used to represent the state space.

Most of the steps from Algorithm 4.1 need to be reconsidered in the light of the generalization using fuzzy membership functions. This is the subject of the following section. Note that for the sake of notation, we will denote $\boldsymbol{\beta}(\mathbf{x}(t))$ as $\boldsymbol{\beta}(t)$, knowing that it is the fuzzy representation of a state $\mathbf{x}$.



(a) The crisp representation of $x = 3.2$ is: $\boldsymbol{\beta}(x) = [0, 0, 0, 1, 0]^{\mathrm{T}}$.



(b) The fuzzy representation of $x = 3.2$ is: $\boldsymbol{\beta}(x) = [0, 0, 0.3, 0.7, 0]^{\mathrm{T}}$.

Figure 4.4: A simple example of a one-dimensional state x that is partitioned with crisp bins and fuzzy membership functions of which the centers are the same. This illustrates the meaning of $\boldsymbol{\beta}(\mathbf{x})$.

4.3.2 Fuzzy ACL Algorithm

In fuzzy ACL, an ant is not assigned to one certain discrete state at a time, but to all discrete states at the same time according to the corresponding degree of fulfillment. Similar to the definition of $\boldsymbol{\beta}(t)$ in (4.8), the vector of all pheromones for a certain action $\mathbf{u}$ at a discrete time index t is denoted as:

$$\boldsymbol{\tau}_{\mathbf{u}}(t) = \begin{bmatrix} \tau_{1\mathbf{u}}(t) & \tau_{2\mathbf{u}}(t) & \dots & \tau_{N_{AB}\mathbf{u}}(t) \end{bmatrix}^{\mathrm{T}}, \tag{4.10}$$

where $N_{AB} = N_A \cdot N_B$. Using $\boldsymbol{\beta}(t)$ and $\boldsymbol{\tau}_{\mathbf{u}}(t)$, we can reformulate the steps from Algorithm 4.1 which used the discrete representation of the state, i.e., the action selection, the local and global pheromone update rules, and the derivation of the control policy.

Action Selection

The action is chosen randomly according to the following probability distribution:

$$\mathbf{u}_c \sim \mathbf{p}_c\{\mathbf{u}|\boldsymbol{\beta}_c\} = \sum_{i=1}^{N_{AB}} \beta_{c,i} \frac{\left(\tau_{i,\mathbf{u}}^{\mathrm{local}}(t)\right)^{\alpha}}{\sum_{\boldsymbol{\ell}\in\mathcal{U}} \left(\tau_{i,\boldsymbol{\ell}}^{\mathrm{local}}(t)\right)^{\alpha}}, \quad \mathbf{u} \in \mathcal{U}. \tag{4.11}$$

This expression involves the weighted sum of the pheromone levels according to the respective membership degrees for the state $\mathbf{x}_c$ after being fuzzified using (4.9). Note that when $\boldsymbol{\beta}_c$ contains exactly one 1 and for the rest only zeros, this would correspond to the crisp case, where the state is discretized to a set of bins and (4.11) then reduces to (4.3).

Local Pheromone Update

When an ant visits a state with the fuzzy representation $\boldsymbol{\beta}_c$ and chooses an action $\mathbf{u}_c$ it performs a local pheromone update. The local pheromone update from (4.4) can be modified to the fuzzy case as follows:

$$\begin{aligned}\boldsymbol{\tau}_{\mathbf{u}_c}^{\text{local}}(t+1) &= (\mathbf{1}-\boldsymbol{\beta}_c)\boldsymbol{\tau}_{\mathbf{u}_c}^{\text{local}}(t) + \boldsymbol{\beta}_c((1-\gamma)\boldsymbol{\tau}_{\mathbf{u}_c}^{\text{local}}(t) + \gamma\tau_0\mathbf{1}) \\ &= (\mathbf{1}-\gamma\boldsymbol{\beta}_c)\boldsymbol{\tau}_{\mathbf{u}_c}^{\text{local}}(t) + (\gamma\boldsymbol{\beta}_c)\tau_0, \end{aligned} \tag{4.12}$$

where all operations are performed element-wise and $\mathbf{1} = \begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix}^{\mathrm{T}}$ is a unity vector of appropriate size. The pheromone levels associated with the chosen action $\mathbf{u}_c$ are thus decayed according to the membership degree of the visited state with the fuzzy representation $\boldsymbol{\beta}_c$. Note that in the crisp case, (4.12) reduces to (4.4).

Global Pheromone Update

In order to derive a fuzzy representation of the global pheromone update step, we introduce the following indicator vectors. The elements of indicator vectors can take a value from the domain $[0,1]$. The most basic indicator vector that we need is $\mathcal{I}_{\mathbf{u},s_i^{(j)}}$, which, in the case of crisp state space partitioning has only one element equal to 1, namely the one for $(\mathbf{q},\mathbf{u}) = s_i^{(j)}$. Since a solution $s_i = (s_i^{(1)}, s_i^{(2)}, \dots, s_i^{(N_{s_i})})$ is an ordered set of solution components $s_i^{(j)} = (\mathbf{q}_j, \mathbf{u}_j)$, another indicator vector, $\mathcal{I}_{\mathbf{u},s_i}$, can be created by taking the union of all $\mathcal{I}_{\mathbf{u},s_i^{(j)}}$:

$$\mathcal{I}_{\mathbf{u},s_i} = \bigcup_{j=1}^{N_{s_i}} \mathcal{I}_{\mathbf{u},s_i^{(j)}}. \tag{4.13}$$

Recall that $\mathcal{S}_{\text{trial}} = \{s_1, s_2, \dots, s_{N_{\mathcal{S}_{\text{trial}}}}\}$ is the multiset of solutions. We can then create $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}$ by taking the union of all $\mathcal{I}_{\mathbf{u},s}$:

$$\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}} = \bigcup_{i=1}^{N_{\mathcal{S}_{\text{trial}}}} \mathcal{I}_{\mathbf{u},s_i}. \tag{4.14}$$

In fact, $\mathcal{I}_{\mathbf{u},s_i}$ can be regarded as a representation of the state-action pair $(\mathbf{q}_i, \mathbf{u}_i)$. In order to generalize the global pheromone update step to the fuzzy case, realize that $\boldsymbol{\beta}(\mathbf{x}(t))$ from (4.8) can be seen as an indicator vector $\mathcal{I}_{\mathbf{u},s_i^{(j)}}$, if combined with an action $\mathbf{u}$. For the union operator, we can take a fuzzy union set operator, such as:

$$(A \cup B)(x) = \max[\mu_A(x), \mu_B(x)],$$

which for a vector operates on its elements. In this operator, A and B are membership functions, x a variable, and $\mu_A(x)$ is the degree to which x belongs to A. Note that when A maps x to a crisp domain $\{0,1\}$, the union operator is still valid. Using these notations, we

can write the generalized global pheromone update rule as:

$$\begin{aligned}\boldsymbol{\tau}_{\mathbf{u}}(k+1) =& \left\{(1-\rho)\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(T) + \rho \sum_{s\in\mathcal{S}_{\text{trial}}} J^{-1}(s)\mathcal{I}_{\mathbf{u},s}\right\}\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}} \\ &+ (\mathbf{1}-\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}})\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(T) \\ =&(\mathbf{1}-\rho\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}})\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(T) + \rho \sum_{s\in\mathcal{S}_{\text{trial}}} J^{-1}(s)\mathcal{I}_{\mathbf{u},s}, \end{aligned} \tag{4.15}$$

where all multiplications are performed element-wise. Here, we have used that from the definitions of $\mathcal{I}_{\mathbf{u},s}$ and $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}$ it follows that $\mathcal{I}_{\mathbf{u},s}\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}} = \mathcal{I}_{\mathbf{u},s}$, when the multiplications are performed element-wise. Note that in the crisp case, (4.15) reduces to (4.5).

Control Policy

Regarding the terminal condition for the ants, with the fuzzy implementation, none of the vertices can be identified as being the terminal vertex. We define a set of membership functions that is used to express the linguistic fuzzy term of the state being *close to the goal*. Here specifically, this is satisfied when the membership degree of the state to the membership function with its core equal to the goal state is larger than 0.5, as in that case, it belongs more to the goal membership function than to any other membership function. If for an ant this has been satisfied, that ant is considered to have terminated its trial.

In order to obtain the control policy for a given state, this state must first be represented by a $\boldsymbol{\beta}$ vector. For each action, the pheromone level is a linear combination of all elements in the pheromone vector and $\boldsymbol{\beta}$. The policy then assigns the action that has the highest pheromone level as follows:

$$\mathbf{u} = \mathbf{h}(\boldsymbol{\beta}) = \arg\max_{\boldsymbol{\ell}\in\mathcal{U}} \left(\sum_{i=1}^{N_{AB}} \beta_i \tau_{i\boldsymbol{\ell}}\right), \tag{4.16}$$

in which ties are broken randomly. Note that this equation indeed shows that the control policy is a mapping from continuous-valued states to discrete actions.

The complete algorithm is given in Algorithm 4.2. The parameters of the algorithm are the same as in Table 3.2.

Algorithm 4.2 The fuzzy ACL algorithm.

Input: $\mathcal{Q}, \mathcal{U}, \mathcal{X}, \mathbf{f}, M, \tau_0, \gamma, \rho, \alpha, T, K$

Initialize the algorithm:
$k \leftarrow 0; \boldsymbol{\tau}_{\mathbf{u}} \leftarrow \tau_0 \mathbf{1}, \quad \forall \mathbf{u} \in \mathcal{U}$
repeat
  Initialize the trial:
  $t \leftarrow 0; \mathcal{S}_{\text{trial}} \leftarrow \emptyset; \mathcal{C} \leftarrow \{1, 2, \ldots, M\}$
  for all ants $c \in \mathcal{C}$ in parallel **do**
    Initialize the partial solution:
    $s_{\text{p},c} \leftarrow \emptyset$
    Initialize the state of the system:
    $\mathbf{x}_c(0) \leftarrow \text{random}(\mathcal{X})$
    repeat
      Fuzzify the state:
      $\boldsymbol{\beta}_c(t) \leftarrow \text{fuzzify}(\mathbf{x}_c(t), \mathcal{Q})$
      Select action:
      $$\mathbf{u}_c(t) \sim \mathbf{p}_c\{\mathbf{u}|\boldsymbol{\beta}_c\} = \sum_{i=1}^{N_{AB}} \beta_{c,i} \frac{\tau_{i,\mathbf{u}}^{\alpha}}{\sum_{\boldsymbol{\ell} \in \mathcal{U}} \tau_{i,\boldsymbol{\ell}}^{\alpha}}, \quad \mathbf{u} \in \mathcal{U}$$
      Update partial solution:
      $s_{\text{p},c} \leftarrow s_{\text{p},c} \cup \{(\mathbf{q}_c(t), \mathbf{u}_c(t))\}$
      Apply action to system:
      $\mathbf{x}_c(t+1) \leftarrow \mathbf{f}(\mathbf{x}_c(t), \mathbf{u}_c(t))$
      Perform the local pheromone update:
      $\boldsymbol{\tau}_{\mathbf{u}_c} \leftarrow (\mathbf{1} - \gamma \boldsymbol{\beta}_c) \boldsymbol{\tau}_{\mathbf{u}_c} + (\gamma \boldsymbol{\beta}_c) \tau_0$
      if $\mathbf{x}_c(t+1)$ is *close to* the goal state $\mathbf{x}_{\text{g}}$ **then**
        If an ant reaches the goal, its trial terminates:
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c\}$
      end if
      $t \leftarrow t + 1$
    until $t = T$ **or** $\mathcal{C} = \emptyset$
    Add the solutions found to the solution multiset:
    $\mathcal{S}_{\text{trial}} \leftarrow \mathcal{S}_{\text{trial}} \cup \{s_{\text{p},c}\}, \quad \forall c \in \{1, 2, \ldots, M\}$
  end for
  Perform the global pheromone update:
  $$\boldsymbol{\tau}_{\mathbf{u}} \leftarrow (\mathbf{1} - \rho \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}) \boldsymbol{\tau}_{\mathbf{u}} + \rho \sum_{s \in \mathcal{S}_{\text{trial}}} J^{-1}(s) \mathcal{I}_{\mathbf{u},s},$$
  with $\mathcal{I}_{\mathbf{u},s_i} = \bigcup_{j=1}^{N_{s_i}} \mathcal{I}_{\mathbf{u},s_i^{(j)}}$ and $\mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}} = \bigcup_{i=1}^{N_{\mathcal{S}_{\text{trial}}}} \mathcal{I}_{\mathbf{u},s_i}$
  $k \leftarrow k + 1$
until $k = K$

Output: $\boldsymbol{\tau}_{\mathbf{u}}, \quad \forall \mathbf{u} \in \mathcal{U}$

4.4 Analysis of the Generalized Pheromone Update

In this section, we analyze the generalized pheromone update rules from Section 4.3.2 and Section 4.3.2.

4.4.1 Serial Execution of the Pheromone Update Rules

In the ACL framework, all ants perform the local pheromone update in parallel. However, when executed on a single core CPU, these operations must be performed in series. The following theorem states that the effect of the local pheromone update is the same for a parallel, or serial implementation.

Theorem 4.1 *The local pheromone update for multiple ants is invariant with respect to the order in which the updates by the individual ants are performed.*

Proof In the crisp case, the local pheromone update from (4.4) may be rewritten as follows:

$$\begin{aligned}\tau_{\mathbf{qu}}^{\text{local}}(t+1) &= (1-\gamma)\tau_{\mathbf{qu}}^{\text{local}}(t) + \gamma\tau_0 \\ &= (1-\gamma)(\tau_{\mathbf{qu}}^{\text{local}}(t) - \tau_0) + \tau_0,\end{aligned}$$

in which t is the discrete time index, counting the interaction steps with the system in a given trial.

Now, when a second ant updates the same pheromone level $\tau_{\mathbf{qu}}^{\text{local}}$, the pheromone level becomes:

$$\begin{aligned}\tau_{\mathbf{qu}}^{\text{local}}(t+2) &= (1-\gamma)(\tau_{\mathbf{qu}}^{\text{local}}(t+1) - \tau_0) + \tau_0 \\ &= (1-\gamma)^2(\tau_{\mathbf{qu}}^{\text{local}}(t) - \tau_0) + \tau_0.\end{aligned}$$

After n updates, the pheromone level is:

$$\tau_{\mathbf{qu}}^{\text{local}}(t+n) = (1-\gamma)^n(\tau_{\mathbf{qu}}^{\text{local}}(t) - \tau_0) + \tau_0, \tag{4.17}$$

which shows that the order of the update is of no influence to the final value of the pheromone level. For the fuzzy case a similar derivation can be made. In general, after n ants have performed the update, the pheromone vector is:

$$\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(t+n) = \left(\prod_{c=1}^{n}(\mathbf{1} - \gamma\boldsymbol{\beta}_c)\right)(\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(t) - \tau_0\mathbf{1}) + \tau_0\mathbf{1}, \tag{4.18}$$

where again all operations are performed element-wise. The order in which the multiplications are performed has no influence on the final value of the pheromone levels after the local pheromone updates. Note that when $\boldsymbol{\beta}_c$ contains exactly one 1 and for the rest only zeros, corresponding to the crisp case, the total fuzzy local pheromone update from (4.18) reduces to the crisp case from (4.17). ■

Similarly, we must also be sure that the order in which the pheromone deposits are processed does not affect the final result of the global pheromone update. We can easily prove the following theorem:

Theorem 4.2 *The global pheromone update for multiple ants is invariant with respect to the order of the pheromone deposits by the individual ants.*

Proof In the generalized global pheromone update rule from (4.15), the operations involving the solutions from $\mathcal{S}_{\text{trial}}$ are all either unions, or sums, which are invariant to the order of their operands. Since crisp ACL is a special case of fuzzy ACL, this automatically holds for crisp ACL as well. ■

4.4.2 Total Pheromone Update

We will derive the generalized total pheromone update from both generalized local and global pheromone update rules. Using the indicator vector from (4.13), the pheromone levels at the end of a trial k, but *before* the global pheromone update, have been updated by the *local* pheromone updates of all ants as follows:

$$\boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(T) = \left(\prod_{s \in \mathcal{S}_{\text{trial}}(k)} (\mathbf{1} - \gamma \mathcal{I}_{\mathbf{u},s}) \right) (\boldsymbol{\tau}_{\mathbf{u}}(k) - \tau_0 \mathbf{1}) + \tau_0 \mathbf{1}. \tag{4.19}$$

Then, the *global* pheromone update is performed as follows:

$$\boldsymbol{\tau}_{\mathbf{u}}(k+1) = (\mathbf{1} - \rho \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}) \boldsymbol{\tau}_{\mathbf{u}}^{\text{local}}(T) + \rho \sum_{s \in \mathcal{S}_{\text{trial}}} J^{-1}(s) \mathcal{I}_{\mathbf{u},s}. \tag{4.20}$$

The generalized *total* pheromone update can now be derived as:

$$\begin{aligned} \boldsymbol{\tau}_{\mathbf{u}}(k+1) = (\mathbf{1} - \rho \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}) & \left\{ \left(\prod_{s \in \mathcal{S}_{\text{trial}}(k)} (\mathbf{1} - \gamma \mathcal{I}_{\mathbf{u},s}) \right) (\boldsymbol{\tau}_{\mathbf{u}}(k) - \tau_0 \mathbf{1}) + \tau_0 \mathbf{1} \right\} \\ & + \rho \sum_{s \in \mathcal{S}_{\text{trial}}} J^{-1}(s) \mathcal{I}_{\mathbf{u},s}, \end{aligned} \tag{4.21}$$

in which k counts the number of trials since the initialization of the algorithm.

Recall that we are considering the update of all pheromone levels for a certain action $\mathbf{u}$ at the same time. Also keep in mind that all operations are performed element-wise.

4.4.3 Lower Bound on the Pheromone Levels

We can derive the same lower bound on the pheromone levels in the generalized case as in Proposition 3.4:

Proposition 4.3 *If τ_0 is chosen such that $0 < \tau_0 \leq J_{\max}^{-1}\rho$, with $J_{\max}^{-1}$ the inverse of the highest possible cost, the lower bound on the pheromone levels is $\boldsymbol{\tau}_{\mathbf{u}}(k) \geq \tau_0 \mathbf{1}, \forall k$.*

Proof Following the same reasoning as in (3.19), we can show by induction that a pheromone

vector can never have elements that are smaller than τ_0:

$$\begin{aligned}
\boldsymbol{\tau}_{\mathbf{u}}(0) &= \tau_0 \mathbf{1} \\
\boldsymbol{\tau}_{\mathbf{u}}(k+1) &= (\mathbf{1} - \rho \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}) \underbrace{\left\{ \left(\prod_{s \in \mathcal{S}_{\text{trial}}(k)} (\mathbf{1} - \gamma \mathcal{I}_{\mathbf{u},s}) \right) (\boldsymbol{\tau}_{\mathbf{u}}(k) - \tau_0 \mathbf{1}) + \tau_0 \mathbf{1} \right\}}_{\geq \tau_0 \mathbf{1} \quad \text{by induction}} \\
&\quad + \rho \sum_{s \in \mathcal{S}_{\text{trial}}} J^{-1}(s) \mathcal{I}_{\mathbf{u},s} \\
&\geq (\mathbf{1} - \rho \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}) \tau_0 + \rho \sum_{s \in \mathcal{S}_{\text{trial}}} J_{\max}^{-1} \mathcal{I}_{\mathbf{u},s} \\
&= \tau_0 \mathbf{1} + \rho \left\{ J_{\max}^{-1} \sum_{s \in \mathcal{S}_{\text{trial}}} \mathcal{I}_{\mathbf{u},s} - \tau_0 \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}} \right\} \geq \tau_0 \mathbf{1},
\end{aligned}$$

in which we have used that $J_{\max}^{-1} \geq \tau_0$ and that $\sum_{s \in \mathcal{S}_{\text{trial}}} \mathcal{I}_{\mathbf{u},s} \geq \bigcup_{s \in \mathcal{S}_{\text{trial}}} \mathcal{I}_{\mathbf{u},s} = \mathcal{I}_{\mathbf{u},\mathcal{S}_{\text{trial}}}$. ■

4.5 Experiments: 2D Navigation with Variable Damping

This section presents an example application of both crisp and fuzzy ACL to a continuous-state dynamic system. The dynamic system under consideration is a two-dimensional (2D) simulated navigation problem and it is similar to the one described in (Buşoniu et al., 2008). Note that it is not our purpose to demonstrate the superiority of ACL over any other method for this specific problem. Rather we want to demonstrate the functioning of the algorithm and compare the results for both its versions.

4.5.1 Problem Formulation

A vehicle, modeled as a point-mass of 1 kg, has to be steered to the origin of a two-dimensional flat surface from any given initial position in an optimal manner. The vehicle experiences a damping that varies non-linearly over the surface. The state of the vehicle is defined as $\mathbf{x} = \begin{bmatrix} c_1 & v_1 & c_2 & v_2 \end{bmatrix}^{\mathrm{T}}$, with $c_1, c_2 \in [-5, 5][\mathrm{m}]$ and $v_1, v_2 \in [-2, 2][\mathrm{m/s}]$ the position and velocity in the direction of each of the two principal axes respectively. The control input to the system $\mathbf{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^{\mathrm{T}}$ is a two-dimensional force. The dynamics of the vehicle in continuous-time are:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -b(c_1, c_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -b(c_1, c_2) \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}(t),$$

where the damping $b(c_1, c_2)$ is modeled by an affine sum of two Gaussian functions:

$$b(c_1, c_2) = b_0 + \sum_{i=1}^{2} b_i \exp \left[- \sum_{j=1}^{2} \frac{(c_j - m_{j,i})^2}{\sigma_{j,i}^2} \right],$$

for which the values of the parameters are $b_0 = 0.5$, $b_1 = b_2 = 8$, and $m_{1,1} = 0$, $m_{2,1} = -2.3$, $\sigma_{1,1} = 2.5$, $\sigma_{2,1} = 1.5$ for the first Gaussian, and $m_{1,2} = 4.7$, $m_{2,2} = 1$, $\sigma_{1,2} = 1.5$, $\sigma_{2,2} = 2$ for the second Gaussian. The damping profile can be seen in Figure 4.5, where darker shading means more damping. The system is sampled with a sampling time $T_\mathrm{s} = 0.2\mathrm{s}$. The parameters used in the ACL algorithm and their values for the experiments in this section are listed in Table 4.1. The values for γ and ρ have been chosen on the basis of the results of Section 3.6. The number of ants M has been chosen to be relatively low compared to the number of state-action pairs in the problems (i.e., respectively 6 561 and 13 608 for the experiments in Section 4.5.2 and Section 4.5.3). The values for τ_0 and α are very standard, T is chosen to be sufficiently large for the ants to find the goal while learning the control policy, and K is chosen large enough for the algorithm to converge. In Chapter 5, the influence of γ, ρ, and M on the learning performance will be studied further.

