Bee-inspired Distributed Optimization

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de rector magnificus,
   Prof. mr. G.P.M.F. Mols,
volgens het besluit van het college van Decanen,
in het openbaar te verdedigen
in de aula van de Universiteit
op donderdag 10 november 2011 om 14.00 uur

door

Nyree Paul-Peter Maurice Lemmens
geboren op 23 november 1979 te Geleen, Nederland
Promotor: Prof. dr. G. Weiss

Copromotores: dr. K.P. Tuyls
dr. A. Nowé

Leden van de Beoordelingscommissie:
   Prof. dr. ir. R.L.M. Peeters (voorzitter)
   Prof. dr. ir. F.C.A. Groen (Amsterdam University)
   Prof. dr. ir. J.C. Scholtes
   Prof. dr. K. Tumer (Oregon State University)

SIKS Dissertation Series No. 2011-38
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN: 978-94-6169-152-1
©Nyree Paul-Peter Maurice Lemmens, Maastricht 2011

Printed by Optima Grafische Communicatie, Rotterdam
Cover design by Steven de Jong & Nyree Lemmens

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronically, mechanically, photocopying, recording or otherwise, without prior permission of the author.
Being a novelty seeker, I easily made the choice of starting a Ph.D. After all, I quitted my industry job to chase after a university degree. Consequently, I was highly motivated and managed to finish my bachelor and master in only 2 years. Knowing what to expect in industry, I indicated to Karl Tuyls, my master thesis supervisor at that time, that I would like to continue working in academia. Although there were no open positions at Maastricht University at that time, Karl knew of an open Ph.D. position in Brussels. I was presented with the choice to take up that job. Additionally, I was offered the possibility to return to Maastricht if a position opened up. I chose to take the position and so I moved to Brussels and I started my Ph.D. career at the Computational Modeling Lab (CoMo) at the Vrije Universiteit Brussel (VUB), under the supervision of Ann Nowé. And what a great choice it was! I want to sincerely thank both Karl and Ann for the opportunities they have offered me and for a job well done in supervising me. Karl, Ann, I have greatly enjoyed cooperating with you and I am grateful for all of your input and support, on work as well as on personal matters.

Although Brussels is only a 1.5 hour trainride from Maastricht, living there initially felt like leaving everything behind (family, friends, and all). A special thank-you to Sven van Segbroeck for making me feel very at home in Brussels and for introducing me to a lot of nice people. Sven, my Brussels experience would not have been as amazing without you. Of course I am also very thankful to all of my other colleagues at CoMo. Thank you Yann-Michael De Hauwere, Ruben Hillewaere, Stijn Meganck, Maarten Peeters, Katja Verbeeck, and Peter Vrancx. I have enjoyed many days inside (and outside) the lab because of you. I still think fondly of the VUB barbecue and the sunny days at the KultuurKaffee. The memories always manage to put a smile on my face.

After one year in Brussels, I was offered the opportunity to go back to Maastricht to continue my Ph.D. Although returning was not an obligation, I made a promise to go back. And a promise is a promise. To emphasize again the wonderful time I had in Brussels: I had a hard time moving back and adapting to Maastricht and my new hometown, Heerlen. Luckily, I was returning to a familiar ‘nest’. Still, it took me a while to adapt to the Dutch way of working again. Of course, having wonderful colleagues at Maastricht University is very helpful. So thank-you roomies: Sander Bakkes, Guillaume Chaslot, Laurens van der Maaten, István Szita, and Philippe Uyttendaele. You made my stay at the first floor of the ‘vrouwengevangenis’ a joy. I would also like to thank Sjriek Alers, Haitham Bou Ammar, Ignacia Arcaya, Hendrik Baier, Daan Bloembergen, Pietro Bonizzi, Daniel Hennes, Michael Kaisers, Pim Nijssen, Jahn-Takeshi Saito, and Maarten Schadd for the entertaining lunches, general chit-chat, and all the smiles. I would also like to express my gratitude to the managing director of the faculty of Humanities and Sciences, Pascal Breuls. Pascal, I hope you realize how helpful our talks have been and that I always enjoy your uplifting mood whenever I stumble across you. Lastly, a very special thank-you goes out to Steven de Jong and Marc Ponsen. Over the years, I have really enjoyed our numerous conversations that put everything into... “perspective”. I could not have thought of anybody else to ask for the honorary jobs of being my paranymphs. ‘Het illustere trio’ strikes again! Also, I want to express my deep appreciation for the friendship we share.
Finally, a very BIG thanks to my family and friends for taking an interest in me, for endless support, and for just being there when it was needed. I cannot express often enough how greatly I appreciate all of you. I send a thankful smile to all of you. To conclude: having, keeping and increasing this circle of wonderful people around me puts a smile on my face every day and I feel it makes me the luckiest man in town.

Nyree Lemmens
November 2011
Contents

Preface
Table of Contents

1 Introduction
  1.1 Swarm intelligence: decentralization and self-organization ............... 1
  1.2 Aim of the thesis ......................................................................... 4
  1.3 Methodology .............................................................................. 5
  1.4 Structure of the thesis .................................................................. 6
  1.5 Contributions of the thesis ......................................................... 6

2 Background
  2.1 Reinforcement learning ............................................................... 8
     2.1.1 Markov Decision Processes .................................................. 9
     2.1.2 Policy Determination .......................................................... 9
     2.1.3 Q-learning ........................................................................ 11
     2.1.4 MDP variations without the Markov property ....................... 12
  2.2 Multi-agent Reinforcement Learning .......................................... 13
     2.2.1 Independent learning vs. Joint Action Learning ................... 14
     2.2.2 Swarm Intelligence .............................................................. 14
  2.3 Biological Swarm Intelligence ...................................................... 15
     2.3.1 Biological Examples ............................................................. 15
     2.3.2 Key principles of SI .............................................................. 19
  2.4 Computational Swarm Intelligence ............................................. 20
     2.4.1 Particle Swarm Optimization .............................................. 21
     2.4.2 Ant Colony Optimization ..................................................... 22
  2.5 Class of problems under study ................................................... 24
     2.5.1 The Foraging Problem .......................................................... 24
     2.5.2 Routing in Mobile ad-hoc networks ..................................... 25

3 Bee Colony Optimization .................................................................. 27
  3.1 Introduction .................................................................................. 27
  3.2 Modeling bee behavior ............................................................... 28
     3.2.1 Recruitment behavior ........................................................... 28
     3.2.2 Navigation behavior ............................................................. 29
  3.3 The Bee System algorithm ............................................................ 31
  3.4 ACO vs. BCO .............................................................................. 33
  3.5 Experimental set-up .................................................................... 35
     3.5.1 Type of measurements ........................................................ 36
     3.5.2 Experiment environments and characteristics ..................... 36
     3.5.3 Simulator settings ................................................................. 38
3.6 Results .................................................................................. 39
3.7 Chapter conclusion ................................................................. 42

4 Bee System with inhibition Pheromones \hspace{1cm} 45
4.1 Introduction .......................................................................... 45
4.2 Additional strategies ............................................................. 46
4.3 The Bee System with inhibition Pheromones algorithm 48
4.4 Experimental set-up .............................................................. 49
4.5 Results .................................................................................. 52
4.6 Chapter conclusion ................................................................. 55

5 Stigmergic Landmark Foraging \hspace{1cm} 57
5.1 Introduction .......................................................................... 57
5.2 Landmark navigation ............................................................. 58
5.2.1 Path planning background .................................................. 59
5.3 The Stigmergic Landmark Foraging algorithm ...................... 61
5.4 Experimental set-up .............................................................. 70
5.4.1 Type of measurements ..................................................... 70
5.4.2 Experiment environments and characteristics .................. 71
5.4.3 Simulator settings ............................................................ 74
5.5 Results .................................................................................. 75
5.6 Chapter conclusion ................................................................. 79

6 Stigmergic Landmark Routing \hspace{1cm} 81
6.1 Introduction .......................................................................... 81
6.2 Routing in Mobile Ad-Hoc Networks ..................................... 82
6.3 The Stigmergic Landmark Routing algorithm ...................... 84
6.3.1 ‘Explorer’ Behavior ......................................................... 85
6.3.2 ‘Carrier’ Behavior ......................................................... 86
6.3.3 ‘Dancer’ Behavior .......................................................... 86
6.3.4 ‘Recruit’ Behavior ......................................................... 87
6.4 Experimental Set-up .............................................................. 87
6.4.1 Simulator settings ............................................................ 88
6.4.2 Type of measurements ..................................................... 89
6.5 Results .................................................................................. 89
6.5.1 The Quantitative Comparison .......................................... 90
6.5.2 The Qualitative Comparison ............................................ 92
6.6 Chapter conclusion ................................................................. 94

7 Conclusion \hspace{1cm} 97
7.1 Answers to the research questions .......................................... 97
7.2 Answer to the problem statement ......................................... 99
7.3 Future research ................................................................. 100
7.3.1 Swarm robotics .............................................................. 101
7.3.2 Physical networks .......................................................... 101
Swarm intelligence offers an alternative way of designing intelligent systems in which autonomy, emergence, and distributedness replace control, preprogramming, and centralization.

Eric Bonabeau

In recent years, we have seen an increasing general interest in how nature efficiently handles different kinds of hard problems (Yager, 2009). Examples are how plants filter sunlight in order to efficiently extract energy from it, how the nano-structure of leaves repel water from their surface, and how combinations of active genes defer or eliminate illness. The general idea is that nature has had a vast amount of time to evolve extremely efficient problem-solving mechanisms. In the field of biology, researchers focus on the question of how and why such mechanisms work. Our focus on the other hand is how we can use such knowledge as an inspiration for our own purposes (e.g., autonomous problem solving). In this thesis, we focus on one of those mechanisms which have evolved in nature. More precisely, we focus on the mechanism that is called Swarm Intelligence (SI).

In this chapter, we provide a general introduction to this thesis. §1.1 will provide the introduction into the field of SI and will set the borders in which the research is done. This will lead to a general problem statement and research questions which together form the aim of the thesis which will be described in §1.2. §1.3 will provide the methodology that is used in order to provide answers to the research questions and the problem statement. §1.4 will describe the structure of the thesis and finally §1.5 will summarize the contributions to the field of SI.

1.1 Swarm intelligence: decentralization and self-organization

A swarm is a group consisting of a large number of individuals such as cells, insects, or even humans. Very often the individuals in a swarm have a common interest. Nature provides many examples of such swarms both at a macroscopic and microscopic level. For example, a large crowd of humans attending a concert may be considered as a macroscopic swarm. Some other examples include animal herds, schools of fish, and colonies of insects. A microscopic example is our human body which consists of microscopic swarms of dedicated cells. An example is a group of liver cells, which could be considered as a swarm. In this thesis, we focus on macroscopic swarms.
Intelligence at the level of individual swarm members is neither a necessary nor a sufficient condition for the emergence of intelligent behavior at the swarm level. The fact that it is not a necessary condition can be easily seen by looking at insect colonies. As an illustration of the fact that it is also not a sufficient condition, consider the case of a crowd of humans who all want to leave an arena through a corridor. Under normal circumstances, such a departure would progress in an orderly fashion and a steady stream of humans leaves the arena. However, when the crowd panics, chaos arises, and the resulting rushing amount of humans clogs the entrance to the corridor and the corridor itself, slowing down the crowd’s escape.

Even though, in both situations, the crowd consists of highly intelligent individuals, the resulting crowd behavior is different. This difference can be ascribed to communication. Under normal circumstances, humans tend to communicate locally, verbal or non-verbal, about who may enter the corridor first. The others will wait patiently until it is their turn to enter the corridor. In panic situations, communication breaks down or is completely lacking. The humans at the back of the arena want to escape as quickly as possible but are not able to see progress. To change the behavior of the crowd, they start to push in order to control the crowd’s movement towards the entrance of the corridor. Although the pushing force propagates through the crowd, it eventually results in clogging of the corridor. Due to the lack or delay of feedback from the humans standing in front of the corridor to the pushers, indicating that there is no space to move on, orderly crowd behavior fails.

If we analyze the above described behavior we can identify a number of properties that seem important for swarm intelligence. First, centralized control is not effective in swarms. Although an individual is able to control part of the swarm, its actions may have insufficient impact for the complete swarm to act upon. The swarm is just too large to be controlled by a single central controller. The fact that a swarm does not exhibit chaotic behavior is due to decentralized control. Second, on a higher level, decentralized control give the swarm the property of self-organization. For self-organization to occur, individuals in the swarm need to communicate. This communication can take place in a number of ways and may be direct or indirect. An example of direct communication may be speaking and listening between individuals. An example of indirect communication may be placing a direction marker in the environment which others can follow. Third, high cognition or intelligence is unnecessary. Although a swarm may profit from individuals with high cognitive power or intelligence, it is not a necessary condition for swarm intelligence to occur. A swarm may therefore both be heterogeneous or homogeneous.

Considering the above described properties, we can use the following definition for Swarm Intelligence:

**Definition 1.1:** Swarm Intelligence is the property of a large multi-agent system, be it natural, artificial or hybrid, in which decentralized, local cooperation between agents results in emerging, self-organizing global structures.

In biology, swarm intelligence is used to manage a large number of problems such as nest-selection, nest-building, foraging, larvae sorting, danger detection, environment observation, labor division, corpse clustering, cooperative transportation, guarding etc. Essentially, virtually any problem that cannot be efficiently dealt with by individuals and is divisible into sub-
problems. For example, animal foraging is characterized by searching for food inside a large and constantly changing environment. Not one individual is able to memorize the complete environment and even it was able to, the memories would become incorrect over time due to changes. For example, a bee is unable to memorize every little bit of information that can be found in their environment due to the size of the environment. Additionally, it would be impractical to keep these memories accurate. Moreover, the cost for observing the whole environment individually would just be too high or even impossible. Therefore, a swarm divides the task over the members of the swarm. Each individual samples part of the environment and gains experience in that particular part of the environment. If an individual finds food, it communicates this to other members of the swarm to enable them to exploit that experience.

The above mentioned problems, such as labor division, clustering, and foraging can all be classified as large optimization problems. More precisely, the problem of finding the best solution from all feasible solutions. Finding the optimal solution is often not feasible. Therefore, swarms often settle for a near-optimal solution. In science, there exist problems which are similar to the problems that biologic swarms try to solve. Such problems are classified as combinatorial optimization problems. Combinatorial optimization problems are characterized by a discrete (and feasible) set of solutions and in which the goal is to find the best solution. Some examples of combinatorial optimization problems are the shortest path problem, the traveling salesman problem, protein folding, routing, patrolling, and selection. We adopt the following definition for combinatorial optimization.

**Definition 1.2:** Combinatorial optimization consists of finding the least-cost solution to a mathematical problem in which each solution is associated with a numerical cost.

Researchers have started to use biological swarm behavior as an inspiration to create algorithms that can deal with combinatorial optimization problems. Currently, most research has been done in SI systems that are inspired by either fish schools or ants (Dorigo and Stützle, 2000; Eberhart and Shi, 2001; Yager, 2009). Both swarm types constitute homogeneous swarms consisting of simple individuals which have low cognition and intelligence and thus creating a virtual swarm is computationally cheap. Fish schools (as well as animal herds and bird flocking) rely on individual proximity to create swarm behavior. The end-result of such swarm behavior resembles the results of genetic algorithms in the sense that they perform population-based optimization. More precisely, each individual samples and evaluates part of the search space. In contrast to evolutionary algorithms, there is no crossover, mutation or evolving generations mechanism after evaluation. Instead, after evaluation, individuals move towards what they think is a better position in the search space. Due to the proximity strategy, the swarm as a whole moves towards this location. For example, if a school of fish is looking for food sources, each fish observes its local environment. Once a fish discovers a food source, it adapts its position. Neighbors try to keep proximity to this individual (and so on for their neighbors) with the resulting effect that the whole swarm moves towards the location where food is present. Alternatively, ants do not rely on proximity for swarming behavior. They use indirect communication strategies via the environment. Ants leave indicator marks inside of the environment to indicate previous experience or ongoing work. Other ants can use these markers to guide their actions. Nature provides many more interesting SI mechanisms. However, not all of these
mechanisms have received considerable attention. An example is the swarm behavior of honeybees. It is therefore unclear whether such SI types provide well-performing mechanisms for solving combinatorial optimization problems.

In this thesis, we focus on the SI mechanism that is displayed by bees. These insects show remarkable performance in many of the previously mentioned biologic problems. In contrast to fish schools or ants, bees do not rely on proximity or indirect communication strategies. They rely on direct communication and segmentation of the search space.

1.2 Aim of the thesis

Bee colonies are able to solve a remarkable amount of problems in nature. Yet, considerable attention, from a computational point of view, for this type of SI is lacking. We argue that algorithms inspired by this swarm behavior may pose a high-performing mechanism for combinatorial optimization problems. Therefore, the problem statement of this thesis is stated as follows.

**Problem Statement:** To what extent can bee-inspired SI algorithms provide a flexible, state-of-the-art mechanism for solving combinatorial optimization problems?

In order to answer the problem statement, we formulate four research questions.

**Research Question 1 (RQ1):** What are the key principles of bee-inspired SI and do they differ from other social-insect SI?

As mentioned previously, nature provides multiple examples of swarms. In this thesis we explore bee-inspired SI. Bees belong to the class of social insects. The best-known example of such insects are ants. Consequently, ant-inspired SI is the most studied social-insect-inspired SI. Although both SI mechanisms appear to be different, they seem to have similar underlying properties. We pinpoint these properties. Chapters 2 and 3 will provide partial answers to this question. More precisely, Chapter 2 provides four principles that can be found in all SI systems. Chapter 3 provides more details about aforementioned principles in social-insect SI.

**Research Question 2 (RQ2):** How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

Being able to model bee behavior in an algorithm, the question rises how such SI performs. In order to indicate performance, we compare various types of social-insect-inspired SI and provide conclusions about their efficiency, scalability, and adaptability. We mainly focus on ant and bee-inspired behavior and combinations of these. The comparisons are done in Chapter 3-5. Chapter 3 focuses on a comparison of algorithms inspired by basic ant and bee behavior. Chapter 4 adds a comparison between the former two and a hybrid algorithm. Finally, Chapter 5 presents a comparison between the hybrid algorithm and a pure bee-inspired algorithm with landmark navigation.
Research Question 3 (RQ3): To what extent can we eliminate potential weak points in bee-inspired SI algorithms?

From the results of the SI comparisons, we can determine strong and weak points of bee-inspired SI and give directions in what ways bee-inspired SI can be improved to eliminate weak points. These improvements do not necessarily have to result in SI mechanisms that are found in nature. More precisely, an improvement may not even be used by the biological counterparts that are used for inspiration of the SI mechanism. Chapters 3-5 each suggest possibilities for improvement. These chapters also introduce descriptions of biology-inspired strategies to realize these improvement possibilities. Lastly, they describe how to create algorithms using these strategies.

Research Question 4 (RQ4): To what extent is a bee-inspired SI algorithm capable of providing improved solutions to current state-of-the-art algorithms in complex, real-world problems?

In order to get an indication of bee-inspired SI in real-world applications, Chapter 6 presents the application of routing in Mobile Ad-hoc Networks. Using the evolution of bee-inspired SI as described in previous chapters, we show that bee-inspired SI is capable of significantly improving current state-of-the-art algorithms in real-world applications.

1.3 Methodology

In the previous sections we identified our research problem and stated the problem statement and the research questions. To answer each of the research questions, we employ the research methodology shown below.

Review of the existing literature. To answer any of the research questions, we perform a literature study in the field of biologic swarm intelligence and in computational swarm intelligence. We focus on social-insect SI and more specific on bee-inspired SI.

Purpose specification. Although it may be obvious that SI is mainly applicable in combinatorial optimization problems, each SI strategy has its own dedicated purpose. Of each strategy that we would like to employ, we indicate what its purpose is.

Algorithm design. Whenever we introduce SI strategies and their purpose, it may still be unclear how and when to use these strategies. Therefore, in each content chapter we will provide a description on how to employ strategies and how to create the corresponding algorithm.

Data collection. In order to get an indication of the performance of SI mechanisms, we have to collect data. The content chapters provide descriptions of the experiment set-up, their purpose and what performance measures are used.

Analysis and interpretation of data. By analysing the results and interpreting them with respect to the strategies, we are able to indicate how good an algorithm performs.

Reporting on and evaluating data. Combining the analysis and interpretation of the data, we are able to answer our research questions and problem statement.
1.4 Structure of the thesis

The thesis is structured as follows. Chapter 1 provides a general introduction. It expresses the aim of the thesis in a problem statement and research questions. Moreover, it also states the methodology. Finally, it gives an overview of the contributions of the research described in this thesis.

Chapter 2 describes the background of the thesis. More precisely, it focuses on reinforcement learning and swarm intelligence. It will also provide a description of the class of problems that are studied. With the content presented in this chapter we are able to partially answer RQ2.

Chapter 3 to 6 constitute the content chapters of this thesis. Each of these chapters addresses one or more of the research questions. Each chapter is structured similarly. More precisely, it provides an introduction to the subject, specifies biologic principles, provides a description of how to employ these principles in an algorithm, presents experiments and results, and finally summarizes the chapter results and provides answers to the specific research questions. Chapter 3 addresses RQ2. Chapter 4 focuses on RQ3. Chapter 5 provides an answer to RQ2-3. Finally, Chapter 6 addresses RQ4.

Chapter 7 concludes this thesis and provides definite answers to each of the research questions and the problem statement. Finally, we will give some pointers for future research.

1.5 Contributions of the thesis

This thesis provides a number of contributions to the field of Swarm Intelligence.

1. It provides a direct comparison between two social-insect-inspired SI mechanisms, i.e., i) bee-inspired SI and ii) ant-inspired SI. It identifies the strong and weak points of the mechanisms.

2. The thesis demonstrates how a hybrid approach is able to keep the strong points and eliminate the weak points of each of the separate parts. It combines the best features of the two mechanisms.

3. Finally, focusing completely on bee-inspired mechanisms, the thesis provides a mechanism with which pheromone dependency can be completely eliminated. For this, we show how to use landmarks in exploration and exploitation strategies. Effectiveness of the landmark mechanisms is shown in the complex, real-world problem of routing in Mobile Ad-hoc Networks.
The research found in this thesis is positioned in the field of Swarm Intelligence (SI) and our focus is on SI techniques inspired by social insects. In social insect colonies, each individual insect is able to perform specific tasks by using fixed actions. The local results of these actions influence the choice of fixed actions by other colony members. Stated otherwise, intelligent behavior emerges by local interaction (i.e., reinforcement or punishment of taking actions) between individuals and is learned at colony level. Therefore, SI is a subfield of Reinforcement Learning (RL) which is a reward-based learning technique (Bonabeau et al., 1999). Classic RL is performed with one learner. In this thesis we consider such RL as a special case of Multi-agent Reinforcement Learning (MARL), namely the case in which the group of agents consist of one. In MARL, we can define two extreme forms of learning. First, Independent Learning (IL) is a technique in which each individual learns behavior independently and without taking other individuals into consideration (Claus and Boutilier, 1998). A group of individuals learning the same task independently is a form of concurrent learning (Panait and Luke, 2005). Second, Joint Action Learning (JAL) is a technique in which individuals have knowledge of the actions of others and their result on the environment. This knowledge is used to learn individual behavior. Stated otherwise, JALs learn the value of their own actions in conjunction with those of other agents (Claus and Boutilier, 1998). Of course, there is a gap between these two extreme forms of learning. For example, an individual could be able to learn behavior without knowledge of another individual’s actions but with knowledge of the outcomes of these actions. As we will see, SI algorithms position themselves between these two extremes. In this chapter, we will provide the reader with the necessary background for the understanding of this thesis. In §2.1, we will provide a brief overview of (single agent) Reinforcement Learning. In §2.2, we will focus on Multi-Agent Reinforcement Learning. In §2.2.1, we will describe two extreme forms of MARL, that is, independent learning and joint action learning. In §2.2.2, we will briefly describe what Swarm Intelligence is and indicate where SI positions itself between these two forms. In §2.3, we will shed light on a number of biologic SI examples and identify a number of key principles in Swarm Intelligence. §2.4 will focus on basic computational SI and it will give a brief description of a number of basic SI optimization algorithms. Finally, in §2.5, we will describe the problems we are going to focus on in this thesis.
2.1 Reinforcement learning

Reinforcement Learning (RL) is the problem faced by an agent that has to learn how to behave in an unknown dynamic environment by trial-and-error, so as to maximize its performance (Kaelbling et al., 1996). An agent may be anything that is able to perceive and interact with its environment. For example, a robot or a software program can both be considered as agents. Panait and Luke (2005) defines an agent as follows: An agent is a computational mechanism that exhibits a high degree of autonomy, performing actions in its environment based on information (sensors, feedback) received from the environment. Typically, such agents have no, or incomplete, knowledge of the environment they function in. In order to behave optimally, they have to explore and gather experience. To be able to distinguish bad experience from good experience, the agent has to acquire feedback from the environment. Such feedback is called reinforcement or reward, and this gives the research field its name.

In a brief overview, the standard RL model functions as follows. An agent is connected to its environment via perception and action. On each step of interaction, the agent receives an input \( i \), which indicates the current state \( s \) of the environment. The agent then chooses an action \( a \) to generate a new output. The action changes the state of the environment and the value of this state transition is communicated to the agent in the form of a positive or negative reward \( r \). The agent behavior \( B \) should choose actions which in the long-run increases the sum of reinforcement-signal values. Such behavior is represented in a policy \( \pi \). A graphical example can be found in Figure 2.1. From this example it may be obvious that the agent may have to be able to learn from delayed reinforcement. An agent may have to perform a sequence of actions, while receiving insignificant reinforcement, in order to arrive at a state with high
2.1 Reinforcement learning

An agent should learn which actions are desirable based on a reward that can take place arbitrarily far in the future. Such rewards are called delayed rewards (Kaelbling et al., 1996).

### 2.1.1 Markov Decision Processes

The environment the agent ‘lives’ in is often modeled as a discrete-time, finite-state Markov Decision Process (MDP). Such a process has the Markov property, i.e., future states depend only upon the present state. Such an environment specifies a sequential decision problem for a fully observable environment with a Markovian transition model and rewards. More formally, an MDP is defined by a 4-tuple \(<S, A, T, R>\), where \(S\) is the set of finite states of the environment, \(A\) is the set of finite actions, \(T\) is the state-transition function, and \(R\) is the reward function. At each time step \(t = 0, 1, 2, \ldots\), an agent perceives the state \(s_t \in S\) of the environment. Depending on \(s_t\), the agent chooses an action \(a_t \in A\). Each action results in an output and possibly alters the state of the environment to \(s_{t+1}\). The probability of a state change from \(s_t\) to \(s_{t+1}\) due to action \(a_t\) is given by \(T : S \times A \rightarrow S\) and is specified in terms of a set of stationary transition probabilities \(P(s'|s, a) \in [0, 1]\). The quality of the state transition is fed back to the agent at the next time step by means of a reinforcement signal or reward \(R : S \times A \rightarrow r\). More precisely, this reward is the immediate or expected immediate reward after transition from state \(s\) to \(s'\) with probability \(P(s'|s, a)\) Over time, by trial-and-error, the agent learns to behave optimally under specific conditions. In other words, the agent tries to maximize its long-term reward. For more information about MDPs we refer to (Sutton et al., 1999).

### 2.1.2 Policy Determination

A solution to a sequential decision problem must specify what the agent should do for any state that the agent might reach. Such a solution is called a policy which is denoted by \(\pi\). A policy maps states to actions, \(\pi : S \times A \rightarrow [0, 1]\). An optimal policy maximizes the expected total reward and is denoted by \(\pi^*\). Given \(\pi^*\), the agent decides what to do by consulting its current percept, which tells it the current state \(s\), and then executing the action \(\pi^*(s)\). The value \(V(s)\) of a state is determined by rewards. Depending on the used model, calculation of rewards varies. A model is everything that allows the environment to be simulated.

In the finite-horizon model, the agent should optimize its expected reward for the next \(h\) steps. Everything behind that horizon is not considered in the agent’s decision. If the agent has a non-stationary policy (i.e., one that changes over time), the agent will take a \(h\)-step optimal action on its first step. That is, the best action available given that it has \(h\) steps remaining in which to act and gain reinforcement. The next step will be the \((h-1)\)-step optimal action. This continues until the agents takes its 1-step optimal action after which it terminates. In finite-horizon models, the value \(V(s)\) of a state is typically calculated by using additive rewards and is calculated as follows,

\[
V(s) = E\left(\sum_{t=0}^{h} r_t\right),
\]  

(2.1)
where \( h \in \mathbb{N} \) and \( r_t \) indicates the reward \( r \) obtained at step \( t \).

In the infinite-horizon model, the agent takes into account the long-run reward. However, future rewards are discounted by a factor \( \gamma \). The factor describes the preference of an agent for immediate reward or future rewards and also sets a bound on the possibly infinite sum. In this model, the \( V(s) \) is calculated as follows,

\[
V(s) = \mathbb{E}\left( \sum_{t=0}^{\infty} \gamma^t r_t \right),
\]

where the discount factor \( \gamma \in [0, 1] \). If \( \gamma \) is close to 0, rewards in the distant future are viewed as insignificant. When \( \gamma \) is 1, rewards in the distant future are equally significant and Equation 2.2 is equal to Equation 2.1. In other words, Equation 2.1 is a special case of Equation 2.2.

RL is primarily concerned with how to obtain the optimal policy when no model is known in advance (Kaelbling et al., 1996). This means that knowledge of the state transition function \( T(s, a, s') \) and the reinforcement function \( R(s, a) \) must be learned. There are two approaches for acquiring this information. First, the model-free approach in which a controller is learned without learning the model. More precisely, an agent tries to learn the optimal policy directly without knowing the state transition function and the reward function. Second, the model-based approach in which a model is learned and afterwards it is used to find a controller. We will only focus on the model-free approach since SI resembles such an approach.

In the model-free approach it is not certain whether an action just taken is a good one. Incorrectly evaluating such an action may have far-reaching consequences. An intuitive strategy would be to wait evaluating/rewarding actions until the end and then reward actions that led to good results and punishing actions that led to bad results. However, deciding what the end is, is difficult and might require a large amount of memory, especially in the infinite horizon model. More precisely, in order to do this, one would need to keep track of all the actions and states. To avoid this problem, a strategy is used in which estimated values of states can be adjusted based on the immediate reward and the estimated value of the next state. Algorithms employing such a strategy belong to the so-called temporal difference methods (Sutton, 1988). Under a policy \( \pi \), the value of a state is calculated as follows:

\[
V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'),
\]

where \( R(s, \pi(s)) \) is the immediate reward for taking action \( a \) recommended by \( \pi \) in state \( s \). With probability \( T(s, \pi(s), s') \), the agent then moves to a state with value \( V^\pi(s', \pi(s')) \). There is at least one optimal stationary policy \( \pi^* \) in which the value of state \( s \) is as follows:

\[
V^*(s) \equiv V^{\pi^*}(s) = \max_{a' \in A} \{ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{\pi^*}(s') \}.
\]

This value represents the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Now that we know how to calculate the optimal value function, the equation for the optimal policy can be stated as follows:
Algorithm 1 Pseudo-code for value iteration

```
initialize V(s) arbitrarily
while Policy is not good enough do
    for s ∈ S do
        for a ∈ A do
            Q(s, a) := R(s, a) + γ ∑ s′∈S T(s, a, s′)V(s′)
        end for
        V(s) := max_a Q(s, a)
    end for
end while
```

Algorithm 2 Pseudo-code for policy iteration

```
choose an arbitrary policy π'
repeat
    π := π'
    // compute the value function of π
    solve: V^π(s) = R(s, π(s)) + γ ∑ s′∈S T(s, π(s), s′)V^π(s′)
    // improve the policy at each state
    π'(s) = argmax_a {R(s, a) + γ ∑ s′∈S T(s, a, s′)V^π(s′)}
until π = π'
```

\[ \pi^*(s) = \arg\max_{a'} [R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V^*(s')] \quad (2.5) \]

To find an optimal policy, there are two possibilities. First, one can try to find the optimal value function. This can be accomplished by using a simple iterative algorithm called value iteration, see Algorithm 1. The basic idea in this algorithm is to discover the true value of each state. The optimal policy would then be the sequence of actions that maximized the expected reward. However, the true value of each state is not known initially. The true value of each state is the immediate reward plus the expected discounted reward if the agent would act optimally from that point on. Value iteration is proven to converge to correct V* values (Bertsekas, 1987), however, this may take a long time.

Therefore, second, instead of finding the optimal policy indirectly via the optimal value function, we can manipulate a policy directly, see Algorithm 2. Starting from a random policy, the algorithm calculates the value in each state given that policy and then selects a new optimal policy.

2.1.3 Q-learning

Value Iteration and Policy Iteration both are able to determine the optimal policy. Both algorithms assume that the state transition function and the reward function are known. However, the agent may not have access to this information. In order to still apply RL one has to learn this information. Q-Learning (Watkins and Dayan, 1992) is an algorithm which is capable of this. The task of a Q-learner is to determine \( \pi^* \) without initially knowing \( R(s, a) \) and \( T(s, a, s') \). For a policy \( \pi \), Q-values (also called estimated action-values) are defined as follows:
Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'). \tag{2.6}

The Q-value is the expected discounted reinforcement of taking action \(a\) in state \(s\), then continuing by choosing actions as recommended by \(\pi\). The goal of a Q-learner is to estimate the Q-values for an optimal policy. Suppose \(V^*(s)\) is the value of \(s\) assuming the best action is taken initially, then \(V^*(s) = \max_a Q^*(s, a)\). An agent’s experience consists of a sequence of distinct stages or episodes. In the \(n\)th episode, an agent observes its current state \(s_n\), and selects and performs an action \(a_n\). Once an action is performed, the agent observes a subsequent state \(s_{n+1}\), receives an immediate payoff \(r_n\), and adjusts \(Q_{n-1}^\pi\) with learning factor \(\alpha_n\). The Q-learning rule is defined as follows:

\[
Q_n(s_n, a_n) = (1 - \alpha_n)Q_{n-1}(s_n, a_n) + \alpha_n(r_n + \gamma \max_{a_{n+1}} Q_{n-1}(s_{n+1}, a_{n+1})) \tag{2.7}
\]

where \(\alpha \in [0, 1]\) and \(\gamma \in [0, 1]\). The initial Q-values, \(Q_0(s, a)\) are initialized randomly. Over time, and as long as all state-action pairs are tried sufficiently often enough, the Q-values will converge to the optimal values. Indeed, it is proven to converge to an optimum (). However, depending on the exploration strategy, it may converge to a good policy quite slowly. Hence, there is a difficult exploration versus exploitation trade-off to be made. Additionally, Q-learning makes inefficient use of the gathered data and therefore often requires a great deal of experience to achieve good performance. Whenever the Q-values are nearly converged to their optimal values, it is appropriate for the agent to act greedily, taking, in each situation, the action with the highest Q-value.

Summarizing, RL is a form of reward-based learning, i.e., there are reinforcement signals indicating the success of actions however there is no expert telling the learner which action is the best action in the long-term. After an agent has performed an action, it is only told the immediate reward for that action. It has to find the best actions by gathering experience. Evaluation of this experience is done on-line, typically concurrent with learning. Exploration of the search space is very important in RL. Settling for an action to fast may result in sub-optimal behavior.

2.1.4 MDP variations without the Markov property

A Semi-Markov Decision Process (SMDP) is a special kind of MDP and is appropriate for modeling continuous-time, discrete-event systems. Actions taken in a SMDP take a variable amount of time and are intended to model temporally-extended courses of action. Such courses of actions are called options which include primitive actions as a special case (Sutton et al., 1999). For a graphical representation, see Figure 2.2

A Partially Observable Markov Decision Process (POMDP) is a process where the underlying environment is still assumed Markov, but the data does not appear Markovian to the learner. The most direct way to create a POMDP is to take a MDP and deprive the learner of perfect information about the state of the environment (Jaakkola et al., 1995).
2.2 Multi-agent Reinforcement Learning

Learning is an essential component of intelligent behavior (Tan, 1993). However, an individual agent may not need to learn everything on its own from scratch. For example, it may take inexperienced agents considerably less time to find a shortest path to the library if other experienced agents can indicate which routes are bad and which routes are best so far. Moreover, some learning tasks may simply be too big for one agent to handle in an acceptable period of time. Consider finding a shortest path from Amsterdam to Paris without having a map. There are simply too many possibilities for one agent to handle in an acceptable period of time. In a multi-agent setting, a number of agents trying to find this route increases the probability of finding the shortest route just by the fact that more routes are sampled in the same amount of time. Additionally, it may be advantageous to exploit experience that others have gathered. For example, knowing that going north from Amsterdam in order to search for Paris results in lengthier routes, eliminates the need for a large number of exploration efforts in that direction.

Previously introduced RL methods can not simply be used in Multi-agent Reinforcement Learning (MARL). In order to use such RL methods, we need to change the representation of the RL model. MARL is represented in the following form. We have $N$ agents, indexed $1, \ldots, n$. Each of these agents has a set of discrete actions $A = A_1, \ldots, A_n$. An environment consists of a set of $S$ states. In each state $s_i$, each action that an agent takes results in a (possibly delayed) reward $R_i : S \times A \rightarrow r$ with which an agent is able to update its action probabilities. This setup is a potential impediment for previously introduced RL algorithms. When an agent is learning value of its actions in the presence of other agents, it is learning in a non-stationary environment, i.e., rewards and even transitions can be changed by actions of other agents. Convergence of Q-values cannot plainly be guaranteed in such environments. Having multiple agents performing actions in the same world does not necessarily mean that there is or has to be interaction. There is a difference between independent learning and so-called joint-action learning in MARL.

![Figure 2.2: A MDP state trajectory consists of small discrete-time steps. An SMDP consists of larger, continuous-time transitions. Options enable a trajectory to be analyzed in both ways.](image)
2.2.1 Independent learning vs. Joint Action Learning

Suppose the Multi-Agent System (MAS) consisting of S states in which 2 agents, A and B, try to learn with each having a set of 2 discrete actions. We denote the actions for agent A as \( a_0 \) and \( a_1 \) and the actions for agent B as \( b_0 \) and \( b_1 \).

An Independent Learner is an algorithm where the agents disregard other agents and learn \( Q \)-values for their individual actions. They perform their actions, obtain a reward and update their \( Q \)-values without taking into account the actions performed by other agents. For example, agent A can learn its individual optimal policy \( \pi^*_A \) consisting of actions \( a_0 \) and \( a_1 \). An agent B is also able to learn its individual optimal policy \( \pi^*_B \) with actions \( b_0 \) and \( b_1 \). A group of agents learning a task independently is a form of concurrent learning. Although each agent learns its optimal policy individually, each individual policy may still depend on the outcome of the actions of agent B. This may be the case in which agent B alters the environment in some way.

In contrast, a Joint Action Learner is an algorithm where the agents learn \( Q \)-values for joint actions as opposed to individual actions. Each agent can observe the actions of other agents. For instance, an agent A learns \( Q \)-values for joint actions, e.g., \( Q(s, < a_0, b_0 >) \), \( Q(s, < a_0, b_1 >) \), etc. Stated otherwise, each agent \( k \) has a action set \( A_k = a_1^k, ..., a_r^k \) where \( k = 1...n \) and \( n \) is the total number of agents inside of the MAS. Transition probabilities \( T(s, a, s') \) depend on the starting state \( s \), succeeding state \( s' \) and joint action \( a = (a_1, ..., a_n) \) with \( a_k \in A_k \). The reward function \( R(s, a) \) is individual to each agent, i.e., each agent can receive different rewards for the same state transition. Learning over joint actions becomes harder and harder the more agents interact with each other. Depending on the size of the state-space, this may take considerable time.

IL and JAL define two extreme forms of Reinforcement Learning. In between lie techniques which combine treats of both. An example of such a technique is Swarm Intelligence.

2.2.2 Swarm Intelligence

Swarm Intelligence is the name given to emerging collective behavior of social insects. Although each individual in a swarm of insects is relatively simple, possessing only meager intelligence (Bonabeau and Meyer, 2001), they are able to solve difficult problems collectively. For this, a swarm relies on simple, local cooperation and strength by numbers. A swarm does not have central control and even local cooperation is unsupervised. The swarm exhibits so-called self-organization which allows the swarm to adapt to changing environments very quickly and makes it very flexible. Moreover, if one individual fails in its task, others might succeed which ensures a high level of robustness. ‘Finding the shortest path to food’ or ‘building a nest’ are some examples of difficult problems such a swarm is able to handle.

Talking in terms we introduced in the previous subsection, we could describe SI as follows. We have \( n \) agents and each of these agents have a fixed set of discrete actions \( A \). Each agent tries to find a policy to solve the problem with disregard of the actions of other agents, i.e., Independent Learning. Each agent takes a sample from the problem space. During the efforts to create an individual policy, an agent is able to ‘store’ individual experience inside of the environment. Other agents are able to see this local result of actions. For instance, ants deposit pheromone during their walks through the environment and this pheromone can be tracked by other ants, see Section 2.3. Previous experience of other agents can be exploited or used...
to guide exploration. By exploiting previous search experience and ‘adding’ own experience to it, policies are reinforced on swarm level. By using previous search experience to guide exploration, agents extend and possibly improve existing policies. Since there is no literal agent action observation or prediction, we cannot consider SI as a JAL.

Therefore, one could say that SI places itself between IL and JAL. Each agent tries to learn an optimal solution independently. For this, it uses the results of other agents’ actions without considering the actual action those agents took. Each agent’s representation of the environment limits itself to the local region so that each agent and its actions can remain relatively simple while global intelligent and possibly complex behavior emerges.

2.3 Biological Swarm Intelligence

Swarm Intelligence describes the collective behavior of decentralized, self-organized systems. It finds its origin in nature where highly evolved forms of such systems can be found, e.g., fish schooling, bird flocking, animal herding, social insects. In SI, each simple individual is able to independently create a solution by following simple rules. By local cooperation between individuals, global intelligent behavior emerges and the optimal solution to a given problem can be attained. This section will provide the necessary background in biological swarm intelligence. §2.3.1 provides a number of available natural systems which can be tagged as Swarm Intelligence. The final subsection will describe the key principles of SI.

2.3.1 Biological Examples

Fish Schooling, Bird Flocking, and Animal Herding

Most swarms (see Figure 2.3(a), 2.3(b), and 2.3(c)) use their vast amounts of individuals as a means for survival and reproduction. For example, by swimming in schools, a swarm of fish (Grünaum et al., 2005; Parrish et al., 2002) looks massive and frightening to predators. Moreover, by the huge number of fish in a school there is the concept of safety by numbers. A predator cannot consume an unlimited amount of prey. Also, at least some eggs will survive a predatory run. Schooling also increases efficiency by reducing water friction. A fish can use the vortex left by a swarm member to swim more efficiently. Predatory fish gain advantage from schooling since a large swarm is better able to search for prey in large numbers. More eyes see more. Schooling fish are able to quickly respond to change in direction and speed of their neighbors. Birds flocks (Reynolds, 1987) and animal herds (Gautrais et al., 2007) use swarming essentially for much of the same reasons.

Social insects

Another (and the most-studied) example in Swarm Intelligence is that of social insects, see Figure 2.4(a) and 2.4(b). Due to the large number of individuals in social insect colonies, central control in such a swarm is impossible. Still, organization at colony level can be observed. Examples of such organizations are nest construction, brood care, nest selection, temperature regulation, foraging runs. Organization emerges through local interactions between individuals, directly or indirect via the environment.
The best-known example of social-insect SI is that of ant foraging. In ant foraging, ants deposit pheromone on the path they take during travel. Using this trail, they are able to navigate towards their nest or food. Ants employ an indirect recruitment strategy by accumulating pheromone trails in the environment. Indirect meaning that there is no direct communication between individuals. Pheromone strength indicates the ‘fitness’ of a trail but is not able to indicate direction. More precisely, an ant is not able to know beforehand to which specific destination it is traveling. When a trail is strong enough, other ants are attracted to it (i.e., there is recruitment) and will follow this trail towards a destination which results in a reinforcement of the trail. This is known as an auto-catalytic process. In particular, the more ants follow a trail, the more that trail becomes attractive for being followed, see Figure 2.5. Short paths are reinforced more quickly and will eventually be preferred (Dorigo et al., 2006; Dorigo and Blum, 2005; Dorigo and Stützle, 2004).

In contrast, honeybees do not use pheromones to navigate through unfamiliar worlds. Instead, for navigation, they use a strategy named Path Integration (PI), see Figure 2.6. A PI vector represents the insect’s knowledge of direction and distance towards its destination. Bees are able to compute their present location from their past trajectory continuously and, as a con-
sequence, can return to their starting point by choosing the direct route rather than retracing their outbound trajectory (Lambrinos et al., 2000; Müller and Wehner, 1988). To construct a PI vector, the insect does not use a mathematical vector summation as a human does, but employs a computationally simple approximation (Müller and Wehner, 1988). Using this approximation, the insect is able to return to its destination directly. More precisely, when the path is unobstructed, the insect solves the problem optimally. However, when the path is obstructed, the insect has to fall back on other strategies such as exploration or landmark navigation (Collett and Durier, 2003; Collett and Collett, 2004) to solve the problem. Obviously, bees are able to fly, i.e., when they encounter an obstacle, they can mostly choose to fly over it. In the remainder of this thesis, we refer to a home-pointing PI vector as a Home Vector (HV), a PI vector indication a goal is called a Goal Vector (GV). PI is used in both exploration and ex-
exploitation. During exploration insects constantly update their HV. It is however, not used as an exploration strategy. During exploitation, the insects update both their HV and the GV, and use these vectors as a guidance to a destination.

For recruitment, bees communicate directly by dancing in the nest, see Figure 2.7. The dance is a representation of a PI vector and therefore communicates distance and direction towards a destination (Von Frisch, 1967). The dance consists of a series of alternating left-hand and right-hand loops, interspersed by a segment in which the bee waggles her abdomen from side to side. The duration of the waggle phase is a measure of the distance to the food. The angle between the sun and the axis of a bee’s waggle segment on the vertical comb, represents the azimuthal angle between the sun and a target location, i.e., the direction in which a recruit should fly (Von Frisch, 1967; Michelsen et al., 1992; Dyer, 2002). Dance strength (i.e., its duration) indicates the ‘fitness’ of a solution. More precisely, bees know which destination they are traveling to. Depending upon the strength of a dance, the dance attracts other bees which may follow the PI vector towards a destination. High evaluated solutions result in stronger bee dances. The more bees follow a danced PI vector, the more that PI vector will be danced for and the more it attracts other bees. Eventually, the best solution prevails. Although we mainly refer to bees when we speak about Path Integration, we have to note that Path Integration can also be found in other insects such as desert ants (Müller and Wehner, 1988).

In this thesis we will focus on social insect foraging. More precisely, our main interest will be on bee foraging behavior.
2.3 | Biological Swarm Intelligence

2.3.2 Key principles of SI

No matter which natural SI system one considers, four properties stand out. First, an individual inside of the swarm does not need any representation, a map or any explicit knowledge of the global structures it helps to produce or participates in. Stated otherwise, an individual is not able to assess the global situation. Moreover, it is unable to centralize information about the state of the swarm in order to control tasks done by other swarm members, i.e., no individual acts as a supervisor. Second, concluding from this, a swarm is a decentralized system. A swarm consists of autonomous individuals which are distributed in the environment and which follow simple probabilistic stimulus-response behaviors (Deneubourg et al., 1983). Interactions between agents are based on local information and it is through these local interactions that organization emerges. Third, interaction propagates information (e.g., search experience, local swarm structure) through the swarm and manipulates the choice of activity of each individual. This is also called Stigmergy (Grassé, 1959). More precisely, information coming from the local environment (and work in progress) can guide individual activity. This essentially means that individuals need feedback mechanisms. These mechanisms can be positive, i.e., for stimulating desired behavior, but also negative, i.e., for discouraging undesired behavior. Fourth, additionally to stigmergy, interaction is also responsible for self-organization inside a swarm. Self-organization can be defined as a set of dynamical mechanisms whereby structures appear at a global level of a system from interactions among its lower-level components, without being

Figure 2.7: Distance and direction determination by means of a waggling dance. Waggling straight up on the vertical comb indicates a food source which is located at an azimuthal angle of 0° while waggling straight down indicates a food source located at an azimuthal angle of 180°.
explicitly coded at the individual level (Garnier et al., 2007). Stated otherwise, global structures arise at a global level through interaction on a local level. In order to self-organize, not only does individual behavior need to be (repeatedly) encouraged or discouraged via feedback, but also new behavior needs to be explored. Therefore, individuals have a stochastic behavior. More precisely, although it may be profitable to exploit current knowledge, it may be wise to explore new knowledge once in a while. By doing this, the swarm ensures that global performance is optimized and structures remain dynamic.

By employing these principles the system acquires a number of features. First, adaptability, which is the degree with which the system can adapt to changes. Second, robustness which is the capability of the system to deal with erroneous local information and still perform well. Third, through stigmergy the systems gains self-organization. Additionally, it enables natural labor division. For example, if a colony of ants searches for food sources, each ant samples part of the environment. Each ant deposits pheromone on its traversed path. Pheromone acts as a stimulant for performing certain actions and directs ants to profitable paths. If a path becomes less profitable, for example due to congestion of the path, pheromone deposits will decrease on this path and another path to the same food source may start attracting attention. Due to the probabilistic nature of pheromone following, the ants will begin to use both paths and divide the labour over two separate foraging paths. Fourth, efficiency, which is the ratio between effort (e.g., in time, or energy spent) and result. Even if individuals would not communicate and each would try to solve a task (for example, searching a shortest path) on its own, the mere fact that a large group is sampling the search space increases the probability that a good result is found in comparison to only one individual sampling the search space. By interaction, the efficiency is further boosted due to the exchange of knowledge.

### 2.4 Computational Swarm Intelligence

With the field maturing, more and more people can see the advantages of SI systems. Therefore the applications involving SI is seeing a slow but steady growth. For example, the U.S. military is investigating swarm techniques for controlling unmanned vehicles. The European Space Agency is thinking about an orbital swarm for self assembly and interferometry. The film industry uses SI for simulating crowds. The low speed of commercial success is due to the fact that a SI system is hard to "program". More precisely, solutions are not pre-defined but emerge as a result from individual action, interactions among individuals and between individuals via the environment. Therefore, building an SI system requires thorough knowledge of what actions an individual should perform and what interactions are needed. One way of acquiring this knowledge is through studying biological examples, such as social insects. By studying how social insects collectively perform specific tasks, modeling their behavior, and then using the model as a basis upon which artificial variations can be developed. These variations consist of tuning model parameters beyond biologically relevant ranges or by adding non-biological features to the model (Bonabeau et al., 1999). This section will describe two basic Computational SI algorithms. We will start with Particle Swarm Optimization (PSO) in §2.4.1. Afterwards, we will focus on Ant Colony Optimization in §2.4.2.
2.4.1 Particle Swarm Optimization

The PSO algorithm is inspired by the behavior of swarms consisting of fish, birds, and herding animals and provides a numerical optimization technique. The algorithm is like a Genetic Algorithm (GA) in that it is population-based and just like a GA, PSO is initialized with a population of random solutions. The approaches differ in the way these solutions evolve. In a GA, after the evaluation of each individual has finished, the population is rebuilt by the best performing individuals through crossover/mutation and evaluation is restarted. In PSO however, each solution, a so-called particle, has a particular position and velocity with which it moves through the problem space. Each particle keeps track of its position in the problem space which is associated with the best solution (i.e., fitness). Each particle keeps track of its (local) best fitness. The swarm as a whole also keeps track of its (global) best fitness.

The concept of PSO consists of the following. Suppose PSO should find the global optimum of a function \( f(x_1, x_2, x_3, \ldots, x_n) = f(\vec{X}) \). The task is to find a \( \vec{x} \) such that the function value \( f(\vec{x}) \) is either a maximum or a minimum. This is denoted by \( f^* \). The total population of particles, \( P \), is called a swarm. Each particle has a particular neighborhood relation, \( N \), defining particles it can interact with. Essentially, \( N \) determines whether a particle \( p_i \) can interact with a particle \( p_j \). The most popular version of PSO uses \( P \) as the neighborhood for each particle (Das et al., 2008). More precisely, any particle can interact with all other particles in the swarm. Each particle has two state variables, namely, its position \( \vec{x}_i(t) \) and its velocity \( \vec{v}_i(t) \in [-V_{\text{max}}, V_{\text{max}}] \). \( V_{\text{max}} \) determines the resolution of the search. A high \( V_{\text{max}} \) results in global exploration which may result in skipping good solutions. A low \( V_{\text{max}} \) results in local exploitation which may result in local optima. The particle is also able to remember its personal best evaluation, \( p_i^* \), and the global best evaluation \( g^* \). At the start of the algorithm, after all
Algorithm 3 Pseudo-code for PSO

```plaintext
//Initialize a population of particles with random positions and velocities on \( d \) dimensions of the problem space
\[ t = 0 \]
while termination condition not reached & \( t < t_{\text{max}} \) do
  //for all particles
  for \( i = 1 \) to \( i = P \) do
    //evaluate the desired optimization fitness function
    if \( f(\vec{x}_i(t)) > f(\vec{p}_i) \) then
      \( \vec{p}_i = \vec{x}_i(t) \)
    end if
    10: if \( f(\vec{x}_i(t)) > f(\vec{g}) \) then
      \( \vec{g} = \vec{x}_i(t) \)
    end if
    //change velocity and position of particle
    \[ \vec{v}_i(t+1) = \omega \cdot \vec{v}_i(t) + c_1 \cdot \phi_1 \cdot (\vec{p}_i - \vec{x}_i(t)) + c_2 \cdot \phi_2 \cdot (\vec{g}(t) - \vec{x}_i(t)) \]
    \[ \vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \]
  end for
  \[ t = t + 1 \]
end while
```

particles are initialized, an iterative process is started. At each time step \( t \) of this process, the algorithm changes the velocity of each particle towards the local best location, \( \vec{p}_i \), and global best location \( \vec{g} \). Acceleration is weighted by two random terms, \( \phi_1 \) and \( \phi_2 \), with separate random numbers for acceleration towards \( \vec{p}_i \) and \( \vec{g} \) respectively. Both are uniformly distributed between \([0, 1]\).

From (Eberhart and Shi, 2001), the basic (global) PSO mechanism can be found in Algorithm 3. A graphical example is shown in Figure 2.8. In Algorithm 3, \( t_{\text{max}} \) indicates the maximum amount of iterations. \( P \) indicates the maximal number of particles used in the swarm. Function \( f(\vec{x}_i(t)) \) evaluates particle \( i \). Each particle stores two values, (i) \( \vec{p}_i \) which indicates the personal best performing \( \vec{x} \) of particle \( i \), and (ii) \( \vec{g} \) which indicates the global best performing \( \vec{x} \). Variables \( c_1 \) and \( c_2 \) represent the weighting of the stochastic acceleration terms that pull each particle towards \( \vec{p}_i \) and \( \vec{g} \) positions. Essentially, they represent the concepts of ‘self-confidence’ and ‘swarm confidence’ respectively. \( \omega \) represents an inertia factor. For more details, we refer the reader to (Eberhart and Shi, 2001).

2.4.2 Ant Colony Optimization

Ant Colony Optimization is a class name for ant-inspired algorithms. Algorithms belonging to it are stochastic search procedures in which the central component is the pheromone model. The model is used to probabilistically sample the search space. In ACO, each virtual ant samples the search space and deposits pheromone on the path leading to a solution. Through the process of accumulation and evaporation of pheromone, the swarm as a whole is guided towards the optimal solution. Dorigo and Blum (2005) provide a formal description of this. First, they formally describe a combinatorial optimization (CO) problem.

A model \( P = (S, \Omega, f) \) of a CO consists of:

- a search or solution space \( S \) defined over a finite set of discrete decision variables and a set \( \Omega \) of constraints among the variables
- an objective function \( f : S \rightarrow \mathbb{R}^+ \) to be minimized
Algorithm 4 Pseudo-code for ACO

initialize pheromone values $\tau$ to a constant value $c > 0$

$s_{\text{best}} \leftarrow \text{NULL}$

$i = 0$

while $\text{fitness} < \text{fitness}_{\text{required}}$ $\&$ $i < i_{\text{max}}$
do

$S_i \leftarrow \emptyset$

//for all agents

for $j = 1$ to $j = n_a$
do

$s \leftarrow \text{construct solution}(\tau)$

if $s$ is a valid solution then

10: $s \leftarrow \text{localsearch}(s)$ [optional]

if $(f(s) < f(s_{\text{best}})) || (s_{\text{best}} = \text{NULL})$ then

$s_{\text{best}} \leftarrow s$

end if

$S_i \leftarrow S_i \cup s$

end if

end for

//apply pheromone update

for $j = 1$ to $j = n_a$
do

for $d = 1$ to $d = D_j$
do

20: $\tau_{jd} \leftarrow (1 - \rho) \times \tau_{jd} + \rho \times \sum_{S \in S_U \| c_i \in S} F(s)$

end for

end for

$i += 1$

end while

return $S_{\text{best}}$

The search space $S$ is defined as follows: Given a set of $n$ discrete variables $X_i$ with values $v^i_j \in D_i = v^i_1, ..., v^i_{|D_i|}$, $i = 1, ..., n$. A variable instantiation is denoted by $X_i = v^i_j$. $D$ is the set of domain values a variable can be set to. A feasible solution $s \in S$ is a complete assignment that satisfies the constraints. If the set of constraints $\Omega$ is empty, then each decision variable can take any value from its domain independently of the values of the other decision variables. A feasible solution $s^* \in S$ is a global optimal solution, if $f(s^*) \leq f(s) \forall s \in S$. The set of globally optimal solution is denoted by $S^* \subseteq S$. To solve the CO problem, one has to find a solution $s^* \in S^*$.

Every algorithm belonging to the class is a variation to the basic ACO mechanism, see Algorithm 4. From (Dorigo and Blum, 2005), the basic ACO mechanism is described as shown in Algorithm 4 where $s$ is a solution found by an agent. $S_i$ is the set with the best solutions found in iteration $i$. $s_{\text{best}}$ is the globally best found solution. $S_U$ is a subset of $S_i \cup s_{\text{best}}$ and indicates the subset that gets pheromone updates. $\tau$ is the vector of all pheromone trail parameters. $n_a$ is the total number of artificial ants. $\text{localsearch}()$ is an optional function and can be used to locally optimize solutions. $f()$ is the fitness evaluation function. $\rho \in [0, 1]$ is the pheromone evaporation rate. $n$ is the number of discrete variables.

$\text{construct solution}$ builds a solution $S$ by combining components $c^i_j$ of the search space. Components are probabilistically selected with use of the pheromone model. The component probabilities are determined as follows:

$$p(C^i_j|S^p) = \frac{[\tau^i_j]^{\alpha}[v(c^i_j)]^{\beta}}{\sum_{c^i_k \in R(S^p)}[\tau^i_k]^{\alpha}[v(c^i_k)]^{\beta}}, \forall c^i_j \in R(S^p)$$ (2.8)
2.5 Class of problems under study

SI is typically applied in the optimization domain. Applications vary from simulated optimization efforts (e.g., shortest path problems) to real-world efforts (e.g., swarm robotics). In this thesis, we focus on simulated optimization efforts.

2.5.1 The Foraging Problem

Foraging is the task of locating and acquiring resources. Typically, this task has to be performed in an unknown and possibly dynamic environment (Stephens and Krebs, 1986). The problem is also known as ‘central place food foraging’ which consists of two phases. First, leaving the start location (e.g., the nest) in search for food. Second, return to the starting location loaded with food (Sudd and Franks, 1987). In its simplest form (e.g., an open-field-like environment), this is a problem domain that can be solved by a single agent or multiple, independent agents which parallelize the effort. If agents want to solve the problem by cooperation, getting to the solution of the problem becomes more complex. Performance can be measured in time used before all items are collected and the number of items collected in a certain time span. Foraging can be seen as an abstract problem for more complex real-world problems such as network routing, information retrieval, transportation, and patrolling. In such problems, item search and item transportation are key problems. Traditionally, foraging algorithms were based on multi-agent planning; however, multi-agent planning does not scale well with a growing number of agents or a growing environment (De Weerdt et al., 2005). Due to the nature of SI, scalability problems are reduced.

We will briefly summarize the characteristics of foraging environments. Usually, foraging takes place in a large environment. In our case, this environment is divided in a square grid with a four-neighbor topology and varying in size. Per time step, agents can make one move from a certain cell to another. See Figure 2.9 for an example. Depending on the problem at hand, this environment can have various other properties, e.g.:
- **Number of constraints.** Some environments have few or no obstacles and thus allow for unconstrained movement. In other environments, movement is constrained, with the most extreme case being a labyrinth.

- **Food-source dynamics.** An environment is said to be *static* with respect to the food sources if the locations of these sources do not change in time; note that food source locations may run out of food on the long run. On the other hand, the environment is *dynamic* if food sources can suddenly appear or be moved to a new location.

- **Obstacle dynamics.** An environment is said to be *static* with respect to obstacles if obstacles cannot suddenly appear or disappear. Otherwise, the environment is *dynamic* in this respect.

### 2.5.2 Routing in Mobile ad-hoc networks

Foraging poses an abstract yet complex problem and is ideal for showing general functionality. However, to illustrate specific functionality a real-world problem is needed. To show that the ideas in this thesis can be applied to real-world problems, we choose the problem of network routing.

The problem of routing consists of selecting paths in a network over which to send data. A path consists of a sequence of nodes connecting the source to the destination. In order to route data, network nodes have to be able to direct data to appropriate nodes in order to deliver the packets to the destination. For this, each node keeps track of routes already known in so-called routing tables. An entry in such a table indicates which node to go next to in order to arrive at the destination. It may be obvious that the construction of these routing tables is very important for efficient routing.

During recent years, we see a shift from wired, always connected networks towards wireless, on-demand connected networks. In this thesis, we focus on the latter. More precisely, we focus on the problem of routing in Mobile Ad-hoc Networks (MANETs). Such networks are typically seen in regions where there is no communication infrastructure or where the use of infrastructure is expensive or difficult to use (Broch *et al.*, 1998).

MANETs are infrastructure-less and dynamic networks based on traditional TCP/IP protocols for communication between nodes. Each node in such a network functions as both a host and as a router, is mobile and can move through the environment. A node functions as long as its battery lasts. A node is able to wirelessly connect to other nodes in its local neighborhood. Stated otherwise, infrastructure is set-up in an ad-hoc manner. For an example, see Figure 2.10(a) where each node is represented by a mobile phone. Each node has a certain communication range and consequently, the network consists of uni-directional, bi-directional, and unconnected nodes. A node is able to connect to its local neighborhood but has no knowledge of the global network structure. In order to route, a node has to forward data to a node in its local neighborhood in order to reach a node outside of its connection range. Connections are typically short-lived due to mobility and possibly radio noise, see Figures 2.10(b) and 2.10(c). Additionally, the limited resources (e.g., memory, battery power, CPU power) of a node add to the complexity of the routing problem. It may be obvious that in a network with failing or congested nodes routing becomes an increasingly complex problem.
A routing protocol should therefore be able to quickly find routes, be able to adapt to changes, and be robust to failure in parts of the network. Moreover, it should also be efficient since each transmission of a packet has a resource cost.

Current routing algorithms are not fit for the routing task. More precisely, they lack efficiency and reliability in the task of routing. Apart from being able to deal with limited resources, the routing protocol should also be able to cope with constantly changing network size, traffic density, load balancing, and network partitioning (Abolhasan et al., 2003).
In this chapter, we focus on RQ2: How do social-insect-inspired SI algorithms compare to each other with respect to their performance? Until now, we described two classes of SI algorithms, namely, PSO and ACO, and also shed light on their biological inspiration. With respect to bee-inspired SI behavior we have only described the biological component in the previous chapter. A computational description is lacking. Therefore, in this chapter, we introduce a new SI-algorithm class, namely ‘Bee Colony Optimization’, and provide a description of an algorithm herein which we use as the basis for all follow-up algorithms described in this thesis. We dubbed this algorithm ‘Bee System’. Moreover, we provide a direct comparison between ACO and BCO.

3.1 Introduction

As mentioned before, biological bee behavior uses two strategies, namely a recruitment strategy and a navigation strategy. The former is used to distribute knowledge to other members of the colony. More precisely, by ‘dancing’, agents are able to directly communicate distance and direction towards a destination (Von Frisch, 1967). The latter is used to efficiently navigate in an unknown world. For navigation, bees use a strategy named Path Integration (PI). Using this, bees are able to compute their present location from their past trajectory continuously and, as a consequence, can return to their starting point by choosing the direct route rather than retracing their outbound trajectory (Lambrinos et al., 2000; Müller and Wehner, 1988).

In analogy to PSO and ACO, Bee Colony Optimization (BCO) is the class name for bee-inspired algorithms. However, in contrast to PSO and ACO, BCO is less extensively studied and research into it only started recently. Examples include, Lucic and Tedorovic (2003); Narkani and Tovey (2004); Chong et al. (2006); Teodorovic and Dell’ Orco (2006) present different BCO algorithms which pose solutions to different types of problems by employing bee

This chapter is based on the following publication.

recruitment behavior. In Lambrinos et al. (2000), the navigation behavior of bees is investigated and applied to a robot. The notable observation here is that these algorithms use only one aspect of bee behavior, i.e., either the recruitment behavior or navigation behavior. As such, there are still two important open issues. First, recruitment and navigation algorithms are currently only studied separately; a combined algorithm is undiscovered land. Second, since a combined bee-inspired algorithm currently does not exist, comparative SI studies have not yet been performed.

The remainder of this chapter is going to focus on these two issues. §3.2 will describe the modeling of bee behavior. §3.3 will provide the description of the basic algorithm, Bee System (BS). §3.4 will focus on the comparison between BCO and ACO. §3.5 describes the simulation environment and the experiments. §3.6 will present the results. Finally, §3.7 will present the chapter conclusions and provide an answer to RQ2: How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

### 3.2 Modeling bee behavior

In contrast to existing algorithms (Lucic and Tedorovic, 2003; Nakrani and Tovey, 2004; Chong et al., 2006; Teodorovic and Dell’ Orco, 2006), BS combines both biological behaviors previously mentioned. First, recruitment behavior is implemented in analogy with biological bees’ dance behavior. Agents share information on previous search experience (i.e., the direction and distance toward a certain food source) only when they are in the hive. Agents in the hive can then decide whether to exploit previous search experience obtained from other agents in the hive, or to exploit their own search experience, if available. Second, the navigation behavior used in BS either exploits previous search experience (of the agent itself or of another agent in the hive) or lets the agent explore the world using an exploration strategy which consists of a random walk. Exploiting previous search experience is guided by the PI vector that agents either have constructed themselves or have adopted from another agent in the hive.

#### 3.2.1 Recruitment behavior

In BS, if an agent has previous search experience, it tries to communicate this to other members of the colony and as a result recruit them for that particular search experience to the goal. Recruitment behavior consists of two components, namely (i) dancing and (ii) observing.

**Dancing**

An agent’s recruitment effort is not literally modeled as a dance. Instead, an agent performs a virtual dance. The virtual dance offers the agent’s PI vector directly to observing agents, by putting it in an array which can be accessed only by agents inside the hive. Each virtual dance is accompanied by a value indicating the estimated profitability of the dance. We call this value the Dance Strength (DS) which is a Real-type value and is calculated according to Equation 3.1.
In Equation 3.1, $DS_i^t \in [0, 1]$ indicates the dance strength for dancer $i$ at time step $t$ of the dance, $m$ indicates the magnitude component of the PI vector, and $\epsilon_d \in [0, 1]$ sets a value for the dance strength decay. Suppose the goal is located next to the hive, then $PI_1 = 1$, setting $\epsilon_d = 0.2$ would result in dancing for the specific goal for $1/0.2 = 5$ time steps. By using the virtual dance, agents share information on previous search experience, that is, the direction and distance toward a certain goal. As such, in BS, dancing is essential for the auto-catalytic process. Over time, DS decays which reflects the aging of search experience. Older search experience may be topped by new experiences and these will get observation priority due to higher DS values.

**Observing**

Non-dancing agents in the hive can decide probabilistically either to explore the environment or observe and exploit (i.e., copy) the previous search experience obtained by themselves or other agents in the hive, if available. Exploration and exploitation probabilities are captured in $P_e$ and $P_r$ respectively. If an agent decides to observe, the agent looks for virtual dances and determines whether the dance is of sufficient quality to adopt. Once an observing agent finds a dancer, the agent copies the corresponding PI vector (indicating the goal) from the dancing agent to its internal Goal Vector (GV). Bees use a (still) unknown decision mechanism to decide whether to exploit another bee’s experience. In BS, the decision is based on distance assessment. More precisely, an agent will exploit another agent’s experience if this experience indicates food sources at a shorter distance from the hive than the food source currently known by the agent.

### 3.2.2 Navigation behavior

In order to obtain knowledge about the environment agents have to navigate the environment. Navigation behavior consists of two components, namely (i) exploration and (ii) exploitation.

**Exploration**

In order to locate randomly located goals whose locations are not known \textit{a priori}, insects use a random walk exploration strategy. A random walk consists of a sequence of random steps and often is assumed to be a first-order Markov process. More precisely, walk history is not taken into account in the present walk. Walk step size can vary according to specific distributions and each distribution results in a different type of random walk (\textit{e.g.,} Brownian or Lévy walk). Figures 3.1(a) and 3.1(b) show two popular random walks, (i) a Brownian walk and (ii) a Lévy walk (Viswanathan \textit{et al.}, 2000). The former can be spotted in the seemingly random movement of particles suspended in a fluid. The latter is based on a mix of long trajectories and short, random movements found in turbulent fluids.
Viswanathan et al. (1999) suggests via analytical and simulated results, that a Lévy probability distribution,

$$P(l_j) \sim l_j^{\mu},$$  \hspace{1cm} (3.2)

of step lengths $l_j$ can lead to the most advantageous searches when $\mu = 2$. Such a walk is also called a Lévy flight. Exploration by insects, birds, and mammals has been found to follow a Lévy flight with $\mu = 2$ (Viswanathan et al., 2000). Therefore, in this thesis, we use Lévy flights as exploration strategy. Essentially, a Lévy flight is characterized by extremely long but rare step lengths. As a result the number of new visited sites is large. Viswanathan et al. (2000) refers to Levandowsky et al. (1988) who suggest that a Lévy flight is advantageous since the probability of returning to a previously visited site is much smaller than for a Gaussian distribution. Related to this, Viswanathan et al. (1996) also argue that the number of new sites visited is much larger for $n$ Lévy walkers than for $n$ Brownian walkers. $n$ Lévy walkers diffuse so rapidly that the competition for resources among the walkers is greatly reduced relative to the competition with $n$ Brownian walkers.

An agent performing a Lévy flight performs the following strategy. If a goal is detected directly, the agent moves to it in a straight line. If no goal is present in view range, the agent chooses a direction at random and a distance $l_j$ from the probability distribution. Next, it moves incrementally to the next position while looking for a goal in view range. If it does not detect a goal, it stops traveling after traversing $l_j$ and chooses a new direction and a new distance $l_{j+1}$. If it does find a goal, it moves to it in a straight line.

**Exploitation**

Exploitation behavior is guided by PI vectors which are created during exploration or by copying from other colony members. In both cases, the vector is updated continuously. More precisely, a PI vector is always calculated with respect to the previous one, see Figure 3.2.
Imagine an agent moving from a starting point $A$ over point $B$ to the end point $C$. Moving from $A$ to $B$ results in a direct return vector $\mathbf{AB}$ with traveled distance $c$ and direction opposite to the start angle. When the agent continues moving from $B$ to $C$, it travels distance $a$ and adjusted its direction angle with $\beta$. In order to calculate the new return distance $b$, we use the cosine rule and rewrite it as:

$$b = \sqrt{a^2 + c^2 - 2ac \times \cos\beta},$$

(3.3)

where $a$ represents the distance traveled since the last turn was made, $c$ the old homing distance, and $b$ the new homing distance. $\beta$ is the angle turned with respect to the old homing angle. Using Equation 3.3 we can now calculate $\alpha$ (the angle used for adjusting the old homing angle), once again by using the cosine rule.

$$\alpha = \arccos \left( \frac{a^2 - b^2 - c^2}{-2bc} \right)$$

(3.4)

Values obtained by Equation 3.3 and Equation 3.4 are used to construct the new PI vector.

Bee behaviour’s main feature is that it naturally constructs a direct, optimal path between a starting point, such as the hive, and a destination, such as a food source.

In BS, an agent is capable of keeping track of two vectors. First, the Homing Vector (HV) which indicates the direct path back towards the hive. Second, the Goal Vector (GV) which indicates the direct path towards the goal.

### 3.3 The Bee System algorithm

The general structure of BS is similar to that of algorithms in ACO (Dorigo and Stützle, 2000). It implements both recruitment and navigation behavior and consists of three functions, see Algorithm 5.
Algorithm 5 BCO Metaheuristic

1: ManageBeesActivity()
2: CalculateVectors()
3: DaemonActions() [Optional]

ManageBeesActivity() handles agents’ activity based on their internal state. Each agent is in one of six internal states. In each state a specific behavior is performed. State changes are outlined in Figure 3.3.

1) Explorer: indicates the agent is exploring the environment for potential goals and carries out Lévy flights to optimally cover the search space (Viswanathan et al., 2000). During exploration, the agent continuously updates its HV. If the agent arrives at the goal, its GV is set to the HV + 180°. This way, the GV represents the vector to get to the goal as if the agent would be in the hive. Afterwards, its internal state changes to Carrier.

2) Carrier: indicates the agent found a goal and is returning an item from the goal to the hive. The actions an agent takes during the return are guided by the HV. During the return, the agent continuously updates its HV in order to have an accurate home-guidance vector. The GV remains unchanged during the return and keeps indicating the goal as if the agent is already in the hive. Once the agent arrives at the hive, its internal state changes to Dancer.

3) Dancer: indicates that an agent arrived at the hive with directional knowledge of a goal (i.e., the GV). In order to recruit other colony members for this goal, the agent starts a virtual dance which other colony members can observe. This behavior is described in Subsection 3.2.1. The dance communicates the distance and directional information directly to the observers. A dance essentially consists of a Real value indicating Dance Strength (DS) which decays over time. This means the dance is only available for a certain amount of time depending on the dance strength. In BS, initial DS depends on the hive-to-goal distance as reflected in the GV. Dance decay $\epsilon_d$ is an algorithm parameter. Once the DS has decayed, the agent’s internal state changes to Rester.

4) Rester: indicates the agent remains in the hive and is ‘resting’. It either did not find a goal by exploration or it did find a goal and recently stopped dancing for it. Resting time is an algorithm parameter. Afterwards, the agent’s internal state changes depending on the number of dancers in the hive. It changes either to Explorer (i.e., no dancers) or Observer (i.e., there are dancers). Agent ‘resting’ is not essential for the system to operate. In biology this state is necessary so that insects can ‘charge their batteries’. There is no such need in a simulation. However, ‘resting’ does increase the exploitation probability of search experience other than the agent’s own search experience.

5) Observer: indicates an agent is in the hive and looking for dancers to observe, i.e., it looks for agents which have a positive value for the DS. If the dance is strong enough, the agent copies the corresponding GV value from the dancing agent and changes its agent state to Recruit.

6) Recruit: indicates an agent adopted the distance and directional information from a dancer and is traveling to the goal. Its actions are guided by the GV which is continuously updated during the travel to the goal. During this exploitation phase, the HV is also continuously updated.
Second, `CalculateVectors()` is used to compute the PI vectors for each agent, i.e., the HV and possibly the GV. A PI vector essentially consists of two values, one indicating the direction and the other indicating the distance. BS uses an exact PI vector calculation which rules out the directional and distance errors that biological PI is prone to make (Müller and Wehner, 1988; Collett and Collett, 2004). It does, however, work in a similar way.

Third, `DaemonActions()` can be used to implement centralized actions which cannot be performed by single agents. For example, collection of global information such as the collection of already discovered goals and their quality. Such information can be used to decide whether it is useful to let an agent dance. In the basic bee-inspired algorithm, `DaemonActions()` is not used.

### 3.4 ACO vs. BCO

In this section, we are going to focus on the comparison between BCO and ACO. Although ant and bee foraging strategies differ considerably, both mechanisms solve the foraging problem efficiently. In the field of Computer Science, researchers have become inspired by the behavior of social insects, since the problems these insects cope with are similar to optimization problems humans wish to solve efficiently, for instance, the Shortest Path Problem. Ant-inspired algorithms are already used to address such problems successfully (Dorigo and Stützle, 2000). We are going to compare BS with an ant-inspired algorithm (with features of Ant Colony System and MAX-MIN Ant System (Dorigo and Stützle, 2000)) in the task of foraging.

The task of foraging consists of gathering objects out of the environment and returning them to a central point, most often the starting location of the participating agents. By foraging, a biological system optimizes its energy intake per unit time (Campo and Dorigo, 2007). The goal of such systems is to find and retrieve the highest quality food source in as short a time as possible. Moreover, such systems perform their tasks in a large, possibly dynamic environment. The properties of the foraging problem fit many other optimization problems, for example, shortest path problems and network routing problems.

In ant foraging, ants deposit pheromone on the path they take during travel. Using this trail, they are able to navigate towards their nest or food. Ants employ an indirect recruitment strategy by accumulating pheromone trails in the environment. Indirect meaning that there is no direct communication between individuals. They communicate indirectly via the pheromone
trails. Pheromone strength indicates the ‘fitness’ of a trail but is not able to indicate direction. More precisely, an ant is not able to know beforehand to which specific destination it is traveling. When a trail is strong enough, other ants are attracted to it (i.e., there is recruitment) and will follow this trail towards a destination which results in a reinforcement of the trail. This is known as an auto-catalytic process. In particular, the more ants follow a trail, the more that trail becomes attractive for being followed. Short paths are reinforced more often and will eventually be preferred (Dorigo et al., 2006; Dorigo and Blum, 2005; Dorigo and Stützle, 2004). The previous described behavior is captured in the ACO meta-heuristic of Algorithm 6.

Algorithm 6 ACO Metaheuristic

1: ManageAntsActivity()
2: EvaporatePheromone()
3: DaemonActions() (Optional)

The content of the functions is pretty straightforward. ManageAntsActivity() handles virtual ant-agents’ activity. In analogy to bee-agents, ant-agents explore their environment via Lévy flights. During exploration, ants keep track of the path they traverse. More precisely, in our simulation environment, ants store every visited cell in a path sequence. Whenever an agent discovers a food source, it uses this sequence to find its way back to the nest. During its return, the agent deposits pheromone in each visited cell in order to indicate its previous search experience. Other agents consequently can exploit this search experience to be guided towards a food source. Once such a recruited agent returns to the nest, it deposits pheromone on the path it traversed and reinforces specific search experience. Ant-inspired SI uses pheromone for both navigation and recruitment. In summary, ants deposit pheromone on the path they take during travel. Using this trail, they are able to navigate towards their nest or food. Ants employ an indirect recruitment strategy by accumulating pheromone trails. When a trail is strong enough, other ants are attracted to it and will follow this trail towards a destination. More precisely, the more ants follow a trail, the more that trail becomes attractive for being followed. This is known as an auto-catalytic process. Short paths will eventually be preferred but it takes a certain amount of time before such pheromone trails emerge. EvaporatePheromone() takes care of the decay of pheromone present in the world. This function decreases the pheromone value of each cell with a constant amount. Pheromone decay has the same function as dance decay. It reflects the aging of search experience. If a specific route is not profitable anymore, it should be forgotten. Pheromone reinforcement and pheromone decay together fuel the auto-catalytic process. The pheromone decay rate is an algorithm parameter. DaemonActions() can be used to implement centralized actions which cannot be performed by single agents, such as collection of global information which can be used to decide whether it is useful to deposit additional pheromone on the path. In this research DaemonActions() is not used.

In contrast, bees do not use pheromones to navigate through unfamiliar worlds. Instead, for navigation, they use a strategy named Path Integration (PI). Bees are able to compute their present location from their past trajectory continuously and, as a consequence, can return to their starting point by choosing the direct route rather than retracing their outbound trajectory (Lambrinos et al., 2000; Müller and Wehner, 1988). For recruitment, bees communicate directly by dancing in the nest. The dance is a representation of a PI vector and therefore commu-
nicates distance and direction towards a destination (Von Frisch, 1967). Dance strength (i.e., its duration) indicates the ‘fitness’ of a solution. More precisely, bees know which destination they are traveling to. Depending upon the strength of a dance, the dance attracts other bees which may follow the PI vector towards a destination. High evaluated solutions result in stronger bee dances. The more bees follow a danced PI vector, the more that PI vector will be danced for and the more it attracts other bees. Eventually, the best solution prevails. Although we mainly refer to bees when we speak about Path Integration, we have to note that Path Integration can also be found in other insects such as desert ants (Müller and Wehner, 1988).

We want to investigate whether the BS provides a better solution to the foraging problem than an ant-inspired algorithm. More precisely, we want to investigate whether paths emerge faster with BS. Such a comparative study would need to focus on the efficiency, scalability, and adaptability of the algorithms.

### 3.5 Experimental set-up

To conduct comparative experiments with BS and the ant-inspired algorithm, we build a simulator, see Figure 3.4. Extensive experiments have been performed with respect to efficiency and scalability. Moreover, we are able to give an indication of the algorithm adaptability.
### Efficient and Scalable Artificial Immune Systems for Multiple Food Sources Optimization

#### Table 3.1: Set-up of the experiments. Colony size indicates the number of agents inside of the colony. Number of runs indicates how many times a problem case is repeated.

<table>
<thead>
<tr>
<th>Problem case</th>
<th>Environment size</th>
<th>Colony sizes</th>
<th>Number of runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unobstructed</td>
<td>110 cells</td>
<td>50, 100</td>
<td>300</td>
</tr>
<tr>
<td>Displacement</td>
<td>440 cells</td>
<td>100, 250</td>
<td>20</td>
</tr>
<tr>
<td>Obstacle</td>
<td>2800 cells</td>
<td>500</td>
<td>20</td>
</tr>
<tr>
<td>Obstacle + Displacement</td>
<td>110 cells</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>Multiple Food Sources</td>
<td></td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

#### 3.5.1 Type of measurements

We examine the performance of the two algorithms based on three properties, (i) efficiency, (ii) scalability, and (iii) adaptability. These are typical performance measures from the MAS literature (Panait and Luke, 2005). First, the efficiency of an approach may be defined by how quickly it completes the foraging task [59, 158], or by the number of items collected in a fixed amount of time. Second, scalability is a property of a system which indicates the ability to gracefully handle an increasing amount of work or enlargement of the system. For example, the dimensionality of the search space may grow rapidly with the number and complexity of agent behaviors, the number of agents, and the size of interactions between them. Third, adaptability is a property of a system which indicates the ability to adapt to occurring changes in the system. These changes can be caused by agents themselves or due to the nature of the environment. For example, agents can change the learning environment of other agents by making additional learning information available. Also, the environment could change itself by becoming partly unavailable to the agents.

#### 3.5.2 Experiment environments and characteristics

We perform four experiments to obtain the data about efficiency, scalability and adaptability. An overview of the experimental set-up can be found in Table 3.1. In all experiments, it’s the agents’ task to gather all the food sources inside of the environment. The speed with which this is done is an indication for the efficiency. It will be represented in the average number of time steps it takes to complete the foraging task. A time step is defined as the period in which each agent performed one action. Scalability is measured with respect to environment size and the number of agents. If efficiency stays high with increasing ‘load’, clearly the algorithm has a higher scalability. As such, we can get an impression of scalability by observing the efficiency when we increase the environment and the number of agents.

The experiments reflect the challenges the algorithm has to deal with. Experiment 1 (E1), Experiment 2 (E2), and Experiment 3 (E3) are used to obtain results for efficiency and scalability. E1 and E2 present a small-sized and a medium-sized environment respectively. Both consist of 4 problem cases which are displayed in Figure 3.5. Figure 3.5(a), 3.5(b), 3.5(c), 3.5(d) show the problem cases of unobstructed foraging, foraging with food-source displacement, foraging...
Figure 3.5: Experiment 1-3: problem cases. ‘h’ and ‘w’ indicate the height and width of the environment respectively. Experiment 1 and 2 represent the small and medium-sized environments and consists of all four cases. Experiment 3 represents the large environment and only consists of Case 1.

with obstacles, and foraging with obstacles and food-source displacement respectively. The final three cases all start out as the unobstructed case. In these cases the obstacle placement and/or food source displacements will be done at some moment in time. In the figures, ‘h’ and ‘w’ indicate the height and width of the environment respectively. The simulator creates a grid over the environment and divides the environment in discrete cells. In E1, ‘h’ and ‘w’ are chosen in such a way that the environment classifies as small and contains 110 cells. In E2, the environment classifies as medium-sized and contains 440 cells. E3 only consists of the problem case as depicted in Figure 3.5(a). In E3, the environment is set to be classified as large and contains 2800 cells. E1 and E2 will be performed twice, both with a different amount of agents. In E1, the colony consists of 50 and 100 agents. In E2, the colony will consist of 100 and 250 agents. E3 will be performed with a colony size of 500 agents. Colony-size variation and environment-size variation is used for scalability results. All experiments consisting of an
unobstructed problem case will also measure the average computation time per time step and indicate the total amount of time used for completing the foraging task.

**Figure 3.6:** Experiment 4: Deneubourg bridge problem case.

In Experiment 4 (E4), we perform a small experiment with the so-called Deneubourg Bridge (Deneubourg *et al.*, 1990). The environment contains two paths towards the food source, one long path and one short path, see Figure 3.6. During the experiment, the short path is blocked and the agents have to adapt to this situation and learn to use the long path instead. The algorithm which is most efficient in completing the task is clearly more adaptive. This experiment is used to obtain an indication of adaptability. Adaptability is not directly measurable. If an algorithm’s efficiency decreases little when changes in the environment occur, we can indicate that adaptability is high. Of course, this is a relative measure. We can only indicate whether BS is less or more adaptive than the ant-inspired algorithm.

### 3.5.3 Simulator settings

Each experiment has a maximum duration in time steps. After this maximum has been reached the experiment is ended. This maximum does not necessarily have to be reached. It is possible that an experiment ends because all the food items have been collected. Maximal duration is set to 2500 time steps. A food source contains 200 items of food. For the algorithms, we use the following simulator settings.

**ACO**

Pheromone deposit rate, $\tau$, determines how much pheromone an agent is able to deposit. Large values for this parameter make the agent’s trails more important but when the ‘Pheromone degrade rate’ is not set properly, a high setting will eventually cause pheromone flooding in the world. The pheromone decay rate, $\rho \in [0, \tau]$, is a parameter the user can set to indicate the degree with which a pheromone trail evaporates. When this setting is high, pheromone trails will evaporate fast and it will be hard to sustain any path. When this setting is low, pheromone trails will prevail longer, even though no agents are traversing it. A too low setting will result in a world flooded with pheromone. Agents will not be able to distinguish any paths anymore. The minimal trail strength, $\tau_{\text{min}}$, indicates how strong a trail has to get before other agents perceive it as a trail. A too low value for this setting lets agents exploit trails even though the
solution is not good. A too high value, however, may cause agents to not follow trails at all. Likewise, the maximum trail strength, $\tau_{\text{max}}$, sets an upper bound to the pheromone that can be deposited on a cell. By setting this upper bound, one can influence the effort agents have to make to sustain a trail. A low setting forces agents to traverse over a trail regularly in order to prevent the trail from degrading. By using a high setting it would take longer before the trail degrades and agents do not have to traverse over the trail as much. Above mentioned settings are determined empirically.

BS
BS starts with a certain percentage of exploring agents. Seeley (1983) indicate that about 5 to 35% of the colony goes out foraging at foraging start, while the rest remains in the hive waiting for these scouts to come back and share their experience. All our experiments are executed with a value of 40% of the total colony for initial foraging. This value is determined empirically and therefore probably differs from the biological percentage. The exploration probability, $P_e$, and exploitation probability, $P_r$, are also determined empirically.

### 3.6 Results

In this section, we will present the results of the algorithm comparison. For simplicity, we call the two algorithms by their SI class name. The comparison is based on efficiency, scalability and adaptability. Concerning efficiency, Figure 3.7 presents the corresponding result figures for the unobstructed problem case of E1. Figure 3.7(a) shows a plot of the average needed time steps for completing the foraging task at hand. Figure 3.7(b) shows a plot of the average total time. In Figure 3.8, we plot the average needed time steps to complete the unobstructed problem case with increasing environment size and increasing number of agents.

Considering efficiency, we can observe that BCO uses significantly fewer time steps to complete the foraging task at hand. Moreover, according to Figure 3.7(b), the average total time spend on the task is also significantly lower. Therefore, we can conclude that BCO is
Figure 3.8: Efficiency: Average needed time steps of ACO vs. BCO with increasing environment sizes and increasing agent sizes in Problem Case 1.

significantly more efficient than ACO in small environments. These are typical results found in all the experiments. This can be observed in Figure 3.8 where we can see how efficiency changes with increasing environment size and increasing number of agents. With respect to efficiency, we can observe that BCO is significantly more efficient in all variations of environment size and colony size. More precisely, if we calculate the $\frac{ACO}{BCO}$ ratios in each variation, we see that BCO is approximately 3 times more efficient in solving the task. For example, the time-step ratio between ACO and BCO in a small environment with 100 agents is $\frac{532.6}{164.4} = 3.2$

The remaining problem cases show similar results as can be observed in Figure 3.9.

Considering scalability, we take into account (i) increasing the number of agents and (ii) increasing the size of the environment. With respect to agent scalability, in Figure 3.10, we can observe that when we increase the number of agents and keep the world size constant, ratios decrease. This is indicated by the solid arrows. This means that the ACO algorithm is more scalable with respect to the number of agents. With respect to environment scalability, we can observe that, when we increase the world size, ratios increase. This is indicated by the dashed arrows and essentially means that the BCO algorithm is more scalable with respect to the size of the world. Note that ratios marked with a * are influenced by the maximum number of time
steps available. These ratios would be higher if no maximum number of time steps would be set. Overall, the BCO algorithm is more scalable than the ACO algorithm since it finishes its tasks much faster.

Considering adaptability, we performed an experiment in which the task set was the Denefbourg bridge. In this two-path environment, the short path is blocked after a certain number of time steps. The ACO algorithm performs better than the BCO algorithm in such a world. More precisely, the BCO algorithm is not able to complete the task at all. This is because of the fact that by using pheromone trails, the ACO algorithm has more information about the environment than the BCO algorithm. The latter will try to exploit its most direct path even when this most direct path is blocked. The ACO algorithm however, will eventually move to-
wards the unblocked path due to the accumulated pheromone on this path. To enable the BCO algorithm to obtain this environmental information, some extra features have to be added, such as landmark navigation. Due to the fact that BS does not complete the task at all, we omit the results and suffice by indicating that the ACO algorithm is more adaptive than the BCO algorithm.

3.7 Chapter conclusion

In this chapter we introduced a new SI class named ‘Bee Colony Optimization’ and we described how to model biological bee behavior. Moreover, we described the bee-inspired algorithm ‘Bee System’. In subsequent chapters, BS will serve as the basis for other bee-inspired algorithms. In order to get an idea of the performance of BS, we performed a number of experiments in which we compared BS as an BCO algorithm with an ACO algorithm in the problem domain of foraging. The results of these experiments provide an answer to RQ2: How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

In foraging, a group of agents has to gather food inside of the environment. The experiments provide a number of problem cases with food-source displacements and obstacles. Foraging is performed in differently sized environments and with differently sized colonies. From the results of this comparison we can conclude the following. First, we can conclude that the BCO algorithm outperforms the ACO algorithm in the task of foraging. According to the efficiency measures, the BCO algorithm appears to be thrice as efficient on average. Second, from the ratios we may conclude that, with growing world sizes, the efficiency of the BCO algorithm grows with respect to the ACO algorithm. It seems that by employing the bee-inspired behavior, especially the environment size is scalable whereas by employing ant-inspired algorithm especially the colony size is scalable. This is a nice feature for the BCO algorithm since multi-agent systems in foraging tasks are mostly bounded in the number of used agents while the
environment can be quite large. Third, the ACO algorithm seems more adaptive than the BCO algorithm since BS is not able to solve the Deneubourg bridge problem.

In summary, although most research in SI optimization algorithms has focused on ACO algorithms, the results presented in this chapter suggest that BCO algorithms would pose a good-performing alternative to, at least, the foraging problem.

Looking at the conclusion in this chapter, we can indicate a number of interesting open questions. First, BS seems to be less adaptive than ACO algorithms. In this chapter we indicated that this might be due to the lacking reinforcement information inside of the environment. ACO algorithms have this information in the form of pheromone. Second, how would a BCO algorithm perform in other domains.

In the next chapter, we will combine bee behavior with ant behavior in order to construct a hybrid algorithm and to get the best of both worlds.
Bee System with inhibition Pheromones

In this chapter we focus on ways to improve Bee System. Although such a bee-inspired SI algorithm proofs to be very efficient, it lacks adaptability in more complex environments. This chapter will therefore provide a partial answer to RQ3: To what extent can we eliminate potential weak points in bee-inspired SI algorithms? We will examine additional strategies and provide a comparison between a hybrid algorithm using these strategies, BS, and ACO.

4.1 Introduction

BS works very well in relatively unobstructed environments; foraging tasks are typically completed in significantly fewer time steps in comparison to a basic ACO algorithm (Lemmens, 2006). However, as environments become more obstructed, the performance of the algorithm drops (Lemmens et al., 2007b,a). With certain environments, no artificial bee is able to reach the hive anymore. This is not surprising; after all, the algorithm is based on the behavior of flying insects, which are typically not really bothered by obstacles. If such insects do fly into an obstacle, they either fly higher or explore for a while and then continue following their path. However, such behavior is not allowed or impossible to perform in most optimization problems. Therefore, the used simulation environment is two-dimensional and the artificial bees can only perform the exploration procedure whenever they bump into an obstacle while following their PI vector. Unfortunately, if the exploration phase does not take long enough or the obstacle is large, the agent will once again bump into the same obstacle. More ‘sophisticated’ methods for obstacle avoidance can help to solve this problem.

Although obstacle avoidance may alleviate the adaptability problem, it does not prevent multiple agents from making the same error. In BS, agents only learn from mistakes inside of the hive. More precisely, at colony level, high performing global solutions will be correctly selected. Local mistakes made inside of the environment itself are forgotten immediately. This means that if one agent has bumped into a wall when following its PI vector back to the hive,

This chapter is based on the following publication.
other agents traveling the same path will make exactly the same error. In fact, even the agent that initially made the error will happily do so again. This is due to its nature to always use a straight, direct path to its destination without taking obstacles into account. In order to further alleviate the adaptability problem and improve the algorithm’s performance, we need to give the agents some additional form of learning.

Obviously, both in the case of more advanced obstacle avoidance and additional learning techniques, special care must be taken to ensure that the resulting algorithm still displays the benefits of the BS algorithm.

The remainder of this chapter is structured as follows. First, in §4.2, we provide the background for the additional strategies that we want to plug into BS in order to enhance its adaptability. Next, in §4.3, we describe the new algorithm, Bee System with inhibition Pheromones. In §4.4, we will describe the experimental set-up with which we want to show the effectivity of the additional strategies. §4.5 will present the results of the experiments. Finally, in §4.6, we summarize the results, present a conclusion and provide an answer to RQ3: To what extent can we eliminate potential weak points in bee-inspired SI algorithms?

## 4.2 Additional strategies

In order to alleviate the adaptability problems of BS, we have to complement it with some additional strategies. We have chosen two strategies. First, to improve the obstacle avoidance behavior, we examine the behavior of wall following. Second, to improve the learning capabilities inside of the environment, we examine the behavior of pheromones. This section will focus on these two strategies.

### Wall following

Wall following is one of the best-known rules for traversing mazes. Such behavior can also be found in the highly evolved behaviors of insects (Cowan et al., 2006). A maze is a collection of branching passage ways in which an agent must find a route. The collection of branching passage ways may represent a complex series of pathways, see Figure 4.1.

Solving a maze is the act of finding or constructing a route through the maze from start position (e.g., the entrance) to a goal position (e.g., the exit). If a maze is simply connected (i.e., all its walls are connected together or connected to the outer boundary, see Figure 4.1),
the maze can be solved by performing wall following to one side. This means, following only walls to the left of the agent or only walls to the right of the agent is guaranteed to solve the maze. If there is an exit in the simply-connected maze, the agent will arrive at this exit. If there is no exit, the agent will find its way back to the entrance. A simply-connected maze is completely static. Adding or changing walls in such a maze could result in loops which the agent can not get out of by left- or right-wall following. Such mazes are in fact the environments we are interested in.

A dynamic maze, i.e., one where the walls and paths can change during the ‘game’ are categorized as tour puzzles. A tour puzzle is not simply connected, see Figure 4.2, and wall following does not guarantee the agent to solve the maze. However, it does provide the agent with a strategy to avoid obstacles and to explore a certain region of the maze. In order to solve such a maze, the agent will need some additional strategy to eliminate already explored paths inside of the maze.

**Learning**

In order to get out of a maze loop, the agent has to be able to memorize the loop location and how to act once this situation occurs. It has to learn what action to perform when a certain situation occurs. The skills to enable the agent to do this can be programmed into the agent. For example, a loop position memory can be stored in the form of a photographic snapshot which is compared to the current percept which triggers an action to perform on this percept. Subsequently, reinforcement learning can associate the percept with the action. However, such a snapshot structure demands a significant increase in the agent’s memory size and cognitive powers to analyze it. Such an increase contradicts with the definition of SI, namely, that the swarm consists of simple agents.

To keep the agents simple, we have to look for other strategies. In the previous chapters, we have read about the navigation and recruitment strategy of ants. Essentially, their search experience memory is stored inside of the environment in the form of attracting pheromone. Since pheromone accumulates, ants are able to reinforce search experience. The pheromone strength indicates which action is best to take and indicates locations which are interesting to navigate to. Using this behavior ensures ants to find food sources. However, accumulating
attractions pheromone may take considerable time before a useful path emerges which decreases efficiency.

Therefore, instead of using attractive pheromones, we use repulsive pheromone to indicate locations we have already seen and are considered uninteresting. For example, such a location is a dead-end in which no food source is found. Such type of pheromone is also found to be used by ants (Montgomery and Randall, 2009). Ants use the ”anti-pheromone” to indicate paths that are not interesting to exploit anymore. In the remaining part of this thesis, we refer to repulsive pheromone as inhibition pheromone in order to indicate the purpose of the pheromone, namely inhibiting the choice of a certain action of the agent.

### 4.3 The Bee System with inhibition Pheromones algorithm

In this section, we will describe the Bee System with inhibition Pheromones (BSP) algorithm. The agents still perform the same recruitment and navigation strategies as used in BS. However, we introduce two additional strategies. First, agents exhibit the wall-following behavior as observed in various insects (Cowan et al., 2006). Such behavior improves obstacle avoidance. Second, agents can place inhibition pheromones in locations that can be considered undesirable, for instance locations that lead to a dead end. This will enhance the algorithm’s adaptability. Essentially, the improved algorithm integrates some features of ACO algorithms into a BCO algorithm, making it a hybrid.

The first strategy improves the obstacle avoidance capabilities of the agents. Wall-following behavior is only used in the internal states ‘Carrier’ and ‘Recruit’ of the agents, see Chapter 3. Wall-following behavior is implemented in a relatively simple way. Whenever an agent bounces into an obstacle while following a PI vector, it randomly selects a direction (left or right) and then starts following the contours of the obstacle in that direction, until following the PI vector becomes possible again. This behavior is illustrated in Figure 4.3.
The second strategy introduces additional learning into the algorithm. The strategy is performed by agents with internal states ‘Explorer’ and ‘Carrier’. Such agents can deposit a certain quantity of inhibition pheromone at a location which is considered uninteresting. In a static environment, this pheromone can be permanent. In a dynamic environment, the pheromone can slowly evaporate, ensuring that the agents adapt to changes in the environment, such as obstacles which are being removed or added. Moreover, in highly dynamical worlds, all the pheromone can be instantly forgotten with a certain, low probability. The pheromone deposited at a specific location is a sign to all ‘Carrier’ and ‘Recruit’ agents (i.e., agents following a PI vector) not to move to this location. Depending on whether the environment is static or dynamic, ‘Explorer’ agents can be instructed to stay out of marked locations as well, or to completely ignore the pheromone marks, respectively. More precisely, in a static environment, locations that were marked once are certainly good to stay out of, whereas in a dynamic environment, an item of food may appear in an area that was previously marked. In this case, the agent finding the food ‘eats’ the pheromone that is still present on its way from the food source to the hive.

We define two situations in which agents deposit inhibition pheromones. The two situations are illustrated in Figure 4.4. First, in Figure 4.4(a), whenever an agent is surrounded by three walls (where walls include locations where pheromone was already deposited), and there is no food or hive present there, this location clearly is a dead end. As a consequence, it should no longer be visited and therefore receives inhibition pheromone. Second, in Figure 4.4(b), the same applies whenever an agent is standing in a corner, but not in a corridor, and there is no food or hive present in this corner.

4.4 Experimental set-up

In a set of experiments, we want to find out whether BSP outperforms both the BS and the ACO algorithm in the problem domain of foraging in relatively unobstructed environments as well as more constrained environments. In order to compare the three algorithms under study under various circumstances, we developed a set of three experiments. We will discuss these experiments here, including an overview of the characteristics of the environments and the settings used for the algorithms. All experiments have been conducted in the same simulator that has previously been used in Chapter 3. The tool is now extended with the hybrid algorithm BSP. The experiment sets are defined as illustrated below.
Figure 4.5: Experiment set 1: Unobstructed environment. ‘h’ and ‘w’ indicate the height and width of the environment respectively.

Experiment set 1 consists of two cases E1.1 and E1.2, only differing in the size of the environment which is determined by ‘h’ and ‘w’, see Figure 4.5. The set presents a unobstructed environment in which the agents are free to roam and are only limited by the edges of the environment.

**E1.1: Unobstructed environment.** As a basic experiment, we run the algorithms in a static environment with no constraints. ‘h’ and ‘w’ are chosen in such a way that the environment consists of 100 cells, of which the outer edge consists of walls. There is a hive in the bottom left corner and a single food source in the top right corner. Since our earlier work (Lemmens, 2006; Lemmens et al., 2007b,a) has already led to the conclusion that the BS is much faster than the ACO algorithm, we do not include the ACO algorithm in this comparison.

**E1.2: Unobstructed environment.** An identical set-up is used with 1600 cells. The task is likely to be too extensive to finish within 2000 time steps by any of the algorithms. This time step limit is introduced to limit experiment time. Using this experiment, we can determine the scalability of the algorithms with respect to a growing environment size.

Experiment set 2 consists of two cases E2.1 and E2.2, see Figure 4.6. Both contain the Deneubourg Bridge (Deneubourg et al., 1990) which BS is unable to solve.

**E2.1: Deneubourg Bridge environment.** The Deneubourg Bridge is a highly constrained environment with obstacle dynamics. ‘h’ and ‘w’ are chosen in such a way that the environment consists of 400 cells, with a short (18 cells) and a long (42 cells) path between the hive and the food source, see Figure 4.6(a). The experiment consists of three phases. Each phase has a max-
maximum duration of 2000 time steps. The first phase presents the Deneubourg Bridge with both paths open. After the first phase is terminated, an obstacle is placed in the short path and the food source is refilled to create the second phase. Once again after the phase is terminated, the obstacle is removed and the food source is refilled to create the third phase. Finally, we wait until the experiment terminates. We then measure the average number of time steps needed over the three phases.

**E2.2: Hard Deneubourg Bridge environment.** An identical set-up to E2.1 is used in a modified, less constrained environment where the long path has a problematic ‘trap’ area, see Figure 4.6(b).

Experiment set 3 consists of 1 problem case E3, see Figure 4.7.

**E3: Alligator environment.** In this environment, ‘h’ and ‘w’ are chosen in such a way that the environment consists of 400 cells and includes food source dynamics and a complicated constraint. The experiment consists of two phases. Each phase has a maximum duration of 2000 time steps. In the first phase, the initial food source is located directly next to the hive, but the two locations are separated by a large, irregular obstacle. After the first phase is terminated, a new food source appears in the upper right corner to create the second phase. Depending on whether the original food source has been emptied or not, there are now either one or two different food sources. Once again, we wait until the experiment is terminated. We then measure the average number of time steps needed over the two phases.

The following settings are selected for the simulator. In every experiment, we simulate using 50 and 100 agents. Initially, 50% of the agents are exploring. Every food source contains 200 items of food and every agent can carry one such item at a time. The experiments are terminated after all the food has been gathered, or after 2000 time steps (whichever comes first) to save on simulation time and to prevent deadlocks which can occur when using the BS. With BS, all agents could get stuck behind an obstacle due to the nature of the behavior. Every
Figure 4.7: Experiment set 3: "Alligator". ‘h’ and ‘w’ indicate height and width of the environment respectively. The blurred food source is moved in the direction of the arrow.

experiment is repeated 30 times to compensate for the inherent randomness in the algorithms. In the experiments involving BSP, ‘Explorer’ agents are allowed to explore inhibited locations, i.e., we do not assume an environment to be static.

The following settings are selected for the algorithms. For the ACO algorithm, we select an evaporation of 1% per time step, 2 units of pheromone deposited per time step when carrying food, maximal trail strength of 10, minimal trail strength for a trail to be followed of 5. The two latter parameters are determined empirically and increase the fairness of the comparison. For BS, no additional settings were needed. For BSP, 1 unit of pheromone is deposited per time step in case of inhibition situations. The maximal pheromone amount in a cell is set to 1. Only ‘Carrier’ and ‘Recruit’ agents can deposit inhibition pheromones. This holds for all experiments except for E1.1 and E1.2, where ‘Explorer’ agents can also deposit pheromones. This is due to the nature of BSP. If ‘Explorer’ agents are not allowed to deposit pheromones in these unobstructed environments, BSP would essentially perform the same behavior as BS. ‘Explorer’ agents may enter inhibited locations. If they find food in a location filled with pheromones, they can ‘eat’ the pheromone on their way back. In E3, we obtain the best performance by ‘forgetting’ all the pheromones with a probability of 0.004 per time step.

4.5 Results

This section will present the results of the algorithm comparison. We compare the performance of the algorithms under study with respect to their efficiency, scalability and adaptability. Efficiency can be measured by determining the average number of time steps the algorithms required to gather all the food present in the environment. Scalability relates to the computation time required to perform this task. Adaptability is not directly measurable in our settings;
### Efficiency

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Experiment 1.1</th>
<th></th>
<th>Experiment 1.2</th>
<th></th>
<th>Experiment 2.1</th>
<th></th>
<th>Experiment 2.2</th>
<th></th>
<th>Experiment 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
<td>Var</td>
</tr>
<tr>
<td>Using 50 agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>252.0</td>
<td>13.9</td>
<td>628.9</td>
<td>189.4</td>
<td>2000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Bee</td>
<td>602.6</td>
<td>97.6</td>
<td>2000.0</td>
<td>0.0</td>
<td>175.9</td>
<td>3.6</td>
<td>707.6</td>
<td>106.5</td>
<td>2000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>BSP</td>
<td>418.8</td>
<td>129.6</td>
<td>2000.0</td>
<td>0.0</td>
<td>170.5</td>
<td>7.1</td>
<td>470.9</td>
<td>189.2</td>
<td>1884.4</td>
<td>188.4</td>
</tr>
<tr>
<td>Using 100 agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>144.5</td>
<td>6.6</td>
<td>394.2</td>
<td>129.4</td>
<td>1977.6</td>
<td>100.2</td>
</tr>
<tr>
<td>Bee</td>
<td>264.2</td>
<td>29.7</td>
<td>1953.8</td>
<td>206.6</td>
<td>95.6</td>
<td>3.4</td>
<td>719.4</td>
<td>148.8</td>
<td>2000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>BSP</td>
<td>191.4</td>
<td>39.7</td>
<td>1953.8</td>
<td>206.6</td>
<td>99.9</td>
<td>4.7</td>
<td>213.6</td>
<td>20.9</td>
<td>1387.5</td>
<td>283.6</td>
</tr>
</tbody>
</table>

| Table 4.1: The efficiency of the three algorithms under study, i.e., the number of time steps used to complete the experiments at hand; both average and variance are displayed. In every experiment, the maximum possible number is 2000. |

(a) Efficiency in E1.1 and E1.2

(b) Efficiency in E2.1, E2.2, and E3

**Figure 4.8:** Efficiency: Average needed time steps for solving foraging task.

However, an algorithm that is not able to learn from its mistakes will perform (a lot) worse than an algorithm that can.

**Efficiency.** We report the results on the efficiency of the three algorithms in Table 4.1. A graphical representation can be found in Figure 4.8. In E1.1, we see that BSP manages to slightly improve on the results of BS. This is due to the fact that ‘Explorer’ agents are not allowed to explore fields of inhibition pheromone in BSP. If they would be allowed to do this, BSP would have an equal performance result. As has been mentioned earlier, BS already constituted an improvement over the ACO algorithm of approximately a factor three with respect to overall efficiency. In E1.2, the algorithms usually fail to complete the task at hand within 2000 time
The scalability of the three algorithms, i.e., the required computation time per time step in milliseconds; average and variance are displayed.

Steps, as expected. In E2.1, BS and BSP perform comparably well, and definitely better than the ACO algorithm. The reason that BS works here, and was reported not to work in Lemmens et al. (2008), is that we used a different problem representation in earlier work, i.e., the grid over the environment was hexagonal in previous work. As a consequence, agents could make circular movements around the hive or the food source when returning from the blocked passage. In the current problem representation, this is no longer possible, and since agents usually prefer any move over turning back, they are able to find the alternative, longer path. In contrast, in E2.2, BS fails to find a solution after the shortest path has been blocked. The ACO algorithm does slightly better, leading to a lower average number of time steps for the whole task. BSP is by far the best-performing algorithm once again; observing it during each of the three phases of the experiments, we see that it quickly finds a good path between the hive and the food source. In E3, BS does not find any solution. The ACO algorithm manages to establish a path, but does so only just before the experiment runs out of time. Meanwhile, BSP has usually already retrieved all the food items, especially with 100 agents.

**Scalability.** We measured the total time required to run the algorithms in the five different experiments and divided this by the number of time steps used. Results are summarized in Table 4.2. In every experiment, the ACO algorithm is much slower than the other two algorithms. BSP is only slightly slower than BS on the computation-time-per-time-step measure, even in E1.2. In E3, it is actually faster than BS, which is caused by the fact that agents in BS get stuck while trying to exploit their PI vector; by implementation, exploiting agents require more computation than exploring agents. When we consider the number of time steps needed by BS and BSP, it is clear that BSP usually finds a solution using less computation time than BS.

**Adaptability.** As has been mentioned earlier, adaptability is not easily measurable. However, by observing the behavior of the algorithms during the experiments and by looking at their efficiency, we may conclude that the added, ant-inspired functionality in BSP made the algorithm as adaptive as the ant-inspired algorithm; BS, in contrast, is only adaptive with respect to changing food sources by ‘advertisement’ in the hive. It lacks adaptiveness outside of the hive.

### Table 4.2

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Experiment 1.1</th>
<th>Experiment 1.2</th>
<th>Experiment 2.1</th>
<th>Experiment 2.2</th>
<th>Experiment 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
<td>Var</td>
<td>Mean</td>
</tr>
<tr>
<td>Using 50 agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>93.9</td>
</tr>
<tr>
<td>Bee</td>
<td>16.0</td>
<td>0.2</td>
<td>17.2</td>
<td>1.5</td>
<td>17.3</td>
</tr>
<tr>
<td>BSP</td>
<td>18.5</td>
<td>0.7</td>
<td>17.2</td>
<td>1.5</td>
<td>18.6</td>
</tr>
<tr>
<td>Using 100 agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>186.9</td>
</tr>
<tr>
<td>Bee</td>
<td>34.0</td>
<td>1.4</td>
<td>36.2</td>
<td>3.3</td>
<td>34.6</td>
</tr>
<tr>
<td>BSP</td>
<td>39.0</td>
<td>1.5</td>
<td>36.2</td>
<td>3.3</td>
<td>36.4</td>
</tr>
</tbody>
</table>
4.6 Chapter conclusion

In this chapter, we presented a hybrid SI algorithm named Bee System with inhibition Pheromones which should combine the efficiency and scalability of the previously introduced BS (Lemmens et al., 2007b,a, 2008) with the adaptive capabilities of ACO algorithms. To create this hybrid, we introduced two additional strategies. First, a wall-following strategy to improve obstacle avoidance. Second, we added the possibility for agents to deposit pheromones. Instead of using attracting pheromones to indicate interesting locations to go to, we let agents deposit repulsive or inhibition pheromones to indicate locations which are of no interest. Effectively, we take an ant-inspired feature and turn it upside down.

Observing the results, we may conclude that the proposed algorithm BSP indeed is a valuable addition to the currently existing collection of insect-inspired foraging algorithms. Both additions increase the performance of the BSP algorithm while keeping its scalability acceptable. In a completely unobstructed and static environment, the additional strategies remain largely unused, which essentially makes BSP perform in the same way as BS, both in terms of number of time steps needed to complete the task at hand and in terms of calculation time required per time step. BSP does get some small improvement in such an environment if the explorers are not allowed to explore in inhibited locations. In that case ‘Explorer’ agents are forced towards the food source which speeds up recruitment in the end. In summary, BSP potentially outperforms BS which in turn significantly outperforms the ACO algorithm. In more obstructed environments, following walls and depositing pheromones definitely increases the ability of the algorithm to navigate back home (or towards food), without requiring too much additional computation. In the most extreme case, for instance in a world with a large number of obstacles, there is pheromone almost everywhere in the environment, leading to performance and computational demands similar to the ant-inspired algorithm; however, pheromones are much less often deposited and updated in BSP. Again summarizing, BSP performs significantly better than an ACO algorithm, which in turn outperforms BS. The required computation time is only slightly higher than that of BS, and drastically lower than that of the ACO algorithm. We can therefore indicate that BSP truly combines the ‘best of both worlds’.

Although BSP outperforms the other algorithms, the algorithm has still one major issue. More precisely, in the process of creating the hybrid algorithm, we introduced pheromone. In simulation environments this may not be a problem. However, if we want to use such an algorithm in an embodied system we have to be able deposit and detect pheromone-like substances. This is not a trivial issue. Therefore, we would like to create a mechanism which gives us good adaptability without the need of pheromone.

In the next chapter, we will introduce the use of landmarks and explore the possibilities how to use such a strategy to improve BS.
We have seen that a pure bee-inspired SI algorithm seems to benefit from additional ant-inspired strategies which enhance the adaptability of the algorithm. However, such a hybrid algorithm depends on pheromone. Such a dependency is not straightforward to include in physical environments. Therefore, in this chapter, we examine landmark-navigation strategies to eliminate such dependencies and thusly provides a partial answer to RQ3: To what extent can we eliminate potential weak points in bee-inspired SI algorithms? We also provide a comparison between the resulting algorithm and the previously introduced hybrid algorithm which provides a partial answer to RQ2: How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

5.1 Introduction

In this chapter, we focus on landmark navigation in large groups. More precisely, we focus on swarm communities like bee colonies. Individuals of such colonies are limited in their means to navigate in typically unknown environments due to limitations in, for example, brain capacity. However, where the individual fails, the swarm as a whole is able to perform complex tasks such as finding the shortest route to a food item.

In biology, bees use landmarks to represent a segment in the total route to the food source (Collett et al., 2002) and the landmark triggers the follow-up action to take to get to the next segment. Detection of landmark depends on the characteristics of the environment. For example, if a specific view of a mountain range is considered as a landmark, then the landmark position may be relatively static, assuming that a specific view of the mountain range does not change over short time. In contrast, if a snapshot of a pile of leaves is considered as a landmark,
then the landmark position is relatively dynamic, assuming that such a pile may move or vanish over time. Therefore relatively speaking, detection of landmarks is inherently dynamic and obviously so is the adaptability of the bees. Moreover, due to route segmentation, landmark navigation ensures robustness and reduces accumulating navigation errors.

The concept of landmark navigation is already used in mainly robotics (Hayet et al., 2007; Loizou and Kumar, 2008; Strasdat et al., 2009). However, current research has an individualistic focus, for example, single robot landmark navigation. Swarm-based landmark navigation is largely unexplored territory. This chapter focuses on such landmark navigation. In particular, we extend the previous introduced BS (Lemmens et al., 2008) with landmark navigation. In the resulting new algorithm, dubbed Stigmergic Landmark Foraging (SLF), each agent is able to contribute to the success of a landmark route. More precisely, agents are able to collectively create a landmark network from and to goals. A landmark can contain multiple routes to one or more goals. The more agents follow a route, the more that route becomes attractive for being followed. Summarizing, we present a stigmergic landmark algorithm. With respect to pheromone-based algorithms, such a solution is easier to realize in the physical world since landmarks are either already physically available, or could easily be placed in the environment by means of, for example, RF-ID tags.

The remainder of this chapter is structured as follows. In §5.2, we present an overview of related work in the area of landmark navigation. Next, in §5.3, we describe the new landmark algorithm, SLF. §5.4 and §5.5 will present the experimental set-up and the results respectively. Finally, in §5.6, we conclude and provide an answers to RQ2 and RQ3.

5.2 Landmark navigation

Although the algorithms mentioned in previous chapters are able to solve combinatorial problems, such as foraging, with high degree of performance, researchers keep looking for ways to boost performance, limit overhead (e.g., communication), and increase adaptability and scalability. Nature pursues the same objectives. For instance, in bee communities, landmark navigation is used as an additional navigation strategy next to PI (Collett, 1996). It actually serves as a strategy to make PI navigation more accurate (Cartwright and Collett, 1983; Collett et al., 1992). Each member of the colony is able to segment the path traveled and measure its length by recognizing landmarks and acting on them once they are recognized. A bee is able to basically classify two types of landmarks, i) global landmarks and ii) local landmarks. Global landmarks are used to guide the bee to the rough area of the goal and can be used over large spatial distances (Collett and Collett, 2002; Vlasak, 2006). Local landmarks are close to the goal (or the hive) and are therefore a natural choice for pinpointing the goal’s position (Cheng et al., 1987; Collett and Collett, 2002; Vlasak, 2006). However, local landmarks are not as reliable as global landmarks. Local landmarks (e.g., a leaf) tend to change over time whereas global landmarks (e.g., a mountain range) tend to be stable. Therefore, bees preferably attend to global landmarks (Vlasak, 2006). The distance that spans between two consecutive landmarks is called a segment and is represented by a so-called local vector. Each local vector is attached to a snapshot of the landmark that is seen at the beginning of the segment, such that when a landmark is recognized, the bee recalls the associated local vector that guides it to the
next landmark. Local vector memories can be linked together in a sequence so that, even in the absence of the triggering landmark, memory recall occurs (Collett et al., 1998, 2002; Collett and Collett, 2002; Menzel et al., 1996). Each bee furthermore keeps an up-to-date global vector which represents the distance and direction between the start location and the destination (Collett et al., 1998). The global vector serves as a backup strategy in case the bee gets lost (Chittka and Kunze, 1995). However, when bees move over familiar territory, local vectors (linked to global landmarks) activated in a sequential fashion take precedence over the global path integration vector (linked to the starting location and the destination).

5.2.1 Path planning background

In recent years, researchers have started to take inspiration from landmark navigation behavior for solving the task of navigation and path planning. Performance results of these studies are mainly illustrated in robotics and agents research. In this subsection we will first briefly overview basic path planning algorithms and their properties in order to provide the reader with sufficient background. Then, we describe recent contributions to path planning and navigation using landmark behavior.

Basic path planning algorithms find their origin in Dijkstra’s algorithm and a special case of Dijkstra’s algorithm, A* (Dijkstra, 1959; Nilsson, 1982). The algorithms uniformly discretize the search space and afterwards plan an optimal path through the discrete search space. Algorithms such as Dijkstra’s and A* show good performance in static and limited-sized path planning problems. However, real-world problems often involve uncertainty and dynamics in a large search space. Traditional path planning algorithms are not able to cope with such problems due to memory issues and computation time (Ferguson et al., 2005). For example, suppose the environment changes but affects the current optimal solution in a minor way. Traditional path planning algorithms such as Dijkstra’s, would need to completely replan to fix the optimal solution which takes precious computation time. In order to overcome these problems, researchers use heuristics to guide path planning, such as potential field techniques (Barraquand et al., 1992). Although such techniques reduce the issues, performance still suffers with increasing problem complexity. Randomized approaches such as Rapidly-exploring Random Trees (LaValle and Kuffner Jr, 2001) and Probabilistic Roadmaps (Kavraki and Latombe, 1998) offer a solution to these issues by randomly sampling points in the (continuous) search space and then grow the current solution towards these points. Environment dynamics can be dealt with by either (i) planning on a snapshot (e.g., a map) of the environment and replanning on a new snapshot, once changes in the environment are detected or (ii) explicitly modeling dynamic objects and taking these objects into account during initial planning. A combination of probabilistic sampling and deterministic planning is able to generate time-optimal trajectories in environments with known dynamic objects. Its application in environments with imperfect information or no information is however not efficient (Van Den Berg et al., 2006). Van Den Berg et al. (2006) propose an anytime path planning and replanning algorithm which is able to efficiently plan routes. Replanning is done using a static roadmap which is updated (online) with local information acquired from sensors. From a single robot point of view, such

---

1 Algorithms such as Dynamic Programming do not require complete replanning. However, such an algorithm is required to have complete knowledge of the network (Barto et al., 1995)
methods are promising. However, in multi-robot environments, redundant work is done. One can imagine that a roadmap for one robot may also be relevant to another robot operating in the same environment. Sharing maps implies communication and its implementation may not be that straightforward. Moreover, a single robot solution would require a higher robot complexity than a multi-robot solution. For example, a single robot would need to solve the problem completely on its own. In a multi-robot solution, each robot only has to solve part of the complete problem.

Therefore, recently researchers have been focusing on using teams of relatively cheap, simple robots to create paths. For this, the robots use their surroundings to store information. Howard et al. (2002) describe a method for localizing members of a robot team. The method uses the robots themselves as landmarks instead of each robot being able to recognize external landmarks. Each robot can determine the relative range, bearing and orientation of every other robot in the team without any external help. A line-of-sight has to be ensured between the robots for the method to work. Nouyan et al. (2008) show that robots are also able to play the role of direction markers. The authors present two swarm intelligence control mechanisms which can be used in distributed robot path formation. Method one, chains, builds linear robot structures between origin and goal while method two, vectorfield, spreads out the robots in space to form a tree-like structure between origin and goal. Robots not belonging to the chain or vectorfield can navigate over the resulting structures following the direction of the structures. Vaughan et al. (2002) also present a localization method with which a robot team can navigate through an unknown environment. The landmarks are not physical but are waypoint coordinates generated online (and communicated) by the robots. Waypoints contain references to features in the world and are common to all robots. Similarly, Payton et al. (2003) present a method in which a team of robots constructs a queue leading to the goal and communicating virtual pheromone values which decrease with distance. Using these markers, one is able to find a route to a certain goal. Another way of using waypoints, or signposts, is to embed a sensor array in the environment over which robots may navigate which would make the need for a map or localization unnecessary (Batalin et al., 2004; Johansson and Saffiotti, 2009). It is also possible to let the robots deposit the signposts (i.e., sensors) themselves based on certain local criteria and let them store directional information inside of these sensors (Batalin and Sukhatme, 2004, 2005; Zecca et al., 2008).

Although applying landmark principles for robot path planning and localization seems a straightforward application, the concept could also be applied in optimization algorithms. The most obvious problem would be shortest path problems, such as network routing. In (Gerla et al., 2000), group mobility of nodes is exploited. Here a landmark is a node which stays connected to the nodes in its group during movement. If a source node wants to send data to a distant node group, the packet is first sent to the landmark of the destination group. Then the packet is sent to its real destination. Such an algorithm increases adaptability and scalability and also decreases overhead.
5.3 The Stigmergic Landmark Foraging algorithm

In order to improve BS and to counter the possible problems of BSP in embodied systems, we extend BS by adding landmark navigation. In biology, such a navigation mechanism enhances basic PI behavior and increases robustness to PI errors. It enables the (physical) recognition of landmarks which segment the path between hive and goal. Each landmark is associated with one or more PI vectors. As such, landmarks provide additional (directional) information about a segment.

Recognizing and remembering physical properties of an object requires significantly increased cognitive capabilities of agents. Such a complexity contradicts the desired properties of a swarm, namely a large group of simple agents. Therefore, in SLF, instead of being able to recognize a landmark based on physical properties of an object, agents are able to create landmarks in the environment once they detect a proper location, see Figure 5.1. In this figure, black squares indicate positions that require obstacles for the landmark location to be detected. Gray squares indicate positions that may contain obstacles, but are not required for a landmark location to be detected. A landmark is essentially a location (e.g., grid block, node depending on the problem domain) in which it is profitable to have additional (heuristic) information in order to guide the search to a better solution. Their placement should be carried out in such a way that solving the sub-problem between two landmarks is trivial. A location is marked as a landmark by setting a Boolean-valued variable connected to that location. Moreover, each landmark can contain multiple route entries (thus multiple PI vectors). A route entry is identified by a ‘globally unique identifier’ (GUID) variable. The route entry represents a segment of the complete route. Such a segment is represented by two PI vectors which each consist of two Real-valued variables indicating distance and direction to another segment or location. First, a Landmark Homing Vector (LHV) indicates the distance and direction to take to get to the hive or possibly an intermediate landmark. Second, a Landmark Goal Vector (LGV) indicates the distance and direction to take to get to the goal or possibly an intermediate landmark. Associated with each landmark vector is a Dance Strength (DS) which indicates the profitability of a vector in analogy to the DS process that can be found in the hive. As such, a landmark essentially serves as a sub-hive. A sequence of landmark PI vectors together form a route between hive and goal.

Figure 5.1: Situations in which a key location (i.e., landmark) is detected. ‘L’ represents the landmark location. Black squares indicate required obstacles positions, gray squares indicate optional obstacles which are not required. Constellations may be rotated and/or mirrored. In none of the cases a hive or goal may be present.
### Abbreviation/Concept

<table>
<thead>
<tr>
<th>Abbreviation/Concept</th>
<th>Full Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>-</td>
<td>the location of a food source</td>
</tr>
<tr>
<td>GUID</td>
<td>Globally Unique Identifier</td>
<td>an ID identifying a route entry in a Landmark</td>
</tr>
<tr>
<td>GV</td>
<td>Goal Vector</td>
<td>a Goal Vector is a goal-directed PI vector in the memory of an agent</td>
</tr>
<tr>
<td>Hive</td>
<td>-</td>
<td>the start location of an agent</td>
</tr>
<tr>
<td>HV</td>
<td>Homing Vector</td>
<td>a Homing Vector is a hive-directed PI vector in the memory of an agent</td>
</tr>
<tr>
<td>L</td>
<td>Landmark</td>
<td>a landmark is a Boolean-valued variable associated with a location, indicating whether that location is a landmark or not. The landmark can contain an index of route entries</td>
</tr>
<tr>
<td>LGV</td>
<td>Landmark Goal Vector</td>
<td>a goal-directed PI vector stored at a landmark and linked to a route entry</td>
</tr>
<tr>
<td>LHV</td>
<td>Landmark Homing Vector</td>
<td>a hive-directed PI vector stored at a landmark and linked to a route entry</td>
</tr>
<tr>
<td>PI</td>
<td>Path Integration vector</td>
<td>a PI vector contains two Real-valued variables indicating the distance and direction (angle), denoted as ((D, A))</td>
</tr>
<tr>
<td>Route</td>
<td>-</td>
<td>a Route is a sequence of PI vectors, stored at subsequent landmarks, that share the same GUID</td>
</tr>
</tbody>
</table>

Table 5.1: This table lists the metaphorical concepts defined in context of the algorithm described in this chapter.

In a physical environment, creating landmarks can be done by placing an information carrying object (such as a RFID tag) inside of the environment (Batalin and Sukhatme, 2004, 2005). For ease of reading, we include a descriptive summary of the abbreviations/concepts used in this chapter in Table 5.1.

The remainder of this section consists of two parts. First, we explain the basic idea of SLF in a concise manner. Second, we extensively describe the full details of the method allowing for a re-production of the presented results.

#### Basic working

SLF’s basic functioning can be described as follows and is graphically depicted in Figure 5.2. During exploration (i.e., a Lévy flight), an ‘Explorer’ agent creates a hive-directed route consisting of a sequence of HVs. More precisely, such an agent keeps track of its own position with respect to the hive (its starting position) or previous landmark by employing PI. This knowledge is represented in the Temporary Vector (TV) which is a PI vector (i.e., two Real-valued variables) stored in the agent’s memory. The vector indicates the distance and direction to take in order to get back to the beginning of the agent’s current segment. During exploration, the agent checks each visited location on landmark conditions, see Figure 5.1 for the rules that determine a landmark. If it detects a possible landmark location, it marks the location by setting the landmark Boolean variable to true (i.e., \(L = true\)). Additionally, it creates a route entry. In this entry, it sets the route ID and copies its TV to the LHV. Moreover, it resets its TV before moving on and stores the route ID in its own memory. As of this moment, the agent is creating vectors for this route.

Once an exploring agent reaches the goal, it evaluates its quality, takes an item (i.e., subtract 1 from the item stack at the goal and becomes a ‘Carrier’ agent), and travels back to the hive by using the route it created during exploration. This behavior is depicted in Figure 5.3. Essentially, it is already exploiting its previous search experience. More precisely, when it leaves the
Figure 5.2: Creation of the HOMING landmark network by an exploring agent starting from the hive. Gray squares indicate landmarks, black squares indicate obstacles. The bee indicates the agent’s current location. A dashed arrow indicates the exploring agents path represented by the TV, a solid arrow indicates the LHV. ‘Landmark found’ essentially means that a potential landmark location is detected.

goal, it copies its current TV to its HV, resets the TV, and retraces it outward path with guidance of the HV. While traveling it keeps track of its own position with respect to the goal or previously visited landmark using the TV. If an agent comes across a landmark, it can perform one of two actions. (i) If the landmark already contains a route entry with the appropriate route ID (created on the way to the goal), it completes the route entry (in case it is visited for the first time for this ID) or reinforces the route entry (in case it has been visited before for this ID) by copying its TV to the entry’s LGV and setting or reinforcing its associated DS. Additionally, it copies the LHV to its HV for further guidance to the next landmark or possibly the hive. (ii) If the landmark does not contain a route entry with the appropriate route ID, which is possible in case the agent ends up in an unexpected landmark due to possible environmental changes, it creates such an entry and copies its TV to the entry’s LGV. In this case, its HV will still be
Figure 5.3: Creation of the GOAL landmark network by a carrier agent starting from the goal. Gray squares indicate landmarks, black squares indicate obstacles. The bee indicates the agent’s current location. A dotted arrow indicates the carrier agent’s path. A dashed arrow indicates the LGV. ‘Landmark found’ essentially means that a potential landmark location is detected.

able to guide it to a known landmark. In any case, it resets its TV and uses its HV to get to the next landmark or possibly the hive.

Once a ‘Carrier’ agent reaches the hive, it is able to recruit other members for its route. For this, a virtual dance is performed which communicates previous search experience and essentially puts the agent’s TV in an array accessible to each agent inside of the hive. Each vector in this array is associated with a Real-valued DS variable indicating how worthwhile the vector is. Other agents can choose to copy such a TV (and the associated route ID) to their own memory (i.e., the GV) and by doing so are recruited for that specific route.

Recruits use the GV to get to the next landmark or possibly the goal. During its travel, the agent keeps track of its own position using the TV. When a recruit arrives at a landmark, it can perform one of two actions. (i) If the landmark already contains a route entry with the appropriate route ID, it reinforces the LHV: the agent copies its TV to the LHV and reinforces the DS of the LHV. Moreover, it copies the LGV to its GV. (ii) If the landmark does not contain
a route entry with the appropriate route ID, it creates such an entry and copies its TV to the entry’s LHV. In this case, its GV will still be able to guide it to a known landmark. In any case, the agent resets its TV and uses its GV to get to the next landmark or possibly the goal.

During the process described above, agents essentially create a network of bidirectional routes between the hive and the goal(s).

**Detailed description**

In order to realize the SLF algorithm as depicted in the previous subsection we extend the BS. In SLF, each agent’s memory is slightly increased with three fields.

First, the Temporary Vector (TV) represents the vector from the last known landmark (or hive/goal) to the current landmark (or hive/goal). On arrival at a landmark (or hive/goal), the TV is copied to the appropriate landmark vector. This depends on the internal state of the agent. For example, an ‘Explorer’ agent copies the TV to the LHV. Afterwards the TV is reset.

Second, an agent is able to remember which route it is actually following. For this, it stores a GUID-value variable indicating the ID of the route. Each landmark might represent multiple routes towards a goal (and/or back to the hive). A landmark is thus able to store multiple route entries, identified by an ID and linked to each entry are two vectors, (i) the LHV and (ii) the LGV. Each of these vectors additionally has a DS indicating how worthwhile the vector is.

Third, an agent is able to remember the quality of a goal it has found. Goal quality is represented by a Real-valued variable and may be set by the user, for example to indicate priority. Obviously, this is less complex than adding physical landmark recognition and landmark memories by appearance which may include extensive cognitive capabilities.

The general algorithm structure is based on BS and thus consists of three functions, i.e., (i) ManageBeeBehavior(), (ii) CalculateVectors(), and (iii) DeamonActions(). With respect to these functions the first and last one have a different implementation than that found in BS. More precisely, ManageBeeBehavior() has to implement agent behavior regarding landmarks which means that agent behavior is different. Moreover, dancing behavior and the choice to explore or observe are more formally described. Previously unused DeamonActions() is now used to handle the decay of landmarks. Each landmark’s DS decays with a factor $\rho$ at each time step, see Equation 5.2b.

The behavior of agents in ManageBeeBehavior() is determined by the six possible internal states and is inspired by agent behavior as described in (Passino and Seeley, 2006) which gives a theoretic background of the biological behavior of each internal state. In Figure 5.4, we show the behavior transitions between each of the internal states.

1) **Explorer**: indicates the agent is exploring the environment for goals and performing Lévy flights to optimally cover the search space (Viswanathan et al., 2000). During exploration, the agent continuously updates its TV. While covering the search space, the agent is able to detect key locations (i.e., landmarks) in the environment which are then used to store the agents’ TV, see Figure 5.2. The landmark detection conditions can be found in Figures 5.1(a), 5.1(b), 5.1(c) and 5.1(d). Essentially, a landmark is found on a corner of an obstacle, an open corner, a corridor in a T-shape, and in a corner of a corridor. These conditions are sufficient for constructing an efficient solution to the foraging problem in our grid-world setting. Connections between such landmarks ensure an efficient path around obstacles. Note however that landmark
Figure 5.4: Behavior transitions between each of the internal states. The numbers which can be found at various edges represent the equations numbered equally in the paper.

<table>
<thead>
<tr>
<th>EID</th>
<th>AR</th>
<th>Explorer Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>Create new landmark with a new route ID</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>Create new landmark with agent route ID</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Not possible</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>IF LHV does not exist: continue exploration without creating new landmarks, ELSE IF LGV exists: probability to become recruit.</td>
</tr>
</tbody>
</table>

Table 5.2: Explorer Landmark creation conditions. These conditions apply when a explorer detects a possible landmark location. EID: there is an existing landmark route ID equal to the agent’s route ID?, AR: Agent Route ID is set?

detection conditions may differ according to the problem domain and environment set-up. For example, in network routing a landmark may be detected if a node has a certain queue, bandwidth, and/or congestion.

Whenever an explorer detects a potentially new landmark (i.e., $L = \text{false}$), the agent creates a new route entry, generates and sets the GUID-valued route ID, stores it in its own memory, and additionally stores its TV in the LHV (see Figure 5.2(c) & 5.2(d)), indicating the next location to move to in order to arrive at the hive. Each explorer is thus able to create a distinct route towards its home. If an agent arrives in a already existing landmark (i.e., $L = \text{true}$), it checks whether its route (if any) is already known inside the landmark. If so, it reinforces the corresponding LHV by copying the TV to the LHV and increasing its related DS. Otherwise, it creates a new route entry. If an agent’s TV is obstructed within the agents

---

2 Explorers are not allowed to extend existing routes. Extensions by explorers may result in route looping due to landmark decay.
5.3 | The Stigmergic Landmark Foraging algorithm

<table>
<thead>
<tr>
<th>EID</th>
<th>AR</th>
<th>Carrier Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>Create new landmark with new agent landmark route</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>Create new landmark with known agent landmark route</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Not possible</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>IF LGV does not exist: set LGV with known agent route, ELSE IF Agent Goal Quality &gt; Landmark Goal Quality then overwrite LGV</td>
</tr>
</tbody>
</table>

Table 5.3: Carrier Landmark creation conditions. These conditions apply when a explorer detects a possible landmark location. EID: there is an existing landmark route ID equal to the agent’s route ID?, AR: agent route ID is set?

view range, it is not allowed to create any more landmarks during its exploration phase since it would create a route which is obviously blocked. Explorer Landmark creation conditions can be found in Table 5.2. The agent’s behavior is depicted in pseudo-code in Appendix A, Section A.1. If the explorer arrives at the goal, its internal state changes to Carrier.

2) Carrier: indicates the agent found a goal and is returning an item from the goal to the hive. If the carrier is located at a goal (i.e., the carrier’s starting location) it determines the quality of the site according to Equation 5.1. Let the quality assessment Q of agent i for goal j be

\[ Q_{ij} = G_j + \epsilon_{ij} \]  

(5.1)

where \( G_j \in [0, 1] \) is the quality of goal j (set by the user) and \( \epsilon_{ij} \in [0, 1] \) the normal-distributed assessment error of the goal quality by agent i due to noise. Goal quality may represent for example goal priority. The actions an agent takes during the return are guided by the LHV of the landmarks the agent visits while following a particular route back home, see Figure 5.3(b). At each visited landmark the particular route segment is completed by creating/reinforcing the LGV, see Figure 5.3(c) & 5.3(d). More precisely, on arrival at a landmark, the agent stores its TV in the appropriate route entry’s LGV and then resets the TV. If a carrier is not able to follow its route back (i.e., the LHV is blocked or the indicated follow-up landmark is non-existent) it starts exploring for landmarks by using a Levy Flight. If it finds one it tries to adopt another landmark-route home (if available). The carrier is not allowed to create any LGVs if it altered its route since this may result in route loops. Carrier landmark creation conditions can be found in Table 5.3.

To prevent excessive carrier exploration efforts, a carrier agent has a certain lifespan (in time steps). In this research, carriers are sure to stay alive for the time it takes to, at least, travel back and forth along the longest route \(((h \times w)^2\) time steps, where \( h \) is the number of grid-rows of the environment and \( w \) is the number of grid-columns of the environment). Once this period is over, the agent has an increasing probability of death in which case it loses its resource. For each agent that ‘dies’, a new agent is ‘born’ inside of the hive. The probability of death increases linearly with the number of visited cells after its minimal lifespan. Such ‘behavior’ is also found in nature where, for instance, bees have a certain lifespan which decreases with increasing amount of time spent foraging (Visscher and Dukas, 1997). Carrier behavior is depicted in pseudo-code in Appendix A, Section A.2.

3) Dancer: indicates that an agent arrived at the hive with directional knowledge of a landmark or possibly the goal directly. This knowledge is represented in the TV and the route the agent followed. In order to recruit other colony members to its goal, the agent starts a virtual
dance which other colony members can observe. Each dance communicates the directional information and route directly to the observers. High-quality goals are preferred over low-quality goals. Moreover, short-distanced goals are preferred over long-distanced goals. Dance strength is a heuristic. It enables the agents to quickly identify the best available goal. To calculate the strength of a dance, as shown in 5.2, we initially normalize the found goal quality (see Equation 5.2a). After initialization, the dance decays every time step (see Equation 5.2b).

\[
DS^i_0 = \begin{cases} 
\gamma \left( \frac{Q_{ij}}{\sum_{i=1}^{B_d} Q_{ij}} \right) : & Q_{ij} > \epsilon_t \\
0 : & Q_{ij} \leq \epsilon_t 
\end{cases} \quad (5.2a)
\]

\[
DS^i_t = \max\{DS^i_{t-1} - \epsilon_d, 0\} \quad (5.2b)
\]

In Equation 5.2 \(DS^i_t\) indicates the dance strength for dancer \(i\) at time step \(t\) of the dance, \(Q_{ij}\) is the quality of goal \(j\) as observed by agent \(i\). \(B_d\) represents the number of dancers in the hive and \(\sum_{i=1}^{B_d} Q_{ij}\) only counts distinct goals for normalization purposes. We do not sum each dancer’s goal quality but only sum the qualities of distinct goals in the hive. More precisely, if two dancers dance for the same goal, and as such promote the same quality, it is only counted once. Not doing this would result in an incorrect normalized dance strength value. \(\gamma > 0\) is a parameter which represents the proportionality between the goal quality and the initial dance strength. It has an impact on the exponential function used for deciding whether to explore or exploit, see Equation 5.3. For example, setting \(\gamma = 300\) will result in a dance strength of 300 for a goal with quality 1. \(\rho\) sets a value for the dance strength decay. Extending the previous example, setting \(\rho = 15\) would result in dancing for the specific goal for 300/15 = 20 time steps. \(\epsilon_t\) sets a threshold on the goal quality before there will be any dancing for the goal and can be set by the user. At the end of the virtual dance, a dancer has to decide whether it rests or recruits itself to its own goal. The self-recruiting probability (i.e., \(P_{sr}\)) can be set by the user.

4) Rester: indicates the agent remains in the hive and is resting. It either did not find a goal by exploration or it did find a goal and recently stopped dancing for it. A resting agent either changes its internal state to observer or explorer. The probability to become an explorer, \(P_e\), is given by Equation 5.3. The equation captures the distributed decision in the search-agreement process in nature (Passino and Seeley, 2006) and can also be applied in computer science. At the start of a search, there are only few dances and consequently many agents explore in order to find high-quality goals. As goals are found, more agents start dancing and by dancing recruit other agents so that fewer agents explore. In the field of reinforcement learning it is suggested that to find optimal solutions, it is best to balance exploration and exploitation by following an exponential function which decays over time (Barto and Sutton, 1998). Although a linear function would also ensure exploration when little dances are available and stimulate exploitation when many dances are available, foraging performance will suffer from premature exploitation since high quality goals may not have been found yet. To ensure prolonged exploration and a higher probability for finding high quality goals, we use an exponential function which is defined as follows:

\[
P_e = e^{\left( -\frac{1}{2} \frac{(s_{total})^2}{\sigma^2} \right)}, \quad (5.3)
\]
where $DS_{\text{total}}$ indicates the total dance strength of all dancing agents, and $\sigma$ essentially determines the tendency of an agent to explore, see Figure 5.5. Notice that if $DS_{\text{total}} = 0$, there is no dancing so that $P_e = 1$ and all the observer bees will explore (e.g., initially $DS_{\text{total}} = 0$ so all observer bees will choose to explore). If $DS_{\text{total}}$ is low, the observer bees are less likely to find a dancer and thus will not likely get recruited to a goal. They will, in a sense, be ‘recruited to explore’ by the lack of the presence of any dance. As $DS_{\text{total}}$ increases, the agents become less likely to explore and, as discussed below, will be more likely to find a dancer and get recruited to a goal.

5) **Observer:** indicates an agent is looking for dancers to observe. The agent looks for a virtual dance (i.e., agents with a positive DS) and determines whether the danced ‘advertisement’ is of sufficient quality to adopt. This behavior is captured in the following equation,

$$P_r = \frac{DS^i}{\sum_{i=1}^{B_d} DS^i}, \quad (5.4)$$

where $P_r$ indicates the probability of an observer to adopt the vector of agent $i$ with dance strength $DS^i$. $B_d$ indicates the total number of dancing agents. This way, agents that dance ‘more strongly’ recruit more agents for their goal. Thus, high quality goals are going to be preferred. Of course, short distanced goals will also have more chance of being exploited since there will be more frequent dancing for them. Once the agent finds a dancer, the agent copies the corresponding TV values (indicating the next landmark of the goal) from the dancing agent to its GV and changes its agent state to **Recruit**.

6) **Recruit:** indicates an agent adopted the directional information from a dancer and is traveling to the goal using landmarks’ LGVs on its route. The recruit essentially validates each LHV and LGV. More precisely, if a recruit is able to arrive at an indicated landmark then obviously the LGV is still usable. Likewise, the agent can compare its TV with the LHV. If the vectors are equal then obviously, the route back home is also still valid. On validation, the recruit reinforces the segments of the routes. Eventually, strong routes will emerge and these routes will prevail over the weak routes. Appendix A, Section A.3 shows the pseudo-code for a recruit.
5.4 Experimental set-up

In this section, we will discuss our experimental set-up. In §5.4.1 we will describe the type of measurements. §5.4.2 presents the environments to experiment on and their characteristics. Finally, §5.4.3 provides the settings of the simulator.

5.4.1 Type of measurements

In this chapter we would like to discover how SLF performs with respect to (a) efficiency, (b) scalability, and (c) adaptability. These three characteristics are most important in MASs. A MAS typically has to deal with a large number of agents and it is used to solve (increasingly) large and complex problems which may be dynamic in nature (Dorigo and Stützle, 2004).

Efficiency relates to the learning performance (i.e., knowledge acquisition and knowledge usage). In this research, we measure efficiency by the average number of time steps the algorithm uses to complete the task. Moreover, by measuring the average computation time per time step, we also monitor time-related resources. Scalability is the ability to cope with an increasing problem size and/or decreasing resource availability. Scalability of a MAS is thus partly related to its efficiency. Obviously, an efficient MAS is more scalable than an inefficient MAS. To measure scalability in this research, we perform the experiments with different (increasing) colony sizes. Moreover, we experiment on two different-sized environments. In these experiments, a MAS that shows a small decrease in efficiency in an increasingly complex problem is more scalable than a MAS that shows a large decrease in efficiency.

Adaptability is the ability to adapt to changing circumstances. An adaptive MAS is able to learn alternative behaviors when the initial learned behavior is not profitable anymore. Learned behavior may become unprofitable due to environmental changes and/or quality-assessment errors by agents. Thus, adaptability relates to robustness. Furthermore, relearned behavior should also be efficient. If a MAS in a dynamic environment is inefficient it apparently learned mediocre or bad performing behavior and as a consequence is not adaptive. In this research, adaptability is measured according to,

\[
A = \frac{\sum_{m=1}^{M} \left(1 - \frac{t_{m, \text{newroute}} - t_{m, \text{change}}}{t_{t}}\right)}{M},
\]

where \(A \in [0, 1]\) is a measure for expected adaptability. A high value for \(A\) indicates high average adaptability, a low value indicates low average adaptability. \(M\) is the total number of environmental changes which is determined by the user. An environmental change is able to block a certain route which may be accomplished by blocking one or more grid cells. We assume that the strongest route is blocked at an environmental change. \(t_{m, \text{change}}\) indicates the time step of the environmental change \(m \in M\). \(t_{t}\) is the total number of time steps of that particular run. \(t_{m, \text{newroute}}\) indicates the time step in which a new strongest route surpasses the previous strongest route before the environmental change \(m \in M\). Route strength is determined according to,

\[
S_{j} = B^j_r + B^j_c,
\]

(5.6)
where $S_j$ indicates the route strength of route $j$, $B^j_r$ represents the number of recruits for route $j$, and $B^j_c$ represents the number of carriers for route $j$. The highest $S_j$ represents the strongest route at the moment of an environmental change.

Summarizing, we keep a count on the number of recruits and carriers for each route. The moment an environmental change happens, the route with the highest count is the strongest route since it is exploited the most. Whenever an environmental change $m$ is performed, we store its corresponding time step (i.e., $t^m_{\text{change}}$) and the current strongest route $S_j$. After some time, a route other than $j$ will surpass $S_j$ and we store the time step, $t^m_{\text{newroute}}$, when this happens. To test adaptability, we experiment with dynamic environments in which obstacles block learned foraging routes towards a goal. Moreover, agents may make quality assessment errors. Consequently, a MAS capable of dealing with such errors is clearly adaptive.

5.4.2 Experiment environments and characteristics

In order to perform the measurements, we apply SLF in a number of problem environments. Each environment consists of a grid of square cells. Each cell has maximally four neighbors and may contain a hive, goal, or obstacle. In the latter case, agents are not able to traverse these cells. Each environment contains one hive (i.e., start location) and one goal (i.e., a collection of items). Each agent’s assignment is to create a route from the hive to the goal, collect one goal-item and return it to the hive. We construct three sets of experiments, (i) BSP comparison set (Figure 5.6), (ii) Heuristic comparison set (Figure 5.7), and (iii) Robustness comparison set (Figure 5.8). We deliberately use relatively simple (yet increasingly complex) environments which may benefit the understanding of the approach and the results.

**EXPERIMENT SET 1: BSP comparison.** In order to get an indication of the performance of
SLF, we compare its performance to the performance of BSP (Lemmens et al., 2007c) when applied to the same set of experiments. The set consists of two experiments, see Figure 5.6. Experiments 1 and 2 are basic experiments without any obstacles between the hive and the goal. ‘h’ and ‘w’ are chosen in such a way that when the simulator creates a grid over the environment, they consist of 400 and 900 cells respectively to test for scalability with respect to an increasing problem size. Both experiments contain a hive in the lower-left corner and a goal in the upper-right corner. Obviously, these experiments can be done in all alternative rotations and with different ‘constellations’ of hive and goal. To test for agent scalability, we perform these experiments with an increasing colony size (i.e., 25 and 50 agents). Each goal contains 300 items.

Figure 5.7: Set 2: heuristic comparison set. The number of flowers indicates the quality of the food source. One flower indicates low quality, three flowers indicates high quality. In order to differentiate between the equal quality food sources in Experiment 3, we indicate them with a ‘I’ and a ‘II’.

**EXPERIMENT SET 2: Heuristic comparison.** In order to test the recruitment heuristic, Set
2 consists of 3 experiments in which each environment has two paths towards two goals, see Figure 5.7. ‘h’ and ‘w’ are chosen in such a way that when the simulator creates a grid over the environment, it consists of 400 cells. We test two recruitment heuristics. First, a basic heuristic based on non-normalized goal quality (Passino and Seeley, 2006). In other words, in Equation 5.2 \[ \sum_{i=1}^{B_d} Q_{ij} = 1 \]. Goal quality does not depend on the number of items present in the goal. Instead, all items have the same quality. In previous chapters, recruitment effort depended on distance assessment. Short-distance goals were preferred over long-distance goals. Indirectly such recruitment effort can also be found in the current heuristic, i.e., a short-distance goal will receive more virtual dancing than a long-distance goal due to the fact that agents visiting the short-distance goal will need less time to return and therefore there will be dances for this goal more frequently. Thus the recruitment probability will be higher for short-distance goals. However, such a heuristic may not be a guarantee that the most profitable route is used. A high quality, long-distance goal may still be more profitable than a low-quality, short-distance goal. Put in another way, distance may offset the priority. Therefore, secondly, to ensure priority paths, we include a normalized heuristic. Recruitment will now be guided by a normalized goal quality. E.g., recruitment for a low-quality goal among recruitment for high-quality goals will have a relatively lower recruitment probability than with the non-normalized heuristic.

Eventually, the most profitable path will be preferred. To show effectiveness of each heuristic, we vary the quality of goals and the distance of each goal to the hive. In Experiment 3, the quality of the short-distance goal and the long-distance goal are equal, see Figure 5.7(a). In Experiment 4, there is a high-quality, short-distance goal and a low-quality, long-distance goal. In Figure 5.7(b), a high-quality goal is indicated by 3 flowers, a low-quality goal is indicated by 1 flower. Experiment 5 switches these goals’ location. Both low-quality and high-quality goals consist of 600 items to clearly show convergence to one of the two paths. For clarity, goal quality can be viewed as a type of priority. A low-quality goal therefore contains 600 low-priority items which are not as rewarding as high-priority items.

**EXPERIMENT SET 3: Adaptability comparison.** As mentioned earlier, adaptability is related to robustness. SLF should be robust (and thus adaptive) to environmental changes and goal-quality-assessment errors. Experiment 6 consists of three cases which together create a dynamic environment, see Figure 5.8. Deneubourg et al. (1990) used a similar experiment to indicate the adaptability of foraging Argentine ants. Case 1 (i.e., upper left in Figure 5.8) shows an environment in which there is a short and a long path to the goal. Moreover, it shows an obstacle construction which the agents have to learn to travel around. Case 2 (i.e., upper right in Figure 5.8) blocks one path (i.e., the short path). Case 3 (i.e., lower right in Figure 5.8) opens up one path (i.e., the short path) and blocks the other (i.e., the long path). Thus, agents first need to learn a route from hive to goal during case 1. At any moment in time, their environment changes (into Case 2) and they have to relearn how to cope with this environment, i.e., forget the short route and relearn/reinforce the long route. Finally, case 3 represent again an environment with the unblocked short route which has to be relearned. This experiment shows the ability of the algorithm to reroute (i.e., ‘relearn’) a route. We perform Experiment 6 twice. First, in Experiment 6a, agents have a ‘no-error’ goal-quality assessment. Second, in Experiment 6b, agents can make goal-quality-assessment errors. Despite of this handicap, the
agents should still be able to learn the shortest route towards the goal and gather the goal-items efficiently. A goal contains 300 items.

5.4.3 Simulator settings

We use different settings for the three experiment sets.

a) In Set 1, due to BSP being simulated in a different simulator which does not support all extensive measurements, we use the following settings. (i) every agent that starts dancing for a goal stays committed to its goal and recruits itself (i.e., $P_{sr} = 1$), (ii) the experiments are run with a colony size of 25 and 50 agents, (iii) no goal quality assessment errors are allowed (i.e., $\epsilon^i_n = 0$).

b) In Set 2, we only experiment with SLF and set the probability to let a dancer recruit oneself to 0.50 (i.e., $P_{sr} = 0.50$). In experiments with the non-normalized heuristic, we set $\sum_{i=1}^{B_d} Q_{ij} = 1$.

c) In Set 3, we test for adaptability and set the dancer’s self-recruitment probability to 0.50 (i.e., $P_{sr} = 0.50$). Note that no adaptability means there is no dancer self-recruitment (i.e., $P_{sr} = 0$) and the initial dance strength is zero (i.e., $\gamma = 0$) and consequently adaptability will be zero (i.e., $A = 0$). In Experiment 6a we set $\epsilon^i_n = 0$ to indicate that no goal-quality assessment errors are made. In Experiment 6b, the goal-quality assessment error of each agent is drawn from a normal distribution over the interval $[0, 1]$.

Furthermore, in each of the experiments we use the following settings. (i) There is no quality threshold (i.e., $\epsilon_t = 0$), every goal will be considered interesting. (ii) Dance strength decay is set to 15 (i.e., $\rho = 15$), (iii) $\gamma = 300$ so that the maximum duration of the initial dance
5.5 Results

To have an indication of performance, we compare SLF’s performance with BSP’s performance when applied to the same experiment set, i.e., Set 1. We compare the algorithms’ performance with respect to their efficiency and scalability. Efficiency can be measured by determining the average number of time steps, $\mu_t$, the algorithms require to gather all the items present in the goal. Scalability relates to the efficiency when the complexity of the problem is increased. In Set 1, we can increase complexity by increasing the environment size or the colony size. As can be observed in the top section of Table 5.4, SLF outperforms BSP with respect to efficiency. A graphical representation of this is given in Figure 5.9. On average, SLF is 2 times faster in solving the problem set. Moreover, we can observe that, if we increase the colony size while keeping environment size equal, in either of the two experiments, both algorithms significantly use less time steps to solve the problem set. Therefore, we can indicate that both algorithms are quite scalable with respect to colony size. With respect to scalability when we increase environment size while keeping the colony size equal, we see a significant increase in the number of time steps. Both algorithms seem to be less scalable with respect to environment size.

In order to get an indication of the rate with which the efficiency decreases or increases, we calculate the corresponding ratios. Figure 5.10(a) shows the colony ratios (i.e., $\frac{\mu_{50}}{\mu_{25}}$) of both al-

<table>
<thead>
<tr>
<th>Algorithm (# agents)</th>
<th>Experiment 1 $\mu_t$</th>
<th>$\sigma$</th>
<th>Experiment 2 $\mu_t$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP (25)</td>
<td>2173.8</td>
<td>14.5</td>
<td>5625.6</td>
<td>52.9</td>
</tr>
<tr>
<td>SLF (25)</td>
<td>1228.6</td>
<td>66.9</td>
<td>2028.2</td>
<td>152.2</td>
</tr>
<tr>
<td>BSP (50)</td>
<td>1160.6</td>
<td>4.8</td>
<td>2600.6</td>
<td>15.4</td>
</tr>
<tr>
<td>SLF (50)</td>
<td>725.9</td>
<td>36.2</td>
<td>1227.6</td>
<td>65.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm (# agents)</th>
<th>Experiment 1 $\mu_c$</th>
<th>$\sigma$</th>
<th>Experiment 2 $\mu_c$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP (25)</td>
<td>4.76</td>
<td>0.0</td>
<td>4.49</td>
<td>0.0</td>
</tr>
<tr>
<td>SLF (25)</td>
<td>18.3</td>
<td>12.2</td>
<td>27.0</td>
<td>17.0</td>
</tr>
<tr>
<td>BSP (50)</td>
<td>9.15</td>
<td>0.0</td>
<td>9.05</td>
<td>0.0</td>
</tr>
<tr>
<td>SLF (50)</td>
<td>44.1</td>
<td>20.5</td>
<td>64.3</td>
<td>32.6</td>
</tr>
</tbody>
</table>

Table 5.4: Results of experiments in Set 1. Top: the efficiency in average needed time steps, $\mu_t$. Bottom: the computation time/time step $\mu_c$. Each experiment contains a goal consisting of 300 items.
algorithms. Figure 5.10(b) shows the environment ratios (i.e., \( \mu_{E2} / \mu_{E1} \)) of both algorithms. Observing the colony ratios, we have to note that the two algorithms don’t differ all that much. The time-step ratios with increasing colony size approximately stay equal. For example, the time-step ratio for SLF with increasing colony size and equal environment size are 725.9/1228.6 = 0.6 and 1227.6/2028.2 = 0.6 respectively. Likewise, the ratios for BSP are 1160.6/2173.8 = 0.5 and 2600.6/5625.6 = 0.5 respectively. The difference between BSP and SLF is more distinct when we increase the environment size while keeping the colony size equal. The time-step ratios with increasing environment size and equal colony size are 2028.2/1228.6 = 1.7 and 1227.6/725.9 = 1.7 for SLF on the one hand and on the other hand 5625.6/2173.8 = 2.6 and 2600.6/1160.6 = 2.2 for BSP. Thus, considering the efficiency, both algorithms are equally scalable with increasing colony size. With respect to increasing environment size, we can observe that the performance of BSP is influenced more negatively by increasing environment size than SLF is.

Considering computation time, SLF needs considerably more computation time per time step as can be observed in the bottom section of Table 5.4 and Figure 5.11. This considerable difference can be explained by looking at the nature of the algorithms. First, BSP uses a binary inhibition pheromone. More precisely, either the pheromone is there or it is not. No ‘administration’ efforts have to be taken. In contrast, SLF needs to perform some ‘administration’ efforts with respect to landmark strength. The larger the world, the more possible landmarks,
the more ‘administration’ needs to be done by the simulator at every time step. Second, both BSP as SLF need the ‘Explorer’ agents to check each cell of the environment for either inhibition locations or landmark locations respectively. Location checking takes considerable effort for both algorithms. However, due to inhibition pheromone, BSP creates a path in a field of pheromone over time. Since the environment is static, eventually no exploration needs to be done by BSP. All agents exploit the resulting path. In contrast, SLF keeps on exploring during the complete experiment and as such uses significantly more computation time/time step. Therefore, on average, SLF uses 6.9 times more computation time/time step, see Figure 5.11. In the end, this results in longer run times for SLF. Since computation time is not independent of exploration effort, we merely indicate the results but do not draw conclusions from it.

Set 2 presents experiments with multiple, different-quality goals. The set is used to explore whether a normalized heuristic improves performance of SLF. The results, which can be
Figure 5.11: BSP and SLF computation time results in Experiment 1 and 2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Experiment 3</th>
<th>Experiment 4</th>
<th>Experiment 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>µ&lt;sub&gt;t&lt;/sub&gt;</td>
<td>σ&lt;sub&gt;t&lt;/sub&gt;</td>
<td>µ&lt;sub&gt;t&lt;/sub&gt;</td>
</tr>
<tr>
<td>Time steps SLF</td>
<td>2993.3</td>
<td>311.0</td>
<td>4246.9</td>
</tr>
<tr>
<td>Time steps SLF (N)</td>
<td>2956.1</td>
<td>442.8</td>
<td>3196.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>µ&lt;sub&gt;c&lt;/sub&gt;</th>
<th>σ&lt;sub&gt;c&lt;/sub&gt;</th>
<th>µ&lt;sub&gt;c&lt;/sub&gt;</th>
<th>σ&lt;sub&gt;c&lt;/sub&gt;</th>
<th>µ&lt;sub&gt;c&lt;/sub&gt;</th>
<th>σ&lt;sub&gt;c&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation SLF</td>
<td>42.8</td>
<td>20.3</td>
<td>76.5</td>
<td>37.8</td>
<td>64.6</td>
<td>29.3</td>
</tr>
<tr>
<td>Computation SLF (N)</td>
<td>54.6</td>
<td>25.0</td>
<td>59.4</td>
<td>24.6</td>
<td>58.7</td>
<td>23.2</td>
</tr>
</tbody>
</table>

Table 5.5: The efficiency (top), computation time (top), and heuristic efficiency (bottom) of the two algorithms under study in Set 2. Efficiency is measured in time steps. Computation time is measured in milliseconds. F1 and F2 represent the first and second depleted goals respectively. ‘I’ and ‘II’ indicate the food sources as shown in Figure 5.7(a). ‘L’ and ‘H’ indicate the low quality and high quality food sources as shown in Figures 5.7(b) and 5.7(c). µ<sub>t</sub> indicates the average time step at which a given goal is depleted. Each goal consists of 600 items.

observed in Table 5.5, indicate that both heuristics perform equally well in an experiment with equal quality goals and there is no significant difference in performance (i.e., double-sided t-test with 99% confidence; t = 0.3766 for efficiency, t = −2.0069 for scalability). With respect to finding the best goal first, we cannot observe any difference either (see bottom section of Table 5.5. However, SLF with normalized shows a higher overall efficiency in experiments with different quality goals. Thus, the normalization heuristic shows an advantage over the
Set 3 (Efficiency & computation time)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Experiment 6a</th>
<th></th>
<th>Experiment 6b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>µ</td>
<td>σ</td>
<td></td>
<td>µ</td>
</tr>
<tr>
<td>Efficiency</td>
<td>4221.1</td>
<td>1101.3</td>
<td>4703.9</td>
<td>763.4</td>
</tr>
<tr>
<td>Computation</td>
<td>59.5</td>
<td>30.5</td>
<td>63.0</td>
<td>31.9</td>
</tr>
</tbody>
</table>

Table 5.6: The efficiency (top), computation time (top), and adaptability (bottom) of SLF in Set 3. Efficiency is measured in time steps. Scalability is measured in computation time per time step.

standard heuristic. By normalization a goal quality is scaled up when it’s the only goal being ‘advertised’ in the hive. Such a scaled goal quality ensures more dancing for the goal and as a consequence more and prolonging recruitment which in the end results in faster item return.

Table 5.6 shows the results for the dynamic experiments, i.e., Experiment 6a and 6b. The former allows no agent quality assessment errors whereas the latter does allow such errors. The results indicate how robust and adaptive SLF is with respect to dynamics and errors. From the results we can observe that SLF is robust to observation errors. Efficiency performance is only slightly lower. A double-sided T-test with 99% confidence interval reveals the difference is not significant ($t = -1.9735$). Moreover, the adaptability measure, as seen in the bottom section, indicates that adaptability is robust to observation errors since the adaptability percentage only decreases with 1%. Adaptability shows the ability of the algorithm to adapt learned behavior to improve performance.

5.6 Chapter conclusion

In this chapter, we presented a novel distributed landmark navigation algorithm for a MAS named Stigmergic Landmark Foraging. More precisely, SLF constructs stigmergic landmark networks over which agents can travel from and to a goal. In contrast to existing algorithms, we present an online learning approach in which found landmarks can immediately be used. Each landmark represents a segment of the total route, ensuring robustness and accuracy by decreasing inherent accumulating PI errors.

Observing our results, we may conclude that SLF is significantly more efficient than BSP. However, its computation time/time step is approximately 7 times larger which in the end results in longer running times for the experiments. Although SLF performs worse with respect to computation time, it is worth noting that this is due to its exploration behavior. More precisely, both BSP as SLF need the ‘Explorer’ agents to check each cell of the environment for either inhibition locations or landmark locations respectively. Location checking takes considerable effort for both algorithms. However, due to inhibition pheromone, BSP creates a path in a field of pheromone over time. Since the environment is static, eventually no exploration needs to be done by BSP. All agents exploit the resulting path. In contrast, SLF keeps on exploring dur-
ing the complete experiment and as such uses significantly more computation time/time step. In a physical environment, computation time may not be a significant problem. For example, if SLF would be applied in mobile routing, each mobile router itself may use a multiple of SLF’s highest computation time to move from one location to another. Moreover, in a physical swarming environment, the task is typically physically distributed over multiple independent processing components as opposed to one central processing simulator. For instance, each landmark could be represented by a RFID tag and this tag could individually manage landmark strength decay. Alternatively, agents could represent landmarks themselves by dancing for a goal at a landmark location. If each agent would have its own processing component, it could individually manage its dance strength. This in contrast to the current simulation in which all tasks were done by only a single processing component.

With respect to scalability we may conclude that BSP is slightly more scalable with respect to colony size. However, SLF shows to be more scalable with environment size. This can be observed from the ratios. Furthermore, we may conclude that extending the SLF’s heuristic is advantageous. By doing so, the efficiency of the algorithm increases. Finally, we note that SLF is robust to quality assessment errors. If the agents’ quality assessment has errors, the performance of the algorithm does not suffer from this.

Taking into account the results in this chapter, we wonder whether SLF could be an interesting approach in other problem domains than the foraging domain. For example, considering an application in Swarm Robotics, SLF may pose an interesting method for navigation. Obviously, in this chapter, we make a number of assumptions such as errorless Path Integration. In a real-world multi-robot application, these assumptions may not hold due to the noisy environment. However, we would like to point out to the interested reader that these assumptions can be relaxed. For instance, Lambrinos et al. (2000); Möller (2000) show PI methods which are able to deal with real-world noisy environments. With respect to creating landmarks, Batalin and Sukhatme (2004, 2005) show that such a mechanism is also possible in real life. Taking into account the possible solutions for our current assumptions, we believe SLF is an interesting method for navigation and could be used in certain disaster scenarios such as a tunnel disaster scenario where the environment does not allow humans to operate due to, for example, bad vision or toxic gases.

As a more direct application, we would like to apply SLF as described in this chapter to the problem domain of network routing. Foraging and routing have a number of properties in common, such as, (i) finding optimal routes, (ii) constructing robust routes, and (iii) load balancing. It might be interesting to see how an algorithm such as SLF performs in comparison to state-of-the-art (swarming) routing algorithms such as Dynamic Source Routing (DSR), Ad-hoc On-demand Distance Vector Routing (AODV), and AntHocNet.

In the next chapter, we will focus on applying SLF to a hard real-world application domain. More precisely, we would like to explore how SLF performs in the problem domain of routing in Mobile Ad-hoc Networks.
In the previous chapters we have focused on applying bee-inspired swarm-intelligence algorithms on the problem domain of foraging. Although this problem domain is a nice abstract representation for other problems, we wonder how such algorithms may actually perform in real-world applications. Therefore, in this chapter, we are going to focus on such a real-world application. Since Swarm Intelligence (SI) is able to cope with large and complex problems which are in need of self-organization, scalability, and adaptability, we have chosen the real-world problem domain of routing in Mobile Ad-hoc Networks (MANETs).

6.1 Introduction

Bees live in colonies and one of the main activities during the day is foraging for food. Due to the limited cognition of bees, most tasks are a shared effort. More precisely, in foraging, bees exploit previous search experience which can be either their own experience or that of a colony member. Previous search experience has a stigmergic nature; it stimulates the performance of an action. Bee foraging mainly consists of two behaviors. First, exploration, which lets bees discover food sources and enables them to navigate through the environment. Second, exploitation, which enables bees to perform recruitment on colony level and communicate previous search experience about where and how to find food sources.

Routing is the task of directing data packets from a source node to a given destination (DiCaro et al., 2005). For a routing protocol to be able to direct data packets, it first has to discover routes from source to destination. Once discovered, the routing protocol stores the route information in routing tables present at each node. Once this information is available to the node, each time a packet has to be sent, the node first checks its routing table for a known route and, if found, sends the packet on its way. If it lacks a known route, it queues the packet and again tries to discover a new route.

This chapter is based on the following publication.

Given a static network, basic (non-load-balancing) routing is trivial since routes never change and can be hard-coded in every router. They may even be added manually depending on the size of the network. Over the years however, networks have grown larger and have become dynamic. Examples of such networks are mobile-phone networks, peer-2-peer networks, satellite-constellation networks, and ad-hoc networks. Especially the latter two may be characterized by mobility and the lack of a preset infrastructure. Moreover, each of the nodes in such a network might be resource limited. Considering these properties, it is obvious that static routing is not possible anymore. Therefore, researchers have started to design protocols which are able to route in such networks. Currently, Dynamic Source Routing (DSR) (Johnson and Maltz, 1996) and Ad-Hoc On-demand Distance Vector routing (AODV) (Perkins and Royer, 1999) are the best known protocols. Although used as the standard for routing in Mobile Ad-Hoc Networks (MANETs), they are not the best performing protocols around. Routing protocols based on Swarm Intelligence outperform the current standard protocols (DiCaro et al., 2005; Wedde et al., 2005).

This chapter presents a novel SI routing protocol, named Stigmergic Landmark Routing (SLR), which is based on bee behavior. It sets itself apart from other SI routing algorithms by using landmarks during the routing process. Landmarks are nodes which can be used as a hub and only in these nodes route information is stored and maintained. Detection and maintenance of such landmarks is a shared effort and each landmark stimulates the choice of a high performing route. Landmarks can be linked together and in such a way that together they create a landmark network. Eventually, packets use the landmark network for routing. Current SI routing protocols, in contrast, store and try to maintain their routing information at every node which may be a heavy burden on each node’s battery and consequently the network. In a comparison between SLR and state of the art algorithms, SLR shows an increase in performance in networks up to 100 nodes.

The remainder of this chapter is structured as follows. First, §6.2 provides the reader with a brief overview of the related work. We will focus on general routing and SI routing. Second, §6.3 describes the SLR protocol for routing. §6.4 continues with our experimental set-up. §6.5 presents the obtained experiment results. §6.6 reviews the experiments and their results and presents the conclusions about the protocol. It also provides an answer to RQ4: To what extent is a bee-inspired SI algorithm capable of providing improved solutions to current state-of-the-art algorithms in complex, real-world problems?

6.2 Routing in Mobile Ad-Hoc Networks

MANETs are infrastructure-less and dynamic networks based on traditional TCP/IP protocols for communication between nodes. Due to node mobility, radio noise, the limited resources of each node, and increasing size of the networks, current routing algorithms are not fit for the routing task. More precisely, they lack efficiency and reliability in the task of routing. Apart from being able to deal with limited resources, the routing protocol should also be able to cope with constantly changing network size, traffic density, load balancing, and network partitioning (Abolhasan et al., 2003)
Protocols that are able to cope with routing in MANETs can roughly be divided into three groups. First, proactive protocols which find routes to all destinations at the start of the routing effort. Over time these routes are being maintained using a periodic route update process (Abolhasan et al., 2003). Destination-sequenced distance vector (DSDV) (Perkins and Bhagwat, 1994) and Wireless Routing Protocol (WRP) (Murthy and Garcia-Luna-Aceves, 1996) are well-known examples of such algorithms. They provide single loop-free paths to each destination. However, both have a considerable overhead cost and are therefore less scalable. Second, reactive protocols which only determine routes on-demand, such as Ad hoc On-demand Distance Vector (AODV) (Perkins and Royer, 1999) and Distributed Source Routing (DSR) (Johnson and Maltz, 1996). AODV makes use of periodic beaconing and sequence numbers for establishing single-path (loop-free) routes. DSR requires each packet to store every node address it comes across in a certain route which increases packet size and bandwidth requirements but with the added advantage that it can create multiple paths to a destination. However, both protocols also have disadvantages. AODV may experience high delays in route construction. DSR is less scalable in large networks due to growing packet sizes. Third, hybrid protocols which combine properties of the other two groups, such as Zone Routing Protocol (ZRP) (Haas, 1997) and Zone-based Hierarchical Link State (ZHLS) (Joa-Ng and Lu, 1999). In the former, the nodes have a routing zone (in hops) to which a node has to maintain connectivity pro-actively. Routes to nodes outside of the zone have to be determined reactively. In contrast, the latter employs a hierarchical structure. The zone is made up of 2 levels, (i) a node level and (ii) a zone level. When a route to a remote zone is required, a zone level location request is submitted to all other zones. Both have the advantage of reducing overhead. Disadvantageous are the zone value settings which may limit the protocol to a pure reactive or pure proactive strategy. Moreover, in ZHLS, all nodes need a pre-programmed static zone map in order to function.

Each of the above mentioned protocols is capable of handling the challenges a MANET poses to a protocol to some degree. However, either routing overhead (i.e., number of routing packets per correctly delivered data packet) is large or the protocols are not (sufficiently) scalable.

In an attempt to deal with these issues, researchers have started looking at Swarm Intelligence. Current SI routing protocols mainly make use of the ant foraging mechanism where virtual ants deposit virtual pheromone on links between nodes. Using pheromone, the virtual ants are able to create efficient routes to destinations. This has led to a number of protocols such as Ant-Based Control (ABC) (Schoonderwoerd et al., 1997), AntHocNet (DiCaro et al., 2005), Ant-colony-based Routing Algorithm (ARA) (Gunes and Spaniol, 2002), and others (Baras and Mehta, 2003; Roth and Wicker, 2003; Shen and Jaikaeo, 2005; Wang et al., 2009).

In recent years, the bee foraging mechanism has drawn attention in a number of dynamic problems such as Combinatorial Optimization Problems (COP) (Lucic and Tedorovic, 2003; Lemmens and Tuyls, 2009) and also routing in MANETs. BeeAdHoc (Wedde et al., 2005) presents a MANET routing algorithm based on bee foraging behavior. The latter is a reactive source routing algorithm which has a lower power consumption due to less control packets during routing. However, routing packets keep track of the complete path between source and destination. A path consists of all visited nodes between source and destination. Keeping this information in a packet increases the size of the packet which makes the algorithm less scalable.
More precisely, such packets decrease node bandwidth and sending such packets drains node batteries.

Although each of these algorithms are capable of outperforming the current standard protocols for routing in MANETs, there is still performance to gain, e.g. by using location information.

6.3 The Stigmergic Landmark Routing algorithm

Stigmergic Landmark Routing (SLR) is a novel distributed routing algorithm which is able to find, create, and maintain multiple routes between a sending source and a receiving destination. Moreover, it is able to do so in an environment with the following three properties. First, nodes are wireless and thus their communication range is limited. Each node can only observe and communicate with its local neighborhood. Second, nodes are able to move freely through the environment with different speeds which may lead to constantly changing local neighborhoods. Third, there is no preset infrastructure. More precisely, an infrastructure has to be created and is set up between nodes in an ad-hoc manner and may often be temporary.

SLR is inspired by the algorithm Stigmergic Landmark Foraging (SLF) (Lemmens and Tuyls, 2009) and its basic functioning can be described as follows. During exploration for a destination, packets keep track of their outbound path by storing every visited node inside the packet’s memory. In the remainder of this chapter, such a sequence of visited nodes is called a vector. Once the destination is found, the packets retrace their path and while traveling back to the source they store part of it in key nodes, indicating the vector to take in order to get to the next key node or the destination. In the remainder of this chapter, key nodes are called landmarks. In the end, such behavior results in a goal-directed route consisting of one or possibly multiple vectors. Once they arrive in the source, bee-inspired recruitment is performed via a virtual dance. A dance merely offers the search experience (in the form of a vector) to other members which can then choose to copy it. The decision to copy it depends on the dance strength for the vector. A landmark vector is associated with an Integer-valued route number and is given a certain Dance Strength (DS). Dance strength is a Real-valued variable (decreasing over time) indicating how good a vector is. Once a vector is copied, data is piggy-backed on routing packets which exploit the copied search experience leading them to landmarks, which may contain another route vector towards the next landmark, or the destination. In a general problem, a location is considered a landmark depending on criteria suitable to the problem. In network routing, we consider a node as a landmark when it has at least $n$ links, where $n$ is an integer to be set by the user. Considering vectors (i.e., sequences of nodes), we maintain four. First, a Global Vector (GV) is maintained in the memory of explorers, spans between the source node and the destination node, and only exists for the duration of that trip. Second, the Temporary Vector (TV), maintained inside the memory of any agent, represents the current segment of the route the agent is on, and exists as long as the agent is traveling on that segment. The Landmark Homing Vector (LHV) and Landmark Goal Vector (LGV), stored inside the memory of a landmark node, represent segments of a specific route, and they exist as long as their Dance Strength (DS), which indicates the goodness of the vector, is positive. The four vectors together capture the essence of the protocol and direct the agents to their destination.
The Stigmergic Landmark Routing algorithm

Table 6.1: Brief description of the agent states and their follow-up states.

<table>
<thead>
<tr>
<th>State</th>
<th>Next state</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explorer</td>
<td>Stochastic Explorer, Carrier</td>
<td>search a route to destination</td>
</tr>
<tr>
<td>Carrier</td>
<td>Dancer</td>
<td>set-up destination-directed route while traveling back to source using GV or LHV’s</td>
</tr>
<tr>
<td>Recruit</td>
<td>Stochastic Carrier</td>
<td>set-up source-directed route while traveling to destination using LGV’s</td>
</tr>
<tr>
<td>Dancer</td>
<td>(Stochastic) Explorer, Recruit</td>
<td>virtual internal state; represents routes in a node’s routing table</td>
</tr>
<tr>
<td>Stochastic Explorer</td>
<td>Carrier</td>
<td>set-up an optimized route using previous search experience found in landmarks</td>
</tr>
</tbody>
</table>

The resulting protocol consists of five agent states which together create the behavior of the protocol, see Table 6.1 for a brief description.

We will now explain the protocol using an example. Suppose a source wants to send a data packet to a certain destination. Whenever a source wants to send a data packet to a destination, it is able to do three things. (i) It checks whether the destination is present in its local neighbor list. If present, it sends the data packet to the neighbor. If not, it (ii) checks its routing table for a known route to the goal. If existing, it sends the data packet on its way. Each data packet is piggy-backed on a recruit in order to decrease packet overhead. A recruit creates the source-directed part of the route during its travel. If no route is known, the protocol (iii) queues the data packet and tries to discover a route by broadcasting an explorer to all neighboring nodes which in turn forward it. Once the destination is found, a destination-directed route is setup by a carrier while returning to the source.

The following subsections describe each internal state’s behavior. Note that in some internal states, a selection policy is used to determine which packets may continue to be send. Such a selection policy limits routing overhead by blocking packets using a route which may be considered bad. A packet’s route is considered good if its property values (e.g., hop count and delay) are within a certain range $r$ of the property values that belong to the first received packet at that node (Marina and Das, 2001). The selection range is a user-defined setting and is a real-valued factor.

6.3.1 ‘Explorer’ Behavior

The explorer is essentially responsible for route discovery. Arriving at a node, the explorer adds its previous visited node to its GV. Furthermore, it checks whether it returned to its source, if so, it is dropped. It also checks whether it arrived at the destination. If so, the agent switches its internal state to carrier and it follows its found route (represented by the GV) back to the source. Additionally, during exploration, at each intermediate node, the agent checks whether there is a known route to the destination and whether it is profitable to exploit this information (see Equation 6.3. A more detailed description will be given in Subsection 6.3.3). If so, it changes its internal state to stochastic explorer and it is send on that route. A stochastic explorer tries to exploit route information to get to the destination and selects a route probabilistically. Each route can be rated according to its dance strength, the total number of hops, total delay and total queue from the current node to the destination. The higher the strength and possibly the
lower the latter three values are, the higher the probability of selecting that route, see Equation 6.1,

\[
P_r = \frac{w_s \times DS_i + w_h \times \frac{1}{h_i} + w_d \times \frac{1}{d_i} + w_q \times \frac{1}{q_i}}{\sum_{i=1}^{B_d} w_s \times DS_i + w_h \times \frac{1}{h_i} + w_d \times \frac{1}{d_i} + w_q \times \frac{1}{q_i}},
\]

where \( DS_i \) represents the dance strength of route \( i \), also see Equation 6.2, \( h_i \) represents the number of hops of route \( i \) to reach the destination, \( d_i \) represents the total delay of route \( i \) to reach the destination, \( q_i \) represents the total observed queue of route \( i \) to reach the destination. The weights can be set by the user to indicate how important each measure is for the route. \( B_d \) indicates the total number of dances for a route. This way, routes that are danced ‘stronger’ for recruit more data packets. Thus, high performing routes are going to be preferred. Additionally, a route will only be chosen if its number of hops is lower than the number of hops of the route the agent is currently on. In any case, if no route is known, the (stochastic) explorer is broadcasted to all neighbors. Appendix B, Section B.1 gives a more detailed description of the explorer behavior in pseudo-code.

### 6.3.2 ‘Carrier’ Behavior

As indicated in the previous subsection, an explorer changes its internal state to carrier the moment it arrives at its destination. The carrier tries to travel back to its source by using the constructed GV of the explorer. During its way back, the carrier is able to create LGVs and in essence creates a route towards the destination. If, during its GV-guided travel back, the carrier is to be send to a node which is not available anymore, it is dropped. Once a carrier returns to the source, the found route is danced for. Appendix B, Section B.2 gives a more detailed description of the carrier behavior in pseudo-code.

**Stochastic Carrier** behavior is much alike the above mentioned behavior. However, stochastic carrier is a state which can only be reached from the recruit state. As we will explain in subsection 6.3.4, the recruit creates a source-directed route consisting of (possibly) multiple landmarks. A stochastic carrier uses the landmark vectors (copied to its TV) to guide it travel to the source. Thus, the GV is not used in this state.

### 6.3.3 ‘Dancer’ Behavior

Once a carrier has returned to a source, dancing for the found destination starts. The internal state Dancer is essentially a virtual internal state. More precisely, there is no packet involved. Once a route is found by the combined efforts of explorers and carriers, the route is added to the routing table of the source. This search experience is only available for a certain amount of time. This property is captured by the so-called dance strength which is a real-valued variable indicating the goodness of the vector in a route. Over time, dance strength decreases and the route becomes less attractive for being followed, see Equation 6.2.

\[
DS_i = DS_i - \epsilon_d
\]
6.4 Experimental Set-up

A source node uses the found routes to send data packets on its way. If it queued packets previously, the arrival of a *carrier* and its subsequent route dancing is a trigger to potentially dequeue the appropriate data packets and send them on their way using the acquired route information. If a data packet is send, it is piggy-backed on a *recruit* to decrease packet overhead. Whether a data packet is dequeued depends on the number of dances for its destination. This behavior is captured in Equation 6.3 which gives the probability of broadcasting explorers in order to discover new routes to the destination and is defined as follows:

\[ P_e = e^{\left( -\frac{1}{2} \frac{DS_i^2}{\sigma^2} \right)} \],

(6.3)

where \( DS_i^t \) indicates the total dance strength of all route dances for destination \( i \), and \( \sigma \) is a scaling parameter and essentially determines the tendency to explore. Notice that if \( DS_i^t = 0 \), there is no dancing so that \( P_e = 1 \) and explorers will be broadcasted in order to discover routes to the destination. If \( DS_i^t \) is low, there are few routes to the destination which additionally may be bad and therefore the data packet is not likely to get recruited to a certain route. The data packets will, in a sense, be ‘recruited to explore’ by the lack of the presence of dances. As \( DS_i^t \) increases, exploration probability decreases and, as discussed below, data packets will be more likely to get recruited by a route. In the latter case, a route is stochastically selected, see Equation 6.1.

6.3.4 ‘Recruit’ Behavior

Once a route to a destination is available, data can be send using the route. The *recruit* is ‘loaded’ with a data packet to a certain destination. To reach the destination, it uses route information found in landmarks on its route. Landmarks are detected by examining the number of links of the node. During its trip, the *recruit* is able to create or reinforce the homing route. If a recruit is not able to follow its route to the destination, it is queued and the receiving node starts to send out explorers to discover a new route. Appendix B, Section B.3 shows a more detailed description of the *recruit* behavior in pseudo-code. Note that if a route fails, an *errorroute* packet is send to the previous visited nodes in order to delete references to the failing route.

6.4 Experimental Set-up

We implemented SLR in the Network Simulator 2 (NS-2) (Fall and Varadhan, 2001). Note that the notion of agent in NS-2 is different than ours. In NS-2, each node has an agent layer. Thus, an agent is connected to a node, has a fixed location and deals with incoming and outgoing traffic, i.e., packets. In SLR, the nodes are viewed as black boxes over which packets are able to travel. Each packet represents an agent which can have a certain agent state.

In order to get a notion of the performance (i.e., efficiency, scalability, and adaptability) of SLR we perform two comparisons. First, we quantitatively compare the protocol to AODV (Perkins and Royer, 1999), which is the de-facto standard in routing at present. Additionally, we provide a qualitative comparison with AntHocNet (DiCaro et al., 2005). The quantitative
comparison is performed using the Network Simulator 2 (NS-2) which runs each experiment for $t_{\text{sim}} = 300$ seconds. To obtain a fair comparison, each protocol has to route traffic through a flat environment of 3000 by 1000 meters. In this environment, we place 25-100 nodes which are able to move freely through the environment. Movement patterns are generated using the Random Waypoint Model (RWM) (Johnson and Maltz, 1996) to assure realistic movement. Each node chooses a random destination point and a random speed and moves to the selected destination. On arrival, the node pauses for a certain time period before deciding on a new destination and speed. The model expects four parameters to be set. The first two set a limit to the minimal ($v_{\text{min}}$) and maximal speed ($v_{\text{max}}$) with which a node can move through the environment. The remaining two parameters limit the minimal ($p_{\text{min}}$) and maximal ($p_{\text{max}}$) pause time for a node. To analyze each protocol’s performance we create a set of environments in which the RWM parameters differ. More precisely, $v^i \in [0, 20] \text{ m/s}$ and $t^i_p \in [0, 480] \text{ s}$ where the index $i$ indicates the particular node.

Data traffic is generated by 20 randomly selected nodes which act as Constant Bit Rate (CBR) sources sending 1 data packet per second to a random destination node. Each of these sources start generating data traffic at a random time but at least in the time frame between 0 and 180 seconds. Once started, the sources keep generating traffic until the simulation ends. To select and set nodes as traffic sources, we use the built-in traffic source generator available from NS-2, i.e., CBRGEN.TCL.

We run experiments for four different-sized networks with 25, 50, 75, and 100 nodes respectively. For each network, 7 different-valued pause times are used, namely, 0, 15, 30, 60, 120, 240, and 480 seconds. In an experiment, pause time of all nodes in the network is fixed on one of these values. We perform 30 experiment runs. One experiment run consists of 4 (i.e., the network size) times 7 (i.e., the pause time) experiments. On each run, new RWM and traffic patterns are generated which are used by the protocols to perform the task of routing.

### 6.4.1 Simulator settings

SLR depends on a number of parameters. Values for these parameters are determined via a Genetic Algorithm (GA). The GA tuned the parameters on six different node traffic/movement scenarios with 100 nodes and 20 CBR sources sending 1 data packet per second. The resulting settings are as follows. SLR routing packet lifetime is set to 30 hops. A node is detected as a landmark when it has 23 links. Each landmark receives a initial dance strength of 0.5. Dance decay is issued each 0.704806 seconds and has a value of 0.031189. This means that a landmark will exist for 16.03 steps or 11.30 seconds. Routes are not reinforced on a revisit by an appropriate packet. A source re-searches a route to a destination every 2 seconds. Routes are evaluated according only to their dance strength and so $w_s$ is set to 1. The selection policy (i.e., a mechanism to limit routing overhead) allows packets to continue being send if their measures (i.e., number of hops, delay) are within a factor 0.005840 of the first received packet. A node’s tendency to explore (i.e., $\sigma$) is set to 0.125888.
6.4.2 Type of measurements

Using the settings as indicated in the previous subsection, we compare the protocols on a number of performance measures. Each measure is recorded as a function of (i) the number of nodes ($n$) and (ii) the pause time ($t_p$). First, we run experiments with a variable number of nodes. A protocol’s routing effort increases as the number of nodes increases. However, we still want the protocol to have a high performance. The number of nodes can thus be seen as an indication of scalability. Second, by using different values for $t_p$, we can set the mobility of the nodes and thus have an indication of adaptability. A low value for the parameter results in a very dynamic network whereas a high value creates a rather static network. In the former, high performance suggest high adaptability. In the latter, adaptability may not be an issue, however, high performance suggests the ability of the protocol to quickly create routes to locations formerly unconnected. Both may pose a difficult problem to the routing protocol. More precisely, a low $t_p$ leaves little time for actual routing since nodes are constantly moving and a found route may become invalid very quickly. A high $t_p$ relaxes this problem, however, due to the random placement of nodes in the environment, nodes may be unreachable for a long time. We use three performance measures.

First, *Packet Delivery Ratio* (PDR) $\in [0, 100]$ which is the ratio between the correctly delivered data packets ($P_d$) and the total amount of data packets send ($P_s$), see Equation 6.4. Obviously, a high value for the PDR is preferred.

$$PDR = \frac{P_d}{P_s} \quad (6.4)$$

Second, *Average end-to-end Packet Delay* (APD) $\in [0, 300]$ is the average time delay between sending a packet ($t_{ps}^p$) and correctly delivering a packet ($t_{pd}^p$), see Equation 6.5. A low APD is preferred. PDR and APD are standard measures of effectiveness in MANETS.

$$APD = \frac{\sum_{p=1}^{P_d} (t_{pd}^p - t_{ps}^p)}{P_d} \quad (6.5)$$

Third, *Routing Overhead* (RO) corresponds to the number of routing packets ($P_r$) per correctly delivered data packet ($P_d$), see Equation 6.6. The measure indicates the routing effort done in order to correctly deliver data packets. A low value is preferred. RO is a measure of efficiency and may also give an indication about scalability.

$$RO = \frac{P_r}{P_d} \quad (6.6)$$

6.5 Results

In this section we will provide the results of two types of comparisons. First, a quantitative comparison between SLR and AODV. Second, a qualitative comparison between SLR and another Swarm Intelligence routing algorithm, namely AntHocNet.
Figure 6.1: Packet Delivery Ratio with increasing node density. \( t_p = 30 \), 20 CBR sources sending 1 data packet a second.

Figure 6.2: Packet Delivery Ratio with increasing pause time. \( \eta_n = 50 \), 20 CBR sources sending 1 data packet a second.

6.5.1 The Quantitative Comparison

Figure 6.1 presents the Packet Delivery Ratio on different-sized networks with high mobility (i.e., \( t_p = 30 \)). We can clearly see that SLR outperforms AODV especially in low-density networks (i.e., \( \leq 100 \) nodes). This is due to the fact that SLR only stores routing information in certain nodes, namely landmarks. Landmarks are detected according to their number of links. Even with high mobility, such nodes may be connected to the network for a prolonged period of time and as such can be used for backup routes. Since SLR is able to create multiple routes, the protocol does not have to reroute as much as AODV once a route gets invalid.

Figure 6.2 and 6.3 show the Packet Delivery Ratio on networks with a fixed number of nodes but with variation in node pause time. The figures show that SLR outperforms AODV in all cases. In low-density networks (i.e., \( \leq 50 \) nodes), we see a trend that SLR outperforms AODV in highly dynamic networks but tends to perform equally in more static networks. Low-density, static networks may not provide a lot of landmarks to make use of which results in source-routing as seen in DSR. Once a route becomes invalid both protocols have to reroute. In higher density networks (i.e., \( 50 < \text{nodes} \leq 100 \)), we see a different trend. SLR tends to outperform AODV both in highly dynamic networks as in more static networks. Such networks
provide the protocol with backup options. Due to the increased number of nodes, the network is better connected which results in more landmarks for a prolonged period of time.

With respect to Average Packet Delay, Figure 6.4 shows a significant difference between SLR and AODV. Although SLR’s packet delay increases with increasing network size, it is considerably lower than the packet delay experienced with AODV. Moreover, as can be seen in Figure 6.5, SLR shows a stable delay at different mobility levels. Both the PDR and APD results suggest that SLR is a more effective, better performing alternative to AODV in networks smaller than 100 nodes.

With respect to scalability, the results suggest that AODV may outperform SLR in very big networks. An indication of this can be observed in Figure 6.6 and 6.7 where SLR shows a significantly higher Routing Overhead (RO). This means that, in comparison to AODV, significantly more routing packets are used to create routes to the destination per correctly delivered data packet. Of course, this may be due to the fact that SLR creates multiple routes while AODV only creates a single route. Moreover, SLR tries to maintain the most promising routes proactively. A striking difference between AODV and SLR is the standard deviation on the RO. AODV manages to keep the standard deviation low with increasing network size and different

Figure 6.3: Packet Delivery Ratio with increasing pause time. $\eta = 100$, 20 CBR sources sending 1 data packet a second.

Figure 6.4: Average end-to-end delay with increasing node density. $t_p = 30$, 20 CBR sources sending 1 data packet a second.
network dynamics. In contrast, SLR shows an increasing standard deviation with increasing network size. This can be explained by the fact that SLR tries to maintain multiple routes. Obviously, the more nodes a network contains, the more connection options for these routes are available. If, in addition, nodes act increasingly dynamic, then the probability of a route break increases. Indeed, in Figure 6.7, we can observe that with higher \( t_p \) (i.e., static node behavior) the standard deviation decreases.

### 6.5.2 The Qualitative Comparison

The AntHocNet algorithm is not readily available in NS2 so a direct and quantitative comparison is not (yet) performed. However, as our settings are similar to those used in AntHocNet we will try to at least give a qualitative comparison between SLR and AntHocNet. For this, we will focus on the individual qualities of each algorithm.

AntHocNet is inspired by the foraging behavior of ants and therefore it uses artificial pheromone for its routing purposes. In AntHocNet, there are two types of packets which are called ants, (i) forward ants and (ii) backward ants. If necessary, forward ants are send out to search
for a route to a destination. During their trip, the forward ants keep track of which nodes they visited, i.e., source routing. Once they have arrived at the destination, a backward ant is send back and it uses the route of the arrived forward ant to get back to the source. During its return trip, it updates the amount of pheromone for each used link at each node. Whenever the backward ant arrives at the source, a route is set up and data can be send. Due to the accumulating properties of pheromone, routes may be linked together or repaired. For a detailed description of AntHocNet we refer to (DiCaro et al., 2005).

As indicated in Section 6.3, SLR is inspired by the foraging behavior of bees and therefore does not use pheromone for its routing purposes. Instead, it makes use of so-called vectors which in essence are sequences of nodes to visit. Vectors may be stored at key nodes, i.e., landmarks, in the network.

Knowing these basic properties allows to draw a number of conclusions with respect to MANETs. First, both are hybrid algorithms, i.e., they combine reactive and pro-active behaviors. A route is setup in a reactive, on-demand manner and once a route is available, it is pro-actively being maintained. Second, AntHocNet stores information at every visited node while SLR only stores information at landmarks. In the former, the network has to perform more administration tasks (i.e., maintaining routes at each node) which may drain node battery life and may result in partial network failure. Third, node-to-node routing, as seen in AntHocNet may have its advantages in larger networks due to an abundant choice of node links. However, in small or low-density networks, node-to-node routing may prove disadvantageous due to link failures and the related amount of time necessary to repair a route containing a link failure. SLR combines the behavior of a source-routing algorithm, such as DSR, and a node-to-node routing algorithm such as AODV. In small or low-density networks, SLR performs source routing which limits pro-active route maintenance and increases performance. In larger networks, SLR combines source routing and node-to-node routing. The latter can be observed by how the packets travel from landmark to landmark. The intermediate nodes between landmarks are source routed. As a consequence, we can suggest that SLR may perform significantly better in low-density or small networks, see Figure 6.8. More precisely, up to approximately 85 nodes, SLR outperforms AntHocNet. Note that Packet Delivery Ratio values of AntHocNet

![Figure 6.7: Average routing overhead with increasing pause time. \( n_n = 100 \), 20 CBR sources sending 1 data packet a second.](image-url)
Figure 6.8: Packet Delivery Ratio with increasing node density. $t_p = 30$, 20 CBR sources sending 1 data packet a second.

are copied from (DiCaro et al., 2005) and as such can only give a suggestion of performance. A direct comparison may increase or decrease differences.

6.6 Chapter conclusion

In this chapter, we presented a novel routing protocol called Stigmergic Landmark Routing (SLR). It draws its inspiration from the foraging/nest-site selection behavior of honeybees and it therefore belongs to the class of BCO algorithms. The contributions in this chapter are used to provide an answer to RQ4: To what extent is a bee-inspired SI algorithm capable of providing improved solutions to current state-of-the-art algorithms in complex, real-world problems?

SLR is capable of performing efficient routing in Mobile Ad-hoc Networks. Such networks lack preset infrastructure and are highly dynamic due to node mobility and wireless node communication. To show the performance of SLR we compared it to the current standard routing protocol for Mobile Ad-hoc Networks, namely AODV. The latter is a reactive protocol which creates a single route between source and destination on-demand. All nodes in this route store routing information, i.e., what node to go next to in order to get to the destination. In contrast, SLR, is a hybrid protocol which is able to create multiple routes between source and destination. Due to the use of landmarks only a limited number of nodes store routing information. Intermediate nodes between landmarks are source-routed, i.e., packets keep track of which sequence of nodes they have to take to get to the next landmark. Since landmarks are more heavily connected to the network, even with high mobility, such nodes may be connected to the network for a prolonged period of time. As such, routes may be available for a longer time period which reduces rerouting and packet delay and increases packet delivery. Due to the property of local cooperation, routing packets are also able to link landmark routes to each other. Moreover, creating and maintaining the landmark network is a shared effort.

The results show that SLR is able to outperform AODV in networks smaller than 100 nodes. More precisely, SLR is able to deliver more packets in less time in comparison to AODV and therefore it is more effective.
The qualitative comparison of SLR with AntHocNet suggests that SLR outperforms AntHocNet, see Figure 6.8. However, a thorough quantitative comparison still has to confirm this conclusion.
This chapter provides conclusive answers to the research questions and the problem statement posed in Chapter 1. First, §7.1, will answer each research question separately. Second, we will combine the answers in such a way that we can provide the answer to the problem statement in §7.2. Finally, §7.3 we will provide recommendations for future research.

7.1 Answers to the research questions

**Research Question 1 (RQ1): What are the key principles of bee-inspired SI and do they differ from other social-insect SI?**

Chapter 2 provides an answer to this research question. More precisely, social-insect SI depends on four principles. First, individuals in a swarm only have local awareness. Their view of the environment is limited to their direct surroundings and they do not take their previous actions into account for future actions. Second, swarms are decentralized. Although central control is possible over short distances with a small number of individuals, the performance of such control deteriorates with distance or increasing number of agents. Therefore, decentralized control is a necessity. Third, to implement control, social insects rely on local interaction. Local interaction is used to solve a sub-problem of the task at hand. For example, find the shortest path in a segment in the total search space. Fourth, by using local interaction the swarm displays self-organization on a global level.

Above mentioned principles potentially describe any type of SI. However, we have to note that each SI is subtly different. For example, fish schools SI depends on proximity between individuals. In contrast social-insect SI does not seem to depend on it. At least, not in the way that fish schools depend on it. Social-insect swarms have specialized in swarming methods that provide them with ways to perform swarming over large spatial areas whereas fish schools can only cover an area as big as the swarm size allows it without loosing swarm density. An additional example is that fish schools do not seem to have a swarm memory. Such swarms seem to be be purely reactive. If a disturbance forces such a swarm in a possibly harmful configuration, this situation may repeat indefinitely. For instance, a group of dolphins may ‘herd’ a school of fish. While some of the dolphins patrol along the borders of the fish school
Conclusion

to control them, other dolphins prey on the ‘captured’ swarm of fish (Morozov, 1970). In contrast, ants and bees are able to keep a temporary swarm memory by depositing pheromone inside of the environment or dancing inside of the hive respectively.

Research Question 2 (RQ2): How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

In Chapter 3, we compare social-insect-inspired SI in the task of foraging. This task is representative for many more tasks such as routing problems and shortest path problems. Such tasks have the characteristic that they can be divided into sub-problems and combination of corresponding solutions result in a global solution. We focus on inspiration from social-insect swarms of ants and bees. Although the skill set of both insects is different, their strategies can still be compared. With skill set we indicate the fact that ants are mostly ground-bound and bees are mostly airborne. Although the skills differ, both types of swarm employ strategies that are not mutually exclusive. More precisely, ground-bound desert ants also employ navigation strategies that are found in airborne bees. Such a strategy can be used whenever an environment allows relatively free unobstructed movement. Similarly, bees also employ pheromones for a variety of tasks, including foraging. However, the foraging pheromone, also called Nasonov pheromone (Smith and Cobey, 1991; Tautz and Sandeman, 2003), is used slightly different. Bees use such pheromone to indicate nest entrances and nectar-containing flowers. They therefore only guide bees once they have arrived at local destinations. Use of a strategy is mainly determined by the environment that the swarm operates in. For example, both land-dwelling insects such as desert ants (e.g., of species *Cataglyphis Fortis*) as airborne insects such as honeybees (e.g., of species *Apis Mellifera*) make use of Path Integration. For clarity, we attribute the strategies to the insect that makes most use of it in foraging. This means, that we attribute pheromone usage to ants and path integration to bees. Although both swarm types are able to efficiently solve the foraging task, they employ different strategies to create a solution. Ant swarms employ indirect communication whereas bees employ direct communication. As such, ants represent their search experience externally while bees represent their search experience internally.

Above mentioned strategies have been used to implement two computational algorithms. In a direct comparison between the algorithms in open environments without obstacles, results show that an internal representation increases the efficiency of foraging. However, due to the lack of external representation, bees lack adaptability in more complex environments, such as a maze. With respect to scalability, the experiment results show that ant-inspired algorithms are most scalable with respect to the number of agents while bee-inspired algorithms are most scalable with respect to the size of the environment. This difference is due to the nature of their strategies. More precisely, ant-inspired algorithms need to perform administrative pheromone tasks which may take considerable time. Increasing the number of agents does not increase or decrease the amount of administrative effort as long as the environment size stays equal. Likewise, bee-inspired algorithms need to perform location calculations which may take considerable time. Increasing the size of the environment does not increase nor decrease this effort while an increasing number of agents does alter the computation time.
Research Question 3 (RQ3): To what extent can we eliminate potential weak points in bee-inspired SI algorithms?

Basic bee-inspired SI lacks adaptability in complex environments since it was not created for use in such environments. However, the efficiency performance that such an SI provides make it interesting to use. This thesis provides three extensions in order to improve the performance of bee-inspired SI, see Chapter 4 and 5. First, wall-following is an extension with which agents are able to move around obstacles and enables agents to get out of lock-out situations. Second, inhibition pheromones provide agents with a skill to indicate no-interest locations. Over time, such a strategy limits the search space that needs to be covered. Using the first two extensions resulted in the algorithm BSP. Results show that it is as efficient as BS but with the added capability of adaptability. However, BSP depends on pheromone and especially in embodied systems, such dependency may prove a hard challenge. Third, landmark navigation enables the agents to break free of literal pheromone dependency and improves the skill of agents to maneuver past obstacles even without the skill to perform wall-following. This extension resulted in the algorithm SLF. By using landmarks that agents can either place or represent themselves, the agents build an adjustable network of landmarks which they can use to navigate. Results show that this extension further increases efficiency but that it decreases scalability. With respect to adaptability, this extension seems to make the algorithm robust to changes and therefore highly adaptive.

Research Question 4 (RQ4): To what extent is a bee-inspired SI algorithm capable of providing improved solutions to current state-of-the-art algorithms in complex, real-world problems?

In order to show performance of SLF in complex, real-world problems, this thesis provides an application of SLF to the domain of routing in mobile ad-hoc networks (MANETs). The resulting routing algorithm, SLR, proves to increase the performance of state-of-the-art MANET routing algorithms especially in networks of up to 100 nodes. We can categorize such MANETs into two groups. First, whenever the network consists out of few nodes or node density is low, connectivity is limited which decreases routing capabilities. Second, whenever the network consists of many nodes or node density is high, the network is flooded with packets which decreases the routing capabilities by limiting bandwidth. SLR provides an emerging and self-organizing routing infra-structure in both types of networks. In the former case, the protocol enables packets to keep track of their own routes and this results in a source-routing behavior. If the network changes towards the latter case, nodes become more interconnected and source-routing is inefficient. By creating landmark nodes over which packets can be routed, the protocol limits source-routing, enables labor distribution, and increases route efficiency.

7.2 Answer to the problem statement

Based on the answers to the research questions in the previous section, we can now answer the problem statement.
Problem Statement (PS): To what extent can bee-inspired SI algorithms provide a flexible, state-of-the-art mechanism for solving combinatorial optimization problems?

Bee-inspired SI algorithms prove to be very efficient combinatorial-optimization-problem solvers. Experiments in both the domain of foraging and mobile ad-hoc networks have shown that bee-inspired algorithms are capable of efficiently creating solutions to the task at hand. High efficiency arises through internal experience representation and direct recruitment behavior. Internal experience representation depends on the problem domain. In foraging, for example, internal representation is spatial. In networks, internal representation is based on sequences of nodes. Direct recruitment behavior ensures that experience is communicated directly and does not require experience to gather momentum before it is used. However, momentum is still necessary to let global (near) optimal behavior emerge. The more agents make use of experience, the more it will be communicated. Considering the ratio between exploration and exploitation, bee-inspired algorithms are able to exploit experience earlier since PI information is as optimal as it can get. Momentum is used for indicating which food source is most profitable.

Basic bee-inspired SI algorithms lack adaptability. Considering the PS, we should therefore indicate that the basic behavior is only applicable to problems where agent movement is unrestricted. However, complex problems often contain dynamics and interruptive events. In order to achieve good performance, basic bee-inspired SI algorithms need to be extended. This can be done by incorporating pheromone inside of the algorithm, resulting in a hybrid algorithm, showing the best features of both mechanisms. In simulated tasks this may be feasible. However, we see challenges when moving to embodied swarms. Especially in such embodied systems, pheromone dependency may be a problem. More precisely, how do we artificially incorporate this? Therefore, one of the main contributions of this research is the elimination of pheromone dependency and the introduction of stigmergic landmark navigation. The experiments in the problem domain of routing in MANETs shows that such a navigation strategy presents a significant improvement in performance compared to state-of-the-art protocols.

Scalability investigates the ability of the algorithms to cope with increasing agent numbers and increasing search space. Bee-inspired algorithms are most scalable in the size of the search space since this does not require the algorithm to perform extra administrative tasks. Increasing the number of agents however, does increase computation effort and therefore bee-inspired algorithms are less scalable in the number of agents. These results may prove a non-issue, especially in embodied systems. More precisely, in simulations, the search space of a system is often much larger than the number of agents used in the system. Moreover, in embodied systems, there is no single CPU that has to do all the calculations. Instead, it is genuinely distributed and each single CPU, although typically less powerful, has lots of time to perform a small calculation.

7.3 Future research

In this section, we mention three directions for future research. In §7.3.1, we focus on the topic of swarm robotics. §7.3.2 focuses on physical networks.
7.3.1 Swarm robotics

A first direction of future research is to study how bee-inspired SI algorithms perform in the physical world. Up until now, research in this area is falling short. Available research mainly focuses on creating a single, highly intelligent robot. Actual swarm robotics research that employs large swarms has received little attention. Apart from the challenges of incorporating the algorithms in these robots, there is also the challenge of physical limitations. In algorithms one can assume that agents can occupy the same location. Other assumptions could be exact movement, no speed variation etc. In the physical world, these assumptions can not be made. Possible application scenarios could be distributed observation and disaster recovery.

7.3.2 Physical networks

Another line of research can be found in the domain of physical networks. We indicate two types of networks that may be interesting.

Adaptive wireless sensor networks

Currently, wireless sensor networks are deployed to gather information. Such networks are static and consist of one or more sinks to which information is relayed. Each sensor has limited resources and optimization algorithms focus mainly on strategies to keep the networks functional as long as possible. These objectives would not necessarily change by making such sensor networks adaptive. However, we see challenges in such networks in which the sensor nodes themselves are able to move. Mobility relaxes the problem of resource saving and enables a network to optimally position itself to gather the best data and relay such data as efficiently as possible.

Smart distribution networks

Recently, we see an increase in renewable energy sources such as wind and sun power. Such energy sources put stress on the network. Classic energy installations based on coal, gas or even nuclear fuel are most efficient if they remain producing at a constant level. Every increase or decrease in effort is costly. Community pressure and climate-change danger forces us to increase the use of renewable energy sources. Moreover, renewable energy production is starting to become available to individuals. Home-installed solar panels may stress the distribution network even more. This trend is resulting in a highly decentralized energy network which is hard to manage by centralized bodies. We see a need for self-organization, local interaction and storage. We envision that SI techniques may play a key role in the development of such networks.

Another example of smart distribution networks may be based on water. Due to climate change, our weather is getting more and more radical. Some regions in the world will be drained from rain and without adaptation may transform in desert environments. Other locations in the world may receive an extremely large amount of rain. In this area, we see possibilities for creating self-organizing distribution networks which, preferentially fair, distribute water around the globe.

Another example of a mobile phone network problem is the explosive growth of smart phones and increasing mobile internet usage, the infrastructure is starting to lack in...
capacity resulting in patchy service. Examples can be found at events where a large number of people gather and make use of mobile services. SI mechanisms may provide ways to create local ad-hoc infrastructures with local communication and self-organization to take stress of the pre-set infrastructure. For example, by caching information inside of the local ad-hoc network instead of getting the information via the pre-set infrastructure.
References


List of Figures

2.1 Basic reinforcement learning example ........................................... 8
2.2 Various MDP state trajectories ..................................................... 13
2.3 Fish schooling, bird flocking, and animal herding .......................... 16
2.4 Ant and bee swarms ................................................................. 17
2.5 Ant foraging auto-catalytic process ............................................. 17
2.6 Biological Path Integration ......................................................... 18
2.7 Distance and direction determination by means of a waggling dance .. 19
2.8 PSO algorithm progression ........................................................ 21
2.9 Example of a simulated foraging environment ............................. 24
2.10 Example of a Mobile Ad-hoc Network ....................................... 26
3.1 Brownian walk vs. Lévy walk ..................................................... 30
3.2 Computational Path Integration ................................................. 31
3.3 BS flowchart .............................................................................. 33
3.4 The BeeHave tool ....................................................................... 35
3.5 Experiment 1-3: problem cases ................................................... 37
3.6 Experiment 4: Deneubourg bridge problem case ........................... 38
3.7 Efficiency results in the small-world unobstructed problem case ...... 39
3.8 Efficiency: Average needed timesteps of ACO vs. BCO with increasing environment sizes and increasing agent sizes in Problem Case 1 40
3.9 Efficiency: Average needed time steps of ACO vs. BCO with increasing environment sizes and increasing agent sizes in Problem Cases 2-4 ................................................................. 41
3.10 Total needed time ratio of ACO vs. BCO with increasing environment sizes and increasing agent sizes in Problem Case 1 .............. 42
4.1 Maze example ............................................................................. 46
4.2 An example of a tour puzzle ....................................................... 47
4.3 Illustration of wall-following behavior ......................................... 48
4.4 Situations in which pheromone is deposited .................................. 49
4.5 Experiment set 1: Unobstructed environment ............................... 50
4.6 Experiment set 2: (Hard) Deneubourg bridge .............................. 51
4.7 Experiment set 3: “Alligator” ...................................................... 52
4.8 Efficiency .................................................................................. 53
5.1 Situations in which a key location (i.e., landmark) is detected .......... 61
5.2 Creation of the HOMING landmark network by an exploring agent starting from the hive ......................................................... 63
5.3 Creation of the GOAL landmark network by a carrier agent starting from the goal ................................................................. 64
5.4 Behavior transitions between each of the internal states ............... 66
5.5 Probability to explore with increasing tendency to explore .......... 69
5.6 Set 1: BSP Comparison set .............................................. 71
5.7 Set 2: heuristic comparison set ........................................ 72
5.8 Set 3: robustness comparison set ....................................... 74
5.9 BSP and SLF efficiency results in Experiment 1 and 2 ............ 76
5.10 BSP and SLF ratios with increasing colony size and increasing environment size in Experiment 1 and 2 .................................................. 77
5.11 BSP and SLF computation time results in Experiment 1 and 2 .... 78

6.1 Packet Delivery Ratio with increasing node density .................. 90
6.2 Packet Delivery Ratio with increasing pause time .................. 90
6.3 Packet Delivery Ratio with increasing pause time .................. 91
6.4 Average end-to-end delay with increasing node density .......... 91
6.5 Average end-to-end delay with increasing pause time ............. 92
6.6 Average routing overhead with increasing node density .......... 92
6.7 Average routing overhead with increasing pause time ............. 93
6.8 Packet Delivery Ratio with increasing node density ............... 94
In this appendix, we provide the pseudo-code of the algorithms mentioned in Chapter 5. In the following, each section provides the pseudo-code for an agent state. First, we describe the pseudo-code for agent state ‘Explorer’. Second, we provide the pseudo-code for agent state ‘Carrier’. Finally, we show the pseudo-code for agent state ‘Recruit’.
A.1 SLF pseudo-code for agent state ‘Explorer’

Algorithm 7 Pseudo-code for explorer

//Agent.routeID = -1 means unset, Agent.routeID = -2 means invalid
if Agent.location == Hive then
    if Agent.previouslocation != Hive then
        //agent returned to hive
        change internal state to rester
    else
        leave hive, start exploring by Levy flight
    end if
else if Agent.location == Goal then
    if Agent.routeID == -2 then
        Agent.routeID = -1
    end if
    Agent.HV = Agent.TV
    change internal state to carrier
else if Agent.location == Landmark then
    //Location.L == true
    if Agent.routeID != -2 then
        if Agent.routeID == -1 then
            Generate GUID
            20: Agent.routeID = GUID
        end if
        if there is a Landmark.route-entry.routeID = Agent.routeID then
            if Landmark.route-entry.LHV does not exist then
                //route-entry became invalid, no new landmarks can be created by agent
                Agent.routeID = -2
            else if Landmark.route-entry.LGV exists then
                if agent wants to be recruit then
                    change internal state to recruit
                    break
                end if
            end if
            create Landmark.route-entry
            Landmark.route-entry.routeID = Agent.routeID
            Landmark.route-entry.LHV = Agent.TV
            reset Agent.TV
        end if
    end if
    end if
else
    continue Levy flight
end if
else if Agent.location qualifies as a landmark location then
    if Agent.routeID != -2 then
        Location.L = true
        if Agent.routeID is not set then
            Generate GUID
            Agent.routeID = GUID
        end if
        create Landmark.route-entry
        Landmark.route-entry.routeID = Agent.routeID
    end if
end if
50: Landmark.route-entry.LHV = Agent.TV
    reset Agent.TV
    continue Levy flight
end if
Algorithm 8 Pseudo-code for carrier

if Agent.location == Goal then
    if Agent.routeID != -1 then
        leave goal, return to hive (or previous landmark) by following Agent.HV
    else
        set Agent.HV to (D = 0, A = 0)
        explore for other landmarks (or the hive) by Levy flight
    end if
else if Agent.location == Hive then
    change internal state to dancer
else if Agent.location == Landmark AND Agent.HV reached (D = 0, A = 0) then
    if there is a Landmark.route-entry.routeID = Agent.routeID then
        if landmark.route-entry.LGV does not exist then
            Landmark.route-entry.LGV = Agent.TV
        else if Agent.goalquality > Landmark.route-entry.LGV.quality then
            Landmark.route-entry.LGV = Agent.TV
        end if
        reset Agent.TV
        Agent.HV = Landmark.route-entry.LHV
        follow Agent.HV back to next landmark (or the hive)
    end if
else
    Agent.routeID = -1
    explore for other landmarks (or the hive) by Levy flight
end if
else if Agent.HV reached (D = 0, A = 0) then
    if Agent.location qualifies as landmark location then
        create Landmark.route-entry
        Landmark.route-entry.routeID = Agent.routeID
        Landmark.route-entry.LGV = Agent.TV
        reset TV
        if alternate Landmark.route-entry.LHV exist then
            Agent.HV = Landmark.route.LHV
            follow Agent.HV back to next landmark (or the hive)
        end if
    else
        //Agent is lost
        Agent.routeID = -1
        explore for other landmarks (or the hive) by Levy flight
    end if
else
    //Agent is lost
    Agent.routeID = -1
    explore for other landmarks (or the hive) by Levy flight
end if
else if Agent.HV is (D = 0, A = 0) then
    continue exploring for other landmarks (or the hive) by Levy flight
else
    continue following Agent.HV
end if
A.3 SLF pseudo-code for agent state ‘Recruit’

Algorithm 9 Pseudo-code for recruit

```plaintext
//reinforcing a vector means that its DS is increased
if Agent.location == Hive then
    leave hive, start following Agent.GV to next landmark (or the goal)
else if Agent.location == Goal then
    change internal state to carrier
else if Agent.location == Landmark AND Agent.GV reached (D = 0, A = 0) then
    if there is a Landmark.route-entry.routeID = Agent.routeID then
        if Agent.TV == Landmark.route-entry.LHV then
            reinforce Landmark.route-entry.LHV
        end if
        reset Agent.TV
        reinforce Landmark.route-entry.LGV
        Agent.GV = Landmark.route-entry.LGV
        follow Agent.GV to next landmark (or the goal)
    end if
else
    //no route found; agent is lost
    change internal state to explorer
end if
else if Agent.GV reached (D = 0, A = 0) then
    //no landmark found; agent is lost
    change internal state to explorer
else
    continue following Agent.GV
end if
```
In this appendix, we provide the pseudo-code of the algorithms mentioned in Chapter 6. In the following, each section provides the pseudo-code for an agent state. First, we describe the pseudo-code for agent state ‘Explorer’. Second, we provide the pseudo-code for agent state ‘Carrier’. Finally, we show the pseudo-code for agent state ‘Recruit’.
Algorithm 10 Pseudo-code for explorer

perform administration; hopcount, lifetime, add previous hop to GV
if explorer.location = Source then
    DROP explorer
else if explorer.location = Destination then
    if selection policy pass then
        found destination: become carrier
    else
        DROP explorer
    end if
else
    //explorer is at an intermediate node
    if selection policy pass then
        if node.route = known AND exploiting is profitable then
            //a homing/goal route to destination is known
            clear TV
            SEND stochastic explorer on route (copy LHV/LGV TO TV)
        else
            BROADCAST explorer
        end if
    end if
else
    DROP explorer
end if
B.2 SLR pseudo-code for agent state ‘Carrier’

Algorithm 11 Pseudo-code for carrier

perform administration; hopcount, lifetime, observed queue, add previous hop to TV, remove last hop from GV

if carrier.location = Destination then
    DROP carrier
else if recruit.location = Source then
    found source: become dancer
else
    //carrier is at an intermediate node
    if carrier.GV != empty then
        if node.landmark = true then
            10: create route segment: copy TV to LGV
                reset TV
                SEND carrier to next GV node
            else
                SEND carrier to next GV node
            end if
        else
            SEND carrier to next GV node
        end if
    else
        carrier is lost; DROP carrier
    end if
end if
Algorithm 12 Pseudo-code for recruit

perform administration; hopcount, lifetime, observed queue
if recruit.location = Source then
    DROP recruit
else if recruit.location = Destination then
    found destination; deliver data, become stochastic carrier
else
    if recruit not at end of TV then
        SEND recruit to next TV node
    else if node.landmark = true then
        10: if agent.route to destination exists in landmark then
            reinforce/create LHV
            clear TV
            SEND recruit on route (copy LGV to TV)
        else
            send errorroute packet to previously visited nodes
            queue recruit
            broadcast explorers
        end if
    else
        20: send errorroute packet to previously visited nodes
        recruit is lost; DROP recruit
    end if
end if
Nature provides highly efficient and good performing ways of solving problems. Whether we look at photo synthesis, gene activation, or animal foraging, nature has had a vast amount of time to evolve such mechanisms. Therefore, researchers are looking at nature for inspiration on how to handle and improve solutions to problems such as traffic control, ‘harvesting’ renewable energy, and defer or eliminate illness.

In this thesis we focus on combinatorial optimization problems which consists of finding the least-cost solution to a mathematical problem in which each solution is associated with a numerical cost. The bio-inspired mechanism we focus on is Swarm Intelligence (SI) which is the property of a multi-agent system, natural or artificial, in which the decentralized, local cooperation between individuals results in global emerging, self-organizing structures.

The best-known examples of Swarm Intelligence in nature are fish schools and ant colonies. Researchers have already begun to take inspiration from the behavior of these creatures to create optimization algorithms. However, nature does provide more examples of Swarm Intelligence such as that found in honeybees. Bee colonies are able to solve a remarkable amount of problems in nature. Moreover, biologists have provided insights in how such SI operates. Yet, in Computer Science, most attention has been given to fish-school-inspired and ant-inspired SI. Considerable attention, from a computational point of view, for bee-inspired SI is lacking. We argue that algorithms inspired by this swarm behavior may pose a good-performing mechanism for combinatorial optimization problems. Therefore, the problem statement of this thesis is stated as follows.

**Problem Statement:** To what extent can bee-inspired SI algorithms provide a flexible, state-of-the-art mechanism for solving combinatorial optimization problems?

In order to answer the problem statement, we formulate four research questions (RQ).

**RQ1:** What are the key principles of bee-inspired SI and do they differ from other social-insect SI?

**RQ2:** How do social-insect-inspired SI algorithms compare to each other with respect to their performance?

**RQ3:** To what extent can we eliminate potential weak points in bee-inspired SI algorithms?

**RQ4:** To what extent is a bee-inspired SI algorithm capable of providing improved solutions to current state-of-the-art algorithms in complex, real-world problems?

Our research starts in Chapter 2 where we position SI in the field of Reinforcement Learning (RL). Furthermore, we provide a number of biologic SI examples and from these examples we can answer RQ1: What are the key principles of bee-inspired SI and do they differ from
other social-insect SI? We identify four properties, i) unawareness of global structures, ii) de-
centralization, iii) stigmergy, and iv) self-organization.

In Chapter 3, we show how to create a basic bee-inspired SI algorithm. In essence, it con-
sists of direct communication in the form of a (virtual) dance and a navigation strategy called
Path Integration. We use the resulting algorithm, Bee System (BS), to compare it to another
social-insect-inspired SI algorithm, namely ACO. This ant-inspired algorithm enables indi-
viduals to communicate by using (virtual) pheromones. The comparison will be done in the
problem task of foraging. This problem can be used as an abstract form for many more real-
world problems such as routing and shortest path problems. The comparison answers RQ2:
How do social-insect-inspired SI algorithms compare to each other with respect to their per-
formance? In essence, the results show a comparison between hive-limited direct communic-
ation plus a non-pheromone-based navigation strategy on the one hand, and environment-only
indirect communication plus a pheromone-based navigation strategy on the other hand. Bee-
inspired SI shows to be 3 times as efficient in solving the problem of foraging. Moreover, it
proves to be most scalable with environment size. Ant-inspired SI proves to be most scalable
in swarm size. However, the former is preferred since most multi-agent systems are bounded
by the number of used agents while the problem size can be quite large.

Above mentioned comparison also reveals a major shortcoming in BS. Namely, due to the
lack of (heuristic) information in the environment, BS is less adaptive. The system is unable to
learn locally and therefore changes in the environment break the performance of BS. Therefore,
in Chapter 4, we focus on RQ3: To what extent can we eliminate potential weak points in bee-
inspired SI algorithms? We provide two extensions to BS which results in the hybrid algorithm
Bee System with inhibition Pheromones (BSP): i) wall-following behavior and ii) inhibition
pheromones. The former increases the ability to avoid obstacles while the latter decreases the
search space and focuses the algorithm’s attention to interesting locations. The extensions allow
BSP to increase adaptability while limiting the impact on efficiency. More precisely, efficiency
does not seem to suffer from the extensions. To show this, we compare BSP to BS and an ACO
algorithm. Therefore, this chapter also provides an answer to RQ2.

Although we have shown that a hybrid algorithm performs well, we have introduced the
use of pheromone in an (in essence) non-pheromone SI mechanism. Moreover, application of
such a hybrid algorithm in embodied systems, such as a swarm of robots, may be problem-
atic. Therefore, in Chapter 5, we again focus on RQ3. We eliminate the use of pheromone and
introduce the use of landmarks to keep track of essential environmental information. These ad-
aptations result in Stigmergic Landmark Foraging (SLF). In this algorithm, landmark locations
and the information stored within are decided on through group effort. Again, efficiency seems
to benefit from this extension. However, computation effort increases.

In Chapter 6, we turn to RQ4: To what extent is a bee-inspired SI algorithm capable of
providing improved solutions to current state-of-the-art algorithms in complex, real-world
problems? We apply SLF in the setting of Mobile Ad-hoc Networks (MANETs) and dub the
resulting routing protocol Stigmergic Landmark Routing (SLR). We compare the routing per-
formance of SLR with the state-of-the-art algorithm AODV. The results show that SLR out-
performs AODV especially in networks consisting of up to 100 nodes. In such networks, node
density and inter-node connectivity limits the ability to route. In low-density and low inter-
connected networks, it pays to source-route. This is a routing mechanism where the sequence
of nodes taken by a routing packet is stored inside the memory of the routing packet. Such a mechanism relaxes infrastructure dependency. More precisely, a routing node does not need to keep track of routing tables to store routing information. With growing density and connectivity, node bandwidth limits the amount of packets that can be send for routing and storing routing information in the network becomes a necessity. SLR provides a natural way of managing such networks. Landmarks are detected based on the number of node links. In low-density networks, there are few node links and therefore, SLR performs source routing. With increasing node density, node links increase, landmarks are detected and gradually routing information is stored in the network environment.

In Chapter 7, we answer the problem statement on the basis of the results and the individual answers to the research questions. Bee-inspired algorithms clearly outperform ant-inspired algorithms in the task of foraging. Moreover, in the less abstract and real-world problem of routing in Mobile Ad-hoc Networks, the mechanism provides an improvement to the state-of-the-art routing algorithms. Especially in networks of up to 100 nodes. Direct communication and limited environment information in the form of stigmergic landmarks results in flexible, balanced SI algorithms. Therefore, we may conclude that bee-inspired SI algorithms provide a flexible and state-of-the-art mechanism for solving combinatorial optimization problems.
Samenvatting

De natuur biedt veel voorbeelden van efficiënte en goed presterende mechanismen om problemen op te lossen. Of we nu kijken naar foto synthese, gen activatie, of het foerageer gedrag van dieren, de natuur heeft ontzettend veel tijd gehad om zulke mechanismen te evolueren. Om deze reden gebruiken onderzoekers de natuur als inspiratie om problemen zoals het controleren van verkeer, het winnen van groene energie en het afwenden of elimineren van ziektes op te lossen. Bovendien zoeken onderzoekers naar wegen om de oplossingen te verbeteren.

In deze thesis concentreren we ons op combinatorische optimisatie problemen welke bestaan uit het vinden van een zo min mogelijk kostende oplossing voor een mathematisch probleem waar elke oplossing geassocieerd is met een numerieke prijs. Het bio-geïnspireerde mechanisme waar we op focussen heet Swarm Intelligence (SI) en is een eigenschap van een multi-agent system, hetzij natuurlijk hetzij kunstmatig, waarin de gedecentraliseerde, lokale samenwerking tussen individuen resulteert in globale, emergerende, zelf-organiserende structuren.

De meest bekende, in de natuur voorkomende, voorbeelden van SI zijn scholen vis en mier kolonies. Onderzoekers gebruiken deze natuurlijke gedragingen reeds als inspiratie voor de creatie van optimalisatie algoritmes. Natuurlijk zijn de twee voorgaande voorbeelden van SI niet de enige voorbeelden. SI kan bijvoorbeeld ook gevonden worden in honing bijen. Bijen kolonies zijn in staat om een verbluffende hoeveelheid problemen op te lossen die ze in de natuur tegenkomen. Biologen hebben gedurende de afgelopen jaren inzicht verschaft in hoe zo’n SI ontstaat. Dit wetende is het bijzonder te noemen dat in het veld van computer wetenschappen toch de meeste aandacht uitgaat naar klassieke SI voorbeelden zoals vis scholen en mier kolonies. Substantiële aandacht voor het gedrag van honing bijen, vanuit het standpunt van computer wetenschappen, ontbreekt.

In deze thesis beargumenteren we dat, door het gedrag van honingbij geïnspireerde, algoritmes goed presterende mechanismen zijn voor combinatorische optimisatie problemen. Om deze reden is de probleem stelling in deze thesis als volgt.

Probleemstelling: In welke mate kunnen honingbij-geïnspireerde SI algoritmes een flexibel, state-of-the-art mechanisme vormen voor het oplossen van combinatorische optimisatie problemen?

Om deze probleemstelling te beantwoorden, formuleren we vier onderzoeksvragen (OV).

OV 1: Wat zijn de hoofd principes van honingbij-geïnspireerde SI en hoe verschillen deze van andere sociale insecten SI?

OV 2: Hoe presteren de verschillende sociale insecten SIs bij een directe vergelijking?

OV 3: In welke mate kunnen we potentiële zwakke punten in honingbij-geïnspireerde SI elimineren?
OV 4: In welke mate is een honingbij-geïnspireerd algoritme in staat om betere oplossingen te bieden voor complexe, echte wereldproblemen in vergelijking met huidige state-of-the-art algoritmes?

Het onderzoek start in hoofdstuk 2 waar we SI plaatsen in het veld van Reinforcement Learning. Daarbuiten geven we een aantal biologische voorbeelden van SI en hiermee beantwoorden we OV 1: Wat zijn de hoofd principes van honingbij-geïnspireerde SI en hoe verschillen deze van ander sociale insecten SI? We identificeren vier eigenschappen: i) het onbewust zijn van globale structuren, ii) decentralisatie, iii) stigmergentie en iv) zelf-organisatie.

Hoofdstuk 3 laat zien hoe een basis honingbij-geïnspireerd SI algoritme geconstrueerd wordt. In essentie bestaat zo’n algoritme uit 2 delen, i) directe communicatie tussen individuen in de vorm van een (virtuele dans) en ii) een navigatiestrategie genaamd Path Integration. Het resulterende algoritme, genaamd ‘Bee System’ (BS), wordt in een directe vergelijking in de taak foerageren vergeleken met een ander sociaal-insect-geïnspireerd SI algoritme, namelijk Ant Colony Optimization (ACO). Dit, door het gedrag van mieren geïnspireerde, algoritme laat individuen met elkaar communiceren via pheromonen. De foerageren taak kan gezien worden als een abstractie van vele ‘echte’ wereldproblemen zoals het routen van netwerk pakketten en kortste pad problemen. De vergelijking beantwoordt OV 2: Hoe presteren de verschillende sociale-insecten SIs bij een directe vergelijking? In essentie geven de resultaten een vergelijking tussen een directe communicatie plus niet-pheromonen gebaseerde navigatiestrategie aan de ene kant en een indirecte communicatie plus pheromonen-gemmerde navigatiestrategie aan de andere kant. Honingbij-geïnspireerde SI (i.e., geen pheromonen) blijkt 3 keer zo efficient te zijn in het oplossen van de foerageren taak. Bovendien blijkt het mechanisme heel schaalbaar te zijn met de grootte van de taak. Mier-geïnspireerde SI (i.e., met pheromonen) blijkt vooral schaalbaar in zwerm grootte. Eerstgenoemde is echter te prefereren in een multi-agent system omdat het systeem meestal gebonden is aan een maximaal aantal agenten terwijl de taak heel groot kan zijn.

De resultaten van bovenstaande vergelijking onthullen ook een grote tekortkoming in BS. Namelijk, door het gebrek aan (heuristische) informatie in de omgeving is BS minder adaptief. Het mechanisme is niet in staat om lokaal te leren en daardoor hebben veranderingen in de omgeving negatieve uitwerking op de prestatie van BS. Om deze reden focussen we in hoofdstuk 4 op OV 3: In welke mate kunnen we potentiële zwakke punten in honingbij-geïnspireerde SI elimineren? We introduceren twee uitbreidingen op BS waarmee het algoritme in principe een hybride algoritme wordt. Het resulterende algoritme noemen we ‘Bee System with inhibition Pheromones’ (BSP). De twee extensies zijn i) wand volgen en ii) inhibitie pheromonen. De eerste vergroot de capaciteit van het algoritme om obstakels te ontwijken terwijl de laatste de zoekruimte verkleint en de aandacht van de agenten richt op interessante locaties. Beide extensies stellen BSP in staat om de adaptiviteit te vergroten en een minimale impact te hebben op de efficientie. Om precies te zijn, efficientie lijkt niet te lijden onder het gebruik van de extensies. Om dit aan te tonen vergelijken we BSP met zowel BS als een ACO algoritme. Hiermee beantwoorden we ondermeer ook OV 2.

Alhoewel we nu hebben laten zien dat een hybride algoritme goed kan presteren, hebben we hiervoor wel pheromonen moeten introduceren in een, in essentie, pheromonen-loos mechanisme. Bovendien, het gebruik van zo’n hybride algoritme in een belichaamd systeem, zoals
een zwerm robots, kan problematisch zijn. Daarom focussen we in hoofdstuk 5 wederom op OV 3. We elimineren het gebruik van pheromonen en introduceren het gebruik van landmarks om lokale omgevings informatie op te slaan. Deze aanpassingen resulteren in het algoritme ‘Stigmergic Landmark Foraging’ (SLF). In dit algoritme wordt de locatie van de landmarks en de informatie die erin opgeslagen wordt, bepaald door samenwerking tussen de individuen van de zwerm. Ook hier zien we weer dat de efficientie niet te lijden heeft onder deze extensie. Echter, computatietijd neemt wel toe.

Hoofdstuk 6 richt zich op OV 4: *In welke mate is een honingbij-geïnspireerd algoritme in staat om betere oplossingen te bieden op moeilijke, ‘echte’ wereldproblemen in vergelijkning met huidige state-of-the-art algoritmes?* We passen SLF toe op een routing probleem in Mobiele Ad-hoc Netwerken (MANETs) en geven het resulterende routing protocol de naam ‘Stigmergic Landmark Routing’ (SLR). We vergelijken de prestatie van dit algoritme met de huidige marktstandaard voor routing in zulke netwerken, namelijk AODV. De resultaten laten zien dat SLR aanzienlijk beter presteert dan AODV in netwerken tot en met 100 knooppunten.

In zulke netwerken limiteren de lage dichtheid en lage connectiviteit van de knooppunten de mogelijkheden om routes te vinden en is het voordelig om te source routes. Deze wijze van routing slaat een volledige route op in de routing pakketten. Zo’n strategie vermindert de afhankelijkheid van de infrastructuur. Om precies te zijn, een knooppunt hoeft geen rekening te houden met routing tabellen om routeinformatie op te slaan. Met groeiende dichtheid en connectiviteit limiteert de bandbreedte per knooppunt het aantal en de grootte van de pakketten die verzonden kunnen worden voor routes en wordt het opslaan van routing informatie in knooppunten een verplichting. SLR biedt een natuurlijke overgang tussen deze netwerk toestanden. Landmarks worden gedetecteerd op basis van knooppuntverbindingen. In lage dichtheid netwerken zijn er weinig verbindingen en daardoor zal SLR source routing toepassen. Met toenemende dichtheid neemt het aantal verbindingen toe en zullen landmarks ontdekt worden en zal graduell routing informatie in het netwerk opgeslagen worden.

Als laatste geven we in hoofdstuk 7 antwoord op de probleemstelling op basis van de resultaten en antwoorden op de OV-s uit de voorgaande hoofdstukken. Honingbij-geïnspireerde algoritmes presteren duidelijk beter dan mier-geïnspireerde algoritmes in de foerageraar taak. Bovendien blijkt dat in een minder abstract en ‘echt’ wereld probleem zoals routeren in MANETs, het algoritme een verbetering oplevert in vergelijking met een huidige standaard. Dit geldt vooral in netwerken tot en met 100 knooppunten. Directe communicatie en gelimiteerde omgevings informatie in de vorm van stigmergerende landmarks resulteert in flexibele, uitgebalanceerde SI algoritmes. Daarom mogen we concluderen dat honingbij-geïnspireerde SI algoritmes een flexibel en state-of-the-art mechanisme vormen voor het oplossen van combinatorische optimalisatie problemen.
Nyree Paul-Peter Maurice Lemmens was born in Geleen, The Netherlands, on November 23, 1979. He graduated from St.-Jans College in Hoensbroek, The Netherlands in 1997. He went on to study Electrical Engineering at Hogeschool Zuyd in Heerlen, The Netherlands, where he received his engineering (ing.) degree in 2002. After his graduation, he took the position of junior software engineer at Laundry Computer Technics where he worked successfully for 2 years. During these 2 years he grew an increasing interest for Artificial Intelligence. In 2004 he decided to quit his job and to start studying Knowledge Engineering at Maastricht University. He received his Bachelor’s degree in Knowledge Engineering in 2005 and his Master’s degree in Artificial Intelligence in 2006. For his Bachelors degree, he performed research into the field of games, more precisely, the classic game of Abalone. He received his Masters degree on the topic of Swarm Intelligence. The resulting thesis received the ‘Leo Coolen Award 2006’ for appropriateness and excellency of the graduation thesis.

After his graduation, in 2006, Nyree took a position as Ph.D. researcher at the Vrije Universiteit Brussel, Belgium, under the supervision of Prof. A. Nowé and under daily supervision of dr. K. Tuyls. In 2007, he returned to Maastricht University to continue his research in Swarm Intelligence. Prof. A. Nowé and dr. K. Tuyls became copromotores while Prof. dr. G. Weiss became his first promotor.

His research has been funded by Fonds Wetenschappelijk Onderzoek (FWO), Integrative Collaborative Information Systems (ICIS), and Transnationale Universiteit Limburg (tUL) respectively. The research was presented at, and published in the proceedings of, numerous (international) conferences and workshops, including the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), the European Conference on Complex Systems (ECCS), and the International Conference on Genetic and Evolutionary Computation (GECCO).

Besides performing research, he was involved in lecturing courses, such as Theoretical Computer Science, Logic, Datastructures & Algorithms, and Introduction to Computer Science.

During his spare time, Nyree likes to do sports (e.g., badminton, rowing, and cycling). He also likes to travel abroad whenever a nice opportunity presents itself. Furthermore he also likes to spend time with family and friends, write blogs, read fantasy books, watch sci-fi tv-shows, attend concerts, play board/card/computer games, keep himself informed in science, and is always on the lookout for learning new stuff.
The scientific work performed during the author’s Ph.D. research resulted in the following publications.

**Conference and workshop proceedings**


2009

1 Rasa Jurgelenaite (RUN) Symmetric Causal Independence Models
2 Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques
3 Hans Stol (UvT) A Framework for Evidence-based Policy Making Using IT
4 Josephine Nabukenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering
5 Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
6 Muhammad Subianto (UU) Understanding Classification
7 Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion
8 Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments
9 Benjamin Kanagwa (RUN) Design, Discovery and Construction of Service-oriented Systems
10 Jan Wielemaker (UvA) Logic programming for knowledge-intensive interactive applications
11 Alexander Boer (UvA) Legal Theory, Sources of Law & the Semantic Web
12 Peter Massuthe (TUE) Humboldt-Universitaet Berlin Operating Guidelines for Services
13 Steven de Jong (UM) Fairness in Multi-Agent Systems
14 Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
15 Rinke Hoekstra (UvA) Ontology Representation - Design Patterns and Ontologies that Make Sense
16 Fritz Reul (UvT) New Architectures in Computer Chess
17 Laurens van der Maaten (UvT) Feature Extraction from Visual Data
18 Fabian Groffen (CWI) Armada, An Evolving Database System
19 Valentin Robu (CWI) Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
20 Bob van der Vecht (UU) Adjustable Autonomy: Controlling Influences on Decision Making

21 Stijn Vanderlooy (UM) Ranking and Reliable Classification
22 Pavel Serdyukov (UT) Search For Expertise: Going beyond direct evidence
23 Peter Hofgesang (VU) Modelling Web Usage in a Changing Environment
24 Annerieke Heuvelink (VU) Cognitive Models for Training Simulations
25 Alex van Ballegooij (CWI) RAM: Array Database Management through Relational Mapping
26 Fernando Koch (UU) An Agent-Based Model for the Development of Intelligent Mobile Services
27 Christian Glahn (OU) Contextual Support of social Engagement and Reflection on the Web
28 Sander Evers (UT) Sensor Data Management with Probabilistic Models
29 Stanislav Pokraev (UT) Model-Driven Semantic Integration of Service-Oriented Applications
30 Marcin Zukowski (CWI) Balancing vectorized query execution with bandwidth-optimized storage
31 Sofiya Katrenko (UvA) A Closer Look at Learning Relations from Text
32 Rik Farenhorst and Remco de Boer (VU) Architectural Knowledge Management: Supporting Architects and Auditors
33 Khiet Truong (UT) How Does Real Affect Affect Affect Recognition In Speech?
34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach
35 Wouter Koelewijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatieuitwisseling
36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks
37 Hendrik Drachsler (OUN) Navigation Support for Learners in Informal Learning Networks
38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) Service Substitution – A Behavioral Approach Based on Petri Nets
40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*

41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*

42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*

43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Environments*

44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*

45 Jilles Vreeken (UU) *Making Pattern Mining Useful*

46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

**2010**

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multi-Display Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

7 Wim Fikkert (UT) *Gesture interaction at a Distance*

8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*

9 Hugo Kielman (UL) *A Politieele gegevensverwerking en Privacy. Naar een effectieve waarborging*

10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*

11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*

12 Susan van den Braak (UU) *Sensemaking software for crime analysis*

13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*

14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*

15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*

16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*

17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*

18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*

19 Henriette Cramer (UvA) *People’s Responses to Autonomous and Adaptive Systems*

20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Improves Agency and Authorship of Emergent Narrative*

21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*

22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*

23 Bas Steunebrink (UU) *The Logical Structure of Emotions*

24 Dmytro Tykhonov () *Designing Generic and Efficient Negotiation Strategies*

25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*

26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*

27 Marten Voulon (UL) *Automatisch contracteren*

28 Arne Koopman (UU) *Characteristic Relational Patterns*

29 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*

30 Marieke van Erp (UvT) *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval*

31 Victor de Boer (UvA) *Ontology Enrichment from Heterogeneous Sources on the Web*

32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*

33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*

34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*

35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*

36 Jose Janssen (OU) *Paving the Way for Lifelong Learning: Facilitating competence development through a learning path specification*

37 Niels Lohmann (TUE) *Correctness of services and their composition*
38 Dirk Fahland (TUE) From Scenarios to components
39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents
40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web
41 Guillaume Chaslot (UM) Monte-Carlo Tree Search
42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach
43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies
44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services
46 Vincent Pijpers (VU) e3alignment: Exploring Inter-Organizational Business-ICT Alignment
47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples
48 Jahn-Takeshi Saito (UM) Solving difficult game positions
49 Bouke Huurnink (UvA) Search in Audiovisual Broadcast Archives
50 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources
51 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
52 Edgar Meijs (UVA) Combining Concepts and Language Models for Information Access

2011
1 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2 Nick Timmemeier (UU) Work flows in Life Science
3 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems
4 Hado van Hasselt (UU) Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference learning algorithms
5 Base van der Raadt (VU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
6 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage
7 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction
8 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues
9 Tim de Jong (OU) Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT) Cloud Content Contention
11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM) Selective Search in Games of Different Complexity
17 Jiyin He (UvA) Exploring Topic Structure: Coherence, Diversity and Relatedness
18 Mark Ponsen (UM) Strategic Decision-Making in complex games
19 Ellen Rusman (OU) The Mind’s Eye on Personal Profiles
20 Qing Gu (UU) Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UvA) System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UvA) Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Quain Jaffry (VU) Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UvA) Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE) Discrimination-aware Classification

30 Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions

31 Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

32 Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science

33 Tom van der Weide (UU) Arguing to Motivate Decisions

34 Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

35 Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training

36 Erik van der Spek (UU) Experiments in serious game design: a cognitive approach

37 Adriana Birlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

38 Nyree Lemmens (UM) Bee-inspired Distributed Optimization