Hugo Hernández Pibernat

Thesis Director: Christian Blum

Swarm Intelligence Techniques for Optimization and Management Tasks in Sensor Networks

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
April · 2012

Acknowledgements

This thesis is the result of my research activities at the ALBCOM research group of the LSI department at UPC. Their institutional support, as well as the help and advice of some of their members has resulted inestimable for my work and personal growth. In particular, two people have been specially near during all this time.

First, Christian Blum, who has been my tutor since we started to work on my Master Thesis. There are not enough words to explain how deeply I appreciate his confidence and help during this period. Therefore I will not even try. For those who do not know him I just want to say that he is someone you are proud to say that you have worked with. I have been his first PhD Student of many to come and I am glad to see that he has mostly enjoyed working with me. I have many things to be thankful for, but I specially appreciate his sincerity. Smart, intelligent and, most of the time, brilliant and friendly, that is Christian in a few words.

Second, Maria Jose Serna, who has helped me from the very first day in the department, even before I met Christian. She offered me the opportunity to collaborate with her, offered advice on both academic and scientific topics, helped me find my path when I was lost and, overall, always talked to me with a big smile on her face. Again, all words would not be enough to describe her, however there is something particular about her that impresses me one day and another. She always has the right question. It doesn't matter if you talk to her informally or on a seminar, for five minutes or an hour, her questions always put you on track for many hours of work and, hopefully, a new achievement at the end. I have been more than lucky of having her at my side every time I have needed her. In particular, the trip we made together to Braunschweig, which included many conversations in planes, trains and restaurants, was one of the main reasons that have allowed me to complete this work with success.

Many more people have contributed to this work. In the following I mention these people and summarize some of their contributions, which have helped me advance on my research and increase the quality and completeness of this thesis. Although not mentioned in the following paragraphs, many more people have contributed to this work. Each of them knows how much I appreciate their help and advice.

I would like to thank Martin Middendorf, Alexander Scheidler, and Kai Ramsch for their contribution to an earlier paper about ant based duty-cycling (see [97]). Moreover, I also thank Martin Middendorf for introducing me to the papers on the calling behavior of Japanese tree frogs and for hosting me for two weeks at the University of Leipzig.

Tobias Baumgartner was essential in the implementation of the ANTCYCLE protocol for Shawn and his contribution to the paper [96]. I also thank Tobias for accepting my invitation to visit our research group and later inviting me to visit the University of Braunschweig, and translate with me the ANTCYCLE into a real implementation. To see the algorithm working in *live* mode was one of the peaks of success of this work.

The help from Enrico Malaguti, who kindly provided experimental results of the centralized algorithm for graph coloring from [137] for a huge set of problem instances, was essential for finishing one of the parts of this thesis in time.

Finally, I would like to thank photographers Jordi Pérez i Grimal (Figure 3.2), John Walker (Figure 3.3), Lee Langstaf (Figure 3.4) and Masaki Ikeda (Figure 5.2) for providing me with rights to reproduce their photographs in the scope of this work. It must be remarked that all originals have been modified to improve printed quality in grayscale.

In addition to the many persons that must be acknowledged for this work, it is necessary to remark the huge institutional support received in order to successfully finish this thesis. I was provided with funds for my living expenses, traveling, computer and other needs. More in detail, I acknowledge support from the following projects and institutions: the ALBCOM research group, the Departament de Llenguatges i Sistemes Informàtics at Universitat Politècnica de Catalunya, the *Comissionat per a Universitats i Recerca del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya*, the *European Social Fund*, the EU project FRONTS (FP7-ICT-2007-1) funded by the European Commission, the EU FET Proactive Initiative on *Pervasive Adaptation*, and the Spanish government project FORMALISM (TIN2007-66523).

I also want to thank for the help, love and affection provided by my close family and friends. Specially to my mother, Eva, and my great friend, Carles, who always take care of me. In addition, I would like to dedicate a deep thought to the memory of Maria Abad Utesà (1931-2010), who always welcomed me warmly and happily.

About Georgina, to whom I dedicate this work, there is not much I can write down in a few sentences. Even if I was a proficient writer, I would probably not have enough words of gratitude, nor enough words of admiration. It is not even a beginning, but thanks for being at my side. I hope to have enough time to show you my gratitude during the years to come. It is difficult to precisely say how much of this thesis is yours, but probably more than what you imagine.

Other family, friends and colleagues who have collaborated in the development of this thesis are:

David Ameller	Patricia Antón	Iván Balaña
Borja de Balle	Judit Cardona	Jordi Coll
Oriol Collell	Daniel Colomer	Iván Couto
Jordi Delgado	Josep Diaz	Rita García
Margarita Garrido	Marquina Hellin	Ana Hernández
Carlos Hernández	Helena Hernández	Nuria Hernández
Rafael Hernández	Ana Ibañez	Rogelio Jiménez
Mercè Juan	Joan Martín	Maria José Merelo
Glòria Mosquera	José Mosquera	Klaus Langohr
Marc Oriol	Jordi Petit	Montserrat Renalías
María Rodríguez	Salvador Roura	Núria Sánchez
Francesc Tuca	Oriol Tuca	Margarita Vancells

Contents

I Preliminaries **1**

1 Introduction **3**
- 1.1 Document Organization . . . 5
- 1.2 Scientific Contributions . . . 8
  - 1.2.1 Self-Synchronized Duty-Cycling in Sensor Networks . . . 8
  - 1.2.2 Distributed Graph Coloring . . . 10
  - 1.2.3 Minimum Energy Broadcast with Realistic Antennas . . . 11

2 Wireless Sensor Networks **13**
- 2.1 Sensing . . . 16
- 2.2 Communication and routing . . . 21
- 2.3 Application Categories . . . 24
- 2.4 Common Design Problems . . . 26

3 Swarm Intelligence **29**
- 3.1 Collective Behaviour of Social Animals . . . 34
- 3.2 Popular Swarm Intelligence Techniques . . . 35
  - 3.2.1 Ant Colony Optimization . . . 35
  - 3.2.2 Particle Swarm Optimization . . . 42
  - 3.2.3 Other relevant examples of SI systems . . . 46

II Management techniques for WSN **49**

4 Self-Synchronized Duty-Cycling **51**
- 4.1 Simulating Ants' Self-Synchronization Behavior . . . 54
  - 4.1.1 Fluid Neural Network Model . . . 54
  - 4.1.2 Experimental Setup . . . 55
  - 4.1.3 Study Concerning Gain Parameter g . . . 57
  - 4.1.4 Study Concerning the Probability of Spontaneous Activation (p^a) . . . 58
  - 4.1.5 Study Concerning Neighborhoods of Different Sizes . . . 62
- 4.2 Adaptation to Sensor Networks . . . 70
  - 4.2.1 System Components . . . 70
  - 4.2.2 Comparison to the FNN Model . . . 72
- 4.3 Adaptation to Changing Energy Availabilities . . . 73
  - 4.3.1 Initial Experiments with ANTCYCLE . . . 74
- 4.4 Further experiments . . . 80

4.4.1 Robustness 80
4.4.2 Application Scenario: Environmental Monitoring 80
4.4.3 Scaling 81
4.4.4 The Case of Static Networks 83
4.4.5 A more Flexible Model for the Light Source 84
4.4.6 Comparison with the Mechanism by Vigorito 84
4.5 Moving to Real Sensor Networks 94
4.5.1 A Duty-Cycling Protocol 94
4.5.2 Experimental Evaluation 96
4.5.3 Additional Realistic Experiments with the Shawn Simulator 97
4.6 Conclusions and Future Work 105

5 Distributed Graph Coloring 107
5.1 Modeling the Calling Behavior of Japanese Tree Frogs 113
5.2 Implementation in Sensor Networks 116
5.2.1 Model Extensions 118
5.2.2 Experiments on the desynchronization of sensor networks 119
5.3 FrogSim: An Algorithm for Distributed Graph Coloring 126
5.3.1 Phase I of FrogSim 127
5.3.2 Phase II of FrogSim 128
5.3.3 Use of the Overlay Tree Structure 130
5.3.3.1 Identifying and Storing the Best Coloring 130
5.3.3.2 Detection of Convergence in Phase I 131
5.4 Experimental Results 132
5.4.1 Resource Consumption 133
5.4.2 Results for Random Geometric Graphs 133
5.4.3 Results for DIMACS Graphs 134
5.4.4 Results for Grid Topologies 136
5.4.5 Results for Small Instances from [124] 141
5.4.6 Summary of Results 141
5.5 Conclusions and Future Work 144

6 MEBRA 147
6.1 Minimum Energy Broadcasting With Realistic Antennas 149
6.2 BIP-Algorithm for the MEBRA Problem 152
6.3 The Ant Colony Optimization Algorithm 153
6.4 Experimental Evaluation 156
6.5 A distributed ACO for the MEBRA 169
6.5.1 Modification of BIP's Greedy Function 169
6.5.2 The Distributed Ant Colony Optimization Algorithm 170
6.5.2.1 Construction of Broadcast Trees 173
6.5.2.2 Updating the Pheromone Values 177
6.5.2.3 Complexity Issues 183
6.5.3 Experimental Results 183
6.6 Conclusions 192

III Conclusions **195**

7 Conclusions and further work **197**

Bibliography **201**

List of Algorithms

1 ACO metaheuristic . . . 38
2 PSO algorithm for minimization . . . 44
3 Fluid Neural Network proposed in [51]. . . . 55
4 Event of sensor node i at time t . . . 72
5 Event of sensor node i with energy harvesting capabilities at time t . . . 75
6 Duty-cycling event of a sensor node i . . . 95
7 Sensor event of node i in the desynchronization algorithm . . . 116
8 Sensor event of node i . . . 127
9 The BIP algorithm for the MEBRA problem . . . 153
10 ACO for the MEBRA problem . . . 154
11 Variable neighborhood descent (VND) . . . 155
12 Task of master node s in DistACO for the MEBRA problem . . . 172
13 Protocol for distributed solution construction . . . 174
14 Protocol for distributed pheromone update . . . 179

List of Tables

2.1 Internet layered model 15
2.2 Commonly sensed measures in WSNs. 20
2.3 Sensor node manufacturers 26

3.1 A selection of pheromone update systems 41

4.1 Reference parameter setting for the FNN model. 56
4.2 System behaviour as a function of g (in numerical form). 59
4.3 System behaviour as a function of p^a (in numerical form). 60
4.4 System behaviour as a function of the neighborhood size $|N_i|$ (in numerical form). 65
4.5 Initial parameter settings for ANTCYCLE. 76
4.6 Behaviour of ANTCYCLE with the parameter settings from Table 4.7 as a function of $r_{\min}$ (in numerical form). 77
4.7 Parameter settings of ANTCYCLE for environmental monitoring. 81
4.8 Behaviour of ANTCYCLE with the parameter settings from Table 4.7 as a function of $p^a_{\min}$ (in numerical form). 82
4.9 Parameter values for networks of different sizes. 83
4.10 Comparison of different duty-cycling mechanisms 86
4.11 Energy parameters. 96
4.12 Distribution of the energy consumption in the duty-cycling protocol 97

5.1 Success rates (in terms of the number of successful applications out of 100) calculated on the basis of an error margin of 5%. 122
5.2 Success rates (in terms of the number of successful applications out of 100) calculated on the basis of an error margin of 15%. 123
5.3 Average distance to the optimal distribution of the theta values 124
5.4 Average number of communication rounds executed before reaching the stopping condition. 125
5.5 Results for random geometric graphs. 135
5.6 Results for the first set of instances from the DIMACS challenge. 137
5.7 Results for the second set of instances from the DIMACS challenge. 138
5.8 Results for grid (respectively, torus) topologies. 140
5.9 Results for the instances introduced in [124]. 142

6.1 The schedule used for values κ_{ib}, κ_{rb} and κ_{bs} depending on cf (the convergence factor) and the Boolean control variable `bs_update`. 156
6.2 Summary of the ACO parameters. 156

6.3 Results for the 30 instances with 50 nodes when using antennas with seven transmission power levels. 165
6.4 Results for the 30 instances with 50 nodes when using antennas with 201 transmission power levels. 166
6.5 Results for the 30 instances with 100 nodes when using antennas with seven transmission power levels. 167
6.6 Results for the 30 instances with 100 nodes when using antennas with 201 transmission power levels. 168
6.7 Summary of the messages handled by a sensor node id for the construction of a solution with identifier sol_id. 178
6.8 The schedule used for values κ_{ib}, κ_{rb} and κ_{bs} depending on $\mathsf{conv\_fact}_{m^{\mathsf{req}}}$ (the convergence factor) and the Boolean control variable bs_update. 181
6.9 Summary of the messages handled by a sensor node id for the pheromone update. 183
6.10 Results for the 30 instances with 50 nodes and antennas with seven transmission power levels. 187
6.11 Results for the 30 instances with 50 nodes and antennas with 201 transmission power levels. 188
6.12 Results for the 30 instances with 100 nodes and antennas with seven transmission power levels.. 189
6.13 Results for the 30 instances with 100 nodes and antennas with 201 transmission power levels. 190

List of Figures

1.1 Side-by-side image of three mobile devices. A Sun Microsystems SunSpot sensor node, a modern Nokia cell phone and an Apple iPod Touch. 4
1.2 Coalesenses iSense sensor node. 5

2.1 Photos of the Canopee project endorsed by Sun Microsystems, India. 14
2.2 The OSI network model divides network communications in seven different levels (layers). 15
2.3 High level architecture of a sensor node. 16
2.4 Basic working of a transducer. 16
2.5 Ethernet message frame. 21
2.6 Working of the basic network handshaking protocol. 22
2.7 A network of ten nodes connected as a (a) flat architecture and (b) a hierarchical architecture. 24

3.1 Evolution of the number of components of microprocessors over time. This figure is published under licensed Creative Commons Attribution Share-Alike 3.0. Both the original file and the data used for the image are available from [198]. 30
3.2 School of barracudas (*Sphyraenidae Sphyraena*). Photograph by Jordi Pérez i Grimal. 32
3.3 Migrating ducks flying in a V-shaped formation. Photograph by John Walker. . 33
3.4 Bees taking care of larvae and eggs in their honeycomb. Photograph by Lee Langstaff. 34
3.5 Experimental setting that shows the shortest path finding capability of ant colonies. 36
3.6 Example of the solution construction for a TSP problem consisting of 4 cities. Figure reproduced from [18]. 40
3.7 Example of a PSO particle position update. 43

4.1 The Moore neighborhood of size 1 that is used in the FNN model. 55
4.2 Evolution of the mean system activity ($A(t)$) of the FNN over time. 56
4.3 System behavior for selected values of g 61
4.4 System behavior as a function of g 62
4.5 System behavior for selected values of p^a 63
4.6 System behavior as a function of p^a 64
4.7 Average system activities displayed by means of gray levels for different combinations of values for g and p^a. 66

4.8 Considered set of neighborhoods. Graphics (a) to (f) show neighborhoods N_1 to N_6. 67
4.9 System behavior for selected values of $|N_i|$ 68
4.10 System behavior as a function of $|N_i|$. 69
4.11 Example of the neighbourhood $N(i, t)$ of a sensor i. 71
4.12 Evolution of the mean system activity ($A(t)$) of a sensor network with 120 sensors for the fourth day of simulation. 72
4.13 Evolution of the mean system activity of ANTCYCLE with the parameters from Table 4.5. 76
4.14 ANTCYCLE behavior for selected values of $r_{\min}$. 78
4.15 ANTCYCLE behavior as a function of $r_{\min}$. 79
4.16 Graphics that show the robustness of ANTCYCLE. 87
4.17 Evolution of the mean system activity of ANTCYCLE with the parameters from Table 4.7. 88
4.18 ANTCYCLE behavior for selected values of $p^a_{\min}$. 89
4.19 ANTCYCLE behavior as a function of $p^a_{\min}$. 90
4.20 Evolution of the mean system activity of ANTCYCLE when different network sizes are concerned. 91
4.21 Behavior of ANTCYCLE for static networks. 91
4.22 Three examples for the distribution of the sun light intensities. 92
4.23 Evolution of the mean system activity of ANTCYCLE when using the stochastic function for the sun light intensities from Equation 4.16. 92
4.24 Comparison of duty-cycling algorithms 93
4.25 Time management between the duty-cycling mechanism and possible user applications. 94
4.26 Different communication topologies in a network with 120 randomly located nodes when different transmission power levels are considered. 100
4.27 Evolution of the mean system activity of ANTCYCLE when run on iSense sensor nodes simulated by the Shawn sensor network simulator 101
4.28 ANTCYCLE behavior for selected values of p_{loss}. 102
4.29 Behavior of the duty-cycling mechanism under varying conditions 103
4.30 Behavior of the duty-cycling mechanism for monitoring applications 104
4.31 Behavior of ANTCYCLE when mobile networks are concerned. 104

5.1 Optimal and suboptimal colorings of a simple graph topology composed of three inner triangles and one outer triangle. 108
5.2 Photograph of a Japanese tree frog (*Hyla japonica*). This photograph was released into the public domain by Masaki Ikeda. No licenses apply. The image is also available from [198]. 109
5.3 Graphical illustration of the working of a system of two coupled oscillators. . . 113
5.4 Two examples of graph topologies that may cause problems for the desynchronization as performed by the model proposed in [3]. 115
5.5 Sensor network topologies used in this work. 119
5.6 Experimental results concerning topology *spare.gph*. 121
5.7 Example of the working of phase II of FROGSIM. 129

5.8 Example of the evolution over time of the quality of the colorings generated by FrogSimfor graphs DSJC1000.9.col and flat300_26_0.col of the DIMACS challenge. . . . 131
5.9 Optimal coloring of a grid topology 136
5.10 Example of the distributed incremental process of coloring a grid of 4×4 nodes. 139
5.11 Summary of results for the different groups of graph topologies. . . . 143

6.1 Energy benefits by the usage of relay nodes. (a) shows a configuration one-to-all, and (b) shows an energy-aware configuration using intermediate nodes. . . . 148
6.2 *Wireless Multicast Advantage* of omnidirectional antennas. . . . 151
6.3 An example for the MEBRA problem and different solutions. . . . 152
6.4 Graphical presentation of the computation times of the BIP+VND algorithm for values of $r_{\max} \in [1,|V|\text{-}2]$ 158
6.5 Graphical presentation of the results. . . . 160
6.6 Summarized algorithm behaviour for different problem versions. . . . 161
6.7 Improvement of the ACO algorithm solutions over the basic BIP algorithm when considering different number of transmission levels. . . . 162
6.8 Solutions for problem instance p50.25 using seven transmission power levels. . . . 163
6.9 Solutions for problem instance p50.25 using 201 transmission power levels. . . . 164
6.10 An example for the working of BIP in the case of a simple sensor network with seven nodes. . . . 171
6.11 An example for the working of BIP^+ in the case of a simple sensor network with seven nodes. . . . 171
6.12 Graphical illustration of the pheromone values related to the transmission power levels. . . . 180
6.13 Solutions provided by BIP, BIP^+, DistACO and the centralized ACO algorithm for instance *p50.26*. The scenario in which nodes are equipped with antennas of 7 different transmission power levels is considered. . . . 191

Part I

Preliminaries

Chapter 1

Introduction

Technological advances of modern society have focused many private and public funds on the development of tiny mobile communication devices as, for example, cell phones or tablet computers. These devices allow their users to communicate from anywhere without requiring much additional infrastructure. The number of services and applications that can be used continuously increases. In fact, most modern devices are small embedded computers whose only limitation is the user interface as, for example, the screen which is restricted by the dimensions of the device. Moreover, smartphones and tablets enable users to create ad-hoc networks for sharing information or running distributed applications.

The advances in mobile technology during the last decade have increased the interest of the research community in the capabilities of wireless networks. This rising interest has encouraged the creation of specific devices for research projects and institutions. While society asks for better graphics, improved interfaces, larger screens, less weight and similar features, researchers dream about technology with a very low energy consumption, radios which are able to communicate to farther distances, higher memory capacity or better methods for energy harvesting from the environment, allowing to obtain power from the sun light or the ocean tides. Figure 1.1 shows a Sun Microsystems SunSpot, side-by-side with a modern Nokia cell phone and an Apple iPod Touch. Notice how these devices have similar sizes although being characterized by quite different uses. For example, the interface provided by the SunSpot is limited to some LEDs while the iPod has a sensitive screen covering the whole device.

The huge efforts aimed at obtaining small devices which suit the requirements of the research community have resulted in a new industrial notation for this technology. In general, any network of tiny artifacts with communication capabilities can be seen as a so-called *wireless sensor network (WSN)*. However, sensor networks are usually composed of nodes with limited computation capabilities, due to both the processor and the available memory. Also energy may be a scarce resource. Often nodes are equipped with batteries to allow their usage in regions where no power infrastructure is available, such as deserts, seas or forests. To cope with this constraint, in addition to new energy-aware algorithms and communication protocols, many sensor nodes are prepared to be attached to energy harvesting devices such as solar panels. Moreover, due to the huge amount of information that may possibly be collected in a WSN, they are generally managed by a remote computer. Consequently, sensor nodes have a rougher appearance than devices aimed for the general public and prescind from a screen to reduce costs and save energy consumption. Figure 1.2 shows a Coalesenses iSense sensor node.

Figure 1.1: Side-by-side image of three mobile devices. A Sun Microsystems SunSpot sensor node, a modern Nokia cell phone and an Apple iPod Touch.

Nowadays, with the price reduction of sensor nodes, networks are becoming every time larger and management techniques based on global control may become inapplicable. The roots of the problem seem to be in the nature of the classical engineering approach to problem solving. In general, engineering methods model complex systems by means of many variables. Next, engineers decide the behavior over time of all these variables. Therefore, management techniques aim at controlling all system parameters and elements such that the desired system behavior arises. Unfortunately, this top-down procedure requires developers to control even the smallest of the system entities. As a result, system exceptions and even minor programming errors may result in undesired system behavior. Moreover, the continuous increases in the size and complexity of systems is pushing this traditional engineering paradigm to its limits. Although the size of tractable problems has been commonly extended by the introduction of new and faster hardware, this solution does not evolve at the same speed as developers expectations do. Moreover, the exponential number of exceptions that may arise can no longer be tackled individually to guarantee the expected system behavior. Modern and fresh development techniques are necessary in order to allow an increase of one or more orders of magnitude of the considered systems. The new goals require distributed programming paradigms, easily parallelizable algorithms and other techniques that simplify system management [188].

Swarm intelligence (SI) is a modern artificial intelligence discipline concerned with the design of multi-agent systems. SI is inspired by the collective behavior of social insects such as wasps, bees or ants. As later explained, swarm intelligence techniques have already been successfully applied to different research fields such as optimization and robotics. Although individual insects are small and quite limited, they are able to achieve huge and/or complex tasks by means of cooperation. SI systems are characterized by their robustness and scalability, which usually appear as a result of the distributed conception of the algorithm and the self-organization principle that governs the whole system.

The global behavior of self-organizing systems is not explicitly defined. Instead, the behavior of entities and their responses to local events are detailed and global behavior arises as a result of the combination of these local interactions. Individuals composing a self-organized system are generally not aware of any global information. Moreover, the amount of possible states of the individual is usually limited. Therefore, even a group of simple individuals can achieve complex tasks by means of self-organized cooperation.

Examples of self-organization in biology include the synchronized flashing of fireflies, the mound construction of termites, and the synchronized resting phases or the shortest-path find-

Figure 1.2: Coalesenses iSense sensor node.

ing behavior of ant colonies. The number of research fields discovering self-organizing processes as valuable alternatives for the management and control of complex or large-scale systems is continuously growing. A prominent example is research on wireless ad-hoc networks. Sensor networks are usually composed of a relatively large number of possibly heterogenous nodes that may be aimed at monitoring large areas for extended periods of time. The broad spectrum of suppliers and manufacturers, as well as the lack of a globally accepted standard, result in management being an arduous task. Furthermore, quite a few sensor network applications require deployment in remote areas such as forests, deserts or, for example, high ceilings of city buildings, which complicates a detailed and global management of all sensor nodes even further. In recent years, many researchers have pointed out that the answer to these problems may be in implementing automated control techniques such as self-organized algorithms. Self-organizing systems are usually more robust, scalable and flexible to subtle changes than those based on detailed algorithmic control.

In this thesis, we explore the development of swarm intelligence methods for the management of sensor networks. We hope that the use of the distributed nature of insect and animal societies can result in an easier management of each individual network node. Furthermore, our goal is to model, obtain and understand the desired behavior of the whole network as a consequence of the interactions of neighboring nodes.

In the rest of this chapter we focus on outlining the document organization and reviewing the publications related to each topic tackled in this thesis.

1.1 Document Organization

As mentioned before, this thesis focuses on the development of mechanisms for optimization and management tasks in sensor networks. All the contributions are inspired by the natural behavior of different species. In addition, they possess the main characteristics of SI systems. The thesis is divided in three parts: Preliminaries, Management techniques for WSNs, and Conclusions.

The first part comprises three chapters. Chapter 1 constitutes this introduction. In Chapter 2 we present a thorough explanation of the main characteristics of wireless sensor nodes and networks and an introduction to the current trends in WSN research. We also include a categorization of the most important applications and a list of issues which concern WSN development. Chapter 3 covers an overview of the swarm intelligence field. We review the origins of swarm intelligence and successful applications for networked sensing.

The second part of the thesis covers the main contributions. We introduce novel solutions and algorithms for three different WSN-related problems. Although each one responds to one of the many open issues in WSN development and contributes in different ways to the development of WSNs, the three solutions share the primary goal of extending network lifetime by means of including energy-aware protocols in the working of a WSN. All the novel solutions share a distributed conception of management. Nodes are programmed individually and reduce their working to communications and tasks in their own local neighborhood. These self-organized solutions avoid the need of a controlling agent that communicates with all nodes for defining their behavior. In the following we briefly describe the problems tackled and the results achieved in the three chapters of part two.

Self-synchronized Adaptive Duty-cycling

Sensor networks are commonly proposed for open-air regions like forests or seas where a reliable energy source may not be available. Consequently, energy is a scarce resource in the context of most applications. As sensor nodes are usually unsophisticated devices with simple and cheap hardware, applications that are conscious of their energy consumption may dramatically increase network lifetime. Duty-cycling is a general technique that schedules the activity of nodes over time. Consider, for instance, a WSN developed in the distribution facilities of a shipping company. There are probably time frames with a lot of activity, and others with hardly any activity. The scope of adaptive duty-cycling is to adjust the network activity to different situations as, for example, the work routine of a factory or the amount of users of a system.

In Chapter 4 we consider the case of WSNs relying exclusively on their batteries due to their location in remote or unaccessible locations. Nodes may be attached to energy harvesting devices such as solar cells. For this scenario, we propose a self-synchronized duty-cycling mechanism which is able to regulate activity depending on the amount of energy remaining in the batteries. Our solution is inspired by the resting phases of ants, which amount to as much as 65% of the time for some species. More interestingly, ants' working and resting phases are synchronized among the members of the colony. The energy generated by solar cells allows nodes to dynamically increase their percentage of activity. Moreover, this method is a self-synchronized mechanism, which means that the presence of a global control agent is not required for the working of the system. More in detail, the proposed algorithm is individually and independently used by each node using solely local information.

We experimentally show that the usage of the resulting mechanism, referred to as ANTCYCLE, results in an extended network lifetime when used in different scenarios and in the context of different applications. Experiments give evidence of the ability of the system to cope with changing energy conditions, static nodes, and other scenarios.

Although ANTCYCLE was initially designed and tested from a swarm intelligence perspective that neglected some of the constraints of real networks, the system was later included in a general purpose library for heterogenous sensor networks, called WISELIB [13], and tested with the sensor network simulator Shawn [68].

Distributed Graph Coloring

Graph coloring is a classical problem of mathematics with more than hundred years of history. It consists in assigning labels (colors in the original problem statement) to the vertices of a graph, without adjacent nodes being allowed to share the same label. Several questions have been studied with respect to graph coloring [107]. Particularly relevant due to its relation to several tasks in communication networks is the *graph coloring problem* (GCP). The GCP, formulated as an optimization problem, consists in finding the minimum number of colors necessary to color a graph satisfying the above-mentioned rule concerning adjacent nodes.

Graph coloring has been extensively studied during the last decades [139, 138, 133] using centralized techniques. However, approaches addressing the problem distributedly still offer room for significant improvements even for relatively small or simple graphs. When being used in real applications, distributed techniques may offer better scalability and lower memory consumption. However, the lack of a centralized control usually prevents mechanisms from globally grasping the underlying graph topology.

In Chapter 5 we discuss the relation of graph coloring with problems arising in the design and management of WSNs. Moreover, we provide a novel mechanism for distributed graph coloring, FrogSim, inspired by the behavior of Japanese tree frogs [3, 2]. Briefly explained, researchers showed that groups of male Japanese tree frogs desynchronize their calls to facilitate females to find them. FrogSim is a distributed mechanism for graph coloring in WSNs which aims at minimizing the number of colors used. Information exchange is exclusively restricted to local neighborhoods.

The results obtained with our algorithm improve upon the results of other recent algorithms. In addition to standard graph coloring benchmark instances, we study the behavior of FrogSim when applied to instances of other types of topologies, such as grids or random geometric graphs, and show that FrogSim has advantages over previous algorithms.

Minimum Energy Broadcasting with Realistic Antennas

It is well known that communication processes are several orders of magnitude more energy-demanding than computing. In WSNs, signal transmission can be identified as the main consumer of energy resources. The minimum energy broadcast problem (MEB) [81, 195] deals with the way in which information is transmitted between nodes in a network. Although the algorithms used for the previous problems only considered local exchange of information, it is quite common that nodes need to broadcast a message to nodes which are not their neighbors. In these cases, control on the routes taken by communication messages is a key issue for properly managing energy consumption.

The MEB problem consists in finding communication trees in a network that minimize the total energy consumed in a broadcast transmission. In previous work [90], we provided a heuristic algorithm that efficiently solved a general formulation of the MEB problem. Unfortunately, the antennas used by real WSN nodes are usually omnidirectional and come with a prefixed set of transmission power levels that may be used for communication purposes. Therefore, we introduce a more adequate problem formulation, the *minimum energy broadcast problem with realistic antennas* (MEBRA).

In Chapter 6 we describe an *ant colony optimization* (ACO) algorithm for solving the MEBRA problem. ACO is a swarm intelligence technique for the solution of combinatorial optimization problems introduced in the early 1990s by Dorigo [59]. ACO is inspired by the foraging behavior of ants.

For comparison purposes we adapt a classical heuristic algorithm (the BIP algorithm [195]) from the literature for the MEB problem to work with realistic antennas. Furthermore, we are able to show that the proposed ACO algorithm clearly outperforms the BIP algorithm for the MEBRA problem.

Additionally, we show that the proposed ACO algorithm can be adapted to a distributed environment. Although results obtained with the distributed algorithm are worse than those achieved by the centralized algorithm, the system still finds reasonably good solutions which mostly improve upon those from the centralized BIP algorithm for the MEBRA problem.

1.2 Scientific Contributions

As already mentioned, this thesis deals with three topics related to wireless sensor networks: self-synchronized duty-cycling in networks with energy harvesting capabilities, distributed graph coloring and minimum energy broadcast with realistic antennas. In the following, we provide a summary of our publications related to each of the topics as well as an overview of the research conducted in each case. All publications were supported by the EU-funded FRONTS project [66]. Other personal, traveling or material grants are listed in the *Acknowledgements* at the beginning of this document.

1.2.1 Self-Synchronized Duty-Cycling in Sensor Networks

Together with Prof. M. Middendorf and his colleagues from the University of Leipzig, we proposed a self-synchronized duty-cycling mechanism for sensor networks. Recall that the main goal of duty-cycling methods is to efficiently choose between different states. In the case at hand, we considered two different states: the asleep state, where communications are not possible and energy consumption is low; and the active state, where communications result in a higher energy consumption.

In order to test the model, we performed synchronous simulations on networks. For this purpose, we made use of mobile sensor nodes equipped with energy harvesting capabilities. The results were presented in the 2009 IEEE Swarm Intelligence Symposium and later published in the corresponding proceedings:

- Hugo Hernández, Christian Blum, Martin Middendorf, Kai Ramsch and Alexander Scheidler. Self-Synchronized Duty-Cycling for Mobile Sensor Networks with Energy Harvesting Capabilities: A Swarm Intelligence Study. In Y. Shi editor, *SIS 2009 – Proceedings of IEEE Swarm Intelligence Symposium (SIS)*. Pages 153-159. IEEE Press, 2009.

In order to deepen our understanding of the working of the system we tested the algorithm on a wider variety of scenarios. In this line, we decided to assess the behavior of the mechanism also in the context of static sensor networks. We considered two different topologies:

random geometric graphs and grids. Moreover, we gave an insight into the performance variations caused by changes in the network size. We obtained successful and promising results in all these experiments. In detail, static nodes result in a rather different activity scheme than mobile nodes, which shows slightly lower levels of synchronicity. Despite higher levels of energy consumption, the system is still able to adapt to changing energy conditions of the environment.

This work was presented at the 11th Annual Conference on Genetic and Evolutionary Computation, and further published in the corresponding proceedings:

- Hugo Hernández and Christian Blum. Self-Synchronized Duty-Cycling in Sensor Networks with Energy Harvesting Capabilities: the Static Network case. In *GECCO 2009 – Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation.* Pages 33-40. ACM Press, New York, 2009.

The simulations that were conducted for the two above-mentioned works were characterized by a synchronous simulation environment. In synchronous scenarios, all nodes are able to perform their tasks at the same time and a global clock is used. However, real sensor networks are asynchronous by nature. Several techniques are available to achieve a certain level of synchronicity among network nodes.

In an attempt of making our simulations more realistic, we changed to asynchronous simulations. More specifically, we used a discrete event simulator that is able to handle events assuming a continuous evolution of time rather than considering a discrete time model. Note that the asynchronous model, in contrast to synchronous simulations, requires sensor nodes to perform idle listening in order to store incoming data packets. The experimental evaluation of the asynchronous simulation of our mechanism was presented at the 2009 Workshop on Bio-Inspired Algorithms for Distributed Systems and further published in the corresponding proceedings:

- Hugo Hernández and Christian Blum. Asynchronous Simulation of a Self-Synchronized Duty-Cycling Mechanism for Mobile Sensor Networks. In *Proceedings of BADS 2009 – Proceedings of the Workshop on Bio-Inspired Algorithms for Distributed Systems.* Pages 61-68. ACM Press, New York, 2009.

Finally we extended the work by assuming a broader point of view and a comprehensive study of the parameters. The results are described in the article:

- Hugo Hernández and Christian Blum. Foundations of ANTCYCLE: Self-Synchronized Duty-Cycling in Mobile Sensor Networks. *The Computer Journal*, 54(9):1427-1448, 2011.

The extended work presented in this article has two aims. First, the mechanism described in previous work suffered from sophisticated formulae. Therefore, we tried to simplify, as much as possible, the proposed mechanism, while maintaining its properties. Second, in previous work we did not provide a profound study of the interaction of different system parameters. However, in order to control the system, these interactions must be sufficiently understood. The resulting system is simpler and additionally improves upon the performance of previous algorithms.

Thanks to a collaboration with the Technical University of Braunschweig (partner site of two European projects, FRONTS [66] and WISEBED [67]), we were able to test our algorithm in the real sensor network simulator SHAWN [68]. This work was supported by the EU-funded PerAda initiative.

The fact that most sensor nodes do not provide a state for *idle listening* forces a minor redesign of the duty-cycling mechanism. In the resulting adaptation, a very small fraction of time is scheduled at the beginning of the cycle of each node. This time can be exclusively used for exchanging messages related to the duty-cycling mechanism and all nodes are active in this period. During this period, nodes are able to decide whether they should be active or inactive (asleep) for the rest of the cycle. Active nodes perform the tasks associated with the user application that is being executed. Sleeping nodes do nothing until the next cycle, when they may reconsider their choice between activity and inactivity. The corresponding experiments were presented at the *6th International Conference on Mobile Ad-hoc and Sensor Networks* and published in the corresponding proceedings:

- Hugo Hernández, Christian Blum, Maria Blesa, Tobias Baumgartner, Sandor Fekete, Alexander Kröller. Self-Synchronized Duty-Cycling For Sensor Networks With Energy Harvesting Capabilities: Implementation in Wiselib. In *MSN 2010, Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks.* Pages 134-139, IEEE Press, 2010.

1.2.2 Distributed Graph Coloring

The second part of this thesis is devoted to the desynchronization of wireless sensor nodes and its application to the distributed graph coloring problem. In particular, our research was inspired by the calling behavior of Japanese tree frogs, where males use their calls to attract females. Interestingly, as female frogs are only able to correctly localize the male frogs when their calls are not too close in time, groups of males that are located nearby each other desynchronize their calls. A theoretical model for this behavior has been proposed in the literature. However, its use in technical applications is, so far, quite limited.

Based on the original model, we proposed a novel algorithm with applications to the field of sensor networks. More in detail, we analyzed the ability of the algorithm to desynchronize neighboring nodes as much as possible. Furthermore, we considered extensions of the original model, which improved its desynchronization capabilities as supported by experimental simulations. A discussion on this work was presented in the Workshop on Bio-Inspired Solutions for Wireless Sensor Networks (part of GECCO 2011) and further published in the corresponding proceedings:

- Hugo Hernández and Christian Blum. Implementing a Model of Japanese Tree Frogs' Calling Behavior in Sensor Networks: a Study of Possible Improvements. In *GECCO BIS-WSN 2011 – Proceedings of the Workshop on Bio-Inspired Solutions for Wireless Sensor Networks.* Pages 615–622. ACM Press, New York, 2011.

To illustrate the potential benefits of desynchronized networks, we then focused on distributed graph coloring. Graph coloring, also known as vertex coloring, considers the problem of assigning colors to the nodes of a graph such that adjacent nodes do not share the same color. The optimization version of the problem concerns the minimization of the number of used colors. We dealt with the problem of finding valid colorings of graphs in a distributed way, that is, by means of an algorithm that only uses local information for deciding the color of the nodes. Such algorithms prescind from any central control.

Due to the fact that quite a few practical applications require to find colorings in a distributed way, the interest in distributed algorithms for graph coloring boosted during the last decade. As an example consider wireless ad-hoc and sensor networks, where tasks such as the assignment of frequencies or the assignment of TDMA slots are strongly related to graph coloring. For this purpose, we proposed an extended algorithm that used desynchronization to achieve better colorings. We experimentally showed that our algorithm was very competitive with the current state of the art by comparison with one of the most competitive algorithms from the literature. These results were presented at the IFIP/IEEE Wireless and Mobile Networking Conference and published in the corresponding proceedings. In addition to invaluable comments and suggestions, the Program Committee of the conference awarded our work with the Best Paper Award.

- Hugo Hernández and Christian Blum. Distributed Graph Coloring in Wireless Ad Hoc Networks: A Light-weight Algorithm Based on Japanese Tree Frogs' Calling Behaviour. In *WMNC 2011 – Proceedings of IFIP/IEEE Wireless and Mobile Networking Conference*. Pages 1–7. IEEE Press, 2011. Best Paper Award.

A more detailed description of the desynchronization-based graph coloring algorithm and the transition from the mathematical model to this algorithm, as well as a broader comparison of the proposed and existing methods, have been accepted for publication:

- Hugo Hernández, Christian Blum. Distributed Graph Coloring: An Approach Based on the Calling Behavior of Japanese Tree Frogs. Swarm Intelligence, 2012. In press.

- Hugo Hernández, Christian Blum. FrogSim: Distributed Graph Coloring in Wireless Ad Hoc Networks – An Algorithm Inspired by the Calling Behavior of Japanese Tree Frogs. Telecommunication Systems, 2012. In press.

1.2.3 Minimum Energy Broadcast with Realistic Antennas

The classical minimum energy broadcast (MEB) problem in wireless adhoc networks, which is well-studied in the scientific literature, considers an antenna model that allows the adjustment of the transmission power to any desired real value from zero up to the maximum transmission level. However, when specifically considering sensor networks, a look at the currently available hardware shows that this antenna model is not very realistic. In this work we reformulate the MEB problem for an antenna model that is realistic for sensor networks. In this antenna model transmission power levels are chosen from a finite set of possible ones. A further contribution concerns the adaptation of ant colony optimization –a current state-of-the-art algorithm for the classical MEB problem– to the more realistic problem version, the so-called *minimum energy broadcast problem with realistic antennas* (MEBRA). The obtained results show that the advantage of ant colony optimization over classical heuristics even grows when the number of possible transmission power levels decreases. Results, together with a formulation of the MEBRA problem, were presented at the *4th International Conference on*

Bio-Inspired Optimization Methods and their Applications (BIOMA 2010) and published in the corresponding proceedings:

- Hugo Hernández and Christian Blum. Ant Colony Optimization for Broadcasting in Sensor Networks under a Realistic Antenna Model. In Bogdan Filipic and Jurij Silc editors, *BIOMA 2010 – Proceedings of 4th International Conference on Bio-Inspired Optimization Methods and their Applications.* Pages 153-162. Published by Jozef Stefan Institute, Ljubljana, Slovenia, 2010. ISBN 978-961-264-017-0.

A more complete and extended paper, including a more extensive experimentation with larger networks and different antenna models, was published in:

- Hugo Hernández and Christian Blum. Minimum Energy Broadcasting in Wireless Sensor Networks: An Ant Colony Optimization Approach For a Realistic Antenna Model. *Applied Soft Computing*, 11(8):5684-5694, 2011.

As previously mentioned, WSNs are generally working in an asynchronous fashion. As a consequence, we decided to develop a distributed version of the ACO algorithm on the basis of the centralized algorithm version. Although the results achieved are worse than those obtained with the centralized version of the algorithm, the distributed algorithm still compares quite favourably against the algorithms from the literature. An article describing the distributed system in detail together with the obtained results has been accepted for publication.

- Hugo Hernández and Christian Blum. Distributed Ant Colony Optimization for Minimum Energy Broadcasting in Sensor Networks with Realistic Antennas. *Computers & Mathematics with Applications*, 2012. In press.

Chapter 2

Wireless Sensor Networks

Wireless sensor networks (WSNs) offer a valuable testbed for distributed algorithms and applications. Therefore, they attract the interest of many computer science research groups. As a result, the work done by computer scientists has produced rich libraries and environments for developing applications based on sensor networks. These advances have encouraged scientists from other fields to start using sensor networks for their experiments. For example, thanks to the sensing devices attached to sensor nodes biologists can monitor the environment (humidity, light, etc.) for extended periods of time.

The conventional approach for sensing considers a reduced set of complex (sometimes semi-manual) devices which are used to obtain the desired measures. WSNs, instead, aim at sensing by means of a large set of unsophisticated tiny devices with networking capabilities. WSNs compensate the lack of more advanced features, such as high resolution or precision, employing aggregated data from a larger set of information feeds. Networked sensing provides several advantages such as, for example, greater coverage, accuracy and reliability while reducing development costs [35, 65]. WSNs are designed to support an enormous number of tasks [5]. Typical examples, by way of illustration, include the monitoring and the control of factories, sensors used for weather forecasting, or medical applications and smart home automation systems. The multiscope nature of WSNs strongly raises the level of complexity of designing, manufacturing and networking sensor nodes. Not surprisingly, a considerable amount of related research activity has been linked to WSNs during the last decade.

Research activity in WSNs can be described as being organized at three levels: the component level, the system level and the application level. Researchers focused on improving the available hardware for WSNs represent the component level. Typical improvements at this level are better sensing capabilities, the efficient use of batteries in communications, and increased computation capabilities. The system level covers innovations in the networking and coordination of the different sensor devices. Usual targets for these improvements are power efficiency and scalability. The application level is related to the goal of the sensor network. The tasks and messages sent by each node depend, mainly, on the application objective. Possible applications include the periodical reporting of the temperature at the sensor node locations or the localization of a tracked object. Figure 2.1 shows photos of a project (*Canopee Project*) which consisted in installing Sun Microsystems SunSpot sensors in the canopees of trees in the rainforest of India.

A remarkable fact, which probably is not obvious, is that the foundations on which WSNs

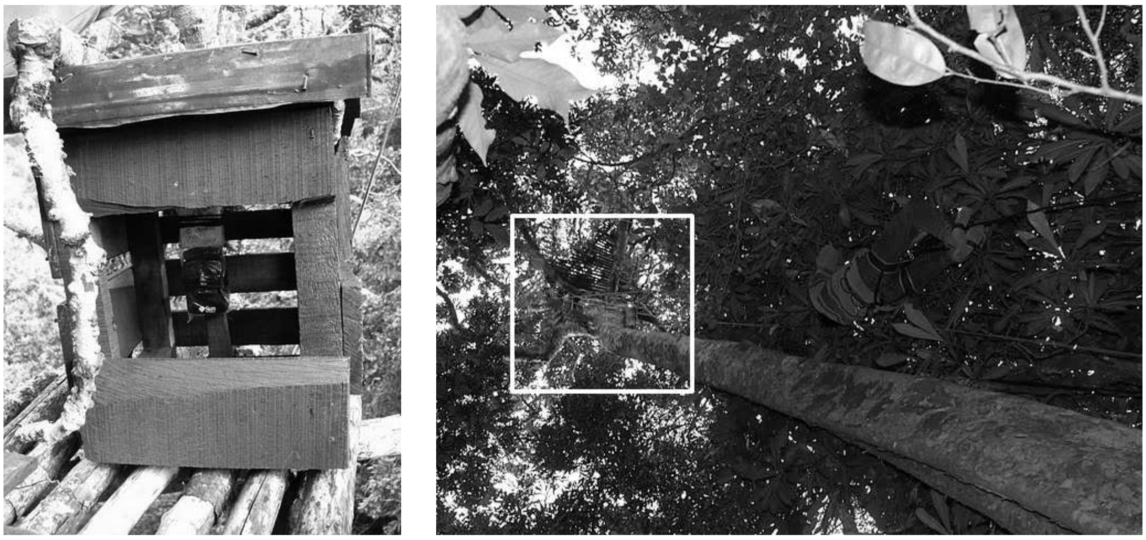


Figure 2.1: Photos of the Canopee project endorsed by Sun Microsystems, India. The first photo shows a box in which one SunSpot is installed. The second photo shows a ground view of a tree on which a sensor is located. Although difficult to appreciate, the red line indicates the presence of one of these container boxes with a sensor inside.

were created are structurally different from those which support traditional networking systems. Home or business networks, as well as the Internet itself, are developed to support different actors actively using the network with independent and selfish goals. The diversity of objectives forces a modular design that has been historically tackled with a layered system. The OSI model described in Figure 2.2 was proposed to divide the working of a network in seven different levels. Another example of a layered network design is the Internet network model. It is a simpler model than the one represented by OSI, dividing network applications in 4 layers. A summary of some of the most used protocols for each layer is provided in Table 2.1. In both models, developers can run any new high-level application on top of the existing network avoiding to enter in physical or routing details managed by lower layers. However, dealing with the multipurpose design of traditional networks may interfere with the goals of WSNs, for example, causing an unnecessary usage of power and computation resources due to an excessive amount of components for a relatively simple network basically supporting broadcasting.

Some core characteristics of WSNs are generally associated to this need for a different network design: a relatively high density of nodes, which has forced the development of simple and cheap devices; application diversity, which causes the need for different applications to use different devices; low memory and power availabilities, which call for highly optimized and lightweight protocols; and, most importantly, in WSNs all nodes generally cooperate for a single and common goal. In some cases multiple goals may concurrently exist. Even if that happens, a single user is still able to configure the sensor node and, therefore, distribute the amount of resources invested in each task.

The basic component of a sensor network is the sensor node. There are many different

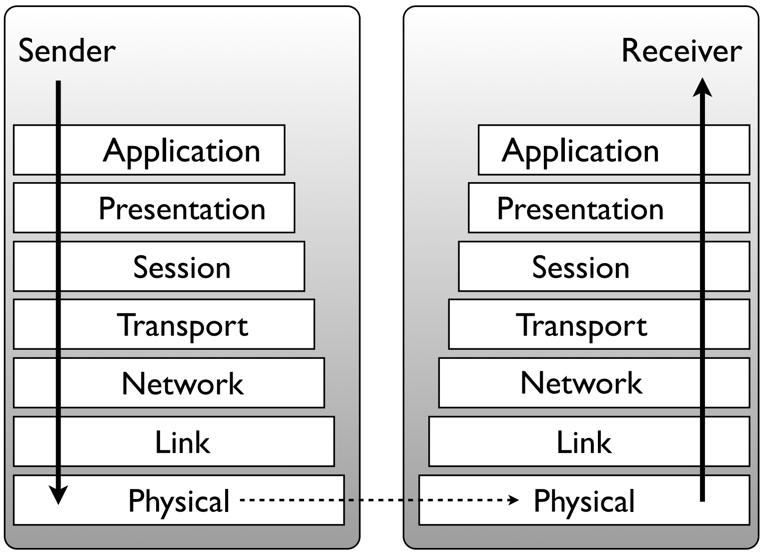


Figure 2.2: The OSI network model divides network communications in seven different levels (layers). Lower levels are closer to physical details of communications. On the contrary, higher levels cope with communications from a logical point of view.

Table 2.1: Internet layered model

Layer	**Protocols**
Application	BGP DHCP DNS FTP HTTP IMAP IRC LDAP MGCP NNTP NTP POP RIP RPC RTP SIP SMTP SNMP SOCKS SSH Telnet TLS/SSL XMPP
Transport	TCP UDP DCCP SCTP RSVP ECN
Internet	IPv4 IPv6 ICMP ICMPv6 IGMP IPsec
Link	ARP/InARP NDP OSPF L2TP PPP MAC (Ethernet, DSL, ISDN, FDDI)

kinds of sensor nodes. Fortunately, the high level description of most of them is quite similar. Figure 2.3 shows the structure of the components of a sensor node. A sensor node is basically composed of one or more sensors connected by means of an *analog-digital converter* (ADC) to a microcontroller. The microcontroller is able to perform calculus with these data, to send and to receive messages from other nodes in the network through the transceiver and to store data for later use in a persistent storage system. In addition, sensor nodes need a power unit that may be either composed of batteries or a continuous power supply.

In the rest of this chapter we will review more in detail the state-of-the-art of WSNs concerning the three levels previously mentioned: the component level, the system level and the application level. In Section 2.1 we deal with recent trends in sensor development and the

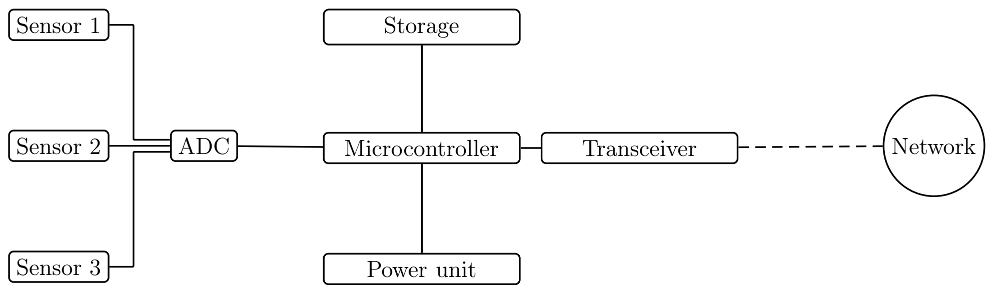


Figure 2.3: High level architecture of a sensor node.

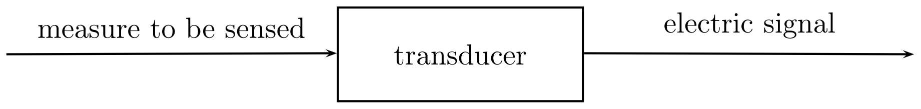


Figure 2.4: Basic working of a transducer. An input measure is used by the transducer to generate another measure or signal. Transducers act as bridges between two measures. In WSNs, transducers generate electric signals that can be processed by a microcontroller.

physical properties of materials that enable the construction of sensors for different measures. In Section 2.2 the networking properties of sensor nodes are reviewed. The discussion includes the differences to other kinds of networks. Next, Section 2.3 shows a categorization of the applications that a WSN can perform. Finally, Section 2.4 reviews some of the most significant challenges, problems and issues of WSN development. These topics are the focus of most of the research in the field.

2.1 Sensing

As previously explained, several physical sensors may be attached to wireless sensor nodes. Nevertheless, the core of sensor nodes are small electronic devices which are only able to measure and process electronic signals. Therefore it is quite common to design sensors using (or directly as) *transducers*. A transducer is a device which is able to transform energy from a specific domain to a different one (see Figure 2.4). More specifically, useful transducers in WSNs are those whose output come in the form of voltages or currents.

Several physical principles can be used for transduction in sensors. Moreover, sensors of the family of *Microelectromechanical Systems* (MEMS) are currently available for most sensing requirements or applications. In the following we describe some of the most used techniques for WSNs. They are grouped by the energy that acts as input of the transducer as in the classification by Lewis [127].

The first group of sensors is composed of those where direct physical contact is necessary. This category of sensors is called *mechanical sensors.*

- The *piezoresistive effect* transforms a pressure applied to a surface into a resistance that can be detected using electronic circuitry. Metals and semiconductors exhibit piezoresistivity. Rather than a recent discovery, the effect was stated by Lord Kelvin in 1856. The relation between the pressure applied and the resistance created was established as:

$$\frac{\Delta R}{R} = S \cdot \varepsilon \tag{2.1}$$

  where R is the resistance, ε the strain applied[1], and S the gauge factor which depends on the properties of the material. A usual technique to increase the effect on silicon circuits considers doping using boron.

- The *piezoelectric effect* (Curie, 1880) transforms material stress into potential difference. The voltage difference V depends linearly on the force F applied:

$$\Delta V = k \cdot \Delta F \tag{2.2}$$

  Hereby, k is a constant that depends on the material charge sensitivity, the material relative permittivity, the crystal thickness and the crystal area. One of the interesting features of the piezoelectric effect is its reversibility (the transformation of a potential difference into physical stress). Current research is exploring the design and usage of dual devices which act both as sensors and actuators.

- The *tunneling* technique is able to measure with high precision nano-metric movements. Tunneling measures the relation between a current I and the surface separation z:

$$I = I_0 \cdot e^{-kz} \tag{2.3}$$

  The main drawback while using tunneling is its highly nonlinear nature that results in a necessary use of feedback techniques.

- *Capacitive sensors* are composed of two plates, one fixed and one movable. A displacement distance Δd of the plate results in a change of the capacitance C:

$$\Delta C = \frac{\varepsilon \cdot A}{\Delta d} \tag{2.4}$$

  where ε is the dielectric constant. Many different circuits can detect changes in capacitance and convert them to changes in voltage or current. Another useful family of senors considers the conversion of displacement into an inductance variation. These sensors are generally known as *inductance sensors.*

[1]Depending on the material used, ε may become a quadratic term.

In those cases where physical contact is not possible or must be prevented, manufacturers can use *magnetic and electromagnetic sensors.* This family of sensors is primarily used for detecting proximity issues [118].

- *Magnetoresistive sensors* are related to the Hall effect (discovered by Edwin Hall in 1879). The Hall effect explains the creation of an induced voltage (Hall voltage) in a plate of thickness T equal to:
$$V_{\text{Hall}} = \frac{R \cdot I_x \cdot B_z}{T} \tag{2.5}$$
where R is the Hall coefficient (usually around 5 times larger in semiconductors than in metals), I_x is the current flow on the x-axis and B_z the magnetic flux in the z-axis.

- *Magnetic field sensors* are efficiently used to detect the presence of metallic objects. A more specific type of sensors are *Eddy-Current sensors*, which use magnetic coils to detect problems in defective metallic structures.

A whole family of sensors, *thermal sensors*, has been developed for monitoring temperature or detecting heat fluxes.

- *Thermo-mechanical transduction* is based on the dilation of materials due to temperature increases (and the reverse effect when temperature decreases). The relation between the temperature change ΔT and the expansion produced on the materials length $\Delta \cdot L$ is linear:
$$\Delta \cdot L = \alpha \cdot \Delta T \tag{2.6}$$
with α being the coefficient for linear expansion which is a property of materials. Thermo-mechanical sensors are common in home and vehicle automation.

- *Thermoresistive sensors* measure temperature by observing the changes in the resistance of a given material. It is difficult to formulate the relation between temperature and resistance for some materials such as silicon. Nevertheless, the relation has been correctly approximated and silicon is commonly used for measuring temperature. The effect of temperature changes ΔT in the resistance R of most metals is easier to explain:
$$\frac{\Delta R}{R} = \alpha_R \cdot \Delta T \tag{2.7}$$
where α_R is the temperature coefficient of resistance, a constant that depends on the material.

- *Thermocouples* consist of two different materials joint together by both ends. If in these circumstances one of the joints is hotter than the other a current flows in the circuit (*Seebeck effect*). If T_1, T_2, are the temperature at the first, respectively second, joint, the Seebeck voltage is approximated by:
$$V \approx \alpha(T_1 - T_2) + \gamma(T_1^2 - T_2^2) \tag{2.8}$$
Thermocouples are quite popular due to their low cost and high reliability. In particular, semiconductor thermocouples are generally more sensitive than metal thermocouples.

- *Resonant temperature sensors* rely on the changing resonant frequency of single-cristal SiO_2 related to temperature changes. The frequency nature of the effect results in higher accuracy than amplitude-change effects even for small temperature changes.

Optical transducers are able to transform light into different effects (a current to flow, the resistance of a material to change, heat generation, etc). Solar cells are also optical transducers which are able to transform light into voltage. Moreover, optical transducers can be adjusted to respond to different light frequencies as in the construction of, for example, infrared or ultraviolet detectors. Optical fiber technology is also commonly used for the construction of accelerometers and other devices.

Chemical and biological sensors are able to detect the presence of specific molecules in the air or other materials. There exist several kinds of transducers which can interact with solids, liquids and gases. Among the possible applications, we could mention environmental or food monitoring, density of NO_x or CO_x particles in the air (pollution), presence of pesticides, anthrax, etc.

- *Chemiresistors* are composed of two interdigitated finger electrodes. They are coated with special chemical products whose resistance changes when exposed to certain chemical agents. The combination of adequate coatings and digital signal processing techniques –as neural network classification– allow to correctly identify different chemical agents.

- *Metal-oxide gas sensors* measure the presence of gases in the environment based on the fact that the resistivity of some semiconductors suffers severe changes due to the absorption of gases as CO_2, CO, NH_3, and ozone (O_3). A sample reaction is the following one:

  $$O_2 + 2e^- \longrightarrow 2O^- \tag{2.9}$$

  As a consequence of the O atoms trapping the electrons, the amount of mobile carriers in the surface is reduced and the material resistance increased.

- *Electrochemical sensors* are very simple and popular sensors. They rely on the effects of reduction of a chemical molecule (oxidation) at the surface of an electrode. A current is caused by the following reaction:

  $$O + ze^- \longleftrightarrow R \tag{2.10}$$

  with R being the molecule that disappears due to oxidation and z being the charge on the ion involved in the reaction.

- *Biosensors* are a wide family of sensors which rely on specific biomolecular reactions. The main drawback of biosensors is that many of the reactions that occur are not reversible. As a consequence, biosensors are not always reusable. Biosensors include a biochemically active substance located on a surface that converts property changes –as mass or resistance variations– into electric signals or light.

Electromagnetic sensors are used for remote sensing in different scenarios. Remote sensors usually work on a specific wavelength which depends on the application. The wavelength

Property family	**Measure**	**Transduction Principle**
Physical	Pressure	Piezoresistive, capacitive
	Temperature	Thermistor, thermo-mechanical, thermocouple
	Humidity	Resistive, capacitive
	Flow	Pressure change, thermistor
Motion	Position	E-mag, GPS, contact sensor
	Velocity	Doppler, Hall effect, optoelectronic
	Angular Velocity	Optical encoder
	Accelaration	Piezoresistive, piezoelectric, optical fiber
Contact	Strain	Piezoresistive
	Force	Piezoelectric, piezoresistive
	Torque	Piezoresistive, optoelectronic
	Slip	Dual torque
	Vibration	Piezoresistive, piezoelectric, optical fiber, sound, ultrasound
Presence	Tactile/contact	Contact switch, capacitive
	Proximity	Hall effect, capacitive, magnetic, seismic, acoustic, RF
	Distance/range	E-mag (sonar, radar, lidar), magnetic, tunneling
	Motion	E-mag, IR, acoustic, seismic (vibration)
Biochemical	Biochemical agents	Biochemical transduction
Identification	Personal features	Vision
	Personal ID	Fingerprints, retinal scan, voice, heat plume

Table 2.2: List of commonly sensed measures in WSNs. Each measure is accompanied by the transduction principle which is typically used in the design of the corresponding sensor. This table is an extension of that shown in [127]

determines the propagation distance, resolution, the ability to trespass solid objects or unclear mediums and the cost of the signal processing. Therefore, the selection of a wavelength depends on both the application and the environment where the sensor will be used. For example, millimetric waves have been used for monitoring satellites or infrared motion detectors are quite popular for night and heat vision. Another application of electromagnetic waves considers the usage of time-to-flight information for establishing the distance to a remote object. More in detail, radar systems use radio frequency (RF) waves and lidar[2] systems use light (laser). Other well known mechanisms relying on electromagnetic transmissions are GPS systems which use RF waves. More recent advances consider the use of visible light imaging for several different applications such as face recognition or motion analysis. However, these systems are still relatively primitive, algorithmically complex and very expensive in computational terms.

Acoustic sensors are similar to electromagnetic sensors but use sound for measuring instead of electromagnetic waves. Applications of acoustic sensors include velocity measurement relying on the Doppler effect, distance measuring relying on sonar techniques and, in the field of mechanical machinery, ultrasounds for the analysis of vibrations, fluid leakage or other system failures. When considering the usage of sound for sensing applications special attention has to be put on the medium, as the propagation speed depends on the environment. Also echoes and other noises must be carefully treated as they may downgrade the signal quality.

Finally, in Table 2.2 we reproduce the table from [127] which summarizes the common measures sensed in WSNs. Each measure is related to the usual transduction methods used for the construction of their corresponding sensors.

[2]Lidar is a detection system that works on the principle of radar, but uses light from a laser.

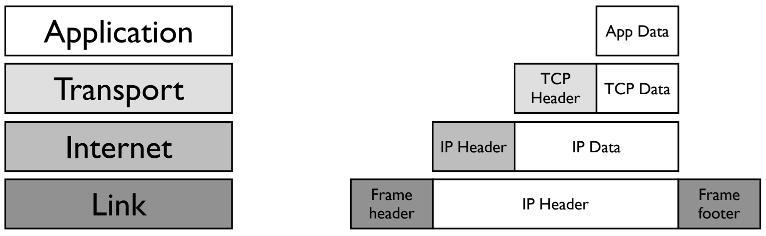


Figure 2.5: Ethernet message frame.

2.2 Communication and routing

Depending on the scope and goals of a network, different protocols and routing techniques may be used. Although the topics and protocols related to networking are quite complex, the high order working of a network can be easily described. In fact, being part of a network enables nodes to perform only one additional task: it allows a node to send a message to another node. Moreover, networks allow all nodes within the network to send messages while energy resources last. Besides, each message may have a different destination. In this section we will first introduce some of the topics and protocols used in networking. Because WSNs are essentially different from general purpose computer networks, later we will also introduce the specific features and mechanisms used in the WSN architecture.

The working of a network usually requires nodes to add some metadata to their messages. The various metadata fields may vary depending on the network protocols used, but are generally used to guarantee the adequate routing of messages. Currently, most networks use *ethernet* for linking the different nodes. The header of an ethernet message is shown in Figure 2.5. In those networks where encoding is possible, headers may also include some parity bits.

The main performance measure of a network is *quality of service* (QoS). Several advances are incorporated into sensor networks in order to improve QoS. For example, packet routing networks divide all messages in *packets* of fixed length. Nodes transmit packets separately and only once all packets have reached their destination they are reassembled into the original message. Although the transmission of a higher number of packets increases the amount of resources used for metadata, the usage of smaller data units reduces the costs of transmission failures.

In some cases, especially before sending large amounts of data, nodes may request other nodes for their availability in order to prevent them to consider other communications for a while. *Handshaking protocols* are used in those cases. Handshaking improves the QoS and enables active detection of lost packets. In Figure 2.6 we show the working and the different phases of a handshaking protocol.

Network traffic may also include other kinds of messages used to report and identify fault conditions. One example are messages used to discover shortest paths or failed links.

In those cases where nodes need to transmit a message using multiple hops, routes are used. The message routes depend on the protocols chosen. However, the behavior of the intermediate nodes is similar in most cases. The most common technique for processing routed messages is *store-and-forward* switching. Whenever a packet is received, it is completely buffered in local memory and retransmitted as a whole. Other switching techniques are, for example, *wormhole*, which divides messages in smaller units known as flow control units or flits; or *virtual-cut-*

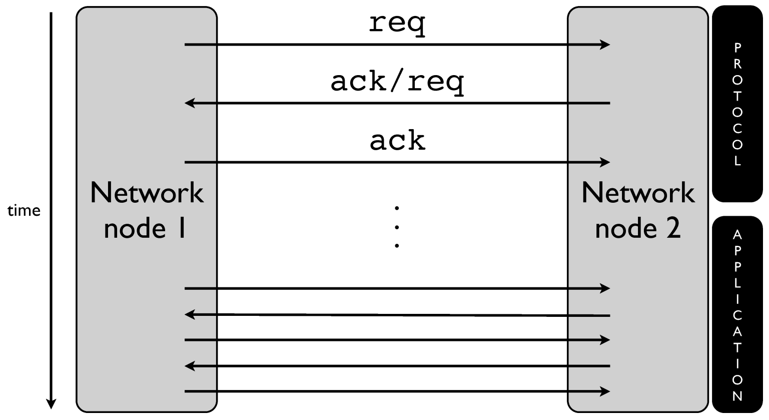


Figure 2.6: The working of the basic network handshaking protocol is fairly easy to describe. It consists of four steps: (1) a network node makes an availability request to a second node; (2) the secondary node answers the request with an acknowledgement message which also includes a request for confirmation of reception; (3) the first node confirms the reception of the acknowledgment; (4) both nodes exchange as many messages as needed. Notice that the first three messages are overhead as they only contain protocol information. From the fourth message on, messages are exclusively determined by the application needs.

through, which forces a header to be routed instantly after reception without waiting for the rest of the packet.

An important aspect, which has gained even more relevance with the appearance of wireless technologies, is that multiple nodes might try to communicate simultaneously. However, nodes can only receive one packet at a time. A general solution for managing *multiple access control* (MAC) was proposed for wired communications and dates back to 1970. In the case of the ALOHA protocol [1, 169], nodes wait for reception acknowledgements of all the messages sent. If after some time no response has been received, the node waits for a random amount of time and sends the packet again. Obviously, although all packets will probably be received by the destination node, the ALOHA scheme involves a huge degree of inefficiency.

Another approach designed exclusively for MAC in wireless communications is *Frequency Division Multiple Access* [122] (FDMA), which assigns different communication frequencies to different nodes. FDMA results in lower bandwidth available at each node. The *Time Division Multiplexing Access* [122, 9] (TDMA) is a more popular approach in sensor networks. Time is divided in discrete time steps which, in turn, are divided in time slots. Each node is assigned a time slot that it may use for communication purposes. During the other time slots, the node is not allowed to communicate. The main drawback of TDMA is that nodes usually need to be time-synchronized in order to share the schedule of time steps and slots.

As previously mentioned, network design is a very complex task. Moreover, the previous paragraphs just mention some of the most important points to consider when working with networks of any kind. Other topics such as routing, management of deadlocks and livelocks or flow control are as much important as the previous ones. In [127] the authors provide a broader introduction to communication in WSN.

Architecture of Wireless Sensor Networks

At the time when the network protocols used in our homes, offices and on the Internet were developed, none of the WSN constraints were taken into account. The OSI[3] [211] network model was developed as a layered system that enables users to develop network applications without dealing with cumbersome physical details about how information is transmitted from one computer to the other. Moreover, the use of physical wires allowed to create different network topologies easily. However, WSNs are essentially different. First, and most importantly, the nature of a wireless sensor network is usually quite specific. WSNs are proposed for attending the needs of a single user. The application designer expects the network to exclusively execute his/her application. Consequently, the layered system design, which indeed results in additional computation and communication overheads, may not be an adequate choice. Although in most cases WSN manufacturers provide interface libraries which enable working from a logical point of view, the network architecture avoids the usage of many intermediate layers in concordance with the much lower energy and computing resources available to sensor nodes.

In general, WSNs are only used as a secondary system for data collection which depends on a mainframe which is responsible for processing data and taking decisions. In most WSNs, communication with the primary system is achieved by means of a *gateway* node. Other usual denominations of the gateway node are *sink* or *master*. The network gateway is commonly a special node which may have additional or more advanced features than the other network nodes. Moreover, gateway nodes may be attached to a continuous power supply, or at least energy can be treated as an unlimited resource from the application perspective.

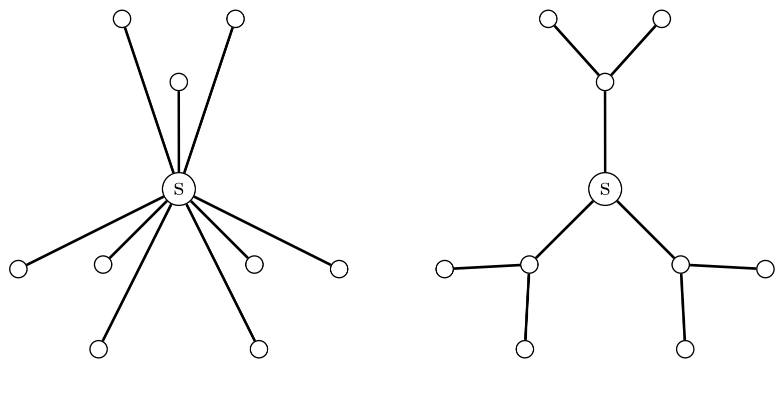
Applications in WSNs are mostly gateway-centric. Communication with the gateway is equally intensive and continuous along the application execution. Even though in some WSN applications nodes only have to interact with their local neighborhood without reporting data or decisions to the sink, most applications include, at least, some kind of data exchange with the sink. As an example for a network where no interaction with the sink is necessary consider a network in charge of turning on a light in a production line if temperature passes a certain threshold.

Moreover, WSNs are, as their name indicates, communicating wirelessly. Wireless communications dramatically reduce the infrastructure cost of networks. However, this comes at the cost of other drawbacks such as, for example, lower security, higher energy consumption, and a higher rate of potential packet collisions. Wireless communications offer the opportunity of creating ad-hoc networks spontaneously and, also, to allow changes in the network topology over time. Topology control is, in fact, one of the fields that has received a lot of attention from the research community [163, 164].

The basic unbreakable premise in the topology of a WSN is that all nodes must be able to communicate with the gateway node (via a direct link or using multi-hop transmissions). Nodes which are not able to reach the network sink become *disconnected*. Disconnected nodes are basically lost for sensing purposes and, therefore, become useless.

A simple classification of WSN architectures is proposed in [147]. In those networks where

[3]The *Open Systems Interconnection* (OSI) is a project started in 1977 by the *International Organization for Standardization* (ISO) in order to normalize the networking field. It was the first attempt to standardize networking and communications between electronic devices that was not driven by governments or vendors. Among the diverse benefits of the model, the simpler and more understandable conception of the different components resulted in major advances in teaching of network concepts.



(a) Flat architecture (b) Hyerarchical architecture

Figure 2.7: A network of ten nodes connected as a (a) flat architecture and (b) a hierarchical architecture.

all nodes are allowed to communicate directly with the gateway, we say that a *plain* or *flat* architecture is being used. Figure 2.7(a) shows an example of a network composed of ten nodes using a flat architecture. This structure is particularly efficient in rather small networks where most nodes are able to physically communicate with the gateway. Nevertheless, in large networks –especially in those where many nodes require multi-hop transmissions to reach the network gateway– the computing and transmission overheads that message routing implies may result in an unweighted consumption of energy resources.

Hierarchical or *clustered* architectures avoid the previously mentioned problems by introducing a set of cluster-heads (see Figure 2.7(b)), which are nodes with a higher degree than the rest . A subset of the network nodes are designated as cluster-heads. Moreover, cluster-heads are the only nodes allowed to directly communicate with the gateway. Other nodes must make use of these nodes in order to reach the gateway. In fact, cluster-heads become *virtual gateways*, each one routing the transmissions from a certain set of nodes. Due to these additional tasks that cluster-heads have to accomplish, and in order to prevent early battery deployment, two solutions are generally conceived. One solution consists in using special nodes which are equipped with larger batteries. The other solution, used when all network nodes are equal, makes use of a protocol for dynamically exchanging the nodes playing the role of cluster-heads.

2.3 Application Categories

According to Iyer and colleagues [103], WSN applications may be classified in four major groups depending on their objectives, traffic characteristics and data delivery requirements. In the following classification, each sensor network has usually a set of one or more distinguished nodes which are called *sink*. It is important to understand the role of these nodes whose main target is to link the network with a computer, panel or other electronic system located outside the network in order to export the information or data collected by the network.

Event Detection and Reporting

The main goal of applications in this category is to report information, generally about detected events to the sink. Examples of applications in this class are detection of failures in a manufacturing process, detection of forest fires, and intruder detection. As expected events usually occur only sporadically –when not rarely– nodes in the network are expected to be inactive most of the time. Only when one of these events is detected the network bursts into activity. Inactivity includes also the communication channels, which remain unused most of the time and are being forced to handle a huge amount of messages when the observed events happen. Usually events are not reported instantly. Instead, the systems waits to obtain a certain consensus between different nodes in order to avoid false positives.

Data Gathering and Periodic Reporting

A common characteristic of systems aimed at periodic data collection is their need to keep the network alive for extended periods of time. In many cases, if energy harvesting is possible, it might be even necessary to reach zero-consumption situations where the amount of energy consumed equals the amount of energy generated at the harvesting units. The most significant examples of this kind of applications are those for monitoring temperature, humidity, light and other physical variables (both indoors and outdoors). In some cases monitoring can be combined with actuation in order to control, for example, the amount of water dispensed by an irrigation system depending on the current humidity, or the allowance of vehicles into a car park depending on the availability of parking places. Actuator modules continuously expect to receive information from the monitoring nodes to adapt the behavior of the system to the current conditions of the environment. Moreover, it is not uncommon that actuators are interested in the information of many nodes located within a bounded region, requiring a distributed computation of some function based on multiple sensor readings.

Sink-initiated Querying

Several applications may require the sink to be able to trigger actions or query information from the network. The usage of sink-initiated querying allows the system to focus on particular regions or to extract information at different resolutions and granularities. For instance, in the case of a malfunction, this mechanism can be used by the sink to collect information about the problem and report or repair as necessary. Additionally, sink initiated querying avoids the need of nodes continuously reporting their state to the sink reducing dramatically energy and bandwidth consumption.

Tracking-based Applications

A frequent scenario considers WSNs for detecting the presence of entities external to the network in the region of deployment. This is the case of applications aiming at controlling the movements of packages in shipping facilities or discovering the areas of influence of certain species. By tracking the movement of animals biologists are also able to understand their habits and patterns. Usual mechanisms in WSNs for tracking include notification of object

Hardware	Firmware/OS	CPU	Language	Dynamic Memory	ROM Size	RAM Size	Bits
iSense	iSense-FW	Jennic	C++	Physical	128kB	92kB	32
SCW MSB	SCW-FW	MSP430	C	None	48kB	10kB	16
SCW ESB	SCW-FW	MSP430	C	None	60kB	2kB	16
Tmote Sky	Contiki	MSP430	C	Physical	48kB	10kB	16
MicaZ	Contiki	ATMega128L	C	Physical	128kB	4kB	8
TNOde	TinyOS	ATMega128L	nesC	Physical	128kB	4kB	8
iMote2	TinyOS	Intel XScale	nesC	Physical	32MB	32MB	32
GumStix	Emb. Linux	Intel XScale	C	Virtual	16MB	64MB	32
Sun Spot	Squawk	ARM920T	Java	Physical	4MB	512kB	32

Table 2.3: Sensor node manufacturers

detection to the sink and management of sink queries requiring nodes near the object to be ready for prompt detection and reporting.

2.4 Common Design Problems

Until now, researchers have come up with various applications for sensor networks. However, just a few of them have evolved to real developments. Most works are experimental studies about the possibilities of sensor networks. Moreover, sensor nodes are named after the sensing devices that they equip and that enable nodes to measure or detect light intensity, humidity, pressure, acceleration, heat and other physical measures of the environment. The benefits of the proposed sensor networks over other alternatives for solving real world problems are, usually, their ease of development, lower infrastructure costs and improved performance. Depending on the problem tackled developers have the choice between a wide range of sensor devices each one equipped with different sensors. Some of these nodes allow the user to choose which sensors to attach while others come already bundled and do not allow extensions.

Diversity of sensor node manufacturers and the lack of standards are the key factors to understand the heterogeneity that prevails in the field of WSN devices. Heterogeneity is, currently, one of the main problems in WSNs. The consequences can be noticed in the following two aspects: firstly, it is not easy to establish and abstract models for sensor networks that can be used by researchers only interested in the logical side of WSN working and communications. Some works, such as [8, 7], tackle this issue and, in fact, the EU-funded WISEBED [67] project is focused on the interconnection of heterogeneous devices. Second, interconnection between different devices is typically difficult as not all devices use the same communication protocols. Even when upgrading a WSN this could be problematic as adding new nodes to an existing network may not be easy if manufacturers no longer supply the original WSN devices used.

In addition to being equipped with different sensors, sensor nodes are characterized by different memory sizes, operating systems and communication protocols. Table 2.3 shows some well-known sensor nodes together with their operating systems, CPUs and other specifications. Due to the lack of standardization in the WSN field, certain design problems appear in most sensor networks and application families. We shortly describe them in the following.

Communication versus Computation

When using wireless channels, communication becomes very expensive in terms of energy-consumption. For example, computing is several orders of magnitude cheaper [5] than communication. Fortunately, methods exist for reducing the amount of intra-network communications.

For instance, in a many-to-one scenario, intermediate nodes may be used to aggregate the data from other nodes on its way to the sink. Advantages achieved by in-network data aggregation may be dramatic when combined with the proper clustering or topology control techniques. In fact, it is quite important to check the amount of information and the type of aggregation performed. Although averaging a set of values is fast, other operations such as signal processing may require more time. Moreover, the information flow in WSNs is usually under certain spatio-temporal constraints which can not be neglected. Otherwise, information requests may expire or significant delays to further events might be caused.

Many-to-one Routing

The many-to-one paradigm is used in most WSN applications. For example, for collecting data from the network, often a single gateway node is used. However, before any information can be retrieved, nodes must provide the gateway node with all the data.

The communication flows required in this scenario are absolutely non-uniform. Nodes strategically located may be forced to consume much more energy than nodes in the network boundaries. Therefore, the usage of routing algorithms which consider this non-uniformity and try to find alternative routes to mitigate the effect of demanding network flows in certain nodes is quite recommended.

From a more general perspective, routing algorithms should combine both techniques for finding energy-efficient routes and for distributing energy-consumption.

Power-saving Algorithms for the Radio

As mentioned before, wireless devices allow the construction of ad-hoc networks without the additional infrastructure costs caused by wires and routers. Unfortunately, there are two main drawbacks. In first place, wireless communications are susceptible to security breaches, as information is transmitted through the air and can be captured by any device within communication range. However, cryptographic algorithms and applications habilitate secure communication scenarios by means of cryptographic keys. In second place, wireless communication can be energetically expensive. Moreover, wireless nodes are usually equipped with batteries to allow their usage in locations where no power supplies are available. This makes energy an essential but scarce resource. Unfortunately, battery capacity has increased only slowly throughout the years giving an important role to energy aware protocols and algorithms. The *Minimum Energy Broadcast/Minimum Energy Multicast* problem [90], which consists in finding energetically-optimal transmission trees in a network, is a good example of these kind of efforts.

Sensor networks are composed of a relatively large number of nodes which usually oscillates between a few dozens and a few hundreds. Nodes are usually equipped with omnidirectional antennas and most applications make intensive use of communication between nodes in order to efficiently achieve the application goals.

Another important way for energy-saving is labelled as duty-cycling, or sleep scheduling [168]. Nodes are usually *inactive*, a state in which energy consumption is almost negligible, sensing is disabled and no incoming/outcoming transmissions can be processed. Nodes are scheduled to become active periodically to execute a certain task and become inactive again.

Tasks may include sensing, processing incoming messages, general computation and message transmission. Obviously, active nodes consume much more energy than inactive nodes. Therefore, developers try to minimize the amount of time nodes stay active. This is, however, not always possible. A generally recognized case is the one in which nodes must wait for incoming transmissions. In this situation, generally known as *idle listening*, nodes do not perform any task. However, they are required to stay active anyway. As pointed out in [173], the battery overheads produced by idle listening can be a major cause of battery deployment. Several solutions have been proposed to avoid this situation. For example, if some kind of synchronization is available on the network, communication can be performed at particular time intervals.

Localization and Synchronization

The goal of most sensor nodes is to report measurements to the sink. Data is usually routed using other nodes. Data may arrive with time delays and from a node different to the measurement source. Therefore, many applications require information to be localized and time-stamped to allow the sink to reconstruct a global image of a whole deployment region at a certain time or to analyze a restricted zone over time.

Although GPS may look as a fairly good solution for self-localization of nodes, the nature of WSNs discourages its usage. WSNs may be deployed indoors or in regions where a GPS signal is difficult to be obtained. Moreover, sensor nodes are small and energy-constrained devices, GPS localization requires a particular antenna, and the process of satellite localization is quite intensive in terms of power consumption.

Therefore, localization is usually achieved from information exchange between nodes. Consider also that consistent time-stamped data requires time-synchronization between nodes. Communication overheads produced from localization and time-synchronization depend on the precision of these methods [166, 76, 185].

Connectivity and Coverage in Unfriendly Terrain

The working of sensor networks is primarily based on the cooperation between nodes. The numerous problems caused by connectivity failures range from incomplete data to structural routing problems and the system being temporarily not operative. Depending on the WSN application at hand, the placement of nodes might be under the control of the user. This is the case, for example, when sensors are attached to vehicles, robots or other mobile entities. Consequently, nodes may sometimes find themselves in positions affected by shadowing. These wireless-unfriendly positions, which may be due to surface topology, vegetation or geographical properties, cause the wireless communication performance to be dramatically downgraded.

Also static networks precisely located by the development team may suffer connectivity problems. For example, abrupt changes in humidity or heat may reduce the coverage area of nodes. Therefore, good knowledge of the terrain and climatic conditions may help in properly dimensioning the number of nodes and the transmission ranges. In [14], the authors explore the effect of shadowing on the network connectivity in the case of randomly located nodes. Other works, such as [82, 171], study the connectivity but also the sensing performance of large sensor networks under possible node failures.

Chapter 3

Swarm Intelligence

Modern computers appeared in the middle 80s. Since the early days, computer development has been focused on increasing processor frequency, available memory, cache hierarchy, and many other performance related properties. For years, Moore's law[1] was the roadguide that manufacturers used to decide in which direction research should be focused. In Figure 3.1 we show the evolution over time of the amount of transistors of many well-known processors from the history of computing. However, the complexity of chips is no longer being pushed that fast. The technology used in processor construction is so precise (up to a nanometric scale) that improvements are every time more difficult to be achieved, to be tested, and to be introduced into industrial processes.

On the other side, software developers, with respect to the recent history of computing, expect the hardware capabilities to increase on a regular basis. Therefore, many engineering projects are developed on the edge of the current possibilities. This means that applications are developed to work perfectly on computers which actually are not available (or at least not for massive production). Under these circumstances engineers start wondering if there may be ways for obtaining a higher benefit from the current technology. It is true that hardware advances have entered in a phase of slow development, but many other technologies have historically suffered similar problems without forcing a dramatic change of the whole paradigm.

For instance, consider the case of batteries. First laptop computers used batteries which lasted for no more than one hour, being equipped with rather small monochrome screens and powered by slow processors. Current batteries keep quad-core computers running for several hours while being connected to WiFi routers and using large displays with millions of colors and hundreds of pixels per inch. Surely, lithium batteries (and other technologies) were a step forward in technological terms. However, by themselves they would have only led to a rather small percentual increase of device lifetime. The significant difference is that batteries used on computers and smartphones are now much more efficiently controlled. Hibernating techniques, reduced display backlighting in sunny conditions, processors which self-adapt to the power source reducing their performance and many other mechanisms have been developed for improving system performance without requiring essential hardware changes. As a result, software developers and hardware manufacturers are now working in the same team with the

[1] Intel co-founder Gordon E. Moore pointed out in 1965 that for seven straight years the number of transistors ensambled in integrated circuits had doubled. Moore also claimed that this trend would hold for at least another decade. As for today, the exponential rise in the number of components of microprocessors still holds year after year.

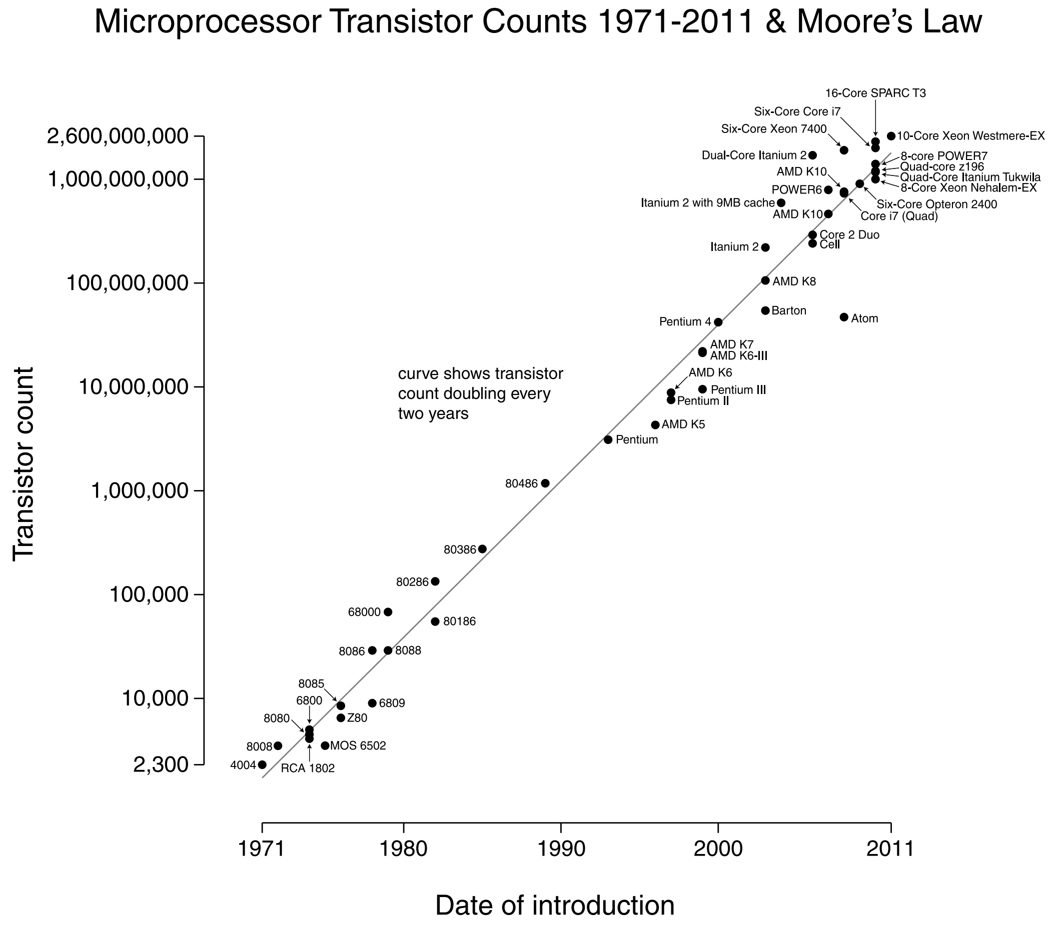


Figure 3.1: Evolution of the number of components of microprocessors over time. The x-axis shows the year of introduction of the chips. The y-axis shows the amount of components of each chip. Note that the y-axis is logarithmically scaled. The solid line is the lineal regression of the whole set of points. This figure is published under licensed Creative Commons Attribution Share-Alike 3.0. Both the original file and the data used for the image are available from [198].

same goals. The outcome of this joint work are the significant advances in battery lifetime that we have seen in the last decade.

Back to the case of information systems, the question is why cannot software become the source of improvement in computers performance during the years to come? Even before the first personal computers were build, the knowledge about algorithms' complexity was already quite advanced. Currently, software advances are generally obtained thanks to a more detailed algorithmic control. However, each new variable or subsystem results in a combinatoric explosion of possible failures and exceptions. Specially in huge systems with user interaction by means of activators or sensors, the large amount of situations that may arise can only be sustained by a similar exponential increase in the capacity of computers. And as previously reasoned, this may no longer be possible.

Some researchers have already raised these points. In [188], Malsburg justifies the need for a transformation in information technology to become more intelligent and conscious. The key role of algorithmic computing in performance is not part of his discussion. Instead it is focused on the broader field of data organization. Malsburg suggests that processor advances are now achieved with multicore technology and, consequently, large-scale massively-parallel systems composed of hundreds or thousands of processors should become much more common in the years to come. However, the current computing paradigm is not able to take profit of such systems. Real benefits of disposing of multiple cores are currently restricted to problems of explicit data-parallel or processing-parallel nature.

Deterministic control techniques are no longer able to deal with such complex systems. So, why not trying to incorporate probabilistic and distributed processes into the control mechanisms of systems? Classically, the success of applications is strongly tied to the programmers knowledge about all their aspects and characteristics. Maybe, if it was possible to avoid this need for such an insight, the barrier to new programming paradigms could be broken.

Nowadays, software projects often set themselves goals which are too complex to be handled. When complex software projects fail, responsibilities are usually assigned to developers. However, the real problem may often be found in the inadequacy of the chosen computing paradigm. Several applications in the long-term plans of governments and research institutions around the world may require for their realization an increase in the complexity of software by an order of magnitude. Is is unimaginable that the existing workforce in the system development sector will be able to handle this complexity with present methods (even considering the pace at which these evolve).

The classical programming paradigm is based on detailed algorithmic control. It is based on a division of labour between human and machine. The machine is deterministic and fast while the programmer has the creativity in the form of goals, methods, interpretation, world knowledge and diagnostic ability. Since their appearance, computers are programmed using the bottom-up approach: first, the individual parts of a system are designed and programmed, and later the resulting global behavior of the system is tested. Unfortunately, this process is error-prone and it often takes many iterations of debugging before the desired behavior is achieved. At first it may sound utopian, but maybe this process should be inverted, limiting ourselves to specifying the global behavior of the system and letting the system figure out how to achieve it.

In other words, we should be heading for information technological applications that require no less than intelligence in the machine. Systems are simply becoming too complex to be programmed in detail any longer. The principles with which programmers formulate programs inside their head have to be installed on the computer, allowing it to decide how to perform tasks while conforming to abstract, human-defined tasks. Although most of us still put in doubt the possible realization of machines able to operate under such a paradigm this is not a dream. Living systems, cells, organisms, brains, ecosystems and society are showing us the way. Living cells are not digital, are not deterministic, are not algorithmically controlled, however they are flexible, robust, adaptable, able to learn, situation-aware, evolvable and self-reproducing.

But, how can an application perform a task without its precise behavior previously being defined? Even artificial intelligence systems used in games are a set of reactions to a huge number of situations. One possible way may be that humans may learn something from certain animal societies. Think for example of a school of fish (see Figure 3.2). It is reasonable to think that fish are not continuously communicating. However, this does not prevent them

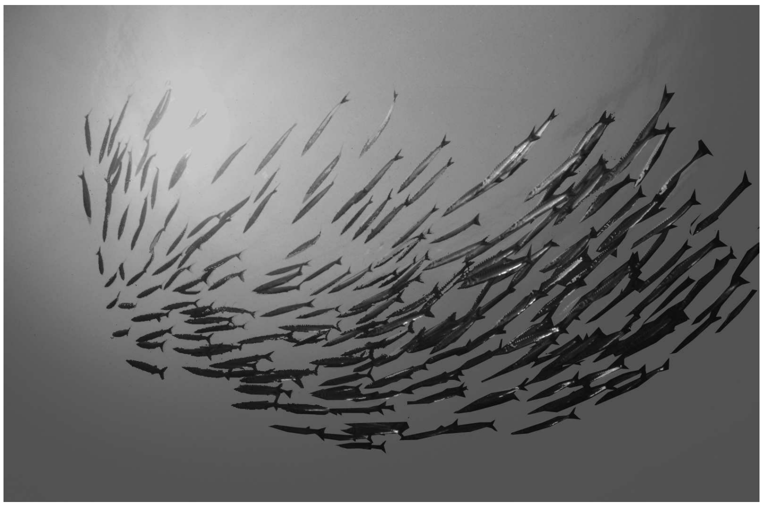


Figure 3.2: School of barracudas (*Sphyraenidae Sphyraena*). Photograph by Jordi Pérez i Grimal.

from moving as a cohesive group. The interested observer may easily note subtle changes in the trajectory of fish schools, and how these happen without breaking the group structure. Moreover, no fish seems to play a distinguished role; there is not a leader, nor a controller or a boss who is organizing the maneuvers.

Many more examples of intelligent behaviors can be found in social insects and other animal societies. Something usual, for example, is the sight of an ant colony with thousands, probably millions of ants, each one working towards its own objective, each one determined to finish with its current task. Ant colonies are an astonishing example of organization. Biologists discovered that there are different roles in ant colonies. Depending on the current needs of the colony each single ant adopts one of these roles. However, when foraging for food or constructing a new anthill, each ant has its own agenda. Individual ants schedule their own tasks and organize their time. Even though ant colonies may be populated by ants sharing common goals, each ant represents an autonomous worker. Nevertheless, ants rely on cooperation to be able to achieve complex tasks they face in their daily job routine.

Another example of an intelligent behavior can be observed in migratory ducks or geese. When flying large distances, energy consumption is the most limiting factor. Ducks are probably not aware of the most recent advances in aerodynamics, and neither are they able to perform complex calculus on how to optimize their flight routine. Nevertheless, many migratory birds (not only ducks) create a V-shape while flying in group formation (see Figure 3.3), which has been proved to be quite efficient from the aerodynamics perspective [45, 192].

Although nature has been traditionally used as an inspiring source in many artistic fields such as painting or dancing, and even in pure sciences such as mathematics or physics, engineers have traditionally disconsidered it. However, over the years nature has been able to come up with very clever solutions to incredibly complex problems. And these are not immediate ideas, these are ideas shaped over millions of years and genetically written in the DNA thanks to evolution.

More recently, and with a much more applied perspective, several researchers started in the mid 1990s to adapt nature's problem solving abilities to engineering problems. Quite soon

Figure 3.3: Migrating ducks flying in a V-shaped formation. Photograph by John Walker.

these techniques showed their ability to outperform or simplify existing solutions and the interest by the research community grew rapidly. Not much later the name *swarm intelligence* was introduced.

Swarm intelligence [112, 22, 20] is a branch of *artificial intelligence* [162] primarily focused on systems inspired by the collective behavior of social insects such as ants, termites, bees, and wasps, as well as by the behavior of other animal societies such as flocks of birds and fish schools. Swarm intelligence approaches share some common properties. Particularly important is the lack of a sophisticated controller which governs the system behavior. Such a controller is typically present in traditional engineering approaches. In contrast to bottom-up approaches, in swarm intelligence approaches global behavior emerges rather than being explicitly defined. Swarm intelligence systems are usually composed of a numerous set of simple and unsophisticated entities which are able to locally cooperate in order to achieve a common goal.

For example, ants, termites and wasps are able to build sophisticated nests in cooperation, without any of the individuals having a global master plan about how to proceed. Another example is the foraging behavior that ants exhibit when searching for food. Ants employ an indirect communication strategy via chemical pheromone trails in order to find shortest paths between their nest and food sources. Also bee colonies are able to exploit efficiently the richest food sources based on scouts that communicate information about new food sources by means of a so-called waggle dance [194]. But note that these are just some examples of many that can be found in nature.

In fact, the existence of dynamic contexts and changing environments is enough to motivate engineering applications of swarm intelligence. Swarm intelligence techniques generally show the scalability, robustness and flexibility that such scenarios demand. However, in the previous paragraphs we wanted to provide a more detailed view of the utility of swarm intelligence, and other nature-inspired techniques, for the beginning of a new era in software engineering and computing.

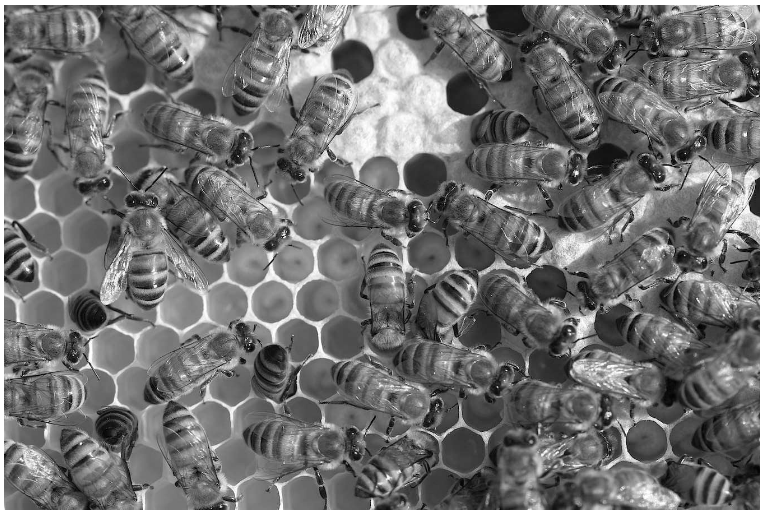


Figure 3.4: Bees taking care of larvae and eggs in their honeycomb. Photograph by Lee Langstaff.

In the rest of this chapter we will first review the basic concepts of collective behavior and self-organization in social animals' societies (Section 3.1). Afterwards we will introduce some of the most successful mechanisms and algorithms proposed under the swarm intelligence design principles (Section 3.2).

3.1 Collective Behaviour of Social Animals

The self-organized nature of social animals is based on a set of mechanisms and interactions between individuals at a local level. As a result, tasks involving the whole colony are successfully completed from a global perspective. In other words, the global behavior of a self-synchronized system emerges as a consequence of the individual acts of lower-level entities. In general, the absence of a regulating agent with a global vision of the system is the distinguishing feature of self-organized systems with respect to others where some sort of global controller is used to achieve synchronization.

In [22], the self-organization phenomenon is discussed. The authors identify four necessary properties of such systems which are summarized in the following paragraphs.

The first ingredient of self-organizing societies is the usage of *positive feedback* (or amplification) techniques. These are constituted by simple behavioral rules that foster the creation of structures. Honeycombs, as the one shown in Figure 3.4, are created cooperatively and distributedly by swarms of bees. Nevertheless, honeycombs are quite regular, made up of thousands of hexagonal storage spaces which increase the resistance of the hive. Among others, recruitment and reinforcement are well-known examples of positive feedback in nature. When foraging for food some ant species lay chemical pheromone trails on the surface of their paths which lead to a food source. These trails can later be followed by other ants when deciding which way to explore for finding food. Also bees looking for nectar use positive feedback for alerting other bess of the presence of an interesting source. However, bees use dancing (specific flying patterns) to indicate the presence of a profitable nectar source.

When the conditions that suggested to provide positive feedback change or disappear, some methods are necessary to recover a neutral state on the collective behavior. *Negative feedback* is used to counterbalance positive feedback and is achieved by means of saturation, exhaustion or competition. Many reasons may cause individuals to start providing negative feedback. For instance, in the case of foraging ants, satiation, food source exhaustion or the appearance of an alternative food source are situations in which negative feedback appears.

Although it might sound quite surprising, error and imperfection are very important for understanding the quality of some structures and solutions achieved in a self-organized manner. *Fluctuations* as random walks or spontaneous switching between the tasks performed by certain entities enable the discovery of better solutions. Again considering the case of foraging ants, the level of error that can appear in the deposited trails may result in some ants getting lost. The initial inefficiency of this phenomenon changes when one of the lost ants finds a richer or, at least, an alternative food source. The previously mentioned positive and negative feedback techniques allow ants aware of these recently discovered sources to recruit other workers for their exploitation.

The fourth necessary feature for achieving self-organization is the existence of a *tolerant* subset of individuals. Although it is true that even single ants would be able to find the shortest path to a food source if enough time is provided, self-organized systems usually rely on the contribution of many entities to a common goal. This means that ants must be able to recognize and be willing to follow trails left by other ants. However, the role of tolerance in groups must not imply an underestimation of the usefulness for a single individual of being able to distinguish its own signals or trails or the advantages that memory can provide.

In addition, as stated by [22], the most important characteristics common to most self-organized phenomenons are:

1. The alteration of an initially homogenous medium with the construction of temporary or long-term structures. Some examples of such structures are nests, trails or honeycombs.

2. More than a single stable state can appear (multistability). Remember that positive feedback may be applied depending on arbitrary decisions or fluctuations. As a result, systems may converge to a specific state neglecting the existence of the others.

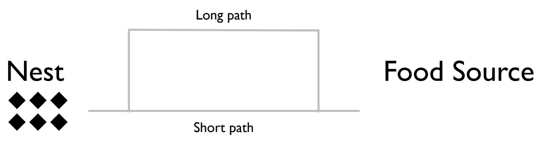
3. The emerging behavior depends on the system, environment or population parameters (bifurcations). Moreover, dramatic differences may exist between the possible behaviors at each bifurcation.

3.2 Popular Swarm Intelligence Techniques

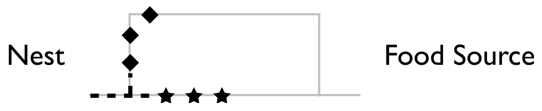
In the final section of this introduction some of the most popular existing swarm intelligence techniques will be reviewed. First, we shortly present two techniques for optimization, *ant colony optimization* and *particle swarm optimization*. Afterwards we briefly mention other relevant swarm intelligence techniques.

3.2.1 Ant Colony Optimization

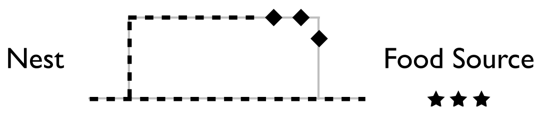
For the following description of the ACO metaheuristic we closely follow the description as given in [18]. Marco Dorigo and colleagues introduced the first Ant Colony Optimization (ACO) algorithms in the early 1990s [55, 57, 58]. The design of these algorithms was inspired by the work conducted by other researchers concerning the observation of ant colonies. Ants



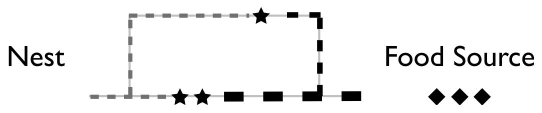
(a) Initial state with no pheromones in the environment. We consider two paths between the nest and the food source, one clearly longer than the other. Pheromone trails will be shown as pencil-like lines whose thickness indicates the pheromone concentration.



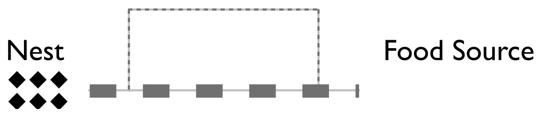
(b) When ants start foraging, the probability of chances to choose both paths are equal. Ants that take, for example, the short path are shown by star symbols.



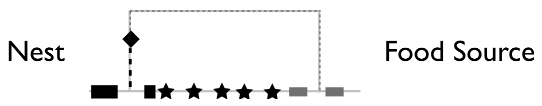
(c) Ants taking the short path are able to reach the food source first.



(d) When the ants that followed the short path start their way back, the pheromone concentration in the short path is higher and the probability to take again the short path increases.



(e) The pheromone trail on the long path has almost disappeared due to evaporation, whereas the pheromone trail on the short path prevails.



(f) Every round the probability for ants to choose the short path will significantly increase.

Figure 3.5: Experimental setting that shows the shortest path finding capability of ant colonies.

are social insects. They live in colonies and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. ACO was inspired by ants' foraging behavior and, in particular, by the ants ability to find shortest paths between food sources and their nest. Ants initially explore randomly the area surrounding their nest. While moving, ants leave a chemical pheromone trail on the ground which can be smelled by other ants. When choosing a path to follow, they tend to choose, in probability, paths marked by strong pheromone concentrations. Moreover, ants have a technique to indirectly communicate the quality of the food source. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the evaluation of the food source. The pheromone trails will guide ants leaving the nest to the best food sources. In [53] it was shown that the indirect communication via pheromone trails (known as stigmergy [78]) enables ants to find shortest paths between their nest and the food sources [53]. In Figure 3.5 a sketch of this behavior is shown.

ACO belongs to the algorithmic family of metaheuristics. Metaheuristics [21] are algorithms sharing a common skeleton. The idea of metaheuristics is to create abstract algorithms which only require to be defined in the behavior of certain functions which are dependent on the particular problem at hand, for example, a neighborhood space navigator or a solution evaluator. Sharing this common structure enables a faster and easier conception of new heuristics. The field of metaheuristics for solving combinatorial optimization problems has received a lot of attention in the last years [21]. The number of their applications has grown significantly, until becoming a huge and independent area of research. A popular example is the *Simulated Annealing* metaheuristic [116, 34].

The ACO metaheuristic is based on the cooperation between ants:

1. Population: The existence of a rather large set of simple, not selfish, entities.
2. Distributedness: No global control of each entity or the tasks performed.
3. Self-organization: Some communication channels must exist between entities.

These three properties enable the ants to accomplish cooperatively complex tasks, without the need of any control agent.

The instinctive foraging behavior of real ants is exploited in ACO in order to solve, for example, discrete optimization problems. In the ACO metaheuristic *ants* decide on the quantity of pheromone deposited, which may depend on an evaluation of the quality of the solutions found. With this additional evaluation and regulation skills, pheromone trails will guide other ants to good solutions, following, in some way, a collective criteria of all the ants working in the colony.

A useful property of the ACO metaheuristic relates to parallel computing. As the algorithm emulates the behavior of a set of ants performing the same task, this metaheuristic can be parallelized, at least, in a naive way, which consists in letting each processor emulate a different ant at the same time [177].

The rest of this section is organized as follows. Firstly, we will analyze the working of the ACO metaheuristic, and secondly we will review some of the applications of ACO algorithms.

Working of ACO

Our model of the foraging behavior of real ants in Figure 3.5 can not be directly applied to combinatorial optimization problems (CO) because it only associates pheromone trails to complete solutions of the problem. This way of modeling implies that the solutions to the considered problem are already known. In contrast, in CO the target is to find an unknown optimal (or good-enough) solution. Thus, when CO problems are considered, solutions are generally split up into several solution components and each one will have its own pheromone value associated. Each ant will choose one solution component at each step, until a full solution to the problem is assembled. Generally, the set of solution components is expected to be finite and of moderate size. $\mathcal{T}$ denotes the set of all pheromone values, more formally, $\mathcal{T} = \{\tau_i \mid c_i \text{ is a solution component}\}$. The complete ACO metaheuristic is shown in Algorithm 1.

The working of the ACO metaheuristic can be explained with a higher level description. At each iteration, one solution is constructed by each of the n_a ants. The process of constructing a solution is probabilistic. However, there exists some bias in the selection of new

Algorithm 1 ACO metaheuristic

```
1:  INPUT: An instance $I$ of a combinatorial problem $P$
2:  InitializePheromoneValues($\mathcal{T}$)
3:  while termination conditions not met do
4:      $S_{iter} \leftarrow \emptyset$
5:      for $j = 1, \ldots, n_a$ do
6:          $s \leftarrow$ ConstructSolution($\mathcal{T}$)
7:          $s \leftarrow$ LocalSearch($s$) [OPTIONAL]
8:          $S_{iter} = S_{iter} \cup \{s\}$
9:      end for
10:     ApplyPheromoneUpdate($\mathcal{T}$)
11: end while
12: OUTPUT: The best solution found
```

solution components. The construction of a solution is generally made by considering a base solution (usually empty) and adding iteratively solution components until a complete solution is reached. The bias on the selection of new components depends on the current pheromone values in $\mathcal{T}$. Initially all pheromone values are equal (although in some cases they can be initialized to stimulate a certain solution or set of components). Each time the pheromone update system is executed, the probability distribution for choosing among new solution components is updated. Generally, the components occurring in the best solutions found obtain an increase in their pheromone values. Once the pheromone distribution converges, the probability distribution used for the construction of new solutions becomes constant. As a result, further new solutions will be difficultly found and the algorithm should finish or reset its pheromone values.

As a summary, let us remark that the ACO metaheuristic uses the pheromone system to specify how the search space should be explored. Therefore, the mechanism used for updating the pheromone system plays a key role in the working of each ACO algorithm. Several pheromone update systems have been suggested in the related literature. In the following sections we review a sample ACO application as described in [18] and later introduce some well-known pheromone update systems.

Example: ACO for the TSP

The first algorithm based on the ACO metaheuristic was one for solving the well known *Traveling Salesman Problem* (see Definition 1). In the following we use an example of this algorithm developed in [18] to explain the main steps of an ACO algorithm.

Definition 1 *In the TSP a completely connected, undirected graph $G = (V, E)$ with edge-weights is given. The nodes V of this graph represent the cities, and the edge weights represent the distances between the cities. The goal is to find a closed path in G that contains each node exactly once (henceforth called a tour) and whose length is minimal. Thus, the search space $\mathcal{S}$ consists of all tours in G. The objective function value $f(s)$ of a tour $s \in \mathcal{S}$ is defined as the sum of the edge-weights of the edges that are in s. The TSP can be modeled in many different ways as a discrete optimization problem. The most common model consists of a binary decision variable X_e for each edge in G. If in a solution $X_e = 1$, then edge e is part of the tour that is defined by the solution.*

While the original behavior of ant colonies was observed for the problem of finding shortest paths, in the algorithm at hand the task of each ant will consist in constructing a feasible TSP solution. Each feasible solution is made up of several solution components. Therefore, it is mandatory to define which elements will be used as solution components. In this example, each edge of the graph becomes a solution component. With TSP solutions being, in essence, feasible tours, this definition of the space of solution components guarantees that ants will be able to build all feasible solutions. As a result, a pheromone value $\tau_{i,j}$ is introduced in $\mathcal{T}$ for each edge $e_{i,j}$ in the graph provided by the TSP problem instance.

As the TSP problem is no longer related with finding shortest paths, the previous notion of nest and food source also change. The nest becomes an initial and incomplete solution, while each different feasible solution is understood as a different food source. Moreover, *the shortest path* is now *the tour $s \in S$ such that $f(s)$ is minimum.*

Regarding the solution construction procedure of each ant, first, a single node is chosen at random and used as the starting node of the tour. Next, the ant must create a full tour in the TSP graph by iteratively visiting one unvisited node at each step. Each time a new node is visited, the traversed edge from the last visited node to the new one is added to the tour under construction. Finally, when no nodes remain unvisited, the ant closes the tour using the edge from the current node to the one used as starting node. More in detail, each construction step of the solution is performed as follows. Assuming the ant to be in node v_i and T be the set of previously visited nodes in the current tour construction, the next construction step consists in visiting a node v_j, such that $v_j \notin T$ and adding the edge $e_{i,j}$ to the tour currently under construction. This is a probabilistic step. The probability of traversing a certain edge is:

$$\mathbf{p}(e_{i,j}) = \frac{\tau_{i,j}}{\sum\limits_{\{k \in \{1,\ldots,|V|\} \mid v_k \notin T\}} \tau_{i,k}} \quad , \forall\, j \in \{1, \ldots, |V|\}, v_j \notin T \ . \tag{3.1}$$

For an example of such a solution construction see Figure 3.6.

The next step in the algorithm execution is pheromone evaporation. Once all ants of the colony have completed the construction of their solution, the following rule is applied:

$$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j} \quad , \forall\, \tau_{i,j} \in \mathcal{T} \tag{3.2}$$

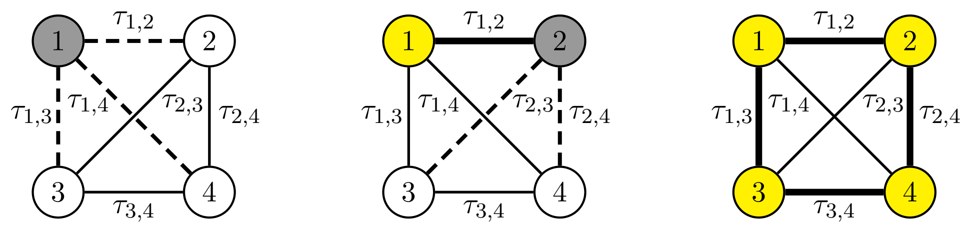
As previously explained, ants leave pheromone trails in their return trips to the nest. Hereby, an ant that has constructed a solution s leaves the following pheromone trails in each $e_{i,j} \in s$:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{C}{f(s)} \ , \tag{3.3}$$

where C is a positive constant and $f(s)$ is the objective function value of the solution s. As it can be seen in the ACO metaheuristic description (Algorithm 1), n_a ants per iteration will perform these tasks until a stopping condition (e.g., a time limit) is satisfied.

ACO variants

One of the crucial decisions in the development of an ACO-based heuristic concerns the choice of an appropriate pheromone model and pheromone update method. The method used in the previous example is known as *Ant System* (AS). AS was the first pheromone update in ACO. The Ant System is composed of two rules:



$$\mathbf{p}(e_{1,j}) = \frac{\tau_{1,j}}{\tau_{1,2} + \tau_{1,3} + \tau_{1,4}} \qquad \mathbf{p}(e_{2,j}) = \frac{\tau_{2,j}}{\tau_{2,3} + \tau_{2,4}}$$

(a) First step of the solution construction.

(b) Second step of the solution construction.

(c) The complete solution after the final construction step.

Figure 3.6: Example of the solution construction for a TSP problem consisting of 4 cities (modeled by a graph with 4 nodes; see Definition 1). Node 1 is the randomly-chosen starting node. Figures (a) and (b) show the choices of the first and the second construction step, respectively. The city where the ant is currently located is shown in dark gray, previously visited cities appear in light gray and unvisited cities are shown in white. In each case, the possible choices of the ant (i.e., the edges it may traverse) are marked by dashed lines. Underneath each graphic the probabilities (Equation 3.1) for the different choices are given. Note that after the first two choices, just one city (node 3) remains unvisited. Therefore, the ant is forced to move into node 3 and then back to node 1 to close the tour. Thick lines in figures (b) and (c) show the edges included in the partial, respectively complete, tour. Figure reproduced from [18].

- Pheromone update: $\tau_i \leftarrow (1-\rho) \cdot \tau_i$
- Reinforcement: $\tau_i \leftarrow \tau_i + \rho \cdot \sum_{\{s \in S_{iter} | c_i \in s\}} F(s)$

where $\rho \in [0,1]$ is the evaporation rate, S_{iter} is the set of solutions generated in the current iteration (being each solution a set of solution components) and F is the quality function $F : S \to \mathbb{R}^+$. Typically, when minimizing, $F(\cdot) = \frac{1}{f(\cdot)}$.

Even though the AS algorithm proved that the ants foraging behavior could be transferred into an algorithm for discrete optimization, it was generally found to be inferior to state-of-the-art algorithms. Fortunately, over the years, several extensions and improvements of the original AS algorithm have been introduced (see Table 3.1) The choice among them may depend as well on the problem characteristics. Besides, note that the pheromone model is highly related to each different problem. In the following, we give a more extended explanation of the ACO variants in Table 3.1.

- **Elitist AS (EAS)**: The EAS [55, 58] keeps track of the best solution found so far by any ant in any iteration. Together with the pheromone update performed by basic AS algorithms, EAS increases the pheromone values of the solution components which compose the best-so-far solution.

Table 3.1: A selection of pheromone update systems

ACO variant	**Authors**	**Main reference**
Elitist AS (EAS)	Dorigo	[55]
	Dorigo, Maniezzo, and Colorni	[58]
Rank-based AS (RAS)	Bullnheimer, Hartl, and Strauss	[29]
$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)	Stützle and Hoos	[178]
Ant Colony System (ACS)	Dorigo and Gambardella	[56]
Hyper-Cube Framework (HCF)	Blum and Dorigo	[19]

- **Rank-based AS (RAS)**: RAS [29] is an extension of AS that creates a ranking of all the solutions found in each iteration. This method is used to regulate the amount of pheromone that each solution may contribute to the pheromone values of its components. Solution components of the best ranked solutions receive a higher increase in their pheromone values than those occurring in poor solutions. In RAS the amount of pheromone added to each pheromone value is independent of the interval of values provided by the objective function, as it depends only on the quality of solutions with respect to other solutions.

- **$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)**: The first notable difference of $\mathcal{MM}$AS algorithms [178] is that pheromone values are restricted to the interval $[\tau_{\mathsf{min}}, \tau_{\mathsf{max}}]$, with $0 < \tau_{\mathsf{min}} < \tau_{\mathsf{max}}$. The τ_{min} value guarantees that the possibility of constructing a solution never completely disappears. Moreover, the τ_{max} value allows the algorithm to detect that due to the high pheromone values of certain solution components, the search is mostly confined to a small region of the search space. $\mathcal{MM}$AS algorithms may use this event to perform a resetting consisting in the reinitialization of all the pheromone values. Finally, updates to the pheromone system are performed using, exclusively, the iteration-best solution, the restart-best solution (i.e., the best solution found since the last restart of the pheromone values), or the best-so-far solution.

- **Ant Colony System (ACS)**: Based upon the original Ant System, Dorigo and Gambardella [56] proposed this ACO variant which aimed at improving the performance of the original system. It introduces three major differences with respect to AS. First, both pheromone update and evaporation are only applied to those solution components which appear in the best solution found so far. Second, each time a solution component is added to a solution its pheromone value is decreased. And third, in the procedure for constructing a solution, some of the steps are not performed in a randomized way. Instead, the new solution component is chosen by means of a *pseudo-random-proportional* method, which is deterministic.

- **Hyper-Cube Framework (HCF)**: The Hyper-Cube Framework developed by Blum and Dorigo [19] is an extension for ACO algorithms that can be used in combination with other pheromone update systems such as the Ant System, the Ant Colony System or the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. When pheromone updates are performed in the HCF the amount of pheromones added to promising solution components are normalized rather than depending solely on the objective function under consideration. Apart from the better performance of the system, HCF allowed the authors to make some theoretical claims about the algorithm behavior and convergence [17].

Finally, notice that ACO algorithms have been regularly used in the optimization of some WSN-related problems. Consider, for instance, routing [151], steiner tree construction [175], data aggregation in wireless sensor networks [131] or energy-efficient routes in networks [90].

3.2.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is an optimization technique inspired by the collective behavior of flocks of birds and fish schools. In 1995, James Kennedy and Russell Eberhart introduced the original PSO model [113] as a novel method for training neural networks. Although the authors initially tried to model the movement of groups of animals, after adapting the model to deal with optimization their graphical interpretation of the system resembled the one of mosquitoes. This fact gave birth to the term that the same authors coined, *particle swarm*.

In the initial experiments presented by the authors, which are concerned with the classification of the Fisher Iris Data Set [70], PSO proved to be as efficient as the traditional backpropagation method used in the neural network field. In further experiments on a data set representing electroencephalogram spike waveforms and false positives [62], the use of PSO improved the overall results. PSO was also compared against some elementary genetic algorithms (Chapter 2 of [48]) for finding the optima of the nonlinear *Schaffer f6* function, which is a difficult function due to its many local optima. In this case, PSO was reported to find the global optimum in each simulation in times comparable to those used by genetic algorithms.

The popularity of PSO among computer scientists has been growing rapidly since its introduction. In the meanwhile, the method has also become popular in more applied fields due to its simplicity, robustness and efficiency for solving hard optimization problems. The working of PSO is based on a relatively large population of entities whose individual behavior stochastically depends on their current state and the current state of neighboring nodes. Individual particles represent feasible solutions. Particles move around the problem search space seeking for efficient solutions. Each particle decides its position in accordance with its *velocity* and the current distance to the best position found by its neighbors and the best position found by itself. When the system is iterated particles increasingly focus on the areas of the search space which contain high-quality solutions.

PSO is founded on previous works in related disciplines which already pointed out that complex, organized group dynamics might arise as a result of local interactions. Some examples are the flocking model *Boids* created by Reynolds [159] or the study on the rules that govern the large synchronized flocks of birds by Heppner [86].

In the rest of this section we will first describe the working of the original PSO algorithm. Then we will mention some noteworthy extensions and improvements to the basic model, as well as the current issues under research. And finally, we will review some of the important applications of PSO with special attention to those related to WSNs.

Working of PSO

The region explored by a particle in PSO depends on its best solution found –remember that solutions are tightly linked to positions– and the best solution found by its neighborhood. More in detail, a PSO system is composed of n particles and a target function f to be minimized or maximized. Each particle has a velocity parameter that depends on its position and on those of its neighbors. The i^{th} particle is defined by the position x_i, the velocity v_i, and the

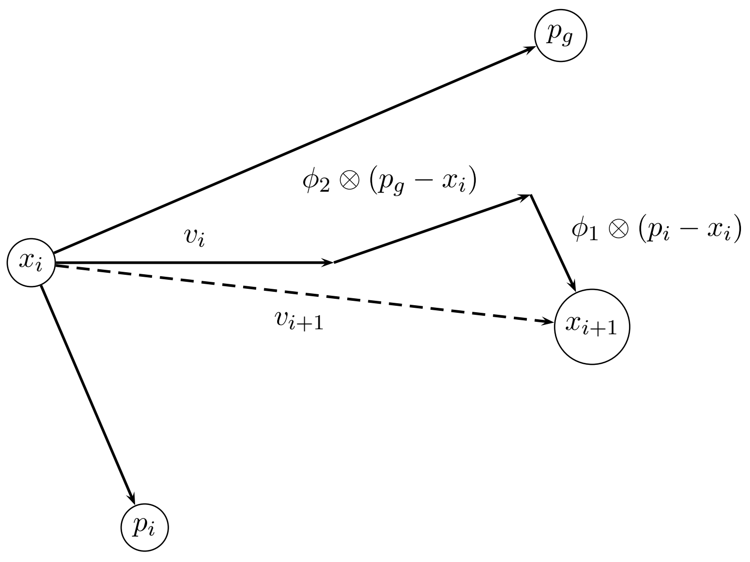


Figure 3.7: Example of a PSO particle position update. For ease of illustration a two-dimensional space is used.

personal, respectively neighborhood, best position p_i, p_g. Notice that the search space is the domain of function f. Therefore, many algorithm variables are vectors as they must cope with the possible multidimensional nature of function f. The rules for synchronously updating the velocity and the position of all particles at each iteration are:

$$v_{i+1} \leftarrow v_i + \phi_1 \otimes (p_i - x_i) + \phi_2 \otimes (p_g - x_i), \tag{3.4}$$

$$x_{i+1} \leftarrow x_i + v_{i+1}, \tag{3.5}$$

where $\otimes$ denotes the point-wise product[2] and ϕ_1, ϕ_2 are defined as follows:

$$\phi_1 = c_1 \cdot R_1, \tag{3.6}$$

$$\phi_2 = c_2 \cdot R_2, \tag{3.7}$$

with c_1, c_2 acceleration coefficients. Additionally, R_1 and R_2 are two different functions that return vectors of random values in $[0, 1]$ following a uniform distribution.

Figure 3.7 shows the different components used for the calculus of the new position of particle i.

As analyzed in [20], from Equation 3.5 it is clear that the new position of a particle strictly depends on its velocity. The velocity of a particle has three different components (Equation 3.4). The *momentum* avoids subtle changes in the traveling direction of a particle while the *cognitive*, respectively *social*, component, deviates particles to the best position visited by each particle, respectively by the neighbours of each particle. The summarized description of the PSO algorithm for finding minima is given in Algorithm 2.

The interested reader might have noticed that the neighborhood topology is an important parameter of the PSO algorithm. Some common neighborhood topologies are ring, star, and

[2]The point-wise product of two vectors $v = (v_1, v_2, \ldots, v_n)$ and $w = (w_1, w_2, \ldots, w_n)$ in the same space is defined as $v \otimes w = (v_1 \cdot w_1, v_2 \cdot w_2, \ldots, v_n \cdot w_n)$.

Algorithm 2 PSO algorithm for minimization

```
 1: INPUT: A function $f$ to be optimized
 2: Assign random positions to each particle
 3: while termination conditions not met do
 4:   for each particle $i$ do
 5:     if $f(x_i) < f(p_i)$ then
 6:       $p_i \leftarrow x_i$
 7:     end if
 8:     $p_g = \mathsf{max}(p_{neighbours})$
 9:     Update velocity (Eq. 3.4)
10:     Update position (Eq. 3.5)
11:   end for
12: end while
13: OUTPUT: The best solution found
```

von Neumann. These lower information propagation schemes are usually recommended for complex problems whereas larger neighborhoods generally achieve better results when simpler problems are concerned [142].

Extended PSO Models

Several modifications for improving the performance of PSO have been proposed. In the following we review those summarized in [20]. Some of them solve natural problems from the initial proposal while others aim at providing significant improvements to the mechanism:

- **Maximum velocity**: In some cases, specially when a particle i is far from the best solutions p_i and p_g, the velocity of particle i may become quite large. Large velocities may involve divergence between the different particles and thus prevent joint movement of the whole swarm. This issue can be tackled by setting a maximum value V_{max} for the velocity in each dimension.

- **Inertia weight**: In the selection of the new velocity, the influence of the best solutions found depends both on a set of random parameters and on their distance to the current position. However, from Equation 3.4, it is clear that the updated velocity is tightly related to the old one. In order to compensate this imbalance, a so-called inertia weight can be added to the velocity component on the right hand side of the update rule in Equation 3.4:

$$v_i \leftarrow \omega \cdot v_i + \phi_1 \otimes (p_i - x_i) + \phi_2 \otimes (p_g - x_i) \ . \tag{3.8}$$

  Apart from the quality of the final solution, the inertia weight allows to state some conditions which favor convergence [39]. Values of $\omega > 1$ may cause particles to diverge over the boundaries of the search space due to the increase of velocities over time. If, instead, values of $\omega < 0$ are used, the reduction of velocity over time causes a convergence tendency in the system. The use of a dynamic inertia weight was proposed in [63]. The proposal considers to gradually reduce the inertia weight from 0.9 to 0.4.

- **Fully Informed Swarms**: Mendes et al. [142] proposed a generalization of the original PSO model influenced by the best positions found by all nodes –instead of considering just the personal best and the neighborhood best. This influence can be captured in a new velocity update rule:

$$v_i \leftarrow v_i + \phi \otimes (p - x_i) \ , \tag{3.9}$$

  where

$$p = \frac{\sum_{k \in \mathcal{N}} r[0, \frac{\mathsf{c_{max}}}{|\mathcal{N}|}] \otimes p_k}{\sum_{k \in \mathcal{N}} \phi_k} \ , \tag{3.10}$$

  $\mathcal{N}$ is the set of particles in the neighborhood, and p_k is the best position found by the k^{th} particle so far.

- **PSO for combinatorial optimization**: Kennedy and Eberhart suggested the use of a threshold in the original PSO algorithm [114]. Depending on the position of the velocity value with respect to this threshold, x_i may be either 0 or 1. Other variants of the algorithm deal with discrete or mixed (continuous and discrete) problems [209, 38]. Direct variable discretization can also be used to adapt PSO to discrete domains. This technique was successfully applied to well-known combinatorial problems such as the knapsack problem, the traveling salesman problem and the quadratic assignment problem [38].

- **Tribes** [38]: This PSO variant is able to dynamically create and destroy particles. Groups of dependent particles are called tribes. The ability to dynamically change the network size seeks a reduction of the computational cost maintaing performance.

- **Bare-bones PSO**: This version of the original PSO algorithm proposed by Kennedy [111] replaces the velocity term with randomly selected values for each dimension. Rather than using a uniform distribution, values are selected from a Gaussian distribution centered around the average of the two attractor points (p_i and p_g) and with standard deviation between both points. Formally, Equation 3.4 is removed and Equation 3.5 is substituted by[3]:

$$x_i \leftarrow \mathcal{N}\left(\frac{p_i + p_g}{2}, \|p_i - p_g\|\right) \tag{3.11}$$

  Further studies considered the usage of different distributions [38, 160].

- **PSO systems with increased diversity**: A popular criticism against PSO is its fast convergence. Although convergence is, generally, a good property, convergence must happen once the algorithm has sufficiently explored the search space. In some systems, fast convergence may imply that the search space has not been adequately analyzed. With this purpose several authors proposed versions of the canonical PSO algorithm where convergence is reduced for the sake of a more exhaustive exploration of the search space. Some of these systems are: ARPSO [161] (attractive and repulsive PSO), which

[3]$\mathcal{N}(\mu, \sigma^2)$ is the Normal (or Gaussian) distribution with mean $\mu \in \mathbb{R}$ and variance $\sigma^2 > 0$.

switches between attraction and repulsion phases; the dissipative PSO model from [208], which focuses on increasing the algorithm randomness; or FDR-PSO [186], a PSO variant relying on the fitness-distance-ratio to dynamically adapt the particle neighborhoods and which achieves better performance than the original PSO. Other works explore the results of constraining the neighborhood topology. For example, in [142] the von Neumann neighborhood has been tested on different problems. The H-PSO [106] (Hierarchical PSO), instead, hierarchically structures the whole particle set. The resulting neighborhoods of a particle contain only their parent in the tree and the particle itself.

- **PSO for multimodal optimization**: PSO has also been applied for finding simultaneously multiple equiparable optima. Some examples are NichePSO [25] and SPSO [25, 129, 15], a niching-based, respectively speciation-based, PSO algorithm.

Applications of PSO Algorithms

There are many applications of PSO algorithms for solving tasks in sensor networks. Some of the most successful ones are the energy-aware clustering methods from [180, 206], the coverage optimization algorithm from [207], the method for optimal dynamic development introduced in [191], the algorithm for finding optimal power scheduling for decentralized detection of a deterministic signal in a wireless sensor network with correlated observations [200], and the method for both clustering in static networks and dynamic reallocation in mobile networks introduced in [205] (obtaining improvements in coverage and energy consumption).

3.2.3 Other relevant examples of SI systems

Although ACO and PSO are the most popular SI techniques, there are others that have been successfully applied to different fields of engineering and computing. In the following we summarize two of them: *firefly synchronization* and *division of labour*.

Firefly Synchronization

Fireflies provide one of the most notable examples of synchronization in nature [176, 27, 26, 85]. At night, in certain regions of southeast Asia, thousands of male fireflies congregate in trees and flash synchronously. Mutual synchronization occurs in many other natural phenomena. Note, for instance, the pacemaker of cells of the heart [99, 104, 181, 155], networks of neurons in the circadian pacemaker [64, 153] and hippocampus [182], and many more. For futher information we refer to [201, 203, 202].

Mirollo and Strogatz [146] extended the seminal work by Winfree [201], focusing on the mathematical analysis of mutual synchronization. They modeled the behavior of these natural phenomena using pulse-coupled oscillators. The model of coupled oscillators of fireflies' synchronized flashing has layed the foundations for several algorithms for activity synchronization in sensor networks (see, for example, [101, 100]). More recently, Lucarelli and Wang [134] showed that this behavior also appears when nodes are located in multihop topologies. Other works, such as [101], showed that these mechanisms work not only in simulations but also in real networks.

Division of Labour

Another important category of algorithms is related to division of labour in ant and wasp colonies. The daily routine of colonies includes several different tasks. Although we know that ants perform most tasks cooperatively, the concept of a specialized worker is clearly present in ant colonies. In the eighties, Wilson studied colonies of the *Pheidole* ant and observed that division of labour allows the colony to adapt to their changing needs [199]. The working force of ant colonies is divided in two groups of individuals: *minors*, in charge of routinary tasks, and *majors*, who work on more on-demand tasks such as nest defense or food storing. The studies by Wilson concluded that externally decreasing the amount of minors, resulted in some major workers switching to become minors.

In the late nineties, Theraulaz [179] and Bonabeau [23] developed the response threshold model for simulating division of labour in insect colonies. Roughly, the model uses thresholds to specify the level of specialization of tasks and workers. Moreover, each individual task emits a certain stimulus that depends on its relevance. After evaluating thresholds and stimulus, ants are able to accept or reject different available tasks.

Many applications of the division of labour principle have been proposed. We mention, for example, the work by Campos et al. [31] on dynamic task allocation considering trucks and facilities. Other noteworthy works are the management algorithm for peer-to-peer networks by Sasabe et al. [165] or the task allocation algorithm for computing systems with reconfigurable components by Merkle et al. [143]. Furthermore, division of labour has also been used in WSN-targeted applications such as the topology control algorithm from [105] and the architectural paradigm for sensor/actuator networks proposed in [123].

Part II

Management techniques for WSN

Chapter 4

Self-Synchronized Duty-Cycling

Ants are generally believed to follow an intensive work routine. Numerous tales and fables refer to ants as conscientious workers. Nevertheless, in the mid 1980's scientists discovered that ants do not work untiringly all day long. On the contrary, they stop and rest for extended periods of time (see [87, 73, 41]). For example, nest workers of the species *L. acervorum* rest about 72% of their time. Moreover, ant species studied in [73, 74, 145] show synchronized activity phases, that is, the ants of a colony work, respectively rest, more or less at the same time. Interestingly, no external signals have been found as a cause for this colony-level synchronization. This suggests that synchronization is achieved by self-organizing processes [43, 42]. Inspired by these works, Delgado and Solé [51] introduced a *fluid neural network* model that simulates the synchronization behavior of ants. The model shows that synchronized activity phases may help to increase the efficiency of the colony in certain situations, for example, for information dissemination. The self-synchronized duty-cycling mechanism for sensor networks that we propose in this work is strongly based on the model proposed by Delgado and Solé.

Self-organization is a process in which pattern at the global level of a system emerge solely from interactions among the lower-level components of the system. Moreover, the behavior of the lower-level components is exclusively governed by local information, without any knowledge about the global state of the system. Apart from the self-synchronized resting of ant colonies, examples of self-organization in biology include the shortest-path finding behavior of ant colonies, the synchronization of fireflies, and the mound construction of termites. We refer the interested reader to [22, 20] for additional information. More and more research fields discover self-organizing processes to be valuable alternatives for the management and control of large-scale systems. A prominent example is research on wireless ad-hoc networks such as, for example, sensor networks. Several authors have recently pointed out the benefits of self-organization for management tasks in sensor networks (see, for example, [44, 61, 144, 4]). Sensor networks generally consist of a large number of small computing elements being equipped with one or more sensors for different physical measures such as light intensity, humidity, and temperature. Mostly they are aimed at monitoring large areas for extended periods of time. Due to several reasons, conventional engineering paradigms may not be very well suited for the control and management of sensor networks. One of the problems is the current lack of standards. Available hardware is very heterogeneous, with a broad spectrum of suppliers and products that are characterized by different processors, memory capabilities, the use of different network protocols, etc. This diversity obstructs research because most of these devices have problems to interact with devices from other suppliers. Furthermore, quite a few sensor network applications require deployment in remote areas such as forests, deserts or, for example, high ceilings of city buildings [141]. Therefore, self-organization has been discovered as one of the possible ways to control and manage sensor networks.

In many application scenarios sensor nodes can not be easily supplied with energy. Moreover, sensor nodes may be mobile. Therefore, most sensor nodes are equipped with batteries [183]. This makes energy a scarce resource which is a restricting factor for the network lifetime. The problem of extending the network lifetime has been dealt by means of different approaches. Hardware advances, for example, try to increase the capacity and the efficiency of batteries. However, the increase in battery capacity has been much slower than the increase in energy requirements [156]. Therefore, some approaches extend network protocols with energy-aware techniques and algorithms [149, 190, 170]. Other interesting solutions are those based on the use of so-called duty-cycles. Unfortunately, sensor nodes consume a large amount of energy when being active and even more when transmitting information. However, measurements have usually to be taken at regular time intervals in which two consecutive measurements are not too close in time. This means that sensor nodes that are active between two measurements consume most of their energy being idle. Therefore, the idea of duty-cycling is to devise a mechanism in which, between two measurements, sensor nodes can switch into a state in which they consume little energy [168]. A more recent approach for energy-saving considers the inclusion of energy harvesting capabilities in sensor nodes, allowing the system to recharge the batteries if possible [158, 157].

Duty-cycling and energy harvesting can also be combined. In the related literature, several authors have studied efficient duty-cycling mechanisms for sensor networks with energy harvesting capabilities [109, 102]. The better performing approaches have to be fed with data about the environment. They allow more energy consumption in those periods of time in which a larger amount of energy is expected to be available from the environment. Unfortunately, two problems are related to these systems. First, data about typical environmental conditions must be obtained before the network is deployed; second, there are regions with considerable changes in environmental conditions and, thus, deciding the actual conditions based on previous data may be inadequate. A recent attempt to solve this problem is presented by Vigorito [187]. This work introduces a technique that uses adaptive control techniques to prescind from the environmental profile. Results show a better network performance and a longer network lifetime. However this approach considers nodes separately, and the lack of synchronicity may cause a considerable downgrade in performance for several types of applications.

Our Contribution

The work presented in this chapter describes a duty-cycling mechanism for sensor networks with energy harvesting capabilities. This mechanism is inspired by the self-synchronization behavior of ant colonies described above. In contrast to related work from the literature, our mechanism does not need any prior information about the environment. The first contribution of this chapter is a study focused on laying the foundations of a self-synchronized duty-cycling mechanism rather than developing a protocol that can directly be employed on available hardware. Hence, neglecting most physical and network constraints.

The second part of this chapter covers the adaptation of the previous mechanism to a real environment where several network variables –such as a MAC layer, routing queues, or packet collisions– are considered. As shown in [96] the reformulated mechanism can be run on the real sensor network simulator Shawn [68, 172].

Most of the contributions of this chapter have been extensively discussed in the scientific community before writing this thesis. With this aim, contributions have been presented in international conferences and journals. The reports provided by the referees have been read carefully. Each and every one of the concerns and comments of the referees has been handled and solved in order to give a broader and more complete explanation of our work. The following list contains all our publications related to novel techniques introduced in this chapter. Remember that a more extended description of the publications is available in Section 1.2. Also note that this chapter is based on the article *Foundations of* ANTCYCLE*: Self-Synchronized Duty-Cycling in Mobile Sensor Networks* [91] which was published in *The Computer Journal*:

- Hugo Hernández, Christian Blum, Martin Middendorf, Kai Ramsch and Alexander Scheidler. Self-Synchronized Duty-Cycling for Mobile Sensor Networks with Energy Harvesting Capabilities: A Swarm Intelligence Study. In Y. Shi editor, *SIS 2009 – Proceedings of IEEE Swarm Intelligence Symposium (SIS).* Pages 153-159. IEEE Press, 2009.

- Hugo Hernández and Christian Blum. Self-Synchronized Duty-Cycling in Sensor Networks with Energy Harvesting Capabilities: the Static Network case. In *GECCO 2009 – Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation.* Pages 33-40. ACM Press, New York, 2009.

- Hugo Hernández and Christian Blum. Asynchronous Simulation of a Self-Synchronized Duty-Cycling Mechanism for Mobile Sensor Networks. In *Proceedings of BADS 2009 – Proceedings of the Workshop on Bio-Inspired Algorithms for Distributed Systems.* Pages 61-68. ACM Press, New York, 2009.

- Hugo Hernández and Christian Blum. Foundations of ANTCYCLE: Self-Synchronized Duty-Cycling in Mobile Sensor Networks. *The Computer Journal*, 54(9):1427-1448, 2011.

- Hugo Hernández, Christian Blum, Maria Blesa, Tobias Baumgartner, Sandor Fekete, Alexander Kröller. Self-Synchronized Duty-Cycling For Sensor Networks With Energy Harvesting Capabilities: Implementation in Wiselib. In *MSN 2010, Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks.* Pages 134-139, IEEE Press, 2010.

Chapter Organization

The outline of this work is as follows. In Section 4.1 we shortly outline the model of ants' self-synchronization behavior as introduced by Delgado and Solé in [51, 50] and analyze the interaction between its most significant parameters. In Section 4.2 we present the adaptation of this model to duty-cycling in sensor networks. Furthermore, in Section 4.3 the proposed system is extended for dealing with energy harvesting capabilities. Section 4.4 offers an extensive experimentation and the presentation of results. Finally, in Section 4.5 we discuss the adaptation of the proposed system to real sensor networks, and in Section 4.6 we present conclusions and an outlook to future work.

4.1 Simulating Ants' Self-Synchronization Behavior

In this section we present an extensive study of the *fluid neural network* (FNN) model that was proposed by Delgado and Solé in [51] for explaining the self-synchronization capability of ant colonies. First, the model is described. Then, after an outline of the experimental setup, a study of the influence of different parameters is presented.

4.1.1 Fluid Neural Network Model

In the FNN model, k automata are located in a two-dimensional $L \times L$ grid with periodic boundary conditions. The automata evolve over time. Each automaton is described by a continuous state variable S_i and a binary variable a_i. The values of these variables at time step $t \in \mathbb{N}$ are denoted by $S_i(t) \in \mathbb{R}$ and $a_i(t) \in \{0,1\}$. In case $a_i(t) = 1$ automaton i is called *active* at time step t, *inactive* otherwise. The value of a_i at time step t is determined as follows:

$$a_i(t) := \Phi(S_i(t) - \theta_{\text{act}}) \ , \tag{4.1}$$

where θ_{act} is the activation threshold, and $\Phi(x)$ is defined as follows:

$$\Phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

At each time step $t \in \mathbb{N}$, the value of the continuous state variables S_i is updated as follows:

$$S_i(t) := tanh[g \cdot h_i(t)] \ , \tag{4.3}$$

where g is a gain parameter and $h_i(t)$ is the amount of stimulus received by automaton i at time t. The amount of stimulus is defined as follows:

$$h_i(t) := S_i(t-1) + \sum_{j \in N(i,t), j \neq i} S_j(t-1) \ , \tag{4.4}$$

where $N(i,t)$ is composed of the automata that are located on grid positions that are part of the Moore neighborhood of size 1 of the grid position on which automaton i is located. See Figure 4.1 for a graphical illustration of the Moore neighborhood.

There are two ways for an inactive automaton i to become active: (1) either due to local interactions (see Eq. (4.3)) or (2) by spontaneous activation. More specifically, if i is inactive at time step t, it may be awakened with probability p^a. If this is the case, the value of its continuous state variable S_i is set to an initial activity level $S^a > 0$, that is, $S_i(t) := S^a$. In addition, at each time step t, active automata either stay at their current location, or they decide to move to one of grid positions in the Moore neighborhood of size 1 of their current location. This decision is done probabilistically without preference. In Algorithm 3 is shown how all the elements described above are coordinated.

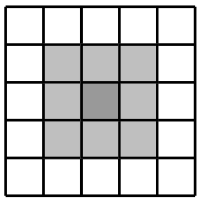


Figure 4.1: The Moore neighborhood of size 1 that is used in the FNN model. Automata located on light-gray shaded grid positions are neighbors of the automaton located on the dark-gray grid position.

Algorithm 3 Fluid Neural Network proposed in [51].

```
Input: A number of simulation steps $m$
Initialize states: $S_i(0) := S^a$, $i = 1, \dots, k$
for $t = 1, \dots, m$ do
  for all automata $i = 1, \dots, k$ do
    Calculate $a_i(t)$ (see Eq. 4.1)
    if $a_i(t) = 0$ then
      Draw a random number $p \in [0, 1]$
      if $p \leq p^a$ then
        $S_i(t) := S^a$
        $a_i(t) := 1$
      end if
    end if
  end for
  for all automata $i = 1, \dots, k$ do
    if $a_i(t) = 1$ then
      Decide randomly to stay at the current location or to move to a neighbour site
      Transmit $S_i(t-1)$ to all automata in $N(i, t)$
      Calculate $S_i(t)$ (see Eqs. 4.3 and 4.4)
    end if
  end for
end for
```

Finally, the behavior of the system is mainly characterized by the evolution of the mean system activity over time, which is for each time step $t \in \mathbb{N}$ defined as follows:

$$A(t) := \frac{1}{k} \sum_{i=1}^{k} a_i(t) \in [0, 1] \tag{4.5}$$

Note that the more automata are active at time t the greater is $A(t)$.

4.1.2 Experimental Setup

We now describe the experimental setup that will be used for all experiments conducted in this section, as well as in later sections (except when otherwise indicated). Assuming that any two time steps are separated by one minute, one day of simulation will consist of 1440 time

Table 4.1: Reference parameter setting for the FNN model.

K	L	S^a	θ_{act}	g	p^a
120	25	0.01	10^{-16}	0.1	0.001

steps. For each experiment we applied the FNN for 10080 time steps, which represents seven days of simulation. The first day is only used for stabilization, that is, no data is registered during the first 1440 time steps.

In [51], Delgado and Solé were able to reproduce the behavior of ants of the species *L. acervorum* with the parameter settings as given in Table 4.1. We reproduced this experiment and present the results in Figure 4.2 in terms of the evolution of the mean system activity ($A(t)$) over time. Note that the oscillating value of $A(t)$ over time indicates the (self-)synchronization of the automata (respectively, the modelled ant colony). In addition, the average system activity for the six days of simulation matches the one observed in real ant colonies (see [74]), in both cases being around 30%.

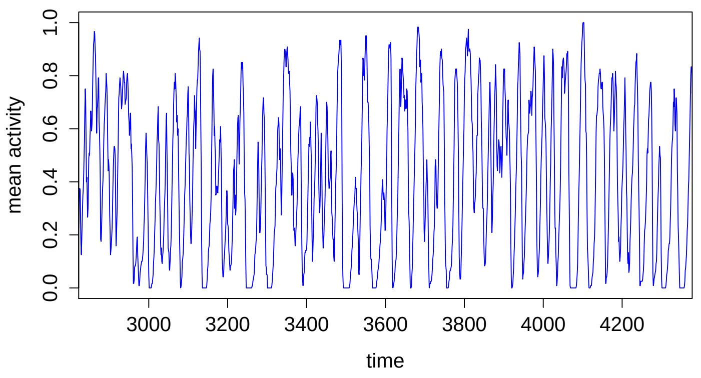


Figure 4.2: Evolution of the mean system activity ($A(t)$) of the FNN over time. Parameter settings are given in Table 4.1.

As mentioned before, our aim is to devise a mechanism for self-synchronized duty-cycling in sensor networks based on this FNN model. Moreover, we aim at a system that automatically adapts to changing energy availabilities. In the following we will therefore present a profound study of the influence of different parameters on the behavior of the FNN model. Moreover, we will show how changes in parameter settings induce changes in the behavior of the model. This study will help us, first, in understanding this complex system, and second, it will help us in devising different duty-cycling schemes that are tailored for different sensor network applications. Our study concerns an exploration of the following parameters:

- The gain parameter g (see Eq. 4.3): this parameter might have an influence on the speed at which values of state variables decrease.
- The probability of spontaneous activation p^a: together with the initial activity level

after spontaneous activation (S^a) and the activity threshold θ_{act}, parameter p^a should have an important influence on the number of activity peaks that can be observed in the evolution of the mean system activity. As all three parameters are highly related, we decided to keep S^a and θ_{act} fixed, whereas p^a is variable.

- The size of the considered neighborhood: in order to study the influence of different neighborhood sizes on the system's behavior, we experimented with different neighborhood definitions.

For each of the above-mentioned parameters we will present a study of its influence on the system behavior. Results will be shown in terms of various measures:

- The evolution of the *mean system activity* ($A(t)$) over time.
- The average fraction of time (over the whole simulation) that an automaton has been active. Note that this can be computed by averaging $A(t)$ over all time steps of the simulation. Henceforth we refer to this measure as the *activity of the system*.
- As shown, for example, in Figure 4.2, the evolution of the mean system activity over time is characterized by peaks and valleys. With this in mind it is possible to measure the average *height of activity peaks* and the *average depth of the valleys* between two peaks. Both measures refer to an average over all the time steps of a simulation.
- Finally, we also measure the *average number of peaks* and the *average width of peaks*, again refering to the average over all time steps per simulation. Hereby, the width of a peak is defined as the number of time steps between the preceeding and the succeding valley. Occasionally we will refer to this measure also as the *length of an activity phase*.

The above mentioned measures are always shown in graphical form, and additionally also in numerical form. Finally, let us mention that, apart from graphics that show the evolution of the mean system activity over time, all other graphics were produced on the basis of 50 independent runs of the system.

4.1.3 Study Concerning Gain Parameter g

Given Eq. 4.3, it is reasonable to assume that the gain parameter g regulates the speed (in terms of the number of iterations) in which automata will turn inactive after activation. In order to confirm this we conducted a series of experiments with varying values of g. More specifically, we used $g \in \{0.0, 0.02, 0.04, \dots, 0.4\}$. The experimental setup is the same as described in Section 4.1.2, and the remaining parameter settings, apart from g, are given in Table 4.1. In Figures 4.3(c) and 4.3(e) we show the behavior of the system for $g = 0.05$, respectively $g = 0.2$. Remember that the reference value for g was 0.1. Indeed, the graphics confirm our initial assumption. With $g = 0.05$, the length of the active phases decreases. Moreover, activity peaks are lower than in the case of $g = 0.1$. This is due to the fact that distant nodes are not able to wait for each other to wake up. On the other hand, when $g = 0.2$ the length of the active phases increases and, moreover, the height of the activity peaks also increases (see Figure 4.3(e)).

It is important to note that the increase in width and height of the activity peaks also results in an overall higher activity of the system. Figure 4.4(a) shows the activity of the system as a function of g. In addition, the average height of activity peaks and the average depth of valleys are also shown as a function of g in the same graphic. In Figure 4.4(b), respectively Figure 4.4(c), is shown the average number of peaks, respectively the average width of peaks, as a function of g. Note that in these three figures the value $g = 0.3$ represents a phase transition between two different system behaviors. For values $g \geq 0.3$, once active, automata do not easily return to the inactive state because they receive too much stimulus from other automata and, hence, activity peaks are higher and longer. Note that in the extreme case just one long peak covers the whole simulation time. In particular, the measurements obtained with $g = 0.3$ are characterized by very large standard deviations. Notice in Table 4.2 that the number of peaks is much smaller than in the case of $g = 0.28$ and at the same time much higher than in the case of $g = 0.32$. In this situation almost all nodes are active during the whole simulation time, but in a few occasions some nodes switch to the inactive state for a few time steps (the average system activity is 0.997745 and the average depth of valleys is 0.955654). Table 4.2 shows the results in numerical form.

4.1.4 Study Concerning the Probability of Spontaneous Activation (p^a)

While parameter g is responsible for the shape of the activity peaks, it is reasonable to assume that parameter p^a can be used for controlling the density of the peaks in time. This is because with fewer spontaneous activations of automata, the probability for activity peaks to arise drops. In fact, this assumption is correct, as shown in Figure 4.5(b) for a case with a lower value of p^a than the one in the reference settings ($p^a = 0.00025$) and in Figure 4.5(f) for a case with a higher value ($p^a = 0.004$). Clearly, the peak density in Figure 4.5(b) is lower than the peak density in Figure 4.5(f).

We applied the FNN for all values of p^a from $\{0, 0.0005, \ldots, 0.009, 0.0095\}$. Figure 4.6 shows the results for all these runs in the same way as in the case of the study concerning parameter g presented in the previous section. Note that the probability of spontaneous activation (p^a) is related in a logarithmic manner to the average system activity, the average number of peaks, height of peaks and depth of the valleys. In other words, in case of a linearly increasing value of p^a, the effect on the system behavior is each time decreasing. In any case, an increasing value of p^a leads to an increase in average system activity. As in the case of parameter g, these results are also given in numerical form in Table 4.3.

In order to study the correlation between parameters g and p^a, we applied the FNN for all different combinations of parameter values, that is, for all (g, p^a) from $\{0, 0.02, \ldots, 0.36, 0.38\} \times \{0, 0.0005, \ldots, 0.009, 0.0095\}$. Figure 4.7 shows the mean activities obtained in the corresponding experiments in a graphical way. Hereby, each mean activity is mapped to a gray level in the following way. An activity of zero is mapped to gray level 0.0. Moreover, all activities in $(i - 0.1, i]$ are mapped to i, for all $i = 0.1, 0.2, \ldots, 1.0$. Therefore, regions with the same colour are produced by couples of parameter values which lead to comparable average system activities. Remember that for the behavior of the FNN as displayed in Figure 4.2 parameter values $p^a = 0.001$ and $g = 0.1$ have been used. As expected, this couple of values is located in a region with an average system activity in $[0.3, 0.4)$ (see Figure 4.7). Generally, similar parameter settings result in comparable system behaviors for what concerns the average system activity, but—at the same time—these changes may slightly increase or decrease the average system activity. Summarizing, this experiment may help in identifying appropriate changes in the parameter values for obtaining different average system activities without changing the shape of the activity peaks.

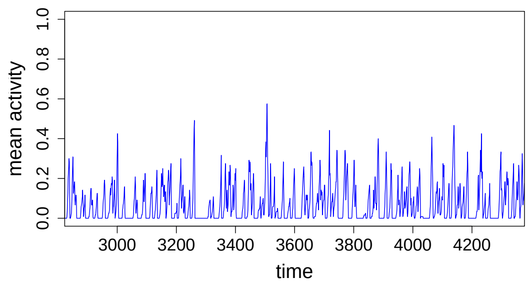
Notice that the graphic of Figure 4.7 also shows that some regions of the parameter domain

Table 4.2: System behaviour as a function of g (in numerical form).

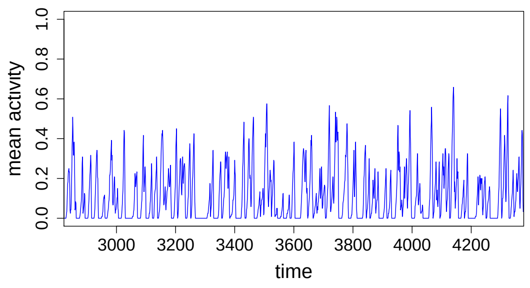
g	**system activity**	**num. peaks**	**peak width**		**peak height**		**valley depth**	
			average	**std**	**average**	**std**	**average**	**std**
0.0	0.001000	866.75	9.957658	6.081839	0.008905	0.001067	0.000000	0.000000
0.02	0.103806	700.10	12.352730	5.131667	0.271804	0.100039	0.065574	0.073027
0.04	0.171583	648.85	13.309465	5.524104	0.398166	0.128090	0.116072	0.119022
0.06	0.241030	607.30	14.226825	6.305007	0.514558	0.148225	0.184839	0.175962
0.08	0.313578	603.65	14.324735	7.036690	0.616273	0.156727	0.267490	0.231205
0.1	0.384835	596.50	14.475030	7.650189	0.713812	0.157414	0.358571	0.288975
0.12	0.456600	624.75	13.827640	8.164415	0.791272	0.149047	0.468442	0.326493
0.14	0.527644	655.20	13.170835	8.243757	0.839356	0.148427	0.549172	0.341591
0.16	0.582746	692.50	12.462020	8.328760	0.865705	0.147216	0.621856	0.339232
0.18	0.641850	770.65	11.213620	7.788833	0.866107	0.160289	0.673595	0.317632
0.2	0.705207	812.00	10.642116	7.555536	0.862662	0.166531	0.709684	0.286974
0.22	0.770005	869.00	9.942079	7.431916	0.860802	0.165265	0.744899	0.252691
0.24	0.832253	935.95	9.218147	6.986033	0.871947	0.146808	0.788468	0.206964
0.26	0.892882	917.80	9.408587	7.497063	0.890035	0.129293	0.830277	0.170012
0.28	0.949724	757.25	11.371255	10.346842	0.928994	0.088881	0.887499	0.116859
0.30	0.999301	63.95	424.473955	553.511940	0.991188	0.011119	0.928796	0.095434
0.32	1.000000	1.10	9161.700000	0.000000	1.000000	0.000000	0.049583	0.049583
0.34	1.000000	1.00	10079.000000	0.000000	1.000000	0.000000	0.000000	0.000000
0.36	1.000000	1.00	10079.000000	0.000000	1.000000	0.000000	0.000000	0.000000
0.38	1.000000	1.00	10079.000000	0.000000	1.000000	0.000000	0.000000	0.000000

Table 4.3: System behaviour as a function of p^a (in numerical form).

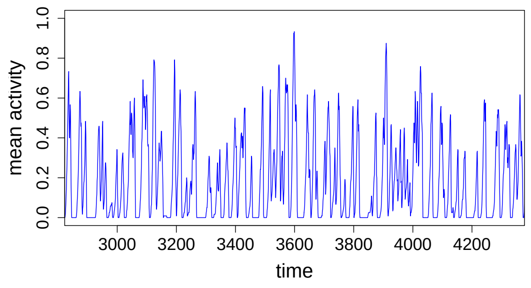
p^a	**system activity**	**num. peaks**	**peak width**		**peak height**		**valley depth**	
			average	**std**	**average**	**std**	**average**	**std**
0.0e+00	0.002596	0.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5.0e-04	0.253731	410.70	21.043815	11.229830	0.661769	0.164758	0.266439	0.270162
1.0e-03	0.385056	603.80	14.308400	7.578235	0.709743	0.155103	0.358297	0.285288
1.5e-03	0.466959	730.45	11.829770	6.510177	0.742673	0.149786	0.430713	0.281512
2.0e-03	0.518951	789.40	10.942415	5.930112	0.768910	0.145675	0.471673	0.281624
2.5e-03	0.560432	829.70	10.418825	5.571964	0.790301	0.142297	0.507839	0.278614
3.0e-03	0.594247	868.25	9.953564	5.332098	0.805478	0.137431	0.535372	0.273640
3.5e-03	0.622923	899.85	9.604948	5.255778	0.820504	0.133013	0.561153	0.270228
4.0e-03	0.642381	916.15	9.432239	4.992922	0.829570	0.130621	0.577958	0.265658
4.5e-03	0.657663	939.45	9.206944	4.962918	0.839562	0.125103	0.597668	0.260923
5.0e-03	0.672819	951.10	9.082183	4.884656	0.847097	0.121990	0.608578	0.259201
5.5e-03	0.688698	951.05	9.094509	4.902898	0.856861	0.118268	0.621955	0.254377
6.0e-03	0.701389	976.90	8.846738	4.772004	0.859158	0.117021	0.633023	0.249062
6.5e-03	0.710483	974.05	8.875148	4.730330	0.867832	0.115722	0.641728	0.253082
7.0e-03	0.720589	965.05	8.960046	4.769763	0.872024	0.114185	0.646461	0.250785
7.5e-03	0.729345	969.15	8.916217	4.811541	0.882102	0.106784	0.659718	0.249752
8.0e-03	0.738944	980.90	8.810171	4.787113	0.883308	0.107846	0.666492	0.246917
8.5e-03	0.742559	969.20	8.920053	4.815140	0.890834	0.102814	0.670474	0.250320
9.0e-03	0.749659	984.85	8.772679	4.782357	0.887795	0.105400	0.675316	0.243724
9.5e-03	0.756470	976.81	8.841410	4.786449	0.893926	0.102695	0.679667	0.244062



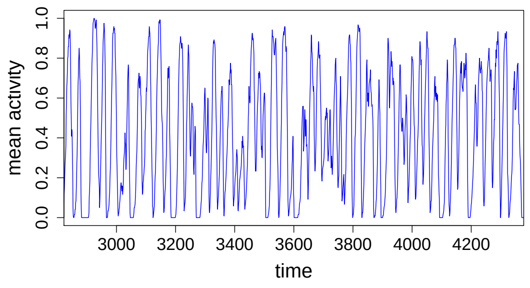
(a) Behavior of the FNN model with $g = 0.0125$



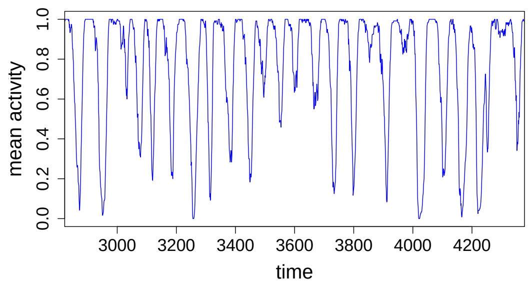
(b) Behavior of the FNN model with $g = 0.025$



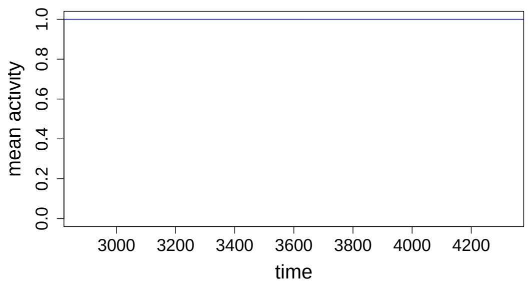
(c) Behavior of the FNN model with $g = 0.05$



(d) Behavior of the FNN model with $g = 0.1$



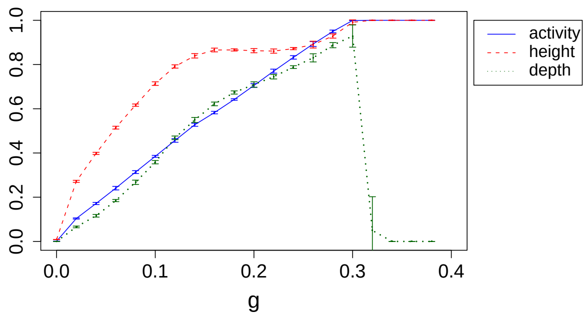
(e) Behavior of the FNN model with $g = 0.2$



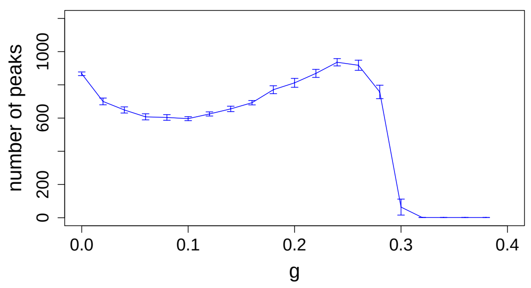
(f) Behavior of the FNN model with $g = 0.4$

Figure 4.3: System behavior for selected values of g

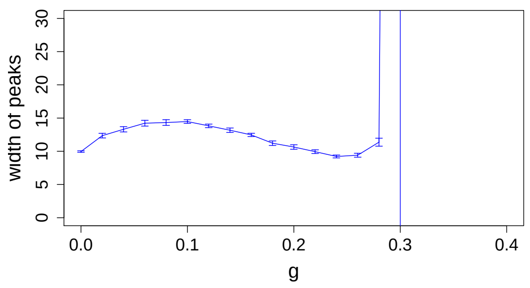
are unstable in the sense that small changes in the parameters may lead to rather considerable changes in the average system activity. As an example observe the region defined by $0.15 \leq g \leq 0.3$ and $0 \leq p^a \leq 0.0005$. On the contrary, if only the average system activity matters (in contrast to the shape of the activity peaks), considering values of $p^a \in [0.003 \leq p^a \leq 0.004)$ results in systems that are not so sensitive to small changes in the values of p^a and g. Considering values in such regions could be helpful when trying to obtain and maintain a



(a) Average system activity, peak height and valley depth



(b) Average number of peaks



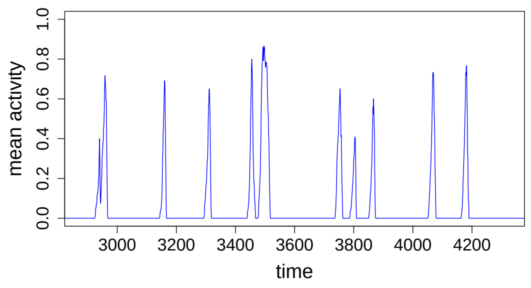
(c) Average width of peaks

Figure 4.4: System behavior as a function of g

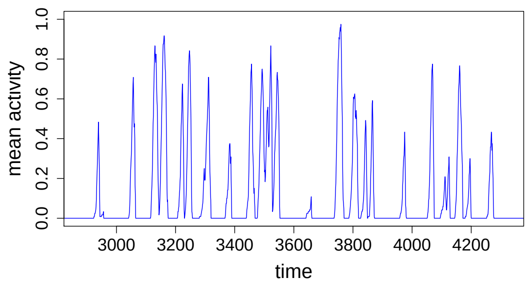
rather fixed amount of average system activity.

4.1.5 Study Concerning Neighborhoods of Different Sizes

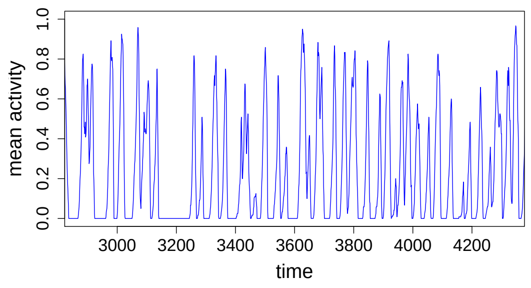
In addition to a change in the value of parameters g and p^a we also studied the effect of neighborhoods of different sizes on the behavior of the FNN. The reason is the assumption that the neighbourhood size is directly related to the height of the activity peaks. This is because the neighbourhood size clearly defines the degree of relation between different automata. In order to confirm that, a series of experiments has been performed with the six different neighborhoods as shown in Figure 4.8. In this context remember that $N(i, t)$ was used in Section 4.1.1 to refer to the set of automata that—at time t—are located at grid postitions of the Moore neighborhood of size 1 of the grid position on which automaton i is located. Note that neighborhood N_3 as shown in Figure 4.8(c) corresponds to the Moore neighborhood of size 1. Also note that $|N_1| < |N_2| < \ldots < |N_6|$, where $|N_i|$ refers to the size of the respective neighborhoods in terms of the covered grid positions. In other words, the sizes of the considered neighborhoods are strictly increasing from N_1 to N_6: $|N_1| = 1$, $|N_2| = 5$, $|N_3| = 9$, $|N_4| = 13$, $|N_5| = 21$, and $|N_6| = 25$.



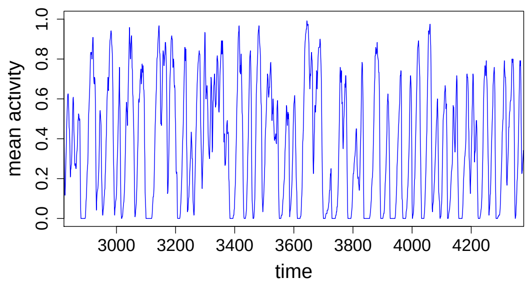
(a) Behavior of the FNN model with $p^a = 0.000125$



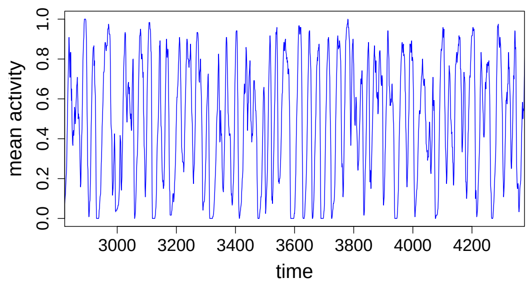
(b) Behavior of the FNN model with $p^a = 0.00025$



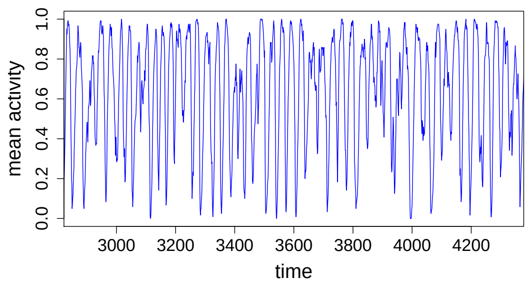
(c) Behavior of the FNN model with $p^a = 0.0005$



(d) Behavior of the FNN model with $p^a = 0.001$



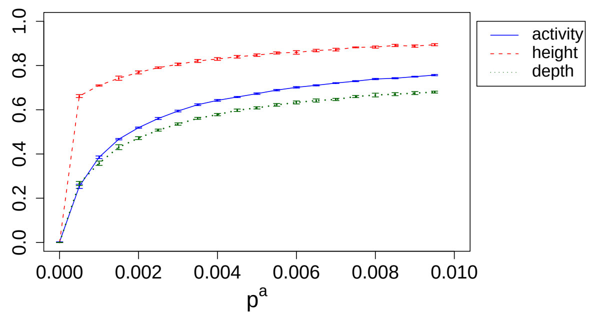
(e) Behavior of the FNN model with $p^a = 0.002$



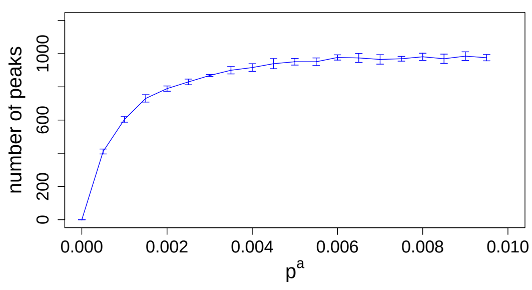
(f) Behavior of the FNN model with $p^a = 0.004$

Figure 4.5: System behavior for selected values of p^a

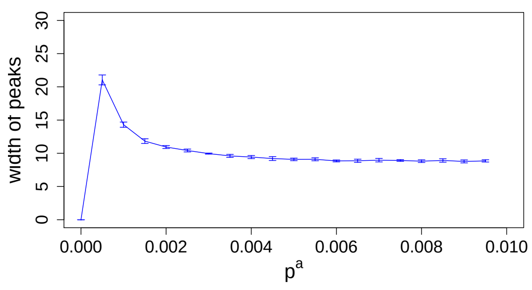
The results are shown in Figure 4.10 in the same way as in Figures 4.4 and 4.6. In Figures 4.9(b), respectively Figure 4.9(d), the behavior of the FNN is shown when a smaller, respectively a larger, neighborhood is chosen with respect to neighborhood N_3, which was used in the initial experiments. The results show indeed that the neighborhood size strongly influences the height of the activity peaks. When using neighborhood N_2 (as in Figure 4.9(b)) the activity peaks are lower than when using neighborhood N_3 (see Figure 4.2). And con-



(a) Average system activity, peak height and valley depth



(b) Average number of peaks



(c) Average width of peaks

Figure 4.6: System behavior as a function of p^a

versely, when using neighborhood N_4 (as in Figure 4.9(d)) the activity peaks are higher than when using neighborhood N_3. The results show that—in general—the larger the neighborhood the higher the stimulus received by the automata on average.

The remaining three graphics in Figure 4.10 show the relation between the neighborhood size and the average system activity as well as other variables of the system's behavior. In general, the larger the neighborhood size the higher the average system activity. The same results are shown in numerical form in Table 4.4.

Table 4.4: System behaviour as a function of the neighborhood size $|N_i|$ (in numerical form).

$\lvert N_i \rvert$	**system activity**	**num. peaks**	**peak width**		**peak height**		**valley depth**	
			width	**std**	**height**	**std**	**depth**	**std**
1	0.041248	690.35	12.520960	5.826813	0.081722	0.035509	0.028929	0.024708
5	0.190860	613.05	14.091955	6.447843	0.386524	0.132995	0.145075	0.129727
9	0.383997	593.15	14.570695	7.700792	0.711960	0.157383	0.355584	0.286901
13	0.537113	573.30	15.088730	8.402342	0.910996	0.101597	0.537132	0.399729
21	0.754239	491.35	17.597060	12.794700	0.924987	0.110344	0.643398	0.363736
25	0.827320	448.38	19.301386	16.989052	0.926782	0.104914	0.703845	0.317331

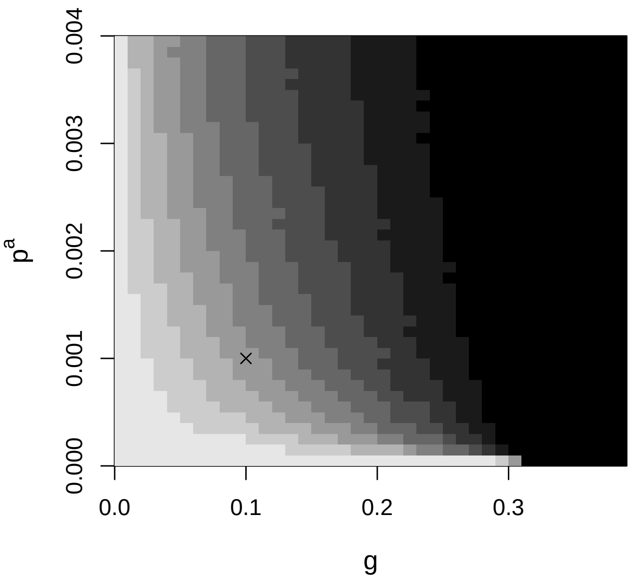


Figure 4.7: Average system activities displayed by means of gray levels (an explanation is given in the text) for different combinations of values for g and p^a. The cross marks the parameter setting that was used for Figure 4.2.

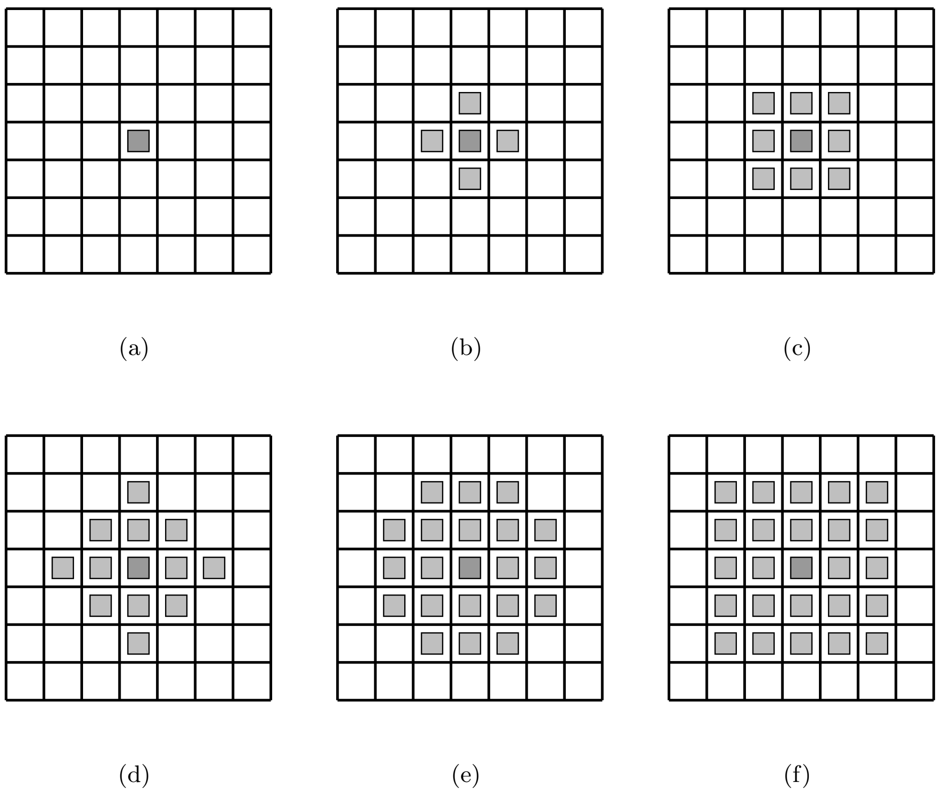
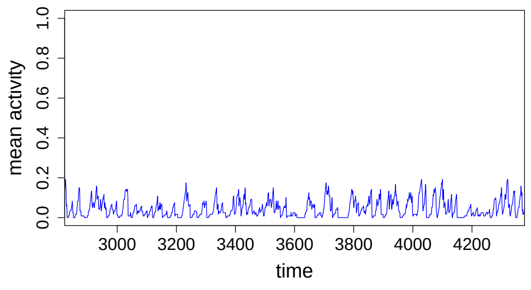
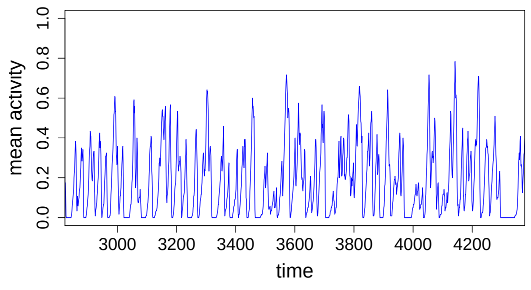


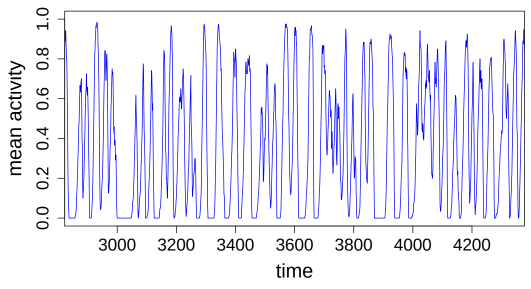
Figure 4.8: Considered set of neighborhoods. Graphics (a) to (f) show neighborhoods N_1 to N_6.



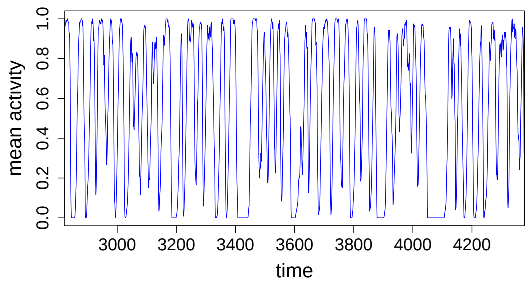
(a) Behavior of the FNN model with N_1



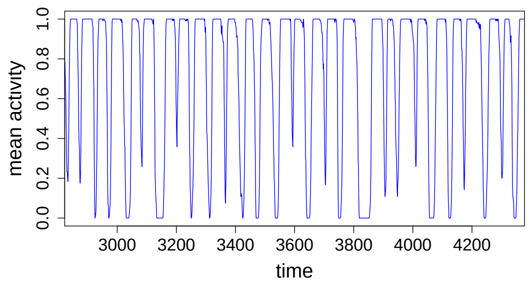
(b) Behavior of the FNN model with N_2



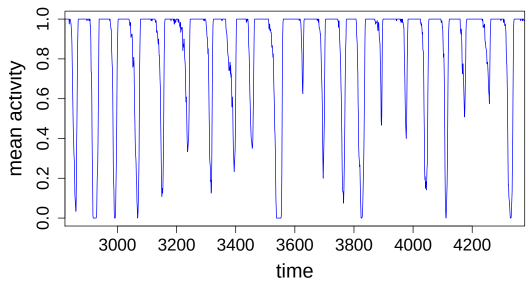
(c) Behavior of the FNN model with N_3



(d) Behavior of the FNN model with N_4

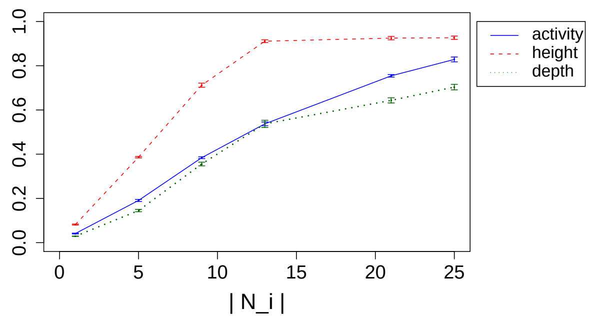


(e) Behavior of the FNN model with N_5

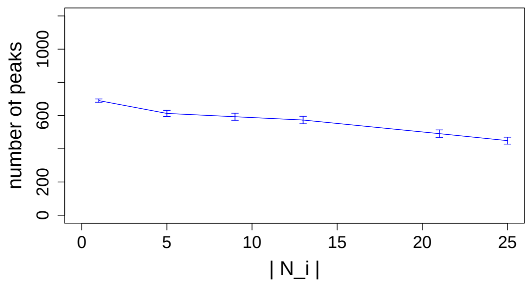


(f) Behavior of the FNN model with N_6

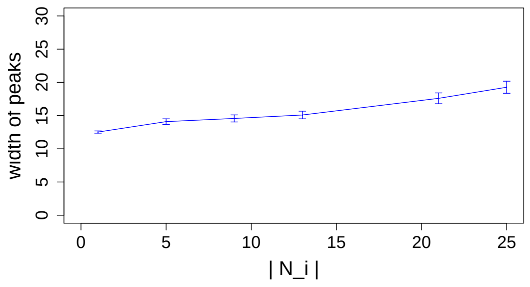
Figure 4.9: System behavior for selected values of $|N_i|$



(a) Average system activity, peak height and valley depth



(b) Average number of peaks



(c) Average width of peaks

Figure 4.10: System behavior as a function of $|N_i|$.

4.2 Adaptation to Sensor Networks

Based on the Fnn model described in the previous section we developed a system intended for the management of duty-cycling in mobile sensor networks. In this context the interested reader should note that the following system is implemented in C++ as a discrete event simulation. We decided against the use of sensor network simulators such as Omnet++/Castalia, NS-2, or Shawn, just to name a few. The reason for this decision was our aim of first studying and understanding the intrinsic properties of the system before adapting it to specific network simulators, MAC protocols, and sensor network devices. In our opinion, this first step is strictly necessary due to the complex nature of the system that we propose. Therefore, the study presented in the following is based on discrete event simulation, neglecting physical obstacles such as, for example, packet loss. Nevertheless, in the last section of this work we present a discussion on the adaptation of the presented system to sensor network simulators. Moreover, we show first results that indicate that the proposed system is very permissive for what concerns packet loss.

As mentioned already in the introduction, sensor networks are composed of numerous small devices which are deployed in some area. In order to account for the fact that the system clocks of the different sensor nodes may not be perfectly synchronized, the discrete event simulation that we present in the following allows a different clock cycle for each sensor node. In the following we assume that the system has an *absolute step frequency* of Δ_g time. These time steps are used for logging information. Moreover, each individual sensor node i has its own step frequency of Δ_i time. For all sensors i, Δ_i was chosen to be a slight variation of Δ_g. More specifically, Δ_i was chosen randomly with mean Δ_g and a standard deviation of $\Delta_g/20$ for each sensor i at the start of each simulation experiment.

4.2.1 System Components

A sensor network consists of k mobile sensor nodes with time-dependent positions in the $[0,1] \times [0,1]$ square. Sensor nodes behave similarly to automata in the Fnn model. More in detail, each sensor is an independent entity which has a state variable $S_i(t)$ and a binary variable $a_i(t)$ that specifies if the sensor node is either *active* or *inactive* depending on the value of $S_i(t)$. This is determined in the same way as in the case of the Fnn; see Eqs. (4.1) and (4.2).

For updating their state variables sensor nodes use Eq. (4.3) from the description of the Fnn model. However, the definition of function $h_i(\cdot)$ is now redefined due to the fact that mobile sensor nodes move in continuous space (in contrast to a grid structure) and are equipped with omni-directional radio antennas for communication. Note that a sensor i at time t can receive the transmission of a sensor j, if and only if $d(p_i(t), p_j(t)) \leq r_j(t)$, where $p_i(t)$, respectively $p_j(t)$, denotes the position of sensor i, respectively sensor j, at time t, $d(\cdot,\cdot)$ is the Euclidean distance, and $r_j(t)$ is the transmission radius of sensor j at time t. Accordingly, the neighborhood $N(i,t)$ of sensor i at a certain time t is defined as the set of all sensors whose transmissions i is (potentially) able to receive at time t; see Figure 4.11 for a graphical example. In this context, given two sensors i and j, the fact that $i \in N(j,t)$ does not necessarily imply that $j \in N(i,t)$, and vice versa.

Formally, function $h_i(\cdot)$ is re-defined as follows:

$$h_i(t) := S_i(t) + S \quad , \tag{4.6}$$

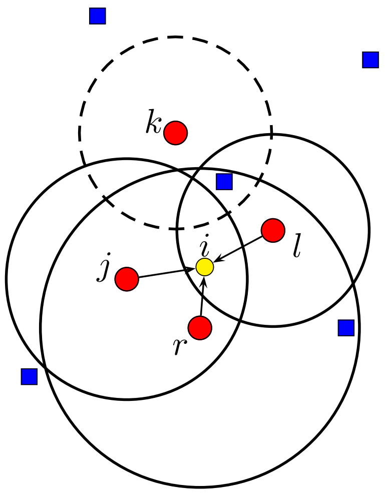


Figure 4.11: Example of the neighbourhood $N(i,t)$ of a sensor i. $N(i,t) = \{j, r, l\}$, because i is located within the transmission radius of sensors j, r and l. However, i is not located within the transmission radius of sensor k. Note that circles represent active sensors, whereas squares represent inactive ones.

where S is the sum of the values $S_j(z)$ of the state variables of other sensors j whose transmissions sensor i has received at any time $z \in (t-\Delta_i, t]$, that is, at any time since the last periodic update of its own state variable. Concerning the movement of sensor nodes, a variation of the *random direction movement (RDM) model* [164] was considered. In the RDM model, each sensor has a velocity v and a direction of movement that is chosen randomly at the beginning of the simulation. All sensors move at their own speed (v space units per Δ_i time) and in their own direction. In case they reach the border of the area of deployment, they choose a new random direction. Our variation of this model is as follows. For the sake of achieving a movement pattern more similar to the one of automata in the FNN model, a new direction is chosen by each sensor at each time step. In case a sensor tries to reach a position outside the area of deployment it will be forced to stay at the border until the next step (when a new direction will be chosen). For simplicity reasons we have assigned the same speed to all sensor nodes, $v = 0.001$. Notice that mobility is not necessarily an active property. Passive mobility can, for example, be observed when sensors are attached to moving objects such as animals or cars, or when sensors are moved by air or water currents.

The system components outlined above are summarized as a so-called sensor event in Algorithm 4. As previously mentioned, the system outlined above is simulated using discrete event simulation. At the start of each simulation, the global system clock, as well as the individual sensor clocks, are initialized to 0. Moreover, sensors' state variables are initialized to an activity level of S^a, that is, $S_i(0) := S^a$, $i = 1, \ldots, k$. Events of the system are either global system events or sensor events. Global system events are scheduled at times $t = z\Delta_g$, where $z = 0, \ldots, 10080$. Assuming that Δ_g corresponds to one minute, the simulations that are performed span 7 days each. For each sensor i, a sensor event is scheduled at times $t = z\Delta_i$ for all $z > 0$ such that $t \leq 10080\Delta_g$. In other words, sensor events are scheduled until the simulation stops. As in the case of the FNN, results from the first day of simulation are discarded to let the system stabilize.

Algorithm 4 Event of sensor node i at time t

```
1: Calculate value $S_i(t)$ for state variable $S_i$ (see Eqs. (4.3) and (4.6))
2: Calculate $a_i(t)$ (see Eq. (4.1))
3: if $a_i(t) = 0$ then
4:    Draw a random number $p \in [0, 1]$
5:    if $p \leq p^a$ then $S_i(t) := S^a$ and $a_i(t) := 1$ endif
6: end if
7: Make a transmission of $S_i(t)$ with radius $r_i(t)$
8: Move according to the mobility model
```

4.2.2 Comparison to the FNN Model

In an initial experiment we wanted to verify that the behavior of a sensor network as outlined above is comparable to the behavior of the automata in the FNN model. Therefore, for parameters S^a, θ_{act}, g and p^a we used the parameter settings as provided in Table 4.1. Moreover, we used a number of $k = 120$ sensors and the transmission radius for each sensor was determined as follows. Remember that to obtain the behavior of the FNN as outlined in Figure 4.2 the Moore neighborhood of size one was used. This means that each automaton received input from automata that resided on the 9 closest grid positions out of $25 \times 25 = 625$ grid positions. In order to achieve neighborhoods of approximately the same size (in terms of the area), the transmission radius $r_i(t)$ was fixed for all sensors i and all time steps t to $r = 0.07$, due to the fact that $\sqrt{\frac{9}{625 \cdot \pi}} \approx 0.07$. The resulting evolution of the mean system activity ($A(t)$) over time is shown in Figure 4.12, exemplary of the fourth day of simulation. Indeed, comparing Figure 4.12 with Figure 4.2 indicates that the behavior of the FNN model and the sensor network system that we developed are comparable.

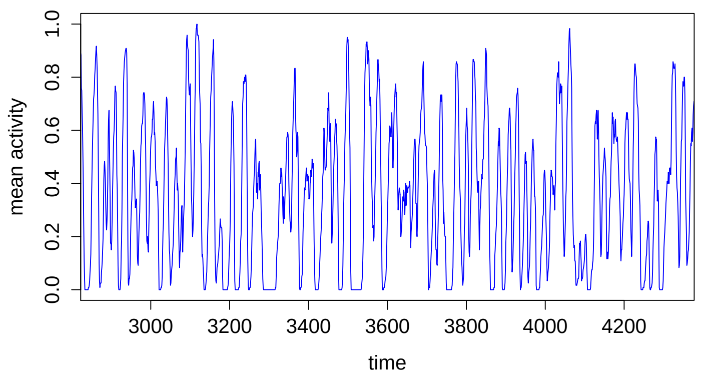


Figure 4.12: Evolution of the mean system activity ($A(t)$) of a sensor network with 120 sensors for the fourth day of simulation.

4.3 Adaptation to Changing Energy Availabilities

Sensor networks are sometimes deployed in remote areas (like the backcountry or oceans) where power supply is not easily available. For this reason sensor nodes are generally equipped with batteries, and energy turns into a scarce resource. Remember that duty-cycling is one possible solution for extending network lifetime. However, in order to achieve a really independent and long-living sensor network, sensor nodes must—in addition—be able to harvest energy from the environment. In this section we introduce an extension of the system described in the previous section in order to obtain an energy-aware duty-cycling mechanism that adapts to changing energy availabilities. This extended system will henceforth be labelled ANTCYCLE. For achieving such a system we will profit from the understanding of the FNN model that was gained in Section 4.1. In particular, a variable transmission radius will be used to control the height of the activity peaks, and a variable probability of spontaneous activation (p^a) to control the frequency of the activity peaks. As as example we assume that sensor nodes are equipped with solar cells, which are able to transform sun light into electrical power. However, we want to clarify at this point that the proposed mechanism is in fact independent of the specific form of energy harvesting.

ANTCYCLE can be described as follows. Each sensor i has a battery. The battery level of sensor i at time t is denoted by $b_i(t) \in [0, 1]$. Hereby, $b_i(t) = 1$ corresponds to a full battery. The proposed system extension consists in the fact that the transmission radius $r_i(t)$ at time t and the probability of spontaneous activation ${p^a}_i(t)$ at time t are now defined with respect to the battery level $b_i(t)$ at time t. More specifically, the variable transmission radius is determined in the following way:

$$r_i(t) := r_{\min} \cdot (1 - b_i(t)) + r_{\max} \cdot b_i(t) \ , \tag{4.7}$$

where $r_{\min}$, respectively $r_{\max}$, is the lower bound, respectively the upper bound, for the variable transmission radius. Note that when the battery of a sensor i is fully charged its transmission radius is set to $r_{\max}$. Moreover, a static transmission radius can be achieved—if desired—by setting $r_{\min} = r_{\max}$. The variable probability of spontaneous activiation (p^a) is also made dependent on the current battery level:

$${p^a}_i(t) := p^a_{\min} \cdot (1 - b_i(t)) + p^a_{\max} \cdot b_i(t) \ , \tag{4.8}$$

where $p^a_{\min}$, respectively $p^a_{\max}$, is the lower bound, respectively the upper bound, for the variable probability of spontaneous activation. In the same way as in the case of $r_i(t)$, when the battery of a sensor i is fully charged its variable probability of activation is set to $p^a_{\max}$. In addition, a static transmission radius can be achieved—if desired—by setting $p^a_{\min} = p^a_{\max}$. The interested reader may note that in contrast to our earlier proposal from [97] the transmission radius and the probability of spontaneous activation depend now in a linear way on the battery level. This reduces the unpredictability of the system.

In the following we describe the energy model that we used for the simulations performed in this work. Again, note that this energy model should be seen as an example, and can be replaced by the real energy model of any of the available physical sensors and solar panels. We assume that at each time interval $(t - \Delta_i, t]$ a sensor i consumes a certain amount of energy that depends on its state. Hereby, e_{sleep} and e_{awake} are constants that determine the energy consumption of sensors in the *inactive* and *active* states, respectively. More specifically, sensors that are *inactive* consume the following amount of energy:

$$e_i(t) := \int_{t-\Delta_i}^{t} e_{\text{sleep}} \cdot dt \tag{4.9}$$

Being inactive represents a state in which a sensor does nothing except for listening to incoming transmissions. In contrast, a sensor i that is *active* in $(t-\Delta_i, t]$ consumes an amount of energy that depends on its current transmission radius:

$$e_i(t) := \int_{t-\Delta_i}^{t} e_{\text{awake}} \cdot (1 + r_i(t - \Delta_i)) \cdot dt \tag{4.10}$$

Values e_{sleep} and e_{awake} strongly depend on the technical properties of the physical sensors used and on the tasks sensors must execute.

Finally, sensors are able to harvest a certain amount of energy at each interval of time $(t - \Delta_i, t]$. Henceforth, the intensity of the light source at time z is denoted by $s(z) \in [0, 1]$. Hereby, $s(z) = 0$ corresponds to absolute darkness. As in previous work we use the following model for the evolution of the sun light intensity—that is, for the evolution of $s(z)$—over time. In this context remember that in this work a global time step, that is, Δ_g, corresponds to one minute. Therefore, one day consists of 1440 global time steps. More in detail, the following function is used for modeling the light intensity:

$$s(t) = \begin{cases} 0 & \text{, if } 0 \leq \dot{t} < 420 \\ \frac{1-\cos(\frac{\dot{t}-420}{1140-420} \cdot 2\pi)}{2} & \text{, if } 420 \leq \dot{t} < 1140 \\ 0 & \text{, if } 1140 \leq \dot{t} < 1440 \end{cases} \tag{4.11}$$

where $\dot{t} := t \mod 1440$. Note that this function can be seen as a model of the sun light intensities of a day (0:00 a.m–24:00 p.m.). However, notice that our system should be able to adapt to any light intensity function. The one that we used is easily replaceable and should only be seen as an example. In accordance to real solar panels we assume that only a fraction f of the available light intensity can be turned into energy:

$$e_i^{\text{harv}}((t - \Delta_i, t]) := f \cdot \int_{t-\Delta_i}^{t} s(z) \cdot dz \tag{4.12}$$

Simulations of ANTCYCLE start with batteries that are fully charged, that is, $b_i(0) = 1$ $\forall i \in \{1, \ldots, k\}$. Algorithm 5 presents a sensor event in ANTCYCLE in the form of pseudo-code. Concerning the values for parameters r_{min}, r_{max}, p^a_{min} and p^a_{max} after initial experiments we decided on the following restriction. In case a variable transmission radius is desired, the relation between r_{max} and r_{min} is always fixed as follows: $r_{\text{max}} = 2 \cdot r_{\text{min}}$. Otherwise, $r_{\text{min}} = r_{\text{max}}$. Similarly, in case of a variable probability of spontaneous activation we require that $p^a_{\text{max}} = 100 \cdot p^a_{\text{min}}$, and $p^a_{\text{min}} = p^a_{\text{max}}$ otherwise. In this way, only suitable parameters for r_{min} and p^a_{min} have to be found. Moreover, initial experiments have shown that these restrictions still allow for a sufficient degree of freedom.

4.3.1 Initial Experiments with ANTCYCLE

The aim of the initial experiments presented in this section is to study the behavior of the system when the transmission radius is variable and the probability of spontaneous activation

Algorithm 5 Event of sensor node i with energy harvesting capabilities at time t

```
1:  $b_i(t) := b_i(t) + e^{\text{harv}}([t - \Delta_i, t])$
2:  if $a_i(t) = 1$ then
3:      $b_i(t) := b_i(t) - e_i(t)$ (see Eq. (4.10))
4:  else
5:      $b_i(t) := b_i(t) - e_{\text{sleep}}$
6:  end if
7:  Calculate value $S_i(t)$ for state variable $S_i$ (see Eq. (4.4))
8:  Calculate $a_i(t)$ (see Eq. 4.1)
9:  ${p^a}_i(t) := p^a_{\text{min}} \cdot (1 - b_i(t)) + p^a_{\text{max}} \cdot b_i(t)$
10: if $a_i(t) = 0$ then
11:     Draw a random number $p \in [0, 1]$
12:     if $p \leq p^a$ then $S_i(t) := S^a$ and $a_i(t) := 1$ endif
13: end if
14: $r_i(t) := r_{\text{min}} \cdot (1 - b_i(t)) + r_{\text{max}} \cdot b_i(t)$
15: Make a transmission of $S_i(t)$ with radius $r_i(t)$
16: Move according to the mobility model
```

is fixed. Remember that the experiments with the FNN model in Section 4.1 suggested that when changing the neighborhood size (which corresponds to changing the transmission radius of sensors) the height of the activity peaks changes. Therefore, we would assume that with a variable transmission radius ANTCYCLE will be able to react to changing energy availabilities by adjusting the height of the activity peaks.

For comparability reasons we decided to apply ANTCYCLE with the same parameter settings as for the experiments with the FNN model, that is, for parameters S^a, θ_{act}, g and p^a_{min} the parameter settings as provided in Table 4.1 were used. Note that the ANTCYCLE parameter p^a_{min} corresponds to parameter p^a from the FNN model. For the purpose of studying the systems' behavior with a fixed probability of spontaneous activation we set $p^a_{\text{max}} := p^a_{\text{min}}$. Moreover, as in Section 4.2, we used a number of $k = 120$ sensors and a setting of $r_{\text{min}} = 0.07$. These parameter settings are summarized in Table 4.5. Note that this table also specifies the settings for the constants e_{awake} and e_{sleep} from the energy model. Remember that these settings should be seen as an example.

The obtained behavior of ANTCYCLE is shown in Figure 4.13, which shows the evolution of the mean system activity over time (continuous line), the evolution of the average battery level of the sensors over time (dashed line) and the evolution of the sun intensity (dotted line). The graphic shows that in addition to exhibiting a self-synchronizing behavior the system is now also nicely adapting to changing energy availabilities. Note that when the average battery level drops, the system reacts by decreasing the height of the activity peaks. On the other side, when the battery level increases the height of the activity peaks reaches the maximum of 1.0, which means that all sensors are awake at the same time. Moreover, the width of the activity peaks increases.

Next, we aimed at studying the effect of changing the setting of r_{min}. The behavior of ANTCYCLE for 20 different values of $r_{\text{min}} \in [0, 0.19]$ is shown in graphical form in Figure 4.15, and in numerical form in Table 4.6. In comparison to the reference value of $r_{\text{min}} = 0.07$, the behavior of ANTCYCLE for a much smaller value ($r_{\text{min}} = 0.325$) is shown in Figure 4.14(b).

Table 4.5: Initial parameter settings for ANTCYCLE.

r_{min}	r_{max}	g	p^a_{min}	p^a_{max}	f	e_{awake}	e_{sleep}
0.07	0.14	0.1	0.001	0.001	0.0027	0.001	0.0002

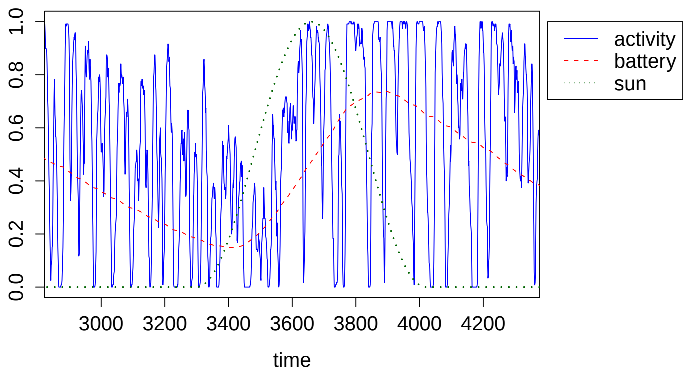


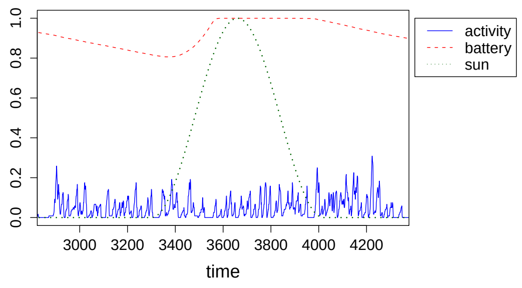
Figure 4.13: Evolution of the mean system activity of ANTCYCLE with the parameters from Table 4.5. Additionally, the dashed line shows the average battery level of the sensors, and the dotted line indicates the evolution of the sun intensity.

Clearly, with this setting the system is not able to adapt to changing energy conditions. This is because the transmission ranges are generally not large enough for the propagation of activation through the network. In contrast, the behavior of ANTCYCLE for $r_{\text{min}} = 0.1$ (which is larger than the reference value $r_{\text{min}} = 0.07$) is shown in Figure 4.14(e). Note that with this setting the system is able to adapt to changing energy conditions. However, the energy consumption is too high. At times of little energy harvesting (during night time) the sensors use up all the available energy, which means that no activity peaks are generated. These experiments indicate that a well-working value for r_{min} strongly depends on the energy harvesting capabilities of the system.

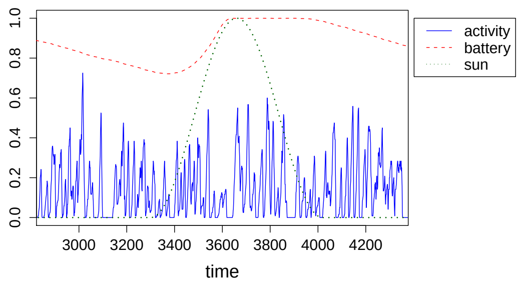
The average system activity over the whole simulation time is shown as a function of r_{min} in Figure 4.15(a). As in the case of the FNN model, this average system activity strongly depends on the transmission radius. The average system activity increases with increasing transmission radius until the sensor nodes start to consume too much energy. At this point the batteries deploy and the average system activity decreases. Finally, Figures 4.15(b) and 4.15(c) show that with increasing value of r_{min} the average number of peaks decreases and the average width of the activity peaks increases. Note, for example, that for $r_{\text{min}} \leq 0.1$ the number of activity peaks is greater than 500, while for bigger values of r_{min} the number of activity peaks decreases dramatically. As mentioned above, this is because activity peaks are much wider for larger values of r_{min}.

Table 4.6: Behaviour of ANTCYCLE with the parameter settings from Table 4.7 as a function of $r_{\min}$ (in numerical form).

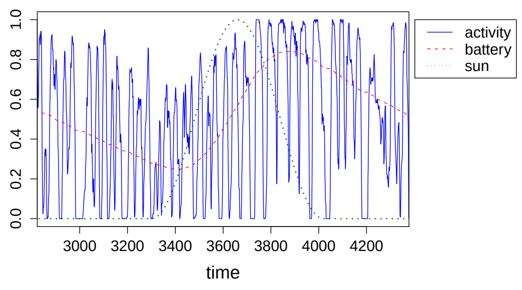
$r_{\min}$	**system activity**	**num. peaks**	**peak width**		**peak height**		**valley depth**	
			width	**std**	**height**	**std**	**depth**	**std**
0	0.014061	454.60	18.996225	8.058430	0.025582	0.008005	0.008484	0.007020
0.01	0.026909	619.35	13.957440	6.417250	0.054634	0.023039	0.019256	0.016229
0.02	0.070994	666.45	12.970205	5.845324	0.146893	0.060532	0.050497	0.045352
0.03	0.139084	647.95	13.350860	6.128064	0.272647	0.101504	0.105212	0.092428
0.04	0.245881	653.90	13.224050	6.766202	0.440283	0.153770	0.204928	0.168808
0.05	0.378638	682.25	12.671640	6.887918	0.625693	0.194127	0.351671	0.260344
0.06	0.487424	735.05	11.759880	6.691377	0.735735	0.198428	0.476536	0.306784
0.07	0.515316	746.35	11.592590	6.682555	0.754579	0.196589	0.505541	0.312408
0.08	0.525643	766.05	11.291905	6.684032	0.746029	0.220336	0.520458	0.328118
0.09	0.538510	717.75	12.047690	7.994258	0.747466	0.250108	0.559079	0.343696
0.1	0.554001	564.70	15.327835	12.785250	0.771961	0.252040	0.602331	0.348809
0.11	0.565143	386.40	22.511200	24.735780	0.772793	0.254945	0.610710	0.345382
0.12	0.569645	265.95	32.484550	42.337455	0.772324	0.260246	0.618263	0.345111
0.13	0.570818	171.40	47.841765	71.477245	0.776612	0.244935	0.620500	0.336090
0.14	0.568790	118.70	64.616475	104.672605	0.762730	0.245914	0.574179	0.335770
0.15	0.564501	90.15	84.597870	145.115285	0.698418	0.254493	0.443957	0.303574
0.16	0.559169	83.85	89.850275	154.756000	0.706741	0.251155	0.429811	0.296664
0.17	0.553926	83.55	89.253165	154.928850	0.698158	0.249364	0.403225	0.283739
0.18	0.549195	94.70	78.444090	136.770100	0.719708	0.237079	0.429383	0.278225
0.19	0.543920	92.00	81.664252	142.019524	0.714629	0.242963	0.410516	0.270141



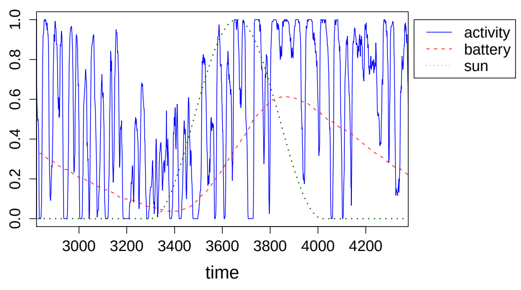
(a) Behavior of ANTCYCLE with $r_{\min} = 0.01625$



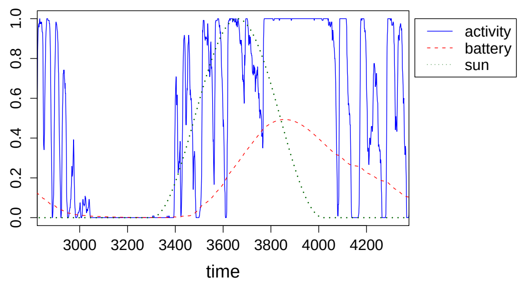
(b) Behavior of ANTCYCLE with $r_{\min} = 0.0325$



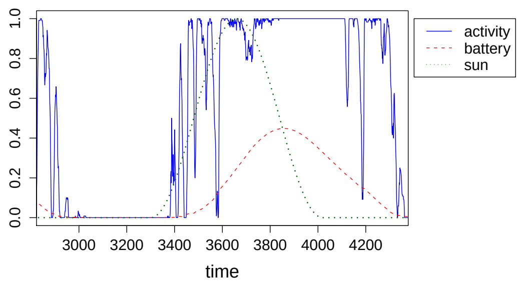
(c) Behavior of ANTCYCLE with $r_{\min} = 0.065$



(d) Behavior of ANTCYCLE with $r_{\min} = 0.081$

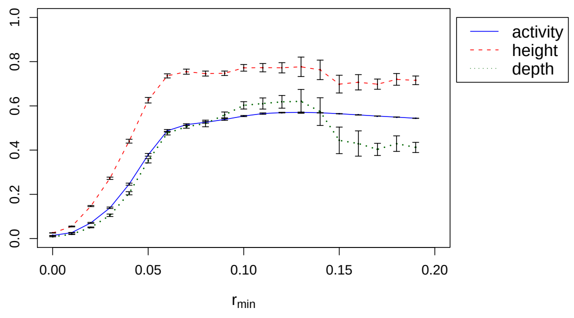


(e) Behavior of ANTCYCLE with $r_{\min} = 0