Table 4.1: The ACL parameters and their values for the experiments in this section.

Parameter	Value	Meaning
M	200	number of ants
τ_0	0.001	initial pheromone level
γ	0.01	local pheromone decay rate
ρ	0.1	global pheromone decay rate
α	3	exponent of pheromone level in action selection rule
T	150	maximal number of interaction steps with the system
K	200	maximal number of trials

The ants are randomly initialized over the complete state space at the start of each trial. An ant terminates its trial when its position and velocity in both dimensions are within a bound of $\pm 0.25\mathrm{m}$ and $\pm 0.05\mathrm{m/s}$ from the goal $\mathbf{x}_\mathrm{g} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\mathrm{T}$ respectively. Each experiment is carried out 30 times and the performance measures from Section 3.6.1 are recorded for every trial.

4.5.2 Regular Partitioning and Quadratic Cost Function

In the first experiments, we consider using a symmetrical partitioning of the state space. The partitioning is as follows:

- For the position in both dimensions c_1, c_2: $\{-5, -2, -0.3, -0.1, 0, 0.1, 0.3, 2, 5\}$.
- For the velocity in both dimensions v_1, v_2: $\{-2, 0, 2\}$.

The action set contains 9 actions, namely the Cartesian product of the sets $\{-1, 0, 1\}$ for both dimensions. The quadratic cost function from (3.2) is used with the matrices $\mathbf{Q} = \mathrm{diag}(0.2, 0.1, 0.2, 0.1)$ and $\mathbf{R} = 0$. The cost function thus only takes into account the deviation of the state from the goal. We will run the fuzzy Q-iteration algorithm from (Buşoniu et al., 2008) for this problem with the same state space partitioning in order to derive the optimal policy to which we can compare the policies derived by both versions of the ACL algorithm.

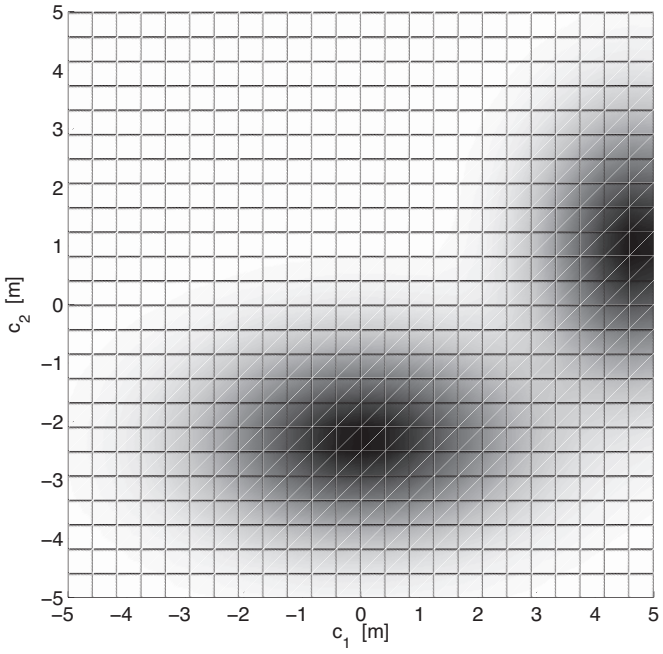


Figure 4.5: Illustration of the two dimensional space from these experiments. There are two Gaussian-shaped regions of stronger damping. Here, darker shades mean more damping.

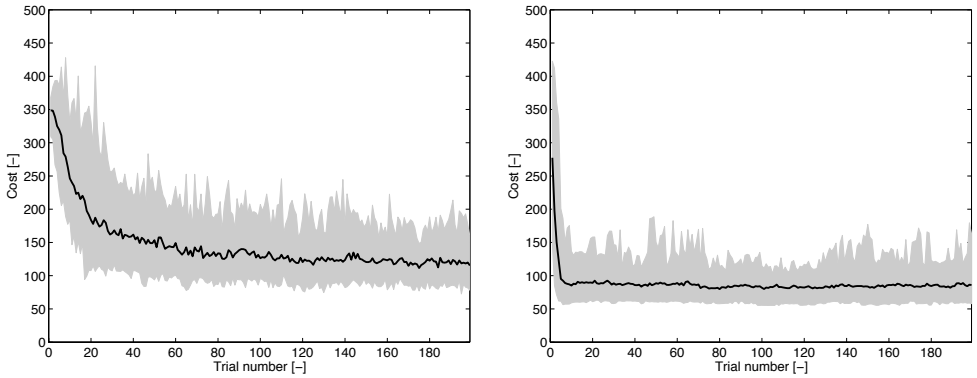
Cost of the Policy

The first performance measure considered represents the cost of the control policy as a function of the number of trials, as was introduced by (3.47). The cost of the control policy is measured as the average cost of a set of trajectories resulting from simulating the system controlled by the policy and starting from a set of 100 predefined initial states, uniformly distributed over the state space. Figure 4.6 shows these plots.

Each experiment is carried out 30 times in order to study the repeatability of the results. The black line in the plots is the average cost over these 30 experiments and the gray area represents max-min range. From Figure 4.6 it can be concluded that for these experiments, the converging trend for the results with the crisp ACL algorithm is not as fast and smoothly compared to that of fuzzy ACL. The average cost for crisp ACL more or less converges to a larger value than the average cost resulting from fuzzy ACL. Furthermore, the gray region for crisp ACL is larger than that for fuzzy ACL. This means that there is a larger variety of policies found at the end of the 30 experiments for crisp ACL than for fuzzy ACL.

Policy Variation

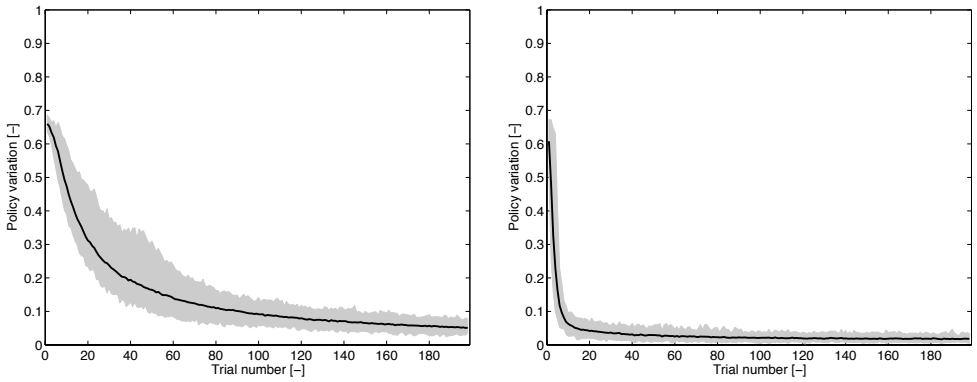
The performance of the algorithm can also be measured by the fraction of discrete bins (crisp version), or cores of the membership functions (fuzzy version) for which the policy has changed at the end of a trial compared to the start of the trial. This measure was called



(a) Crisp ACL (b) Fuzzy ACL

Figure 4.6: The cost of the control policy as a function of the number of trials passed since the start of the experiment. The black line is the average over the 30 experiments and the gray area represents the max-min range.

the *policy variation* in Section 3.6.1. The policy variation as a function of the trials for both crisp and fuzzy ACL is depicted in Figure 4.7.



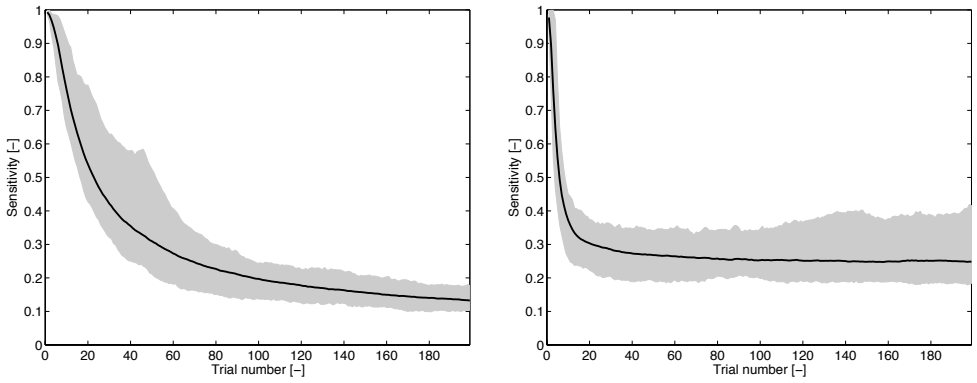
(a) Crisp ACL (b) Fuzzy ACL

Figure 4.7: Performance of the algorithm in terms of policy variation. The black line is the average over the 30 experiments and the gray area represents the max-min range.

It shows that for both ACL versions, the policy variation never really becomes completely zero and confirms that crisp ACL converges slower compared to fuzzy ACL for these experiments. The observation that the policy variation never becomes completely zero indicates that there are some states for which one or more actions result in nearly the same cost.

Sensitivity of the Policy

Figure 4.8 shows the evolution of the sensitivity of the policy, as defined by (3.48).



(a) Crisp ACL (b) Fuzzy ACL

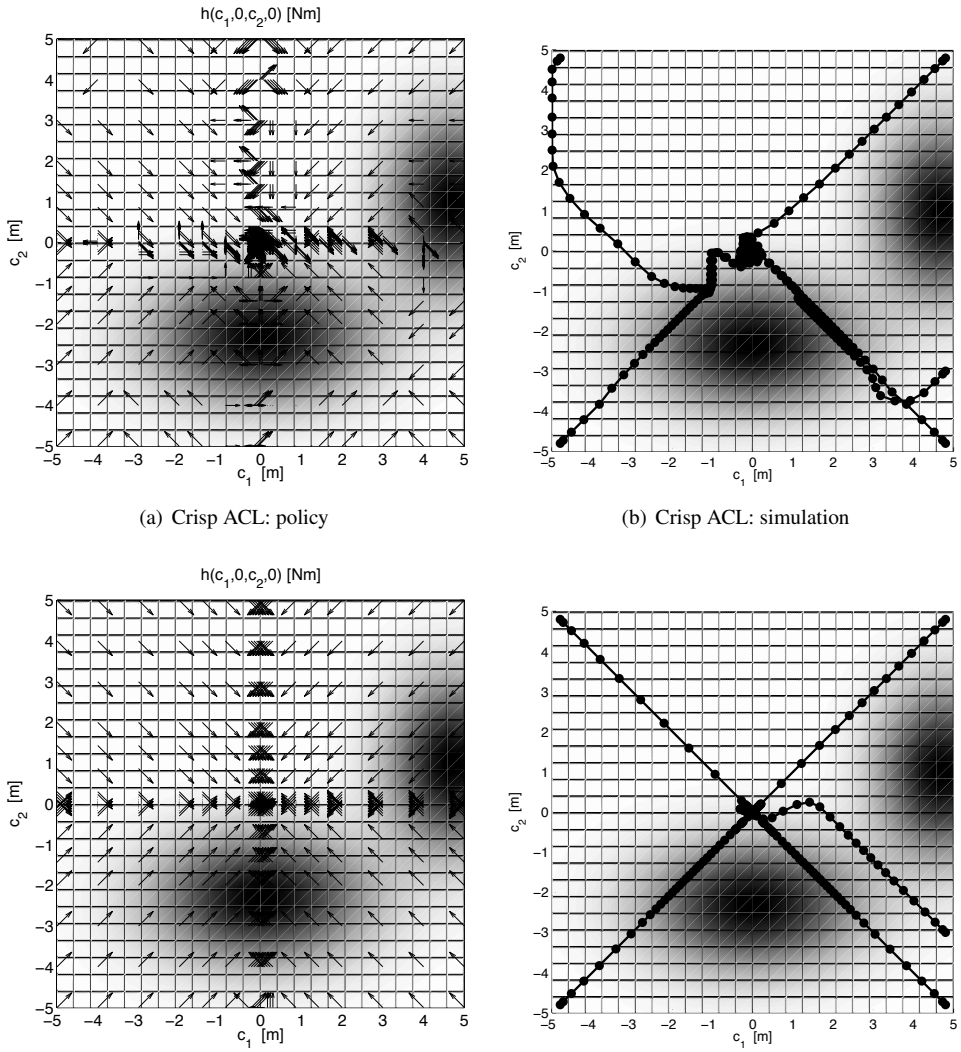
Figure 4.8: Evolution of the sensitivity of the control policy as a function of the number of trials. The black line is the average over the 30 experiments and the gray area represents the max-min range.

The figure shows that the sensitivity decreases with the number of trials, explaining the decrease in the policy variation from Figure 4.7. The fact that it does not converge to zero explains that there remains a probability larger than zero for the policy to change. This seems to confirm the hypothesis that ACL algorithms are potentially capable of adapting the policy to changes in the cost function, due to changing system dynamics. The observation that crisp ACL results in a lower sensitivity than fuzzy ACL may have to do with the fact that in crisp ACL, the pheromone update for a given continuous state-action pair results in a pheromone deposit at one discrete bin only, while in fuzzy ACL such a pheromone deposit is distributed over several fuzzy membership functions. In crisp ACL, this may result in a strong difference between two pheromone levels associated with the same discrete state, but different action. In fuzzy ACL, the difference between two pheromone levels associated with the same membership function and different actions may be less strong because of this.

The Final Policy and Simulation of the System

Finally, in Figure 4.9, we present the best policies resulting from the experiments and the behavior of a simulated vehicle controlled by these policies for both crisp and fuzzy ACL.

In the case of crisp ACL, Figure 4.9(a) depicts a slice of the policy for zero velocity. It shows the mapping of the positions in both dimensions to the input. Figure 4.9(b) presents the trajectories of the vehicle for various initial positions and zero initial velocity. The mapping is shown for a grid three times finer than the partitioning grid used in these experiments. For the crisp case, the states in between the centers of the partition bins are discretized to the nearest center. Figure 4.9(c) and Figure 4.9(d) present the results for fuzzy ACL. Comparing these results with those from the crisp ACL algorithm, it shows that for crisp ACL, the trajectories of the vehicle go almost direct to the goal state, but not as direct as with fuzzy ACL. Especially the top-left trajectory seems to sub-optimal. For fuzzy ACL, the trajectories are very smooth and seem to be more optimal; however, for both crisp and fuzzy ACL, the vehicle does not avoid the regions of stronger damping at all. The policy of fuzzy ACL is much more regular



(a) Crisp ACL: policy

(b) Crisp ACL: simulation

(c) Fuzzy ACL: policy

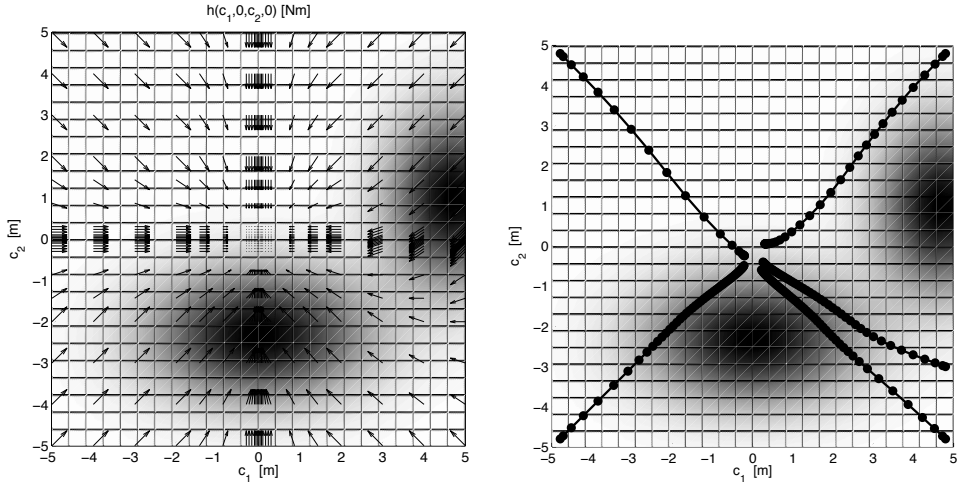
(d) Fuzzy ACL: simulation

Figure 4.9: The figures on the left present a slice of the best resulting policy for zero velocity obtained by crisp ACL (top) and fuzzy ACL (bottom) in one of the 30 runs. The figures show the control input for a grid of positions, three times finer than the partitioning grid in these experiments. The multi-Gaussian damping profile is shown, where darker shades represent regions of more damping. The figures on the right show the trajectories of the vehicle under these policies for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

compared to the one obtained by crisp ACL.

In order to verify the optimality of the resulting policies, we also present the optimal policy and trajectories obtained by the fuzzy Q-iteration algorithm from (Buşoniu et al., 2008).

This algorithm is a model-based reinforcement learning method developed for continuous state spaces and is guaranteed to converge to the optimal policy on the partitioning grid. Figure 4.10 present the results for fuzzy Q-iteration.



(a) Fuzzy Q-iteration: Policy

(b) Fuzzy Q-iteration: simulation

Figure 4.10: The baseline optimal policy derived by the fuzzy Q-iteration algorithm. The left figure presents a slice of the policy for zero velocity. The figure on the right shows the trajectories of the vehicle under this policy for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

It can be seen that the optimal policy from fuzzy Q-iteration is more similar to the one obtained by fuzzy ACL, but that it manages to avoid the regions of stronger damping somewhat better, though still not perfectly. The term "perfectly", must however be understood in the light of the cost function that has been used. An optimal control policy behaves perfectly by definition. If we feel that it should behave even better, according to some other notion of the actual control objective, we should adapt the cost function to better match our intended objective.

Even with the optimal policy, the regions of stronger damping are not completely avoided. The reason for this may be that the cost function is ill defined and does not completely relate to our intended control objective. Another reason may be that our particular choice of the cost function in these experiments is not capable of clearly discriminating between the real optimal policy and several sub-optimal policies. In the next section, we will use a different cost function, as well as a finer state space partitioning in order to see if we can improve the learning performance like this.

4.5.3 Non-Regular Partitioning and Time-Spent Cost Function

In these experiments, we consider using a non-regular partitioning of the state space. This partitioning of the position space is composed of a baseline grid $\{-5, -0.3, 0, 0.3, 5\}$ and adding to it, extra grid lines around each Gaussian damping region. This partitioning is

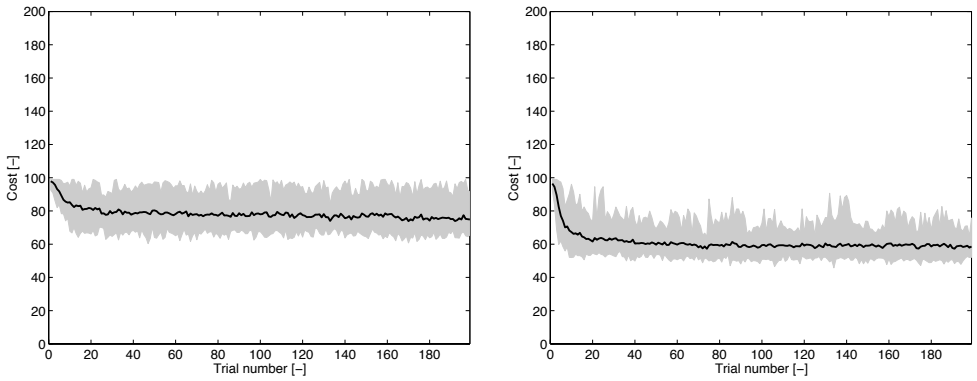
identical to what is used in (Buşoniu et al., 2008) and is as follows:

- For the position c_1: $\{-5, -3.75, -2, -0.3, 0, 0.3, 2, 2.45, 3.5, 3.75, 4.7, 5\}$.
- For the position c_2: $\{-5, -4.55, -3.5, -2.3, -2, -1.1, -0.6, -0.3, 0, 0.3, 1, 2.6, 4, 5\}$.
- For the velocity in both dimensions v_1, v_2: $\{-2, 0, 2\}$.

The action set is the same as in the Section 4.5.2 and contains 9 actions, namely the Cartesian product of the sets $\{-1, 0, 1\}$ for both dimensions. As the cost function, the number of time steps to the goal is used. We will run the fuzzy Q-iteration algorithm from (Buşoniu et al., 2008) for this problem with the same state space partitioning in order to derive the optimal policy to which we can compare the policies derived by both versions of the ACL algorithm.

Cost of the Policy

Again, the first performance measure considered is the cost of the policy from (3.47). Figure 4.11 shows these plots.



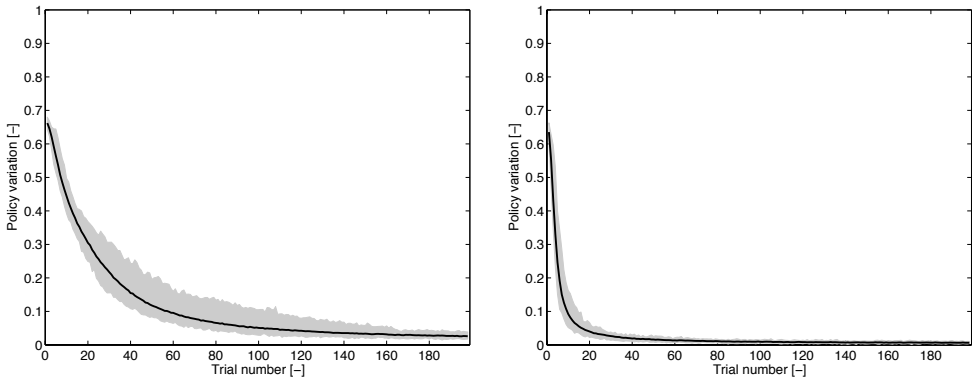
(a) Crisp ACL (b) Fuzzy ACL

Figure 4.11: The cost of the control policy as a function of the number of trials passed since the start of the experiment. The black line is the average over the 30 experiments and the gray area represents the max-min range.

It can be seen that the difference between the initial cost (of a random policy) and the final policy is much smaller compared to what we have seen in Figure 4.6. For both crisp and fuzzy ACL, the trend of the average cost converges in about the same number of trials, but the average final cost using fuzzy ACL is smaller. With both algorithms, there is a lot of variation between the 30 runs of the experiment.

Policy Variation

The policy variation as a function of the trials for both crisp and fuzzy ACL is depicted in Figure 4.12.



(a) Crisp ACL (b) Fuzzy ACL

Figure 4.12: Performance of the algorithm in terms of policy variation. The black line is the average over the 30 experiments and the gray area represents the max-min range.

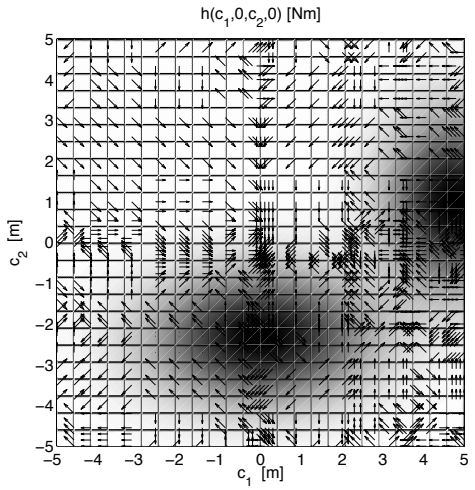
Here it is seen that the policy obtained by fuzzy ACL more rapidly takes on its final form, compared to crisp ACL. These results vary only slightly over the 30 runs of the experiments, indicating that considering the results from Figure 4.11, although the cost of the policy may vary a lot from experiment to experiment, the cost of the policy itself always converges quickly. This also indicates that in these experiments, the algorithm seems to get easily stuck in a local optimum. The reason may be found in the random proportional action selection rule that is used, because the exploration depends on the difference between the pheromone levels for the various actions. If the cost function that is used results in a rapid increase of the pheromone levels associated with sub-optimal actions, it immediately becomes unlikely for the ants to choose different actions.

The Final Policy and Simulation of the System

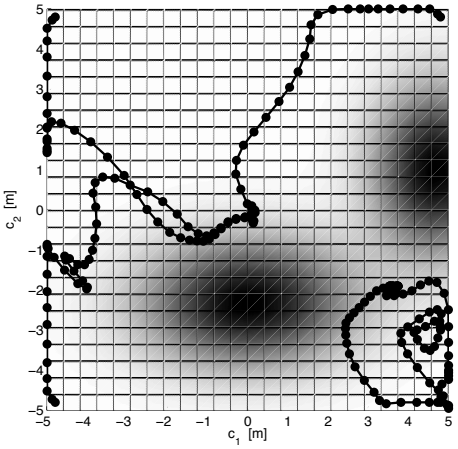
The best policy resulting from the experiments and the behavior of a simulated vehicle controlled by these policies are shown in Figure 4.13 for both crisp and fuzzy ACL.

There is a clear difference in the policy and the behavior of the vehicle controlled by this policy between the crisp and the fuzzy case. With crisp ACL, in some regions of the state space, the policy seems to steer the vehicle around the regions of stronger damping. However, in other regions, the policy appears to be quite random. When looking at the trajectories of the vehicle, the policy controls the vehicle to the goal state in a very sub-optimal way. Especially when the vehicle starts in the lower-right corner, the control policy is incapable of controlling the vehicle to the goal. For fuzzy ACL, however, the policy looks very smooth and also appears to be able to steer the vehicle around the regions of stronger damping much more optimally. In order to verify the optimality of the resulting policies, we show the results obtained by the fuzzy Q-iteration algorithm in Figure 4.14.

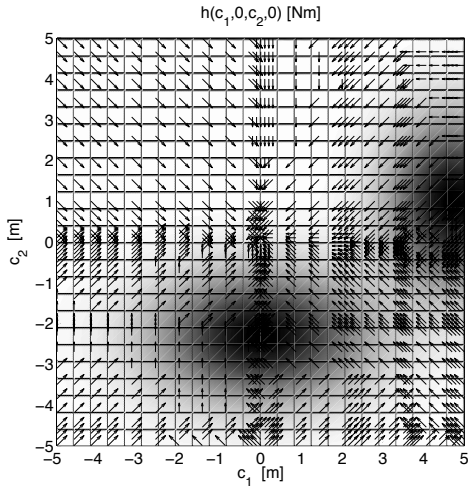
Indeed, the control policy and the trajectories of the vehicle are very similar to those obtained by fuzzy ACL. The cost function that is used appears to represent more closely the objective of avoiding the regions of stronger damping, while driving as quickly as possible to the center of the field before coming to a standstill. Also the state space partitioning, which



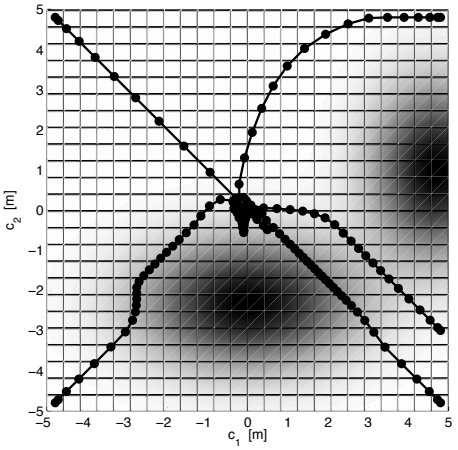
(a) Crisp ACL: policy



(b) Crisp ACL: simulation



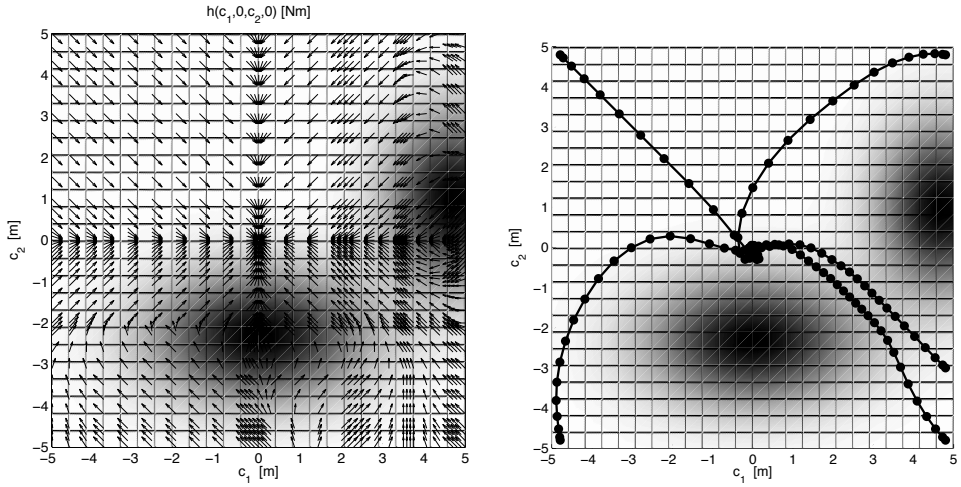
(c) Fuzzy ACL: policy



(d) Fuzzy ACL: simulation

Figure 4.13: The figures on the left present a slice of the best resulting policy for zero velocity obtained by crisp ACL (top) and fuzzy ACL (bottom) in one of the 30 runs. The figures show the control input for a grid of positions, three times finer than the partitioning grid in these experiments. The multi-Gaussian damping profile is shown, where darker shades represent regions of more damping. The figures on the right show the trajectories of the vehicle under these policies for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

was denser around the regions of stronger damping, helped to make the policy more accurate in these regions. The slice of the policy for zero velocity clearly shows the action vectors "curving" around these Gaussian regions. Of course, it can still be questioned if this is the



(a) Fuzzy Q-iteration: Policy

(b) Fuzzy Q-iteration: simulation

Figure 4.14: The baseline optimal policy derived by the fuzzy Q-iteration algorithm. The left figure presents a slice of the policy for zero velocity. The figure on the right shows the trajectories of the vehicle under this policy for various initial positions and zero initial velocity. The markers indicate the positions at twice the sampling time.

desired behavior. If the answer to this question is negative, the cost function and possibly also the state space partitioning should be tuned even further.

For these experiments, fuzzy ACL has turned out to be much more capable of finding the optimal control policy compared to crisp ACL. Also, these experiments illustrate the importance of choosing an appropriate cost function and state space partitioning. In general, the more information is available about the problem structure and the dynamics of the system, the better it is possible to find a suitable state space partitioning and cost function.

4.6 Concluding Remarks

In this chapter, we have discussed ACL in continuous state domains. The partitioning of the state space is a crucial aspect for ACL and two versions have been presented. In crisp ACL, the state space is partitioned using bins, such that each value of the state maps to exactly one bin. Fuzzy ACL, on the other hand, uses a partitioning of the state space with fuzzy membership functions. In this chapter, the fuzzy membership functions have been chosen to be triangular shaped, but the algorithm allows for differently shaped membership functions as well, as long as they are normalized. In the case of fuzzy ACL, each value of the state maps to the membership functions to a certain membership degree. Both ACL algorithms are extensions of the ACL framework, presented in Chapter 3.

The applicability of ACL to optimal control problems with continuous-valued states has been outlined and demonstrated on the non-linear control problem of two-dimensional navigation with variable damping. We have studied two sets of experiments that differed in the choice of state space partitioning and the cost function. For both sets of experiments, fuzzy

Q-iteration was used to compute the optimal policy to which the results for crisp and fuzzy ACL could be compared. The first set of experiments used a regular state space partitioning grid and a quadratic cost function. The results showed a converging trend of both the crisp and fuzzy version of the algorithm to suboptimal policies. However, with fuzzy ACL this converging trend was much steeper and the cost of its resulting policy did not change as much over repetitive runs of the algorithm as it did for crisp ACL. Its resulting policy was also better compared to crisp ACL. We have studied the convergence of the pheromone levels in relation to the policy using the performance measures of policy variation and sensitivity. These measures showed that ACL progresses as a function of the number of trials to a decreasingly sensitive control policy meaning that a small change in the pheromone levels will not result in a large change in the policy. In all cases, even with the optimal policy from fuzzy Q-iteration, however, the control policy could not avoid the regions of stronger damping well. Therefore, in the second set of experiments, the cost function was changed to reflect the number of time steps needed to arrive at the goal, and the partitioning was made denser around the regions of stronger damping. As a result, the optimal policy derived by fuzzy Q-iteration turned out to be well capable of steering the vehicle around the Gaussian regions. The crisp ACL algorithm was incapable of finding a close to optimal policy, while fuzzy ACL did derive a near-optimal policy. In the next chapter, we will perform a more extensive computer simulation study of the effect of the ACL learning parameters on the behavior of both crisp and fuzzy ACL.

Chapter 5

Simulation Experiments

This chapter presents a series of simulation experiments of crisp and fuzzy ACL on the pendulum swing-up and stabilization problem. The performance of both algorithms is evaluated and compared for various parameter settings.

5.1 Introduction

The ACL framework has been presented in the previous chapters as a multi-agent control policy learning methodology. The updating of the pheromone levels act as a reinforcing mechanism and stimulates the ants to choose actions that are likely to lead to the goal state in an optimal way. The pheromone update rules have been generalized to the continuous domain using fuzzy membership functions that partition the continuous state space. So far, we have analyzed the convergence properties of the algorithm and studied some properties of the local and global pheromone update rules. We have also discussed the behavior of ACL when applied to grid search problems and to the problem involving two-dimensional navigation with variable damping.

In this chapter, we will perform a number of simulation experiments in order to analyze the behavior of ACL on a non-linear control system with a continuous state space. The non-linear control problem that is used in all these simulation experiments is the inverted pendulum swing-up and stabilization problem, which will be introduced in Section 5.2. Using the same control problem in all the experiments allows us to properly compare the results. We analyze the influence of several important parameters in Section 5.3: the local and global pheromone decay rates, the number of ants, and the partitioning of the state space.

5.2 Pendulum Swing-Up and Stabilization

The pendulum swing-up and stabilizing problem is a nice abstraction of more complex robot control problems, like the stabilization of a walking humanoid robot. The behavior can be easily analyzed, while the learning problem is challenging. This problem is used in all the following simulation experiments with ACL. We analyze how the behavior and the learning performance relate to some of the parameters in the algorithm, such as the global and

local pheromone decay rate, the number of ants, and the number of quantization levels (for crisp ACL) or membership functions (for fuzzy ACL). The performance of crisp and fuzzy ACL is compared. This section first formulates the problem and describes the set-up of the experiments.

5.2.1 Problem Formulation

The pendulum is modeled as a pole, attached to a pivot point at which a motor exerts a torque. The objective is to get the pendulum from a certain initial position to its unstable upright position, and to keep it stabilized within a certain band around that unstable position. The torque is, however, limited such that it is not possible to move the pendulum to its upright position in one movement. The pendulum must thus swing back and forth to accumulate enough energy for swinging up. The pendulum must also stop in time in order to be able to balance in the unstable equilibrium. This makes the solution non-trivial and suitable for learning algorithms. The non-linear state equations of the pendulum are given by:

$$\begin{aligned} \dot{\theta}(t) &= \omega(t) \\ J\dot{\omega}(t) &= K_{\mathrm{m}}u(t) - mgL\sin(\theta(t)) - D\omega(t), \end{aligned} \tag{5.1}$$

with $\theta(t) = x_1(t)$ and $\omega(t) = x_2(t)$ the state variables, representing the angle and angular velocity of the pole in continuous time respectively. The positive direction of the state variables is indicated in Figure 5.1. Furthermore, $u(t)$ is the applied torque and the other parameters with their values as used in the simulations are listed in Table 5.1.

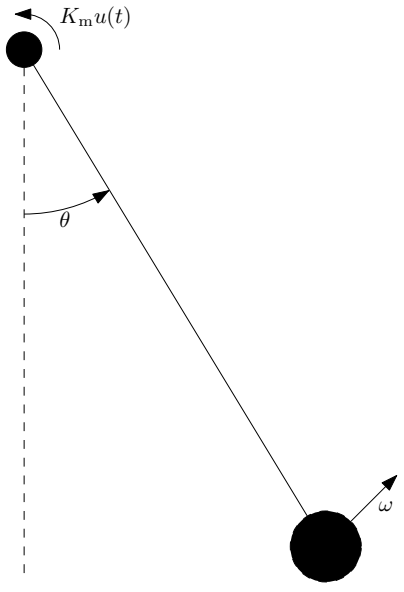


Figure 5.1: Positive direction of θ and ω for the pendulum.

Table 5.1: The parameters of the pendulum model and their values used in the experiments.

Parameter	Value	Unit	Meaning
J	0.005	$\mathrm{kg} \cdot \mathrm{m}^2$	pendulum inertia
K_m	0.1	-	motor gain
D	0.01	$\mathrm{kg} \cdot \mathrm{s}^{-1}$	damping
m	0.1	kg	mass
L	0.1	m	pendulum length
g	9.81	$\mathrm{m} \cdot \mathrm{s}^{-2}$	gravitational acceleration

5.2.2 Set-Up of the Experiments

The system is sampled with a sampling time $T_\mathrm{s} = 0.1\mathrm{s}$ and the states are discretized using bins or fuzzy membership functions, the centers of which define the discrete state space:

$$\mathcal{Q}_\theta = \left\{ 0, \frac{2\pi}{N_\theta}, \ldots, \frac{2\pi(N_\theta - 1)}{N_\theta} \right\} \tag{5.2}$$

$$\mathcal{Q}_\omega = \left\{ -\omega_\mathrm{max}, -\omega_\mathrm{max} + \frac{2\omega_\mathrm{max}}{N_\omega - 1}, \ldots, \omega_\mathrm{max}, \right\}, \tag{5.3}$$

where N_θ and $N_\omega \geq 3$ are the number of discretization bins for θ and ω respectively and ω_max is the maximum (absolute) angular velocity expected to occur. Note that N_θ must be even for crisp partitioning to make sure that both equilibria coincide with centers of the partitioning bins rather than fall exactly in between two elements, which would result in chattering of the discretized state when the pendulum is near one of the equilibria. For similar reasons, N_ω must be odd. The angle will be observed as $\theta \bmod 2\pi$. Unless stated otherwise, we take $N_\theta = 40$ and $N_\omega = 41$.

In all experiments, the action selection method is the Max-Boltzmann rule (3.6), or its fuzzy version (4.11). The ACL parameters that have the same value for all experiments in this chapter are listed in Table 5.2. These parameters are fixed in order to fairly compare the algorithm for the scenarios in Section 5.3. The initial pheromone level τ_0 is set to a sufficiently small value. The amount of exploration is constant during all experiments and the values of ϵ and α have been shown to work well. The maximal number of interaction steps with the system, T, is sufficiently long for the ants to find the goal from any initial state. The maximal number of trials, K, is sufficiently long for the algorithm to converge, or to be able to say that the algorithm will not be able to improve the policy any further. The results of the experiments in Section 5.3 are not expected to be much different for slightly different values of these parameters. Basically, smaller values for τ_0 and ϵ and larger values for α will result in less exploration, which may cause ACL to prematurely settle at a local optimum. Likewise, larger values for τ_0 and ϵ and smaller values for α will result in more exploration, possibly preventing ACL from settling at any optimum at all. However, as said before, slightly different values for these parameters are not expected to change much the behavior of ACL. The values of T and K should be taken large enough in order to have sufficient interaction with the system. Taking these values too large, however, unnecessarily increases the learning time.

At the start of each trial, the ants are randomly initialized over the state-space with a

Table 5.2: The ACL parameter values that are the same for all experiments in this chapter.

Parameter	Value	Meaning
τ_0	0.0001	initial pheromone level
ϵ	0.1	exploration probability
α	3	exponent of action selection rule
T	300	maximal number of interaction steps with the system
K	100	maximal number of trials

uniform probability. The goal state is $\mathbf{x}_g = \begin{bmatrix}\pi & 0\end{bmatrix}^{\mathrm{T}}$ and an ant is said to be in the goal state if its state satisfies $|\pi - \theta| \leq 0.1$ rad and $|\omega| \leq 0.1$ rad/s. A quadratic cost function (3.2) is used to measure the performance:

$$J(s) = J(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}) = \sum_{t=0}^{T-1} \mathbf{e}^{\mathrm{T}}(t+1)\mathbf{Q}\mathbf{e}(t+1) + \mathbf{R}u^2(t), \tag{5.4}$$

$$\mathbf{Q} = \begin{bmatrix}1 & 0\\ 0 & 0.1\end{bmatrix}, \quad \mathbf{R} = 0.05,$$

which has been explained in Section 3.2.1. The action set $\mathcal{U} = \{-0.8, 0, 0.8\}[\mathrm{Nm}]$ contains 3 actions, which will be referred to as full negative torque, zero torque, and full positive torque respectively. All experiments are carried out 30 times. The plots will show the average performance and the min-max area of worst-case and best-case performance.

5.3 Results

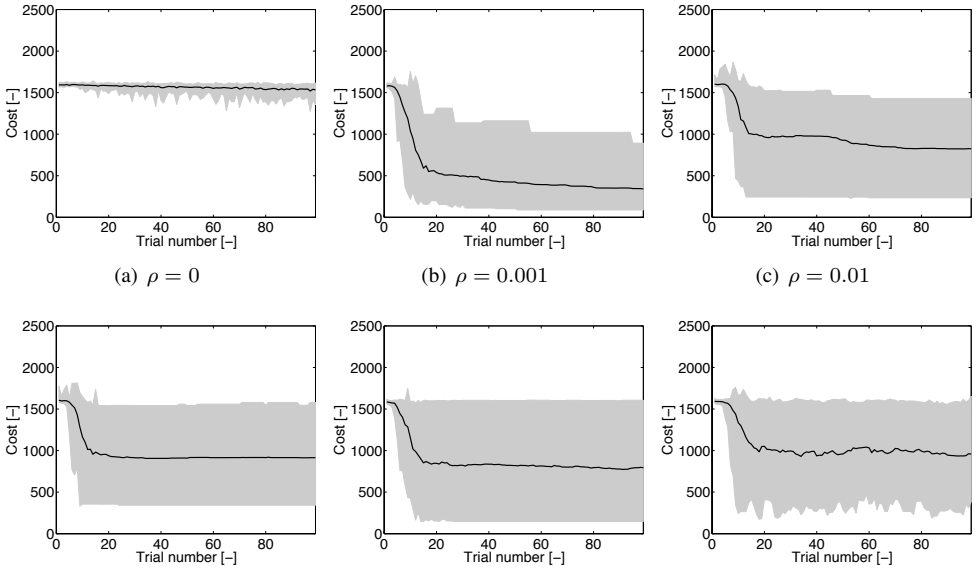
In the following simulation experiments, we study the effect of the global and local pheromone decay rates, the number of ants, and the number of state space partitioning levels on the performance of crisp and fuzzy ACL. The application that is used is the pendulum swing-up and stabilization problem.

5.3.1 Global Pheromone Decay Rate

In these experiments, we study the effect of the global pheromone decay rate, ρ, on the learning performance, in a similar way as we did in Section 3.6.3 for the case of a 1-dimensional grid search problem. The values of ρ are from the set $\rho \in \{0, 0.001, 0.01, 0.1, 0.5, 1\}$. We keep the other parameters constant over the experiments. The local pheromone decay rate is $\gamma = 0.01$, the number of ants is 250, and the other parameters are listed in Table 5.2. We study the behavior of both crisp and fuzzy ACL.

Crisp ACL

Figure 5.2 shows the simulation results obtained with the crisp ACL algorithm for varying values of the global pheromone decay rate, ρ.



(a) $\rho = 0$ (b) $\rho = 0.001$ (c) $\rho = 0.01$

(d) $\rho = 0.1$ (e) $\rho = 0.5$ (f) $\rho = 1$

Figure 5.2: Policy performance of crisp ACL for $\gamma = 0.01$ and varying ρ. The black line is the average over the 30 experiments and the gray area represents the max-min range.

The results do not show a very good convergence behavior of crisp ACL. For $\rho = 0$, which is actually a value that is not within the allowed range for ρ, viz. $(0, 1]$, the algorithm does not converge to any meaningful policy. This is obvious, as $\rho = 0$ means that there is no

global pheromone update, so that the quality of the policies found by the ants are not taken into account at all. The very slow trend downwards is due to the local pheromone update that stimulates ants to choose actions that have not been chosen before. This results in some convergence of the pheromone levels that is not related to the performance of the policies. The policy performance is also not expected to become much better with this mechanism. For large values of ρ, the pheromone deposits contribute significantly to the pheromone update and it can indeed be seen that this does not lead to convergence within 100 trials. For small values of ρ, the convergence is better. The best behavior can be seen for $\rho = 0.001$, where there is a continuing trend downwards, which would have continued if the experiment would not have stopped at 100 trials. The typical value of $\rho = 0.1$, as used in most papers on ACO, results here in fast convergence of the average performance for the 30 experiments, but to a higher cost than for $\rho = 0.001$. A simulation of the inverted pendulum controlled by the best final policy obtained with $\rho = 0.001$ from $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.3.

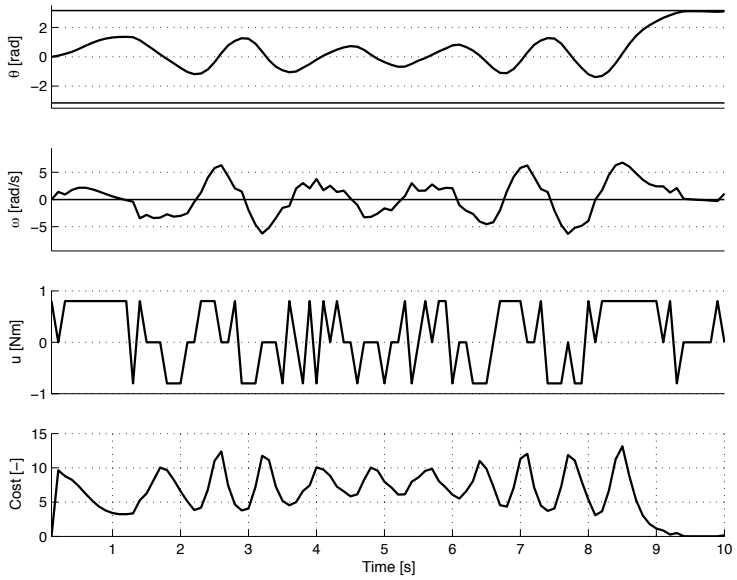
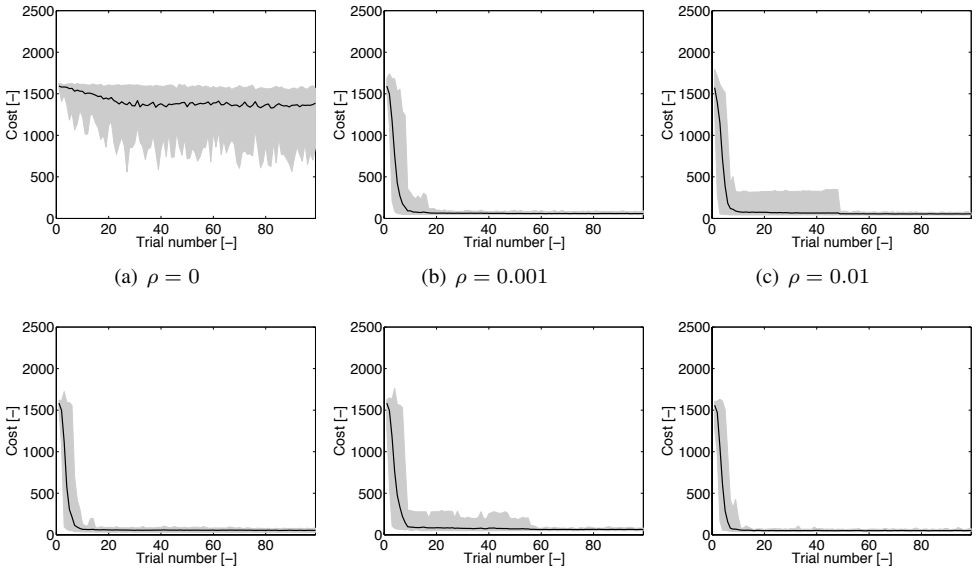


Figure 5.3: The inverted pendulum controlled from an initial state $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ by the final policy obtained by crisp ACL with $\rho = 0.001$ and $\gamma = 0.01$. The top two graphs show the trajectories of the states, while the third shows the input and the bottom graph shows the cost.

In this figure, from the top to the bottom plot, we see respectively the trajectory of the angle and the angular velocity over time, the control input, and the corresponding cost. It shows that the control policy manages to swing up and stabilize the inverted pendulum in the unstable equilibrium and that it does so by swinging the pole a couple of times back and forth before it accumulates enough energy to be swung up completely.

Fuzzy ACL

Figure 5.4 shows the simulation results obtained with the fuzzy ACL algorithm for varying values of the global pheromone decay rate, ρ.



(a) $\rho = 0$ (b) $\rho = 0.001$ (c) $\rho = 0.01$

(d) $\rho = 0.1$ (e) $\rho = 0.5$ (f) $\rho = 1$

Figure 5.4: Policy performance of fuzzy ACL for $\gamma = 0.01$ and varying ρ. The black line is the average over the 30 experiments and the gray area represents the max-min range.

Compared to the results with crisp ACL, the convergence of the control policy is much better with fuzzy ACL. For $\rho = 0$, we see a similar behavior, which is obvious, as it means that there is essentially no global pheromone update. For the other values of ρ, the convergence behavior is more or less similar to each other. Higher values of ρ mean faster adaption of the pheromone levels to newly obtained pheromone deposits, which can slightly be seen in the slightly slower convergence for $\rho = 0.001$ and the greater variation for $\rho = 0.5$. The clean convergence for $\rho = 1$ is difficult to explain, as we would have expected even more variation in the cost over the different experiments than with $\rho = 0.5$. The reason could be that because there is no noise in the system, a value of $\rho = 1$ is allowed and leads to the fastest possible convergence. We expect that if the state measurement of system would be corrupted by noise, the complete and immediate adaptation of the pheromone levels to an update with $\rho = 1$ would not result in convergence as is the case with crisp ACL, which suffers from state-discretization noise. From these experiments, it is not possible to conclude which value for the global pheromone decay rate is the best. The differences between the simulation results for $\gamma = 0.001$, 0.1, and 1 are small. The simulation experiments do show that the usual $\rho = 0.1$ leads to good results. The big improvement of the learning behavior with fuzzy ACL over crisp ACL can be explained by the absence of artificially introduced discretization noise. A simulation of the inverted pendulum controlled by the control policy learned by fuzzy ACL, $\rho = 0.1$, and for $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.5. This policy

swings up the pendulum much faster compared to the results in Figure 5.3.

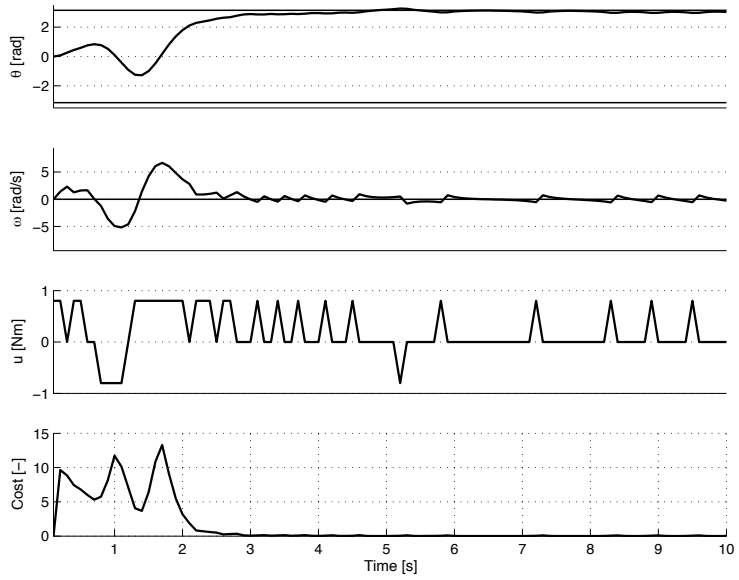
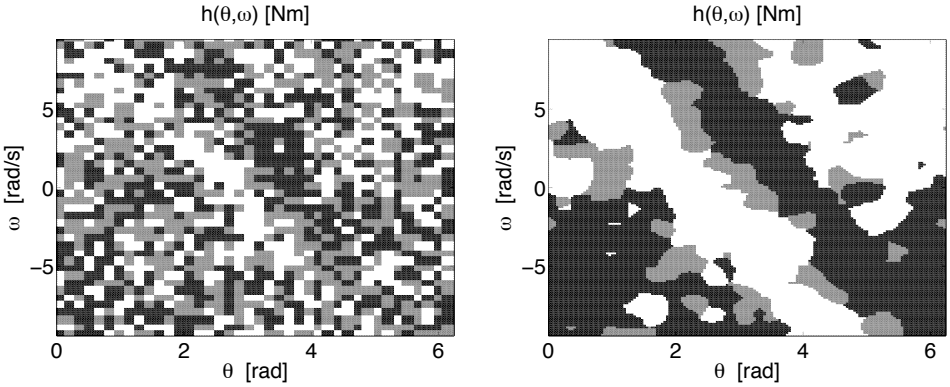


Figure 5.5: The inverted pendulum controlled from an initial state $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ by the final policy obtained by fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.01$. The top two graphs show the trajectories of the states, while the third shows the input and the bottom graph shows the cost.

The control policies for both crisp and fuzzy ACL are shown in Figure 5.6. These figures show the mapping from the state of the system to the control action, where white denotes maximal positive torque, gray is zero torque, and black is maximal negative torque. The mapping is plotted for a state grid four times finer than the state space partitioning used by the algorithms. It shows that for the crisp policy, evaluating the policy for more states does not result in a finer mapping. This is because the policy is not interpolated between the centers of the crisp partitioning bins. For fuzzy ACL, however, the finer grid reveals a meaningful interpolation between the centers of the membership functions and illustrates that this policy is really a continuous mapping. The other main observation from these plots is that the policy in Figure 5.6(b) is much more structured compared to the policy in Figure 5.6(a). From Figure 5.6(b) the control policy can be understood as follows. For angles near $\theta = 0$, or $\theta = 2\pi$, corresponding to the pendulum in its stable equilibrium, the control policy destabilizes the pendulum by providing maximum torque in the direction where the pendulum is moving. There is a curved transition plane where the control action changes from maximum negative torque (black) to maximum positive torque (white), or vice versa and results in the pendulum to slow down. For states near the unstable equilibrium $\theta = \pi$, the control policy is stabilizing and in a small area around $\theta = \pi$ and $\omega = 0$, the control action is zero torque. A similar structure can only very vaguely be identified in Figure 5.6(a). The policy appears to be heavily corrupted by noise. The reason why such a policy is still capable of controlling the pendulum, though sub-optimally, is that the momentum in motion systems makes the system somewhat robust to small variations in the control action.



(a) Crisp control policy.

(b) Fuzzy control policy.

Figure 5.6: The final control policy obtained for the inverted pendulum problem with both crisp and fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.01$. The colors black, white, and gray represent the inputs -0.8, 0.8, and 0 Nm respectively. For angles near $\theta = 0, 2\pi$, the policy represents a destabilizing controller, while near $\theta = \pi$, the policy represents a stabilizing controller.

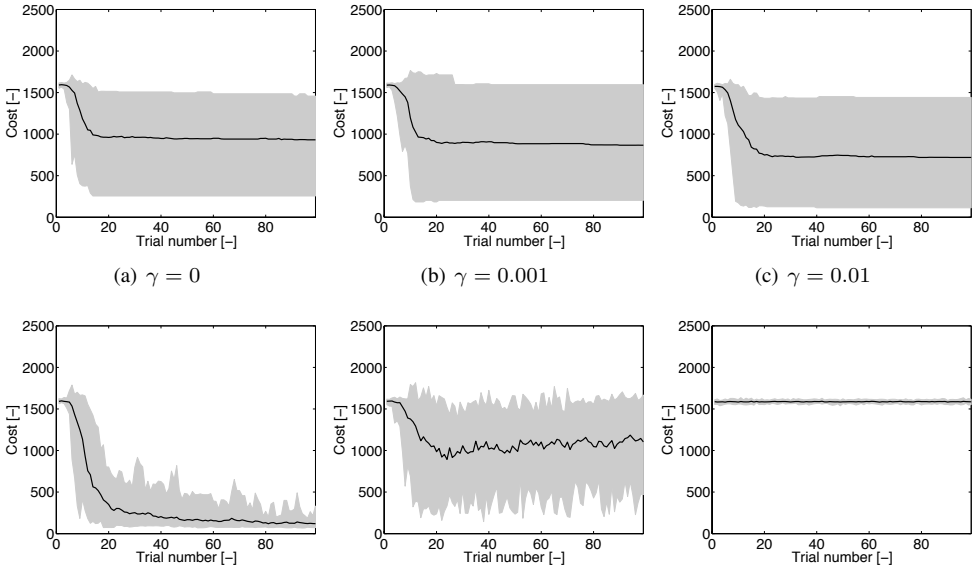
5.3.2 Local Pheromone Decay Rate

Similarly to Section 3.6.4, the effect of the local pheromone decay rate on the learning performance is studied. We keep the other parameters constant over the experiments. The global pheromone decay rate is equal to $\rho = 0.1$, the number of ants is 250, and the other parameters are listed in Table 5.2. The values of γ that we use are from the set $\gamma \in \{0, 0.001, 0.01, 0.1, 0.5, 1\}$. We study the behavior of both crisp and fuzzy ACL.

Crisp ACL

Figure 5.7 shows the simulation results obtained with the crisp ACL algorithm for varying values of the local pheromone decay rate, γ.

The first observation of the results is that the local pheromone update can really improve the learning performance, but only when it is chosen carefully. For low values of γ, the effect is minimal. For $\gamma = 1$, which is outside the allowed range for γ, viz. $[0, 1)$, there is no convergence at all, because the pheromone levels are reset to τ_0 after each step of an ant. For $\gamma = 0.5$, there is also no convergence. For $\gamma = 0.1$, however, the local pheromone update really improves the learning performance, as the average cost of the resulting control policy is much lower than that of the algorithm for smaller values of γ and the variation over the experiments also reduces considerably. The policy for this case is shown in Figure 5.11. A simulation of the inverted pendulum controlled by this control policy for $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.8. The control policy has indeed been improved as it swings up and stabilizes the pendulum faster than the policy that was learned with crisp ACL and $\gamma = 0$, as shown in Figure 5.3. It is however still much less optimal than the policy learned by fuzzy ACL for $\gamma = 0$ that could swing up and stabilize the pendulum within three seconds (Figure 5.5).



(a) $\gamma = 0$ (b) $\gamma = 0.001$ (c) $\gamma = 0.01$

(d) $\gamma = 0.1$ (e) $\gamma = 0.5$ (f) $\gamma = 1$

Figure 5.7: Policy performance of crisp ACL for $\rho = 0.1$ and varying γ. The black line is the average over the 30 experiments and the gray area represents the max-min range.

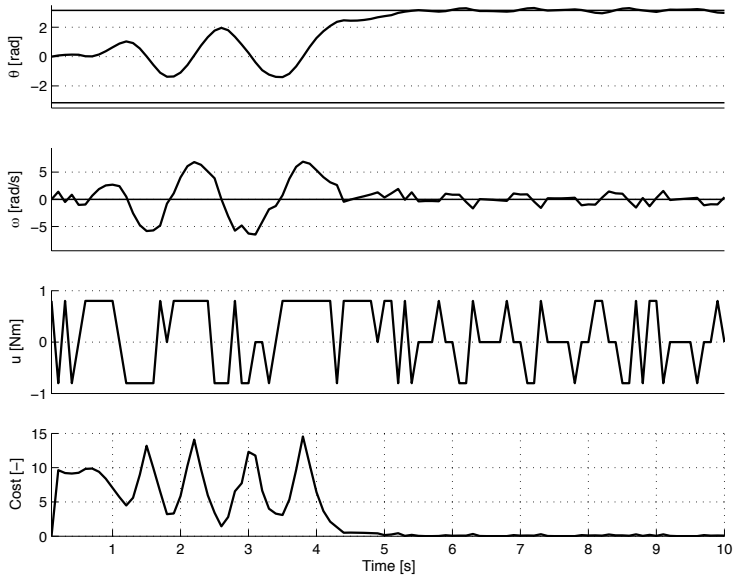
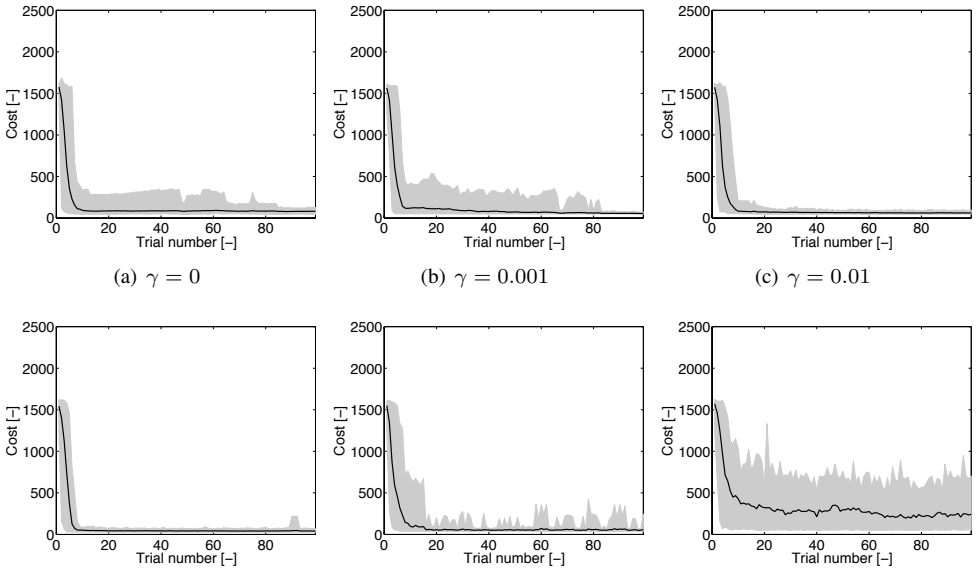


Figure 5.8: The inverted pendulum controlled from an initial state $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ by the final policy obtained by crisp ACL with $\rho = 0.1$ and $\gamma = 0.1$. The top two graphs show the trajectories of the states, while the third shows the input and the bottom graph shows the cost.

Fuzzy ACL

Figure 5.9 shows the simulation results obtained with the fuzzy ACL algorithm for varying values of the local pheromone decay rate, γ.



(a) $\gamma = 0$ (b) $\gamma = 0.001$ (c) $\gamma = 0.01$

(d) $\gamma = 0.1$ (e) $\gamma = 0.5$ (f) $\gamma = 1$

Figure 5.9: Policy performance of fuzzy ACL for $\rho = 0.1$ and varying γ. The black line is the average over the 30 experiments and the gray area represents the max-min range.

The effect of the local pheromone update rule on the learning performance of fuzzy ACL seems to be much smaller than it has on crisp ACL. For crisp ACL, choosing an appropriate value of γ clearly improved the performance of the algorithm. For fuzzy ACL, the value of $\gamma = 0.1$ results in the best performance, but smaller values are almost equally good. The hypothesis that can be derived from these results is that for control problems with noisy state transitions, the local pheromone update can greatly improve the results. If there is, however, no such noise in the system, the local pheromone update has only a minor effect. A simulation of the inverted pendulum controlled by the control policy for $\gamma = 0.1$ and for $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.10. The control policy is indeed even more optimal than the one obtained by fuzzy ACL and $\gamma = 0.01$ as shown in Figure 5.5, as the time to swing up and stabilize the pendulum is now only two seconds. Another difference can be seen in the amount of chattering around the unstable equilibrium. With only three discrete actions, stabilizing the pendulum is quite difficult. Comparing the graphs of the input in Figure 5.8 and Figure 5.10, it can be seen that there is less chattering with fuzzy ACL. This is understandable as with a continuous policy it is possible to define the exact state for which the discrete stabilizing action should be carried out. For a discrete policy, this is most likely not possible, and a stabilizing action would almost always result in some overshoot.

The control policies of both crisp and fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.1$ are shown in Figure 5.11. It is difficult to see the differences of these policies with the ones obtained

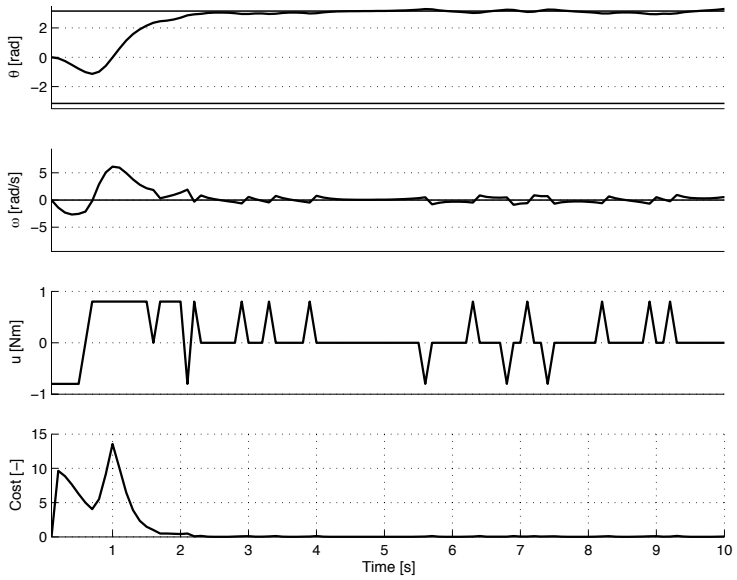
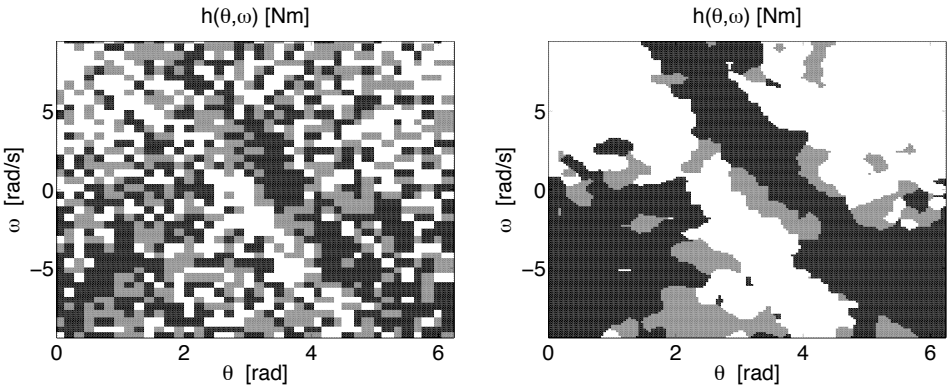


Figure 5.10: The inverted pendulum controlled from an initial state $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ by the final policy obtained by fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.1$. The top two graphs show the trajectories of the states, while the third shows the input and the bottom graph shows the cost.

for $\gamma = 0.01$ as shown in Figure 5.6. Both seem to show a bit less variation of the policy in neighboring states. There appears to be a bit more structure in the stabilizing and destabilizing regions of the policy, but especially for the crisp control policy this is difficult to say.



(a) Crisp control policy.

(b) Fuzzy control policy.

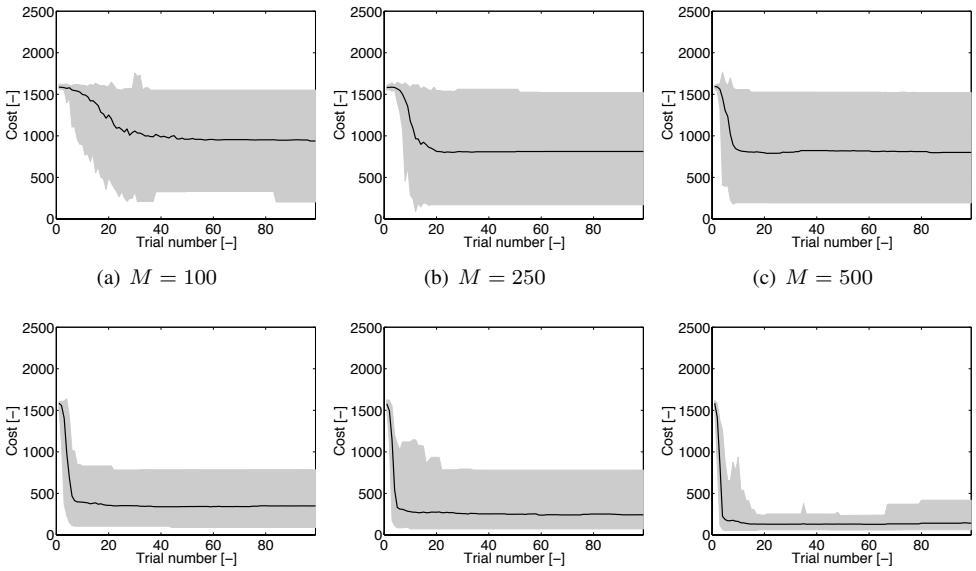
Figure 5.11: The final control policy obtained for the inverted pendulum problem with both crisp and fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.1$. The colors black, white, and gray represent the inputs -0.8, 0.8, and 0 Nm respectively. For angles near $\theta = 0, 2\pi$, the policy represents a destabilizing controller, while near $\theta = \pi$, the policy represents a stabilizing controller.

5.3.3 Number of Ants

In the previous experiments, the number of ants has always been taken equal to $M = 250$. The power of a multi-agent approach to control policy learning lies in the fact that multiple agents can jointly sample the state space contributing to the learning of one control policy for the complete state space. For ACL, we expect that with an increasing number of ants, the control policy converges faster. With the state space partitioning of 40×41 quantization levels, or membership functions, and 3 actions, there are 1640 states and 4920 possible state-action pairs. In these experiments, we study the effect of the number of ants on the learning performance, much in a similar way as for the case of a 1-dimensional grid search problem in Section 3.6.5. The values of M that we use are from the set $M \in \{100, 250, 500, 1000, 1500, 2000\}$. We keep the other parameters constant over the experiments. The global and local pheromone decay rates are $\rho = 0.1$ and $\gamma = 0.01$ respectively, and the other parameters are listed in Table 5.2. We study the behavior of both crisp and fuzzy ACL.

Crisp ACL

Figure 5.12 shows the simulation results obtained with the crisp ACL algorithm for varying numbers of ants, M.



(a) $M = 100$ (b) $M = 250$ (c) $M = 500$

(d) $M = 1000$ (e) $M = 1500$ (f) $M = 2000$

Figure 5.12: Policy performance of crisp ACL for varying number of ants. The black line is the average over the 30 experiments and the gray area represents the max-min range.

It can clearly be seen that for an increasing number of ants, on average over 30 experiments, the policy converges faster and to a better value with a smaller variation. The maximum number of ants used in these experiments $M = 2000$ turned out to result in the best

learning performance. In this case, Figure 5.12(f) shows a little increase of the cost of the policy near the end of the experiments. Why this happens is not very clear. It could result from the larger amount of exploration that this many ants exhibit. Exploration always comes with the risk of a decrease of performance, especially when the exploration probability ϵ is not annealed, as is the case for all experiments in this chapter. The final policy is shown in Figure 5.15. A simulation of the inverted pendulum controlled by this control policy for $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.13.

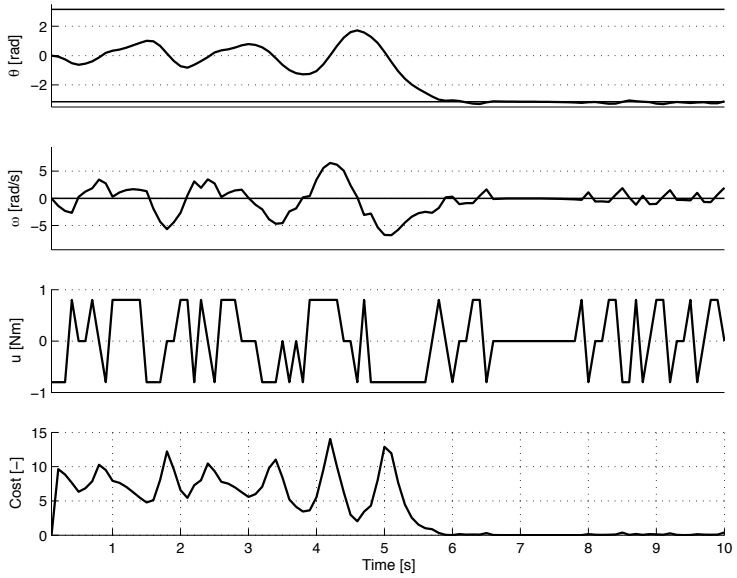
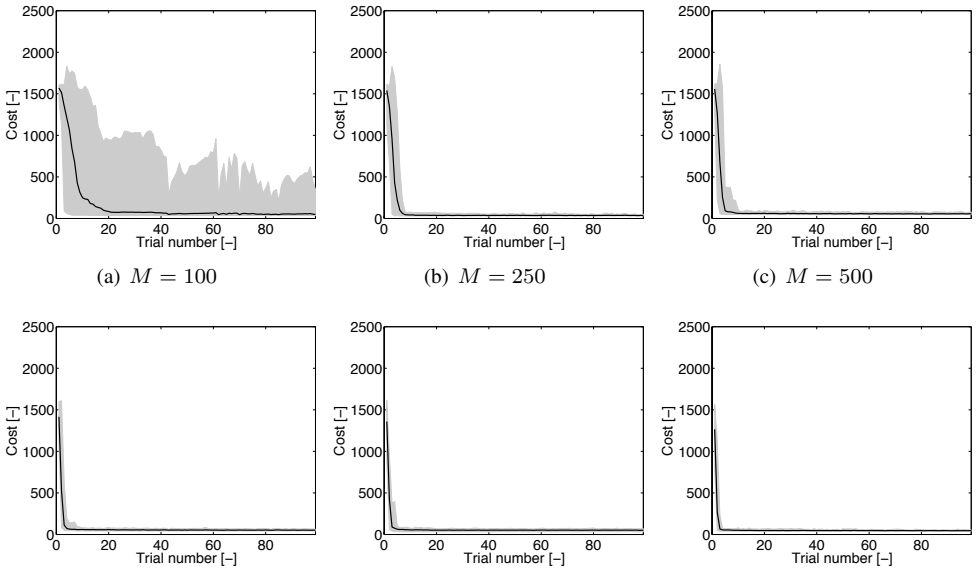


Figure 5.13: The inverted pendulum controlled from an initial state $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ by the final policy obtained by crisp ACL with $\rho = 0.1$, $\gamma = 0.01$ and the number of ants $M = 2000$. The top two graphs show the trajectories of the states, while the third shows the input and the bottom graph shows the cost.

The simulation behavior of the pendulum controlled by the final policy obtained by crisp ACL and 2000 ants is very similar to the policy learned by using 250 ants, as shown in Figure 5.3. Likewise, it is worse than that in Figure 5.8, where a larger local pheromone decay rate, viz. $\gamma = 0.1$ was used. This indicates that although the policy has a lower cost, the simulation from an initial state $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}$ is not much better.

Fuzzy ACL

Figure 5.14 shows the simulation results obtained with the fuzzy ACL algorithm for varying numbers of ants, M.

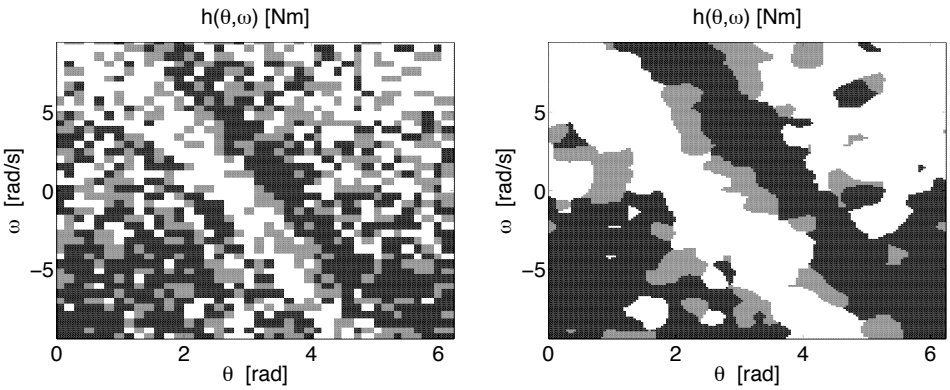


(a) $M = 100$ (b) $M = 250$ (c) $M = 500$

(d) $M = 1000$ (e) $M = 1500$ (f) $M = 2000$

Figure 5.14: Policy performance of fuzzy ACL for varying number of ants. The black line is the average over the 30 experiments and the gray area represents the max-min range.

Like with crisp ACL, an increasing number of ants results in a better learning performance. However, for 250 ants and more, the policy already converges very rapidly to a low cost, with almost no variations over the various experiments. Apparently, much fewer ants are required in fuzzy ACL compared to crisp ACL. Moreover, the final policy is better and barely varies for repetitive runs of the algorithm. The best value for the number of ants with fuzzy ACL in these experiments is thus 250, also considering that more ants require more computational resources. As the parameters for this case are the same as the ones resulting in the best performance for fuzzy ACL in Section 5.3.1, a simulation of the inverted pendulum controlled by the final policy for these parameters and for $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$ is shown in Figure 5.5. The final policy of crisp ACL with $M = 2000$ is shown in Figure 5.15 next to the final policy of fuzzy ACL with $M = 250$ that has already been shown before in Figure 5.6(b). The main result is that the crisp control policy in Figure 5.15(a) shows much more structure compared to the earlier plots of crisp control policies for the inverted pendulum problem. The several regions that can clearly be seen in Figure 5.15(b) now also become visible in Figure 5.15(a). Still, as observed earlier, this did not result in a much faster swinging up and stabilization of the pendulum from $\mathbf{x}_0 = [0 \quad 0]^{\mathrm{T}}$.



(a) Crisp control policy, $M = 2000$.

(b) Fuzzy control policy, $M = 250$.

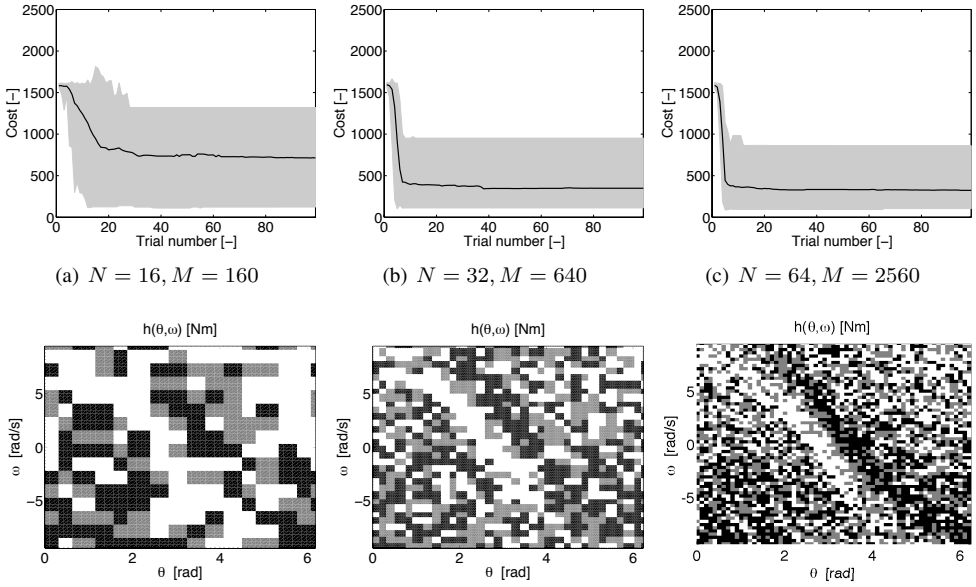
Figure 5.15: The final control policy obtained for the inverted pendulum problem with both crisp and fuzzy ACL with $\rho = 0.1$ and $\gamma = 0.01$. The colors black, white, and gray represent the inputs -0.8, 0.8, and 0 Nm respectively. For angles near $\theta = 0, 2\pi$, the policy represents a destabilizing controller, while near $\theta = \pi$, the policy represents a stabilizing controller.

5.3.4 State Space Partitioning

Another issue in the successful learning of control policies is the partitioning of the state space. Some regions of the state space may require a denser quantization than other regions, but the process of determining these regions is far from trivial and requires prior knowledge of the system to be controlled and possibly also of the optimal controller. In the experiments in this section, we look at different quantization densities, but with a homogeneous grid that does not require any prior knowledge of the system. A finer homogeneous partitioning grid may result in a better description of the policy in certain regions, but may not have this effect in other regions, only leading to an increase of computation time and memory requirements. The purpose of these experiments is therefore not to derive the most optimal partitioning grid for this particular control problem, but to study the scalability of ACL. The state space is partitioned using (5.2)-(5.3) and the density is characterized by only one parameter $N = N_\theta = N_\omega - 1$. A finer quantization results in a larger state space that must be sampled by the ants. Therefore, keeping the number of ants constant for various quantization densities would not result in a fair comparison. Also, we have observed before that for fuzzy ACL the number of ants can be much smaller than the number of ants in crisp ACL. We have chosen for crisp ACL a number of ants approximately equal to 60% of the number of states and for fuzzy ACL 15%. The values of N that are tested are 16, 32, and 64. We keep the other parameters constant over the experiments. The global and local pheromone decay rates are $\rho = 0.1$ and $\gamma = 0.01$ respectively, and the other parameters are listed in Table 5.2. We study the behavior of both crisp and fuzzy ACL.

Crisp ACL

Figure 5.16 shows the simulation results obtained with the crisp ACL algorithm for varying numbers of state space partitioning levels, N, and the corresponding numbers of ants, M.



(a) $N = 16, M = 160$ (b) $N = 32, M = 640$ (c) $N = 64, M = 2560$

(d) $N = 16, M = 160$ (e) $N = 32, M = 640$ (f) $N = 64, M = 2560$

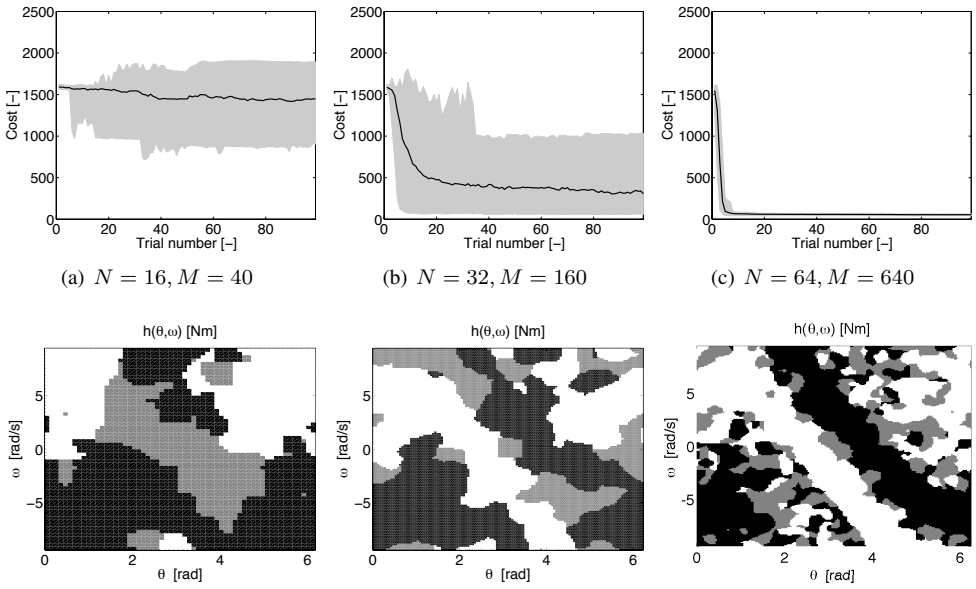
Figure 5.16: Policy performance and the policies for crisp ACL and varying number of quantization levels and ants. In the top three figures, the black line is the average over the 30 experiments and the gray area represents the max-min range. In the bottom three figures, the colors black, white, and gray represent the inputs -0.8, 0.8, and 0 Nm respectively. For angles near $\theta = 0, 2\pi$, the policy represents a destabilizing controller, while near $\theta = \pi$, the policy represents a stabilizing controller.

The performance plots show that $N = 16$ results in a cost of the control policy that is on average larger than for $N = 32$ and $N = 64$, while for $N = 64$ the performance is only slightly improved compared to $N = 32$. This leads to the conclusion that for this problem and crisp ACL, $N = 16$ is probably not a sufficient quantization density, or that the number of ants used in this case is too small. The convergence speed in terms of numbers of trials does not change much for increasing quantization density. The policy plots basically show similar control policies, but with increasing level of detail.

Fuzzy ACL

Figure 5.17 shows the simulation results obtained with the fuzzy ACL algorithm for varying numbers of state space partitioning levels, N, and the corresponding numbers of ants, M.

Contrary to the results for crisp ACL in Figure 5.17, these plots do show an improving learning behavior for increasing state space partitioning density. It may indicate that the partitioning density for $N = 16$ and $N = 32$ is not large enough for obtaining good control policies, or that the number of ants used for these densities is too low. The number of ants used for $N = 64$ is large enough and results in very fast convergence with almost no variation over repetitive runs of the experiment.



(a) $N = 16, M = 40$ (b) $N = 32, M = 160$ (c) $N = 64, M = 640$

(d) $N = 16, M = 40$ (e) $N = 32, M = 160$ (f) $N = 64, M = 640$

Figure 5.17: Policy performance and the policies for fuzzy ACL and varying number of membership functions and ants. In the top three figures, the black line is the average over the 30 experiments and the gray area represents the max-min range. In the bottom three figures, the colors black, white, and gray represent the inputs -0.8, 0.8, and 0 Nm respectively. For angles near $\theta = 0, 2\pi$, the policy represents a destabilizing controller, while near $\theta = \pi$, the policy represents a stabilizing controller.

5.4 Concluding Remarks

In this chapter we have performed a number of experiments and studied the behavior of ACL for various parameter settings. The system used in all experiments was the inverted pendulum with limited input, for which the control goal was to swing it up from any initial state and to stabilize it in its unstable equilibrium. We have studied the influence of the global and local pheromone decay rates, the number of ants, and the density of the state space partitioning grid on the learning performance. The learning performance was evaluated in terms of convergence, convergence speed, the performance of the resulting control policy, and the variation of the results over several runs of the experiment. Simulation plots of the system controlled by the learned policy were presented as well.

With respect to the global and local pheromone decay rates, the main result is that the performance of crisp ACL is much more sensitive to the choice of ρ compared to fuzzy ACL. Too large a value for ρ does not lead to convergence. State transition noise in the system causes fluctuations in the cost of the trajectories found by the ants, which are amplified for large values of ρ in the pheromone update. State transition noise is introduced in crisp ACL due to the discretization of the state space. In fuzzy ACL, the continuity of the state space is retained by the state space partitioning with fuzzy membership functions. Therefore, no

artificial non-determinism is introduced with fuzzy ACL and even a choice of $\rho = 1$ leads to convergence of the algorithm. The local pheromone decay rate appeared to have a far greater (and positive) influence on the convergence behavior of crisp ACL than it has on fuzzy ACL. The convergence properties of crisp ACL greatly improved for a value of $\gamma = 0.1$. It can be hypothesized that the local pheromone update helps in finding optimal control policies in the presence of state transition noise.

The experiments with varying numbers of ants showed that more ants lead to a faster convergence to better policies, with less variation over several runs of the experiment. However, the number of ants does not have to be extremely large. Especially with fuzzy ACL for a certain number of ants, adding more ants does not further improve much the performance of the algorithm. The main result is that crisp ACL requires many more ants for satisfying performance compared to fuzzy ACL. The required number of ants for fuzzy ACL was about ten times lower than the number required for crisp ACL for the considered application.

The scaling of ACL with increasing state space partitioning density was studied as well. The convergence speed and the quality of the resulting policy turned out to be related to the density of the state space partitioning as well as to the number of ants. That is, a finer partitioning results in a larger state space that must be sampled by the ants and therefore requires more of them. For crisp ACL, there is a slow, but gradual improvement of the learning for increasing state space partitioning density. For fuzzy ACL, the learning performance is improved more rapidly for increasing state space partitioning density and requires fewer ants. More research is required to make these claims more exact.

At this point, ACL has been introduced and studied both analytically and experimentally. In the next chapter of this thesis, we will broaden the ACL framework and see how it fits within the more general scope of swarm intelligence.

Chapter 6

Generalization of the ACL Framework

This chapter relates ACL to other swarm intelligence methodologies. It presents a general modeling framework for swarms of moving agents and we argue that optimization as well as control methodologies from the realm of swarm intelligence can be modeled in this way. It particularly serves to put ACL in a broader context.

6.1 Introduction

In Section 2.2, we have addressed the principles of self-organization and swarm intelligence from both a biological and engineering point of view. It was stated that optimization methods such as ACO and Particle Swarm Optimization as well as swarms of mobile agents, such as robots and UAVs all belong to the class of swarm intelligence. In Chapters 2 - 5 of this thesis, however, the focus has been on ACO and more particularly on ACL. In this chapter, we present a general modeling framework for swarms of moving agents that we have originally published in (van Ast et al., 2008b). This framework formally links the various forms of swarm intelligence and can be used for the integration of analysis and design methodologies that have been developed for the individual algorithms and controllers. The development of such a framework is an essential first step before any analysis can be made of the behavior of more intelligent moving agents in a swarm.

The rest of this chapter is structured as follows. The proposed framework is introduced and discussed in detail in Section 6.2. Examples of how the framework relates to the state of the art are discussed in Section 6.3, and Section 6.4 indicates which opportunities the framework brings for future research and development and concludes this chapter.

6.2 Modeling Framework for Swarms of Moving Agents

A swarm is a system of multiple cooperating autonomous agents. Generally, in multi-agent systems, the coordination of the agents is achieved by complex strategies, often in a fixed topology. In case a central controller is used to determine the optimal action for each of the

agents, such methods scale poorly with the number of agents. Swarm intelligence aims at controlling a large number of cooperative autonomous agents, in a varying topology, with simple, local rules. The analysis of a swarm intelligence system typically focuses on the dynamics of the swarm as a whole, rather than on the dynamics of the individual agents. A particular control problem considered in swarm literature is that of swarm aggregation, which deals with controlling the moving agents to form a cohesive swarm and which is briefly described in Section 6.3.3. The results for swarm aggregation are too limited to be generally applicable to a wide class of control problems. This demonstrates the difficulty of applying swarm intelligence to practically relevant control problems.

This section presents a new framework that enables a more structured approach to the development of swarm intelligence for distributed sensing and control and that provides better insight in the structure of swarm systems. The framework separates the physical parts and behavior of the swarm members from their decision making capabilities. It serves to integrate research fields focusing on the physical behavior of the swarm member with research studying dynamic agent system and enables the development and analysis of more sophisticated swarm systems.

In the considered framework, the members of the swarm are called *moving agents*. The two key features of a moving agent are that:

1. it can move through its environment and

2. it is capable of decision making based on its input and recollection of the past.

These features are represented by two strictly distinct classes, called *particles* and *dynamic agents*. A diagram of the framework is given in Figure 6.1.

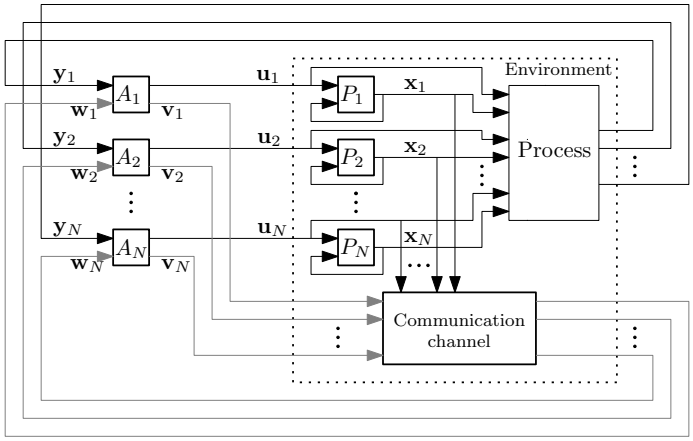


Figure 6.1: Block diagram of the framework.

The diagram shows N dynamic agents $A_1, \ldots, A_N$ interacting with a common environment. The dynamic agents represent the intelligence of the moving agents. The particles $P_1, \ldots, P_N$ represent the physical part of the moving agents, such as their position and speed. Each dynamic agent, indexed by i, senses the environment by observations represented by a vector $\mathbf{y}_i$ and produces an input $\mathbf{u}_i$ to the environment. The environment contains the

particles, a process, and a communication channel. The process may represent the system with which the moving agents need to interact in order to achieve a mission, or a control task. Furthermore, each dynamic agent A_i may send and receive messages $\mathbf{v}_i$ and $\mathbf{w}_i$ to and from the environment, respectively. Within the communication channel, it is determined which messages are received by which dynamic agents, based on the state of all the particles $\mathbf{x}_\mathrm{s} = \begin{bmatrix}\mathbf{x}_1^\mathrm{T} & \dots & \mathbf{x}_N^\mathrm{T}\end{bmatrix}^\mathrm{T}$. Each dynamic agent A_i is associated with a particle P_i. The state of each particle changes based on the input from its associated dynamic agent and the restrictions posed on it by the environment. The process state may be influenced by the inputs $\mathbf{u}_\mathrm{s} = \begin{bmatrix}\mathbf{u}_1^\mathrm{T} & \dots & \mathbf{u}_N^\mathrm{T}\end{bmatrix}^\mathrm{T}$ from the dynamic agents and the state $\mathbf{x}_\mathrm{s}$ of the particles. The rest of this section discusses the elements of the framework in more detail. Without loss of generality, it is assumed that the dynamics are modeled in discrete time.

Example 6.1 *In order to exemplify the discussion throughout this section, a swarm of identical robots is considered, the task of which is to guard a building. This is clearly a task that may benefit from the advantages of a swarm over a single agent. As it is assumed that the robots are frequently cut off from radio communication, a certain level of autonomy is required. It is also desired for such a system to be scalable and robust to malfunctioning of the robots. The robots are assumed to be able to drive over the terrain surrounding the building and they have some short-range communication capabilities, some processing power, and sensors for sensing some properties of their immediate environment.*

6.2.1 Particles

In research on swarm intelligence, the members of the swarm are usually modeled as particles (Gazi and Passino, 2005; Kennedy and Eberhart, 1995), though the exact definition of a particle is not always given. In this section, a particle is defined as follows:

Definition 6.1 *A particle P_i is an entity having a state $\mathbf{x}_i \in \mathcal{X}$, containing the physical states of the moving agent.*

Particles at the least represent the position of the moving agent in the environment, but may include many other states, ranging from velocity and orientation to shape and color. The particle is in fact the physical body of the moving agent and it is not responsible for the decision making. The states of the particles influence the neighborhood for all the agents, the local values of the sensor inputs, and the distances between the agents.

Example 6.2 *The physical position and speed of each robot in our previous example belong to the state of its associated particle. The state changes as a result of the actions chosen by the robots, according to the dynamics of the robots and the environment. The particle may also represent other physical properties of the robot, such as size, shape, color, etc., depending on what is useful for the control problem.*

6.2.2 Dynamic Agents

The decision making power of a moving agent is taken care of by a dynamic agent. Dynamic agents are the basis of modern artificial intelligence (Russell and Norvig, 1994). A dynamic agent is defined as an entity that observes the environment, possesses an internal state that

changes as a function of the observations, and acts on the environment based on this internal state and the observations. Moreover, it is able to send and receive messages, which is discussed further in Section 6.2.6.

Definition 6.2 *A dynamic agent A_i, is a tuple $(\mathcal{Z}_i, \mathcal{Y}_i, \mathcal{U}_i, \mathcal{W}_i, h_i, \pi_i, \phi_i)$, with:*

- *$\mathcal{Z}_i$ the internal state space of agent A_i,*
- *$\mathcal{Y}_i$ its observation space,*
- *$\mathcal{U}_i$ its input space,*
- *$\mathcal{W}_i$ a set of tuples (messages), representing its message space,*
- *$h_i : \mathcal{Z}_i \times \mathcal{Y}_i \times \mathcal{W}_i \rightarrow \mathcal{Z}_i$ its internal state transition function, describing how the internal state evolves as a result of observations and the messages received,*
- *$\pi_i : \mathcal{Z}_i \times \mathcal{Y}_i \times \mathcal{U}_i \rightarrow [0, 1]$ its decision probability distribution, which maps its internal state and observations probabilistically to an input to the environment, and*
- *$\phi_i : \mathcal{Z}_i \rightarrow \mathcal{W}_i$ its message generating function.*

In particular, the internal state of agent A_i is denoted by $\mathbf{z}_i$, its observation of the environment is denoted by $\mathbf{y}_i$, its input to the environment is denoted by $\mathbf{u}_i$, and $\mathbf{w}_i$ and $\mathbf{v}_i$ denote its incoming and outgoing message respectively.

In this definition, the behavior of a dynamic agent is thus defined as a probabilistic mapping of states and observations to actions. This allows for making decisions that are aimed at exploring different regions of the environment. This is essential to decision making in an environment of which there is no model available to the dynamic agents. This stochasticity in π_i is explicitly separated from the dynamic agent's deterministic internal state transition function h_i, stressing the algorithmic nature of h_i.

Example 6.3 *The dynamic agent of each robot is the software routine that receives input from sensors and communication links, processes this input based on its current state, and produces an output. This output generally drives the actuators of the robot and usually results in a change of the position or orientation, thus a change of the particle state. According to this state and the environment, the dynamic agent is presented with other pieces of information based on which it has to make a new decision.*

According to the principle that the physical state of the robot is not a part of the dynamic agent, the position of the robot is not known to the dynamic agent right away. Everything the dynamic agent needs to know, it has to draw from its observations of the environment.

The framework also stipulates that the decision making part does not necessarily need to be present at the same location as the rest of the robot. Remote controlled robots are in this sense identical to robots with an on-board controller.

For the application to swarms, dynamic agents are assumed to be *cooperative*. Cooperativeness is usually defined in the way that all the agents aim at achieving the same objective (Panait and Luke, 2005; Vlassis, 2007). Within the swarm framework, this needs a little more explanation, as the agents are all autonomous and thus act based on their own local objective function, or strategy. The control objective is generally defined for the whole swarm, so the

cooperativeness must hold for a global objective that is not necessarily known to the agents. As this is a contradictio in terminis, dynamic agents in a swarm are usually said to be cooperative if the agents do not aim at preventing other agents from achieving their individual goals.[1]

6.2.3 Moving Agents

Using Definitions 6.1 and 6.2 a *moving agent* can be defined.

Definition 6.3 *A moving agent $M_i = (A_i, P_i)$ is a pair of a dynamic agent and a particle.*

In this way, the dynamic agent can operate without directly taking into account its motion through the environment, and the motion of the particle can be considered without directly taking into account the decision making of the dynamic agent. Existing results on either part can be more easily combined to form more sophisticated swarms than those representing the current state of the art. When discussing the behavior of the swarm, one can now refer to its members by the clearly defined moving agents. Regarding stability and performance of the swarm, conditions can be determined for the signals $\mathbf{u}_i$ and $\mathbf{y}_i$ and the messages $\mathbf{w}_i$ and $\mathbf{v_i}$ that couple the dynamic agents and the environment.

Example 6.4 *The physical robots (particles) including their software for the decision making (dynamic agents) together form the moving agents.*

6.2.4 Process

Everything that must be sensed or controlled by the moving agent is called the process. It may include a real process, e.g., a chemical reactor, but also a virtual process, such as an artificial potential field (see Section 6.3.3).

Definition 6.4 *A process is characterized by its state $\boldsymbol{\psi} \in \Psi$, which contains the variables that must be sensed and/or controlled by the dynamic agents.*

The process state may change based on the output of the dynamic agents $\mathbf{u}_\mathrm{s}$ and the state of the particles $\mathbf{x}_\mathrm{s}$. Typically, the state of the process is distributed in space when a swarm approach is chosen to sense or control it. The observations of the dynamic agent may vary for varying positions as a result of this.

Example 6.5 *The task of the robots is to detect and respond to threats in the environment. These threats can be seen as the process, which state must be observed by the robots. If the interaction between the robots and the threats is modeled by an artificial potential field, this field can be interpreted as a virtual process.*

6.2.5 Environment

As illustrated in Figure 6.1, the environment encompasses everything that is outside of the dynamic agent. It consists of everything that is physically present within the problem setting. It holds the physical state of the moving agents $\mathbf{x}_\mathrm{s}$ (the particles), the process, and the communication channel. It also defines the state space of the particles, i.e., the world state with

[1] A better term would be *indifferent* rather than cooperative.

boundaries and obstacles. As the particles move through the environment and potentially are obstacles to each other, this state space is dynamic. The environment also includes the state transition functions of the particles.

The term environment, as used in this definition, comes from the dynamic agent community (Sutton and Barto, 1998). It is different from what is standard in the systems and control community, where the environment denotes everything that is outside of the controller and the process, and is usually just held responsible for the disturbances of the signals in the system. Disturbances can also easily be added to the framework, as well as communication delays and errors, although these are left out from the discussion in this section.

Definition 6.5 *The environment is a tuple* $(\mathcal{X}, \mathcal{U}, g, \Psi, f, C)$*, where*

- $\mathcal{X} = \bigtimes_{i=1}^{N} \mathcal{X}_i$ *is the joint state space of the particles,*
- $\mathcal{U} = \bigtimes_{i=1}^{N} \mathcal{U}_i$ *is the joint action space of the dynamic agents,*
- $g : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0, 1]$*, is the particles' state transition function, describing how the states of the particles evolve as a result of the action taken by all dynamic agents,*
- Ψ *is the state space of the process,*
- $f : \Psi \times \mathcal{U} \times \Psi \to [0, 1]$*, is the process transition probability distribution, describing how the process evolves as a result of the dynamic agents' actions, and*
- C *is the communication channel.*

Example 6.6 *The world outside the software of the robots is the environment. It contains the physical robots, the building, its surroundings, possible threats, the communication signals, and other signals that may be received by the sensors of the robots.*

6.2.6 Communication and Interaction

A key characteristic of a swarm is that the dynamic agents within the swarm interact strongly in order to enhance the performance of the swarm as a whole. One important aspect of interaction constitutes the observations by the dynamic agents of the states of the particles. This is closely related to a second form of interaction, namely *communication*. Agents may communicate in order to share information about, e.g., the observations and strategies. The moving agents can communicate in various ways. One way is through their particle state, e.g., by making meaningful movements. This is observed in nature with honey bees. Another way is by changing the environment in a meaningful way through its state. This is called stigmergy and is observed in nature with ants, which deposit pheromones to communicate. A third way is by sending and receiving messages through the *communication channel* C. In our framework, this is incorporated by the message signals $\mathbf{v}$ and $\mathbf{w}$, the message space $\mathcal{W}_i$, and the message generating function ϕ_i, defined in Definition 6.2.

As the moving agents in a swarm are typically low powered, with short-range sensors, they are only capable of communicating with other agents within a certain *neighborhood*. The neighborhood of a moving agent is defined as the set of moving agents that the given agent can receive messages from and is a function of the positions of the moving agents and their communication parameters, such as their broadcasting power, signal bandwidth, and constraints on the number of simultaneous connections.

Definition 6.6 *The neighborhood of a moving agent M_i is denoted by the set $\mathcal{N}_i(\mathbf{x}_\mathrm{s}, \boldsymbol{\sigma})$, with $\mathbf{x}_\mathrm{s}$ the vector stacking the particle states and $\boldsymbol{\sigma}$ the vector of communication parameters of all the moving agents respectively.*

The neighborhoods of the dynamic agents are dynamic, as they are dependent on the particle states $\mathbf{x}_\mathrm{s}$. The framework allows for any method to model the communication channel. With limited broadcast power, the neighborhood of a moving agent consists only of the other moving agents that are within a certain radius. The communication parameter $\boldsymbol{\sigma}$ then consists of these radii. The notion of neighborhood is defined in our framework to be part of the environment, as it depends on the physical properties of the moving agents and acts as a filter on the communication. Also properties like the bandwidth of the communication channel and the delay and attenuation of the signals traveling through the communication channel play a role here.

It must be noted that in order to retain flexibility and scalability of the swarm, the dynamic agents are not able to directly address other agents. The agents thus broadcast their messages and the neighborhood defines which agents receive them.

Example 6.7 *The robots broadcast the messages that are constructed by their software. Their position, broadcast power, and the properties of the environment determine their neighborhoods, which dictate which of the other robots receive their messages. The messages can be used to improve the internal world model of the dynamic agents and their decision making.*

6.2.7 Swarms

Finally the notion of a swarm can be formalized.

Definition 6.7 *A swarm is a subset of the set of moving agents, $\mathcal{S}$, with the dynamic agents being cooperative.*

In Definition 6.2, the index i to the spaces and transition functions has been added to stress that the framework allows for all the moving agents to have different properties. This gives rise to the notions of *homogeneous* and *heterogeneous* swarms.

Definition 6.8 *If all the moving agents in the swarm have the same state transition functions, the swarm is said to be homogeneous. If at least one of the state transition functions is different, the swarm is said to be heterogeneous.*

6.3 Relation to the State of the Art

This section relates the proposed framework to ACL, as well as to two important swarm intelligence methods in optimization and control, namely Particle Swarm Optimization and artificial potential fields for swarm aggregation.

6.3.1 Ant Colony Learning

ACL is the main focus of this thesis and has been introduced as a multi-agent model-free learning methodology. We can regard the algorithm in the light of the moving agent modeling framework as follows. Each ant c is considered to be a moving agent, M_c in a set of M ants.

They can be thought of moving through the state space of the system, for which they need to learn the control policy. The state $\mathbf{x}$ of the particle is equal to the "position" $\mathbf{x}_c$ of the ant. Since we regard a moving agent to be composed of a particle and dynamic agent, $\mathbf{x}_c$ is thus the state of the particle P_c. As seen in Figure 6.1, a particle is part of the environment. In ACL, particles do not directly influence each other by their state, like they would if they would represent moving hardware in a physical world. The environment also consists of a process and a communication channel. As discussed before, communication in ant-based systems is done through stigmergy, i.e., through modifying the process state. In ACL, this state is determined by the pheromone levels. The dynamic agent A_c of an ant c determines the input $\mathbf{u}_c$ to the process. The process contains the state transition function $\mathbf{f}$ of the system to be controlled, which dictates the response of the system for a certain state-action pair. The process can therefore determine the new state $\mathbf{x}_c \leftarrow \mathbf{f}(\mathbf{x}_c, \mathbf{u}_c)$, which in turn is observed by the dynamic agent A_c and stored in its internal state $\mathbf{z}_c$. The observation vector of the dynamic agent thus contains the new state, but also the pheromone levels associated with this state and all possible inputs. Let $\mathcal{U}(\mathbf{x}_c) = \{\mathbf{u}_1, \ldots, \mathbf{u}_{N_c}\}$ be the set of actions available to the ant c in its current state $\mathbf{x}_c$. The observation vector is then defined as $\mathbf{y}_c = \begin{bmatrix}\mathbf{x}_c^{\mathrm{T}} & \tau_{\mathbf{u}}(\mathbf{x}_c) & \ldots\end{bmatrix}^{\mathrm{T}}$, with $\mathbf{u} \in \mathcal{U}(\mathbf{x}_c)$ and is used by the dynamic agent to determine the action, using one of the action selection rules from Section 3.3.1 as its decision probability distribution. Note that the actual action selection rule depends on the method of representing $\mathbf{x}_c$ by the dynamic agent, which may be by crisp partitioning bins, by fuzzy membership functions, or by other means not discussed in this thesis. The selected action and the state are stored in the internal state of the dynamic agent representing the solution found so far (c.f. $s_{\mathrm{p},c}$, the partial solution of an ant c). The selected action is fed to the process, which in turn responds again by changing the system state. In addition to that, the local pheromone update is carried out by the process by slightly decaying the pheromone level associated with the particular state-action pair. The previous steps are repeated until the dynamic agent realizes it has reached the goal, either because the observed state matches with an internal representation of the goal state, or because the observation vector is augmented by a binary signal from the process telling whether or not the moving agent has reached the goal. In the usual case of synchronized global pheromone updates, each dynamic agents sends this binary signal as a communication signal $\mathbf{v}_c$ to all other dynamic agents through the communication channel. Once all moving agents have reached the goal, or a synchronized internal clock in each dynamic agent has triggered a time-out, all dynamic agents proceed with the global pheromone update. As the state-action pairs visited by an ant c are stored in its internal state $\mathbf{z}_c$, each ant can request the cost of its final solution (trajectory of state-action pairs) from the process. All ants then have the process update the pheromone levels accordingly by "walking back" from the goal state along their found trajectories to their initial states.

6.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) is an optimization heuristic, in which the goal is to find the parameter vector associated with the global optimum in a problem space according to a certain objective, or fitness function. It is different from most other optimization heuristics as it uses a swarm of agents instead of only one agent. In this way, it is closely related to evolutionary computation, which is also population-based. For information about similarities and differences between these two population-based optimiza-

tion methods, the reader is referred to (Kennedy and Eberhart, 2001). The agents in PSO are called particles and partly conform with the particles from the framework introduced in this section. A particle i is defined by a state $\mathbf{x}_i = \begin{bmatrix}\boldsymbol{\theta}_i^{\mathrm{T}} & \boldsymbol{\nu}_i^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ denoting its position and velocity in the problem space Θ. A fitness function $F(\boldsymbol{\theta}) : X \to \mathbb{R}$ maps the parameter space to a fitness landscape, which as a result associates each particle to a fitness value.

In PSO, the particles evolve in discrete time, because of the algorithmic nature of the optimization problem. In the basic setting, at each iteration of the algorithm, the particles update their state with the following rule:

$$\boldsymbol{\theta}_i(k+1) = \boldsymbol{\theta}_i(k) + \boldsymbol{\nu}_i(k), \tag{6.1}$$

$$\begin{aligned}\boldsymbol{\nu}_i(k+1) =& w(k)\boldsymbol{\nu}_i(k) + c_1 r_1(k)[\boldsymbol{\theta}_{i,\mathrm{pbest}}(k) - \boldsymbol{\theta}_i(k)] \\ &+ c_2 r_2(k)[\boldsymbol{\theta}_{i,\mathrm{lbest}}(k) - \boldsymbol{\theta}_i(k)]\end{aligned} \tag{6.2}$$

where k is the current time step, $\boldsymbol{\theta}_{i,\mathrm{pbest}}$ is the *personal-best* position, $\boldsymbol{\theta}_{i,\mathrm{lbest}}$ is the *local-best* position, $w(k)$ is the inertia weight, $r_{1,2}(k)$ are random variables, and $c_{1,2}$ are positive acceleration constants. The personal- and local-best positions are the values of $\boldsymbol{\theta}_i(k)$ that are associated with the highest fitness value attained since $k = 0$ for particle i and any particle in the neighborhood of that particle respectively.[2] Each particle in the swarm is attracted towards its personal-best solution and the local-best solution. In this way, it learns to find the optimum of the fitness function, not only by its own experience, but from other members of the swarm as well. The values of the inertia weight $w(k)$ and the range of the random variables $r_{1,2}(k)$ influence the convergence properties of the particle swarm. The positive acceleration constants $c_{1,2}$ trade off exploration and exploitation. More information can be found in (Kennedy and Eberhart, 1995) and (Clerc and Kennedy, 2002).

The equations (6.1) and (6.2) can be written in a state space form that separates the particle from the dynamic agent according to the framework proposed in this section:

$$\mathbf{x}_i(k+1) = \begin{bmatrix}1 & 1 \\ 0 & w(k)\end{bmatrix}\mathbf{x}_i(k) + \begin{bmatrix}0 & 0 \\ c_1 r_1(k) & c_2 r_2(k)\end{bmatrix}\begin{bmatrix}\boldsymbol{\theta}_{i,\mathrm{pbest}} - \boldsymbol{\theta}_i \\ \boldsymbol{\theta}_{i,\mathrm{lbest}} - \boldsymbol{\theta}_i\end{bmatrix}(k).$$

From this form, the particle P_i can be identified by the relation:

$$\mathbf{x}_i(k+1) = \begin{bmatrix}1 & 1 \\ 0 & w(k)\end{bmatrix}\mathbf{x}_i(k) + \mathbf{u}_i(k),$$

with

$$\mathbf{u}_i(k) = \begin{bmatrix}0 & 0 \\ c_1 r_1(k) & c_2 r_2(k)\end{bmatrix}\mathbf{z}_i(k) \tag{6.3}$$

the input from the dynamic agent A_i and $\mathbf{z}_i = \begin{bmatrix}(\boldsymbol{\theta}_{i,\mathrm{pbest}} - \boldsymbol{\theta}_i)^{\mathrm{T}} & (\boldsymbol{\theta}_{i,\mathrm{lbest}} - \boldsymbol{\theta}_i)^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ its internal state.

The process represents the fitness function $F(\boldsymbol{\theta})$ and the observation signal is defined as $\mathbf{y}_i = \begin{bmatrix}F(\boldsymbol{\theta}_i) & \boldsymbol{\theta}_i^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. Each dynamic agent determines its personal-best position as:

$$\boldsymbol{\theta}_{i,\mathrm{pbest}}(k) = \arg\max\left(F(\boldsymbol{\theta}_i(k)), F(\boldsymbol{\theta}_i(k-1))\right)$$

[2] Sometimes, the neighborhood is considered to cover the complete swarm. In that case, the local-best position is called the *global-best* position of a particle, $\boldsymbol{\theta}_{i,\mathrm{gbest}}$.

and broadcasts it as the message $\mathbf{v}_i(k) = (F(\boldsymbol{\theta}_i), \boldsymbol{\theta}_{i,\mathrm{pbest}}(k))$.

The communication channel determines the neighborhood of each moving agent and produces the message $\mathbf{w}_i$. Let $\{j_1, \ldots, j_N\} = \{j \mid M_j \in \mathcal{N}_i(\mathbf{x}_\mathrm{s}, \sigma)\}$ be the set of indices belonging to the moving agents within the neighborhood of a moving agent i with σ the size of the neighborhood, which is equal for all the moving agents. Then, the message $\mathbf{w}_i$ is defined as $\mathbf{w}_i = (F(\boldsymbol{\theta}_{j_1}), \boldsymbol{\theta}_{j_1,\mathrm{pbest}}, F(\boldsymbol{\theta}_{j_2}), \boldsymbol{\theta}_{j_2,\mathrm{pbest}}, \ldots)$. Dynamic agent A_i processes this message by taking the maximum over the fitness values in this set to determine the local-best position $\boldsymbol{\theta}_{i,\mathrm{lbest}}$. This provides the dynamic agent with enough information to update the internal state $\mathbf{z}_i$ and produce its input to the environment $\mathbf{u}_i$ according to (6.3). Here, the stochastic nature of the decision making is expressed by the random variables $r_{1,2}(k)$.

6.3.3 Swarm Aggregation by Potential Functions

One of the multi-robot control problems studied in literature is *swarm aggregation*, in which the agents have to aggregate to form a cohesive swarm (Gazi and Passino, 2005). For analyzing the swarm behavior, most of the research has focused on a simple model of the particle dynamics and their interaction. In continuous time, the particles are modeled by the following kinematic model:

$$\dot{\mathbf{x}}_i(t) = \mathbf{u}_i(t), \tag{6.4}$$

where the position of a particle i at time t is denoted by $\mathbf{x}_i(t)$ and its corresponding input by $\mathbf{u}_i(t)$. This model allows proof-of-concept design of swarm systems, where at a later stage (6.4) can be replaced by a more realistic, more complex model, like a point mass model or full actuator model (Gazi and Passino, 2005; Gazi and Fidan, 2007). The input to the particle dynamics is the local value of an artificial potential field. In the environment, all the objects, such as the particles and obstacles are assigned a *potential function* that defines a virtual force acting upon a particle at a certain distance. The value of the artificial potential field is the sum of the values of all the potential functions. The general class of attraction/repulsion functions studied in (Gazi and Passino, 2004) is of the type:

$$g(\mathbf{y}) = -\,\mathbf{y}[g_\mathrm{a}(||\mathbf{y}||) - g_\mathrm{r}(||\mathbf{y}||)], \tag{6.5}$$

where $g_\mathrm{a}, g_\mathrm{r} : \mathbb{R}^+ \to \mathbb{R}^+$ represent the magnitude of the attraction and repulsion term respectively, the vector $\mathbf{y} = \mathbf{x}_i - \mathbf{x}_j$ represents the distance between two particles $i, j \in \{1, \ldots, M\}$, and $||\mathbf{y}|| = \sqrt{\mathbf{y}^\mathrm{T}\mathbf{y}}$ is the Euclidean norm. The input $\mathbf{u}_i$ is generated by the dynamic agent A_i based on measurement of the distance between its own particle and other particles in the environment, $\mathbf{y}_i = \{\mathbf{x}_i - \mathbf{x}_j \mid j : M_j \in \mathcal{N}_i(\mathbf{x}_\mathrm{s}, \sigma_i)\}$. The neighborhood is defined by the set $\mathcal{N}_i(\mathbf{x}_\mathrm{s}, \sigma_i) = \{M_j \mid ||\mathbf{x}_i - \mathbf{x}_j|| < \sigma_i\}$, with σ_i the sensing radius of P_i. The function g is typically predefined and identical for all the moving agents. Dynamic agents may also infer g from observed physical properties of other moving agents. For example, the color of a moving agent may be associated with a certain g according to a prespecified database. Parameters that define g may also be communicated by the signals $\mathbf{v}$ and $\mathbf{w}$.

6.4 Concluding Remarks

This chapter has proposed a general modeling framework for swarm systems of moving agents. The framework separates the physical parts and behavior of the swarm members

from their decision making capabilities. This facilitates the integration of current swarm intelligence research, which focuses mainly on the physical behavior of the swarm members, with research on more sophisticated decision making in dynamic agent systems and artificial intelligence. The framework enables a more structured approach to the development of new applications of swarm intelligence, particularly for distributed sensing and control. It provides a better insight in the structure of swarm systems and theoretical results as well as results based on simulation experiments can be more directly mapped to the application domain. The proposed framework aims to integrate both fields to enable the development and analysis of more sophisticated swarm systems.

We have related ACL as well as two of the most established methods from the swarm community, namely Particle Swarm Optimization (PSO) and artificial potential functions for swarm aggregation to the proposed framework. Although ACL is a cooperative learning method, PSO is an optimization method, and swarm aggregation is a control problem, it has been demonstrated how these methods can all be decomposed into similar elements. Future research will focus on the development and analysis of swarms of moving agents for distributed sensing and control, based on the proposed framework.

Chapter 7

Conclusions and Recommendations

In this thesis, we have presented the ant colony learning framework and studied its behavior both analytically and through computer simulations. Ant colony learning is a methodology for control policy learning using a set of cooperating agents, based on the principles of ant colony optimization. We have presented an algorithm for discrete state spaces and two algorithms for continuous state spaces. This chapter summarizes the main topics and contributions, lists the main conclusions, and gives an overview of the open issues and recommendations for future research.

7.1 Summary of Contributions

Chapter 2 has provided the necessary background for this thesis. The main concepts of swarm intelligence have been introduced in relation to natural and engineered swarms. In nature, swarming provides benefits to the individuals, which is one of the driving forces behind the local behavior of the individuals. Particularly, we have discussed the behavior of ants. Communication between ants is done through pheromones. When an ant forages for food, it is biased to search along trails of stronger pheromone concentrations. When it then finds food, it will walk back to the nest while depositing pheromones and thereby contributing to the reinforcement of a successful trail. The shortest path finding capability of ants and their method of depositing pheromones has been used as an inspiration by Marco Dorigo and other researchers to develop the Ant Colony Optimization (ACO) metaheuristic for combinatorial optimization problems. In ACO, the optimization problem is formalized using the concept of the construction graph, in which the nodes represent the elements that need to be combined optimally, and the edges represent all the possible combination of these elements. In Chapter 2, we have presented the ACO framework, two of its most important algorithms, namely the Ant System and the Any Colony System, and a proof of convergence from literature.

In Chapter 3, we have introduced the general framework of Ant Colony Learning (ACL). This cooperative control policy learning approach is based on the ACO framework, and shares the ant and pheromone metaphors. When the states are discrete, as is the assumption in Chapter 3, the nodes of the construction graph represent the states. In each state, there is

a set of possible actions that can be chosen and each action will bring the system to a new state. The actions can thus be represented by the edges, connecting the nodes. If the state transitions are deterministic, there is a one-to-one relation between the current state and action and the next state. We have theoretically analyzed the convergence properties of ACL and have derived upper and lower bounds for the pheromone levels and for the expected value of the pheromone levels. We have shown that the expected policy converges to the optimal policy in the case of using only one ant. For an increased number of ants, convergence of the expected policy is only guaranteed in some specific scenarios and is otherwise not formally guaranteed. In experiments on control problems that have a discrete state space and deterministic state transitions, we have found that ACL converges quickly to the optimal solution. More specifically, we have found for these experiments, that the values of the local and global pheromone decay rates, γ and ρ respectively, should be chosen in the range of $(0.01, 0.1)$ so to obtain the best performance. We have also seen that increasing the number of ants in the algorithm results in a decrease of the number of trials needed for convergence to the optimal policy. The effect of state transition noise on the behavior of the algorithm has also been studied. Although noise makes the learning problem much more challenging, we have found that our ACL algorithm performs very well in these cases.

Chapter 4 has introduced two ACL algorithms for control policy learning in continuous state spaces. The key issue here is that the state measurements need to be represented in a discrete way. We have studied two ways of doing this. In the first one, called crisp ACL, the state space is partitioned using crisp partitioning bins. In that case, the state measurement is assigned to exactly one bin. A problem here is that crisp partitioning introduces discretization noise, which turns an originally deterministic system into a non-deterministic system. The second way of representing a continuous-valued state by a finite number of elements is by partitioning the state space by means of fuzzy triangular membership functions. In this algorithm, which is called fuzzy ACL, the continuous state measurement is said to belong to multiple membership functions, each to a certain degree. With fuzzy partitioning, the continuity of the state variables is preserved. No non-determinism is introduced in this case either. We have performed an experimental analysis of both crisp and fuzzy ACL by applying them to the non-linear control problem of two-dimensional navigation with variable damping. The control goal here was to steer a vehicle to the center of the area avoiding the regions of stronger damping, before coming to a standstill. The results show convergence of both versions to suboptimal policies. However, fuzzy ACL converged much faster and the cost of its resulting policy did not change as much over repetitive runs of the algorithm compared to crisp ACL. Fuzzy ACL also converged to a more optimal policy than crisp ACL. We have also showed the importance of choosing a good cost function and state space partitioning. Especially fuzzy ACL showed to be capable of learning control policies that are close to the optimal policies derived by fuzzy Q-iteration.

In Chapter 5, we have presented a series of experiments to demonstrate the performance of both crisp and fuzzy ACL. The system used in all experiments was the under-actuated inverted pendulum. The control goal was to swing up the pendulum from any initial state and to stabilize it in its unstable equilibrium. We have studied the influence of the global and local pheromone decay rates, the number of ants, and the density of the state space partitioning grid on the learning performance. The learning performance was evaluated in terms of convergence, convergence speed, the performance of the resulting control policy, and the variation of the results over several runs of the experiment. We have found that the best

values for ρ and γ are in the order of 0.1 and 0.01 respectively, confirming the results from Chapter 3. Especially, the performance of crisp ACL improved for a small local pheromone decay rate, while fuzzy ACL outperformed crisp ACL over the whole line. In general crisp ACL is much more sensitive to choices of ρ and γ than fuzzy ACL. We found that more ants lead to faster convergence, but that the number of ants does not need to be extremely large to have satisfying performance. Furthermore, the required number of ants for fuzzy ACL was about ten times lower than the number of ants required for crisp ACL. With regard to the scaling of ACL with increasing state space partitioning density, crisp ACL revealed a slow, but gradual improvement of the learning for increasing state space partitioning density. Fuzzy ACL, on the other hand, improved more rapidly and required fewer ants to learn a better and more regular control policy.

Chapter 6 has presented a general modeling framework for swarms of moving agents. We have shown how ACL fits in this framework and as such can be unified with other swarm intelligence techniques, such as particle swarm optimization and swarm aggregation using artificial potential functions. This may result in beneficial integration of elements from other swarm intelligence techniques into ACL, or the other way around.

7.2 Main Conclusions

The main conclusions of the research presented in this thesis are the following:

- We have developed the ACL framework. The convergence properties of ACL have been theoretically analyzed for the case of a discrete state space with noiseless state transitions. We conclude that the expected policy converges to the optimal policy in the case of one ant. For an increased number of ants, convergence of the expected policy is only guaranteed for specific parameter values. For other cases, the policy is not formally guaranteed to converge.

- We conclude that crisp and fuzzy ACL can be unified using the generalized pheromone update developed in this thesis. Non-determinism is introduced in crisp ACL due to the discretization of the state space. In fuzzy ACL, the continuity of the state space is retained by the state space partitioning with fuzzy membership functions. Fuzzy ACL outperforms crisp ACL in all experiments and fuzzy ACL is capable of learning control policies that are close to optimal.

- Based on all experiments with ACL in this thesis, we conclude that the best values of the global and local pheromone decay rates are in the order of 0.1 and 0.01 respectively. The performance of crisp ACL is much more sensitive to the choice of the global pheromone decay rate, ρ, compared to fuzzy ACL. State transition non-determinism causes fluctuations in the cost of the trajectories found by the ants, which are amplified for large values of ρ in the pheromone update. Since no artificial non-determinism is introduced with fuzzy ACL, even a choice of $\rho = 1$ leads to convergence of the algorithm, contrary to crisp ACL. We conclude that in a noise-free setting, the local pheromone decay rate has a far greater (and positive) influence on the convergence behavior of crisp ACL than it has on fuzzy ACL.

- More ants lead to a faster convergence to better policies, with less variation over several runs of the experiment. However, the number of ants does not have to be extremely

large. Crisp ACL requires many more ants for satisfying performance compared to fuzzy ACL. For the applications considered in this thesis, the required number of ants for fuzzy ACL is about ten times lower than the number required for crisp ACL.

- The convergence speed and the quality of the resulting policy is related to the density of the state space partitioning as well as to the number of ants. That is, a finer partitioning results in a larger state space that must be sampled by the ants and therefore requires more of them. For crisp ACL, there is a slow, but gradual improvement of the learning for increasing state space partitioning density. For fuzzy ACL, the learning performance is improved more rapidly for increasing state space partitioning density and requires fewer ants.
- ACL can be integrated with other swarm intelligence techniques in a common generalized modeling framework of swarms of moving agents, as presented in this thesis.

7.3 Open Issues and Directions for Future Research

From the research presented in this thesis, some open issues have surfaced. This section presents a list of these issues, as well as directions for future research.

Extensions

- In this thesis, we have used quite standard cost functions. For instance, in Chapter 3 the cost was simply represented by the number of steps to the goal and in Chapter 4, the cost was represented by a quadratic cost function. The latter one is already somewhat more involved, as it reflects the error of the state trajectory as well as the error of the input used on this trajectory. In fact, the ACL framework allows for any kind of cost that can be computed on the basis of the solutions found by the ants. Remember that the solutions consist of state-action pairs from the initial state of the particular ant to the goal state. The cost may thus reflect any characteristic of the solutions found, like overshoot, rise time, settling time, etc. We recommend future research to investigate the performance of ACL for cases where the cost is measured in such terms.
- One way of improving the learning time may be the inclusion of prior knowledge. Prior knowledge may be incorporated in ACL in a number of ways. One way is to translate the prior knowledge into heuristic variables, like in conventional ACO. The heuristic variables introduce a bias for the decision making of the ants, which may speed up the learning process. However, without decaying its exponent, the heuristic information will stay equally important during the course of the learning, which may in fact hamper the learning after some number of trials. Another way of including prior knowledge in ACL is by encoding the knowledge in the initial value of the pheromone levels themselves. In this way, the prior knowledge similarly biases the decision of the ants towards more promising solutions, but these values are immediately adapted after the first trial, potentially preventing a negative effect in the long run. Regarding the learning performance, it is expected that prior knowledge may be of great help, but only if applied with care. Future research must study the effects of the various ways of incorporating prior knowledge on the learning performance.

- When the dynamics of a controlled system change, the performance of the controller usually degrades. With some (non-adaptive) automatic control approaches, the controller must be re-tuned, or the optimization algorithm restarted, in such a case. The ACL framework is however potentially very useful for tracking time-varying dynamic systems, as the ants continuously explore the state-action space. Future research must study to which extent such changes can be tracked. Namely, if the changes are too big, adapting the current pheromone levels to the new situation may be more time consuming than simply restarting the learning completely.

On-line Application

- With respect to the on-line application of ACL, it is useful to develop an implementation of ACL that does not rely on off-line learning on the basis of a model of the system. As we have discussed before, ACL is in principle a model-free learning methodology, but the practical implementation so far required off-line learning based on a model. It would be of great practical importance to study ways of circumventing this current limitation. For instance, given the current control policy and some exploration probability, the real-world system may be sampled along the trajectories it travels. These samples may be stored in a database with the ants continuously exploring the data for better trajectories. The ants then optimize the policy on the basis of this evolving database. Snapshots of the evolving control policy may then be applied to the system, resulting in an indirect interplay between the ants and the physical system.
- For an on-line application of ACL, it may be required that a sufficient performance of the evolving control policy is guaranteed. Especially with respect to exploration, certain random actions can cause the system to behave in a way that is dangerous to anything in the vicinity of the set-up, or in a way that is harmful to the set-up itself. Guaranteeing performance could involve having an inner control loop that at least gives a baseline performance to the system. The learning may then take place on a higher level, e.g., tuning the parameters of the inner control loop within certain safety limits. Clearly, guaranteeing performance requires prior knowledge about the system to be available.
- When, in an on-line setting, ACL is applied to many physical copies of the same system, it is especially undesirable that the ants have to wait for each other to complete the trial before the global pheromone update takes place. For this reason, an asynchronous implementation of ACL is recommended. Also for ACL interacting with a software model of the physical system, the learning speed may be greatly improved if ants are allowed to perform the global pheromone update directly after reaching the goal, or after timing out. The result is that ants currently working on their solution experience changing pheromone levels. Asynchronous implementations of ACO have not been published yet. The reason for this is that in the conventional ACO setting and for most combinatorial optimization problems, all ants will terminate their trial at the same time, as each ant travels across the same number of nodes in the construction graph. Since this is not the case in the state-action framework in which ACL operates, an asynchronous implementation makes sense for ACL. This would also more closely follow the way natural ants work, since obviously, the foraging behavior of real ants is not synchronized at all.

State-Action Space Partitioning

- The optimal choice of state space partitioning is also an important subject for future research. In this thesis, we have either used a regular partitioning, or in the case of the two-dimensional navigation with variable damping problem, one with a denser partitioning around the regions of stronger damping. These choices are however quite arbitrary and may very well not be optimal. Future research is needed to study ways to automatically derive the optimal number of partitioning bins or membership functions and their optimal distribution across the state space. In absence of prior knowledge of the system or the controller, it may be necessary to make the partitioning adaptive during the learning, e.g., by refining the partitioning in regions of the state space where the pheromone levels show a stronger variation of the policy. Also for changing processes, adapting the partitioning density along with adapting the pheromone levels may improve the learning performance.

- We have shown that fuzzy partitioning leads to much better results than crisp partitioning. However, the problem with fuzzy partitioning using triangular membership functions is that the number of membership functions required grows exponentially with the number of dimensions of the state space. For state spaces with a dimension larger than 6, this may already become a serious problem, both in terms of memory required to store these membership functions, as well as the computational time needed to evaluate the many state-action pairs. Amongst others, Gaussian basis functions have the potential to solve this problem, as the number required does not have to grow exponentially with the state space dimension. Still, their optimal positioning across the state space, as well as the choice of their variance is an open issue. Future research must study the inclusion of various basis functions in the ACL framework, their optimal distribution in the state space, and their effect on the learning performance.

- In addition to continuous state spaces, we may also consider continuous action spaces. The current implementation of ACL relies on choosing an action from a set of discrete actions. This step must be reconsidered when allowing for the action to be continuous-valued. Also the storage of pheromone levels for state-action pairs must be re-examined. The action space may be partitioned similarly as the state space and the optimal action may then be defined as the average action weighted by the associated pheromone levels. Future research must explore the implication of these changes as well as the performance and convergence properties of the resulting algorithms.

Other recommendations for future research

- In this thesis, we have briefly discussed the similarities and differences between ACL and reinforcement learning. An important direction for future research is to quantitatively compare these two control policy learning techniques. In particular, the more recent development of parallel implementations of Q-learning may prove to be strong competitors to ACL. A thorough numerical and qualitative study should be performed to reveal the strong and weak points of both techniques, also in terms of the learning cost. Furthermore, it can be fruitful to study a hybrid of ACL and Q-learning, for instance by applying ACL off-line to learn a good control policy fast, and applying

Q-learning on-line for fine-tuning the control policy and adapting it to changes in the environment.

- With respect to the practical application of ACO algorithms we recommend to develop implementations for processors capable of massive parallel computations, such as Graphics Processing Units (GPUs). In this way, the massive parallelism that is at the foundation of the ACO framework, can be exploited to its full extend. Such processors can bring the full power of on-line optimization by ACO to practical systems and still be very cheap. For genetic programming, implementations on GPUs have already been published (Harding and Banzhaf, 2007; Langdon and Banzhaf, 2008).

- Other directions for future research on ACO algorithms for optimal control involve more general issues seen in, e.g., the reinforcement learning community. Such issues include performance guarantees in on-line learning, learning in environments where not all state variables are measured (in the RL community, such environments are known as Partially Observable MDPs), the control policy learning for distributed systems, the exploration-exploitation dilemma, and the need for standardized benchmark problems allowing for better comparison of the many different learning algorithms.

- The modeling framework for moving agents that we have presented in Chapter 6 is a first step towards the integration of theoretical results and more realistic moving-agent models. Our expectation is that future research in this direction will be of great importance to ant colony optimization for control as well as to the whole field of swarm intelligence.

Bibliography

Alaykran, K., Engin, O., and Dyen, A. (2007). Using ant colony optimization to solve hybrid flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 35(5-6):541–550.

Åström, K. J. and Wittenmark, B. (1990). *Computer Controlled Systems – Theory and Design*. Prentice-Hall, Englewood Cliffs, New Jersey.

Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. (2006). Metaheuristics in stochastic combinatorial optimization: a survey. Technical Report 08, IDSIA, Manno, Switzerland.

Bilchev, G. and Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces. In Fogarty, T., editor, *Selected Papers from AISB Workshop on Evolutionary Computing*, volume 993 of *Lecture Notes in Computer Science*, pages 25–39, London, UK. Springer-Verlag.

Birattari, M., Caro, G. D., and Dorigo, M. (2002). Toward the formal foundation of Ant Programming. In *Proceedings of the International Workshop on Ant Algorithms (ANTS 2002)*, pages 188–201, Brussels, Belgium. Springer-Verlag.

Blum, C. (2005). Ant colony optimization: introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2008). Continuous-state reinforcement learning with fuzzy approximation. In Tuyls, K., Nowé, A., Guessoum, Z., and Kudenko, D., editors, *Adaptive Agents and Multi-Agent Systems III*, volume 4865 of *Lecture Notes in Computer Science*, pages 27–43. Springer.

Clerc, M. and Kennedy, J. (2002). The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.-M. (1990). The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 3(2):159–168.

Deneubourg, J. L., Gross, S., Franks, N., and Pasteels, J. M. (1989). The blind leading the blind: Modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, 2:719–725.

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

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344(2-3):243–278.

Dorigo, M. and Gambardella, L. (1997). Ant Colony System: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M., M. Birattari, M., and Stützle, T. (2006). Ant colony optimization – artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

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

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization.* The MIT Press, Cambridge, MA, USA.

Fan, X., Luo, X., Yi, S., Yang, S., and Zhang, H. (2003). Optimal path planning for mobile robots based on intensified ant colony optimization algorithm. In *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP 2003)*, pages 131–136, Changsha, Hunan, China.

Fox, B., Xiang, W., and Lee, H. P. (2007). Industrial applications of the ant colony optimization algorithm. *International Journal of Advanced Manufacturing Technology*, 31(7-8):805–814.

Gambardella, L. M. and Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning*, pages 252–260, San Francisco, CA. Morgan Kaufmann Publishers.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman.

Gazi, V. and Fidan, B. (2007). *Coordination and control of multi-agent dynamic systems: Models and approaches*, volume 4433 of *Lecture Notes in Computer Science*, chapter Swarm Robotics: SAB 2006, pages 71–102. Springer-Verlag.

Gazi, V. and Passino, K. M. (2004). A class of attractions/ repulsion functions for stable swarm aggregations. *International Journal of Control*, 77(18):1567–1579.

Gazi, V. and Passino, K. M. (2005). Stability of a one-dimensional discrete-time asynchronous swarm. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(4):834–841.

Gordon, D. M. (1999). *Ants at Work: how an insect society is organized.* Free Press, Simon and Schuster.

Gordon, D. M. (2003). Deborah gordon digs ants. URL: `http://www.ted.com/talks/deborah_gordon_digs_ants.html`.

Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J.-M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12):579–581.

Grounds, M. and Kudenko, D. (2007). Parallel reinforcement learning with linear function approximation. In *Proceedings of the International Conference on Autonomous Agents*, pages 213–215.

Harding, S. and Banzhaf, W. (2007). Fast genetic programming on GPUs. In Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., and Esparcia-Alcázar, A., editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 90–101, Valencia, Spain. Springer.

Huang, R. H. and Yang, C. L. (2008). Ant colony system for job shop scheduling with time windows. *International Journal of Advanced Manufacturing Technology*, 39(1-2):151–157.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Western Australia.

Kennedy, J. and Eberhart, R. C. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Korosec, P., Silc, J., Oblak, K., and Kosel, F. (2007). The differential ant-stigmergy algorithm: an experimental evaluation and a real-world application. In *Proceedings of the 2007 Congress on Evolutionary Computation (CEC 2007)*, pages 157–164, Singapore.

Kushida, M., Takahashi, K., Ueda, H., and Miyahara, T. (2007). A comparative study of parallel reinforcement learning methods with a pc cluster system. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2006)*, pages 416–419.

Langdon, W. and Banzhaf, W. (2008). A SIMD interpreter for genetic programming on GPU graphics cards. In O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcazar, A. I., De Falco, I., Della Cioppa, A., and Tarantino, E., editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 73–85, Naples. Springer.

Laurent, G. and Piat, E. (2001). Parallel q-learning for a block-pushing problem. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 286–291.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Purnamadjaja, A. H. and Russell, R. A. (2005). Pheromone communication in a robot swarm: necrophoric bee behaviour and its replication. *Robotica*, 23(6):731–742.

Reinelt, G. (1991). TSPLIB–A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384.

Russell, S. and Norvig, P. (1994). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Schoonderwoerd, R., Holl, O., Bruten, J., and Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5:169–207.

Sim, K. M. and Sun, W. H. (2003). Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 33(5):560–572.

Socha, K. and Blum, C. (2007). An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing & Applications*, 16(3):235–247.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173.

Stützle, T. and Dorigo, M. (2002). A short convergence proof for a class of ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365.

Stützle, T. and Hoos, U. (2000). MAX-MIN Ant Systems. *Journal of Future Generation Computer Systems*, 16:889–914.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Thrun, S. and Möller, K. (1992). Active exploration in dynamic environments. In Moody, J. E., Hanson, S. J., and Lippmann, R., editors, *Advances in Neural Information Processing Systems*, pages 531–538. Morgan Kaufmann, Denver, Colorado, USA.

Tsutsui, N. D., Suarez, A. V., Holway, D. A., and Case, T. J. (2001). Relationships among native and introduced populations of the argentine ant (linepithema humile) and the source of introduced populations. *Molecular Ecology*, 10(9):2151–2161.

Tsutsui, S., Pelikan, M., and Ghosh, A. (2005). Performance of aggregation pheromone system on unimodal and multimodal problems. In *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, volume 1, pages 880–887, Edinburgh, Scotland.

van Ast, J. M. (2010). Swarm intelligence. URL: `http://ziedaar.nl/article.php?id=385`.

van Ast, J. M. and Babuška, R. (2006). Dynamic exploration in Q-learning. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN 2006)*, pages 41–46.

van Ast, J. M., Babuška, R., and De Schutter, B. (2008a). Ant colony optimization for optimal control. In *Proceedings of the 2008 Congress on Evolutionary Computation (CEC 2008)*, pages 2040–2046, Hong Kong, China.

van Ast, J. M., Babuška, R., and De Schutter, B. (2008b). A general modeling framework for swarms. In *Proceedings of the 2008 Congress on Evolutionary Computation (CEC 2008)*, pages 3796–3801, Hong Kong, China.

van Ast, J. M., Babuška, R., and De Schutter, B. (2008c). Particle swarms in optimization and control. In *Proceedings of the 17th IFAC World Congress*, Seoul, South Korea. Delft University of Technology, Delft Center for Systems and Control.

van Ast, J. M., Babuška, R., and De Schutter, B. (2009a). Fuzzy ant colony optimization for optimal control. In *Proceedings of the American Control Conference (ACC 2009)*, pages 1003–1008, Saint Louis, MO, USA.

van Ast, J. M., Babuška, R., and De Schutter, B. (2009b). Novel ant colony optimization approach to optimal control. *International Journal of Intelligent Computing and Cybernetics*, 2(3):414 – 434.

van Ast, J. M., Babuška, R., and De Schutter, B. (2010a). Ant colony learning algorithm for optimal control. In Babuška, R. and Groen, F. C. A., editors, *Interactive Collaborative Information Systems*, volume 281 of *Studies in Computational Intelligence*, pages 155 – 182. Springer Berlin / Heidelberg.

van Ast, J. M., Babuška, R., and De Schutter, B. (2010b). Generalized pheromone update for ant colony learning in continuous state spaces. In *Proceedings of the 2010 Congress on Evolutionary Computation (CEC 2010)*, pages 2617–2624, Barcelona, Spain.

Vlassis, N. (2007). *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Synthesis Lectures in Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Walker, M. (2009). Ant mega-colony takes over world. URL: `http://news.bbc.co.uk/earth/hi/earth_news/newsid_8127000/8127519.stm`.

Wang, J., Osagie, E., Thulasiraman, P., and Thulasiram, R. K. (2009). HOPNET: A hybrid ant colony optimization routing algorithm formobile ad hoc network. *Ad Hoc Networks*, 7(4):690–705.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

Glossary

This glossary presents a summary of the mathematical conventions used throughout this thesis, as well as the list of abbreviations and mathematical symbols most frequently used in this thesis.

Conventions

The following mathematical conventions are used throughout this thesis.

- Boldfaced non-capital characters represent vectors. All vectors are considered to be column vectors and the transpose of a vector is denoted by the superscript T. For instance, a continuous state vector is denoted by $\mathbf{x}$, with its transpose being $\mathbf{x}^{\mathrm{T}}$. The vector of ones is denoted by $\mathbf{1}$, with its size depending on the context.
- Boldfaced capital characters represent matrices. For instance, $\mathbf{R}$ represents a matrix with $\mathbf{R}^{\mathrm{T}}$ its transpose.
- Functions operating on matrices or vectors are also displayed in boldface. For instance, the action (vector) defined by the deterministic policy in a given continuous state (vector) is denoted as: $\mathbf{u} = \mathbf{h}(\mathbf{x})$.
- The $\sim$ operator is used to denote drawing a sample from a probability distribution. For instance, when the policy is probabilistic, drawing an action from the policy in a given continuous state is denoted as: $\mathbf{u} \sim \mathbf{h}(\mathbf{x})$.
- The probability of i given j is denoted by $p\{i|j\}$.
- Sets are denoted using the calligraphic font. The empty set is denoted by $\emptyset$ and $\setminus$ denotes the set difference operator. The neighborhood set is always denoted by $\mathcal{N}$.

List of symbols

The following lists contain the mathematical symbols most frequently used in this thesis.

States and actions

$\mathcal{X}$	continuous state space
$\mathbf{x}$	continuous state vector
$\mathcal{Q}$	discrete state space
$\mathbf{q}$	discrete state vector
$\mathcal{U}$	discrete input, or action space
$\mathbf{u}$	discrete input, or action

Time and iterations

t	time index, used for counting the iteration within a trial
k	trial index, used for $\tau_{\mathbf{qu}}$
κ	trial index, used for $\tau_{\mathbf{qu}}^{\mathrm{upd}}$
K	maximal number of trials
T	maximal number of steps in a trial
T_{s}	sampling time

Ant colony optimization

c	an ant
$\mathcal{C}$	set of ants that are not in the goal
M	number of ants
$\mathbf{h}$	control policy
τ	pheromone variable
α	exponent of the pheromone variable in action selection
η	heuristic variable
β	exponent of the heuristic variable in action selection
ϵ	exploration probability in action selection
$\Delta\tau$	pheromone deposit
ρ	global pheromone decay rate
γ	local pheromone decay rate

Solutions and performance

s	a solution
$s_i^{(j)}$	the j^{th} solution component of a solution i
s_{p}	a partial solution
s_{gb}	the global best solution
s_{ib}	the iteration best solution
$\mathcal{S}$	set of solutions, solution space
$\mathcal{S}_{\mathrm{upd}}$	multiset of solutions used in the global pheromone update
$\mathcal{S}_{\mathrm{trial}}$	multiset of solutions found in a given trial
J	cost function
F	fitness function

Reinforcement learning

R	return
r	reward
V	(state) value function
Q	(state-action) value function
$\mathcal{P}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$	state transition probability function
$\mathcal{R}^{\mathbf{u}}_{\mathbf{q}\mathbf{q}'}$	expected immediate reward following the state transition
e	eligibility trace
γ_{rl}	discount factor in reinforcement learning
α_{rl}	learning rate in reinforcement learning

Particle swarm optimization

$\boldsymbol{\theta}_i$	position of a particle i
$\boldsymbol{\nu}_i$	velocity of a particle i
$\boldsymbol{\theta}_{i,\mathrm{pbest}}$	personal best position of a particle i
$\boldsymbol{\theta}_{i,\mathrm{lbest}}$	local best position of a particle i
$\boldsymbol{\theta}_{i,\mathrm{gbest}}$	global best position of a particle i

General modeling framework for swarms of moving agents

$\mathcal{S}$	swarm
M	moving agent
A	dynamic agent
$\mathbf{z}$, $\mathcal{Z}$	internal state (state space) of a dynamic agent
h	internal state transition function of a dynamic agent
$\mathbf{y}$, $\mathcal{Y}$	observation vector (observation space) of a dynamic agent
π	decision probability function of a dynamic agent
$\mathbf{u}$, $\mathcal{U}$	action (action space) of a dynamic agent
$\mathbf{v}$, $\mathbf{w}$, $\mathcal{W}$	messages (message space) of a dynamic agent
ϕ	message generating function of a dynamic agent
P	particle
$\mathbf{x}$, $\mathcal{X}$	state (state space) of a particle
g	state transition function of a particle
ψ, Ψ	process state (state space)
f	process function
C	communication channel

Miscellaneous

$\mathcal{I}$	an indicator vector
φ	sensitivity
$\mu_i(x)$	membership degree of x to the membership function i
$\beta_{ij}(x)$	degree of fulfillment of x to the membership functions i and j

Subscripts and superscripts

max	maximum
min	minimum
avg	average
exp	expected
upd	update
upper	upper bound
lower	lower bound
0	initial value
g	goal
$*$	optimal

List of abbreviations

Below, a list is given collecting the abbreviations used in this thesis.

ACL	Ant Colony Learning
ACO	Ant Colony Optimization
ACS	Ant Colony System
AI	Artificial Intelligence
AS	Ant System
MDP	Markov Decision Process
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
TSP	Traveling Salesman Problem

Summary

The very basis of this thesis is the collective behavior of ants in colonies. Ants are an excellent example of how rather simple behavior on a local level can lead to complex behavior on a global level that is beneficial for the individuals. The key in the self-organization of ants is communication through pheromones. When an ant forages for food, it is biased to search along trails of stronger pheromone concentrations. The moment it finds food, it will walk back to the nest while depositing pheromones and thereby contributing to the reinforcement of a successful trail. Inspired by this mechanism, research within an engineering context has led to the development of the field of Ant Colony Optimization (ACO). Specifically developed for efficiently solving combinatorial optimization problems, ACO has been successfully applied to routing in road traffic and Internet networks.

In this thesis, we take the principles behind ACO to the domain of control policy learning. A control policy is a mapping from states to actions and our objective is to develop methods to learn the optimal control policy for a given dynamic system by interacting with it. We call our methods Ant Colony Learning (ACL) and their power lies in the fact that there is a set of ants, from which each ant interacts with the system and influences the other ants through updating pheromone levels associated with the visited state-action pairs. In experiments involving control problems that have a discrete state space and deterministic state transitions, it turns out that ACL converges quickly to the optimal solution. We also observe that increasing the number of ants in the algorithm results in a decrease of the number of trials required for convergence to the optimal policy. An analytical study of the convergence behavior of ACL reveals that for systems with discrete and noiseless state transitions, the expected policy converges to the optimal policy in the case of using only one ant.

Another major part of this thesis deals with the application of ACL to control problems with continuous state spaces. In order to capture a continuous space with a finite number of elements, we study two ways of partitioning the state space and their incorporation in the ACL framework. In crisp ACL, the state space is partitioned using bins. Each state measurement is assigned to exactly one bin, which leads to the introduction of discretization noise, rendering an originally deterministic system non-deterministic and restricting the performance of the algorithm. We find that a better way of partitioning the state space is by using fuzzy triangular membership functions. The continuous state measurement then belongs to multiple membership functions to a certain degree. With fuzzy partitioning, the continuity of the state variables is preserved and no non-determinism is introduced. We call this method fuzzy ACL. The developed generalized ACL algorithm unifies both crisp and fuzzy ACL.

The behavior and performance of crisp and fuzzy ACL are further studied using simulation experiments. We study the influence of the local and global pheromone decay rates, the number of ants, and the density of the state space partitioning grid on the learning performance. Especially, the performance of crisp ACL improves for a small local pheromone decay rate, while fuzzy ACL outperforms crisp ACL over the whole line. In general, crisp ACL is much more sensitive to the choice of the pheromone decay parameters than fuzzy ACL. We find that using more ants leads to faster convergence, but that the number of ants does not need to be extremely large to obtain a satisfactory performance. With regard to the scaling of ACL, crisp ACL reveals a slow, but gradual improvement of the learning for an increasing state space partitioning density. Fuzzy ACL, on the other hand, improves more rapidly and requires fewer ants to learn a better control policy.

Finally, we present a general modeling framework for swarms of moving agents. It turns out that ACL fits within this framework and as such can be unified with other swarm intelligence techniques. In the future, this could result in beneficial integration of elements from other swarm intelligence techniques into ACL, or the other way around.

Samenvatting

De basis voor dit proefschrift wordt gevormd door het collectieve gedrag van mieren in kolonies. Mieren zijn een uitstekend voorbeeld van hoe vrij eenvoudig gedrag op een lokaal niveau kan leiden tot complex gedrag op een globaal niveau waarvan de individuen weer profiteren. Een sleutelrol in het zelforganiserend gedrag van mieren ligt in de communicatie door feromonen. Wanneer een mier op zoek gaat naar voedsel, wordt zij gestuurd in haar zoektocht door sporen met een sterkere feromoonconcentratie. Op het moment dat de mier voedsel vindt loopt zij terug naar het nest terwijl zij een feromoonspoor achterlaat. Hiermee draagt de mier bij aan het versterken van succesvolle paden. Geïnspireerd door dit mechanisme heeft onderzoek binnen een technische context geleid tot de ontwikkeling van het veld van *Ant Colony Optimization* (ACO). Specifiek ontwikkeld voor het efficiënt oplossen van combinatorische optimalisatie problemen is ACO succesvol toegepast op het routeren van verkeer en in Internet netwerken.

In dit proefschrift brengen we de principes die ten grondslag liggen aan ACO over naar het domein van het leren van regelaars. We zien een regelaar als een afbeelding van een toestandsruimte naar een actieruimte en ons doel is methodes te ontwikkelen voor het leren van optimale regelaars voor een gegeven dynamisch systeem door hiermee te interageren. We noemen onze methodes *Ant Colony Learning* (ACL) en hun kracht zit hem in het feit dat er een verzameling van mieren is, waarvan elke mier interageert met het systeem en de andere mieren beïnvloedt door het aanpassen van de feromoonwaarden die zijn geassocieerd met de bezochte toestand-actie paren. In experimenten met regelproblemen met een discrete toestandsruimte en deterministische toestandsovergangen blijkt dat ACL snel naar de optimale oplossing convergeert. We zien ook dat een toenemend aantal mieren in het algoritme leidt tot een afname van het aantal iteraties dat vereist is voor het convergeren naar de optimale regelaar. Een analytische studie naar het convergentiegedrag van ACL toont aan dat voor systemen met discrete en ruisvrije toestandsovergangen, de verwachte regelaar convergeert naar de optimale regelaar bij het gebruik van slechts één mier.

Een ander belangrijk gedeelte van dit proefschrift behandelt de toepassing van ACL op regelproblemen met een continue toestandsruimte. Om een continue ruimte te kunnen beschrijven met een eindig aantal elementen, bestuderen we twee manieren om de toestandsruimte te kunnen partitioneren en in te bedden in het ACL raamwerk. In *crisp* ACL wordt de toestandsruimte gepartitioneerd met discrete cellen. Hierbij wordt er discretisatieruis geïntroduceerd wat er voor zorgt dat een anderszins deterministisch systeem niet-deterministisch wordt en dat de prestatie van het algoritme afneemt. We zien dat een betere manier van het partitioneren van de toestandsruimte kan worden bereikt door het gebruik van *fuzzy* driehoekige

lidmaatschapsfuncties. De continue toestandsmetingen behoren nu tot een zekere graad toe aan verscheidene lidmaatschapsfuncties. Met *fuzzy* partitionering blijft het continue karakter van de toestandsvariabelen behouden en wordt er geen niet-determinisme geïntroduceerd. We noemen deze methode *fuzzy* ACL. Het ontwikkelde gegeneraliseerde ACL algoritme verenigt *crisp* en *fuzzy* ACL.

Het gedrag en de prestatie van *crisp* en *fuzzy* ACL is verder bestudeerd door middel van simulatie-experimenten. We bestuderen de invloed op de leerprestaties van de lokale en globale feromoonparameters, het aantal mieren en de dichtheid van de partitionering. In het bijzonder verbetert de prestatie van *crisp* ACL wanneer een lage lokale feromoonafnamesnelheid wordt gebruikt, terwijl *fuzzy* ACL het beter doet dan *crisp* ACL over de gehele lijn. In het algemeen kan worden gesteld dat, vergeleken met *fuzzy* ACL, *crisp* ACL veel gevoeliger is voor de keuze van de feromoonparameters. We zien dat het gebruik van meer mieren leidt tot een snellere convergentie, maar dat het aantal mieren niet extreem groot hoeft te zijn voor een voldoende prestatie. Met betrekking tot de schaalbaarheid van ACL bij toenemende dichtheid van de partitionering van de toestandsruimte laat *crisp* ACL een langzame, maar zekere verbetering van de geleerde regelaar zien voor een fijnere partitionering. *Fuzzy* ACL daarentegen verbetert aanzienlijk sneller en vereist minder mieren voor het leren van een zelfs nog betere regelaar.

Uiteindelijk presenteren we ook een algemeen modelleringsraamwerk voor zwermen van bewegende agenten. Het blijkt dat ACL in dit raamwerk past en daarmee kan worden verenigd met andere zwermintelligentie-technieken. In de toekomst zou dit kunnen resulteren in een goede integratie van elementen uit andere zwermintelligentie-technieken in ACL, of omgekeerd.

Curriculum Vitae

Jelmer Marinus van Ast was born in 1981 in Oud-Beijerland, the Netherlands. In 1999, he graduated from the secondary school R.S.G. Hoeksche Waard of the same town. He then studied Electrical Engineering at Delft University of Technology, the Netherlands, where he obtained his Bachelor of Science degree in 2003 and his Master of Science degree in 2005. During his M.Sc. research work, his research topic was Reinforcement Learning and he was advised by Prof.dr. Robert Babuška, M.Sc. For half a year, he worked at the European Patent Office in order to be able to spend four months in Kathmandu, Nepal, to teach elementary English and science to children in the age ranging from 6 to 16.

Since 2006, Jelmer van Ast has been working on his PhD project at Delft Center for Systems and Control, Delft University of Technology. His PhD research has dealt mainly with Ant Colony Optimization algorithms for optimal control purposes, and has been performed under the supervision of Prof.dr. Robert Babuška, M.Sc. and Prof.dr.ir. Bart De Schutter. During his PhD project, Jelmer van Ast obtained the DISC certificate for fulfilling the course program requirements of the Dutch Institute for Systems and Control, and advised a number of M.Sc. and B.Sc. students.

Jelmer van Ast's research interests include swarm intelligence and reinforcement learning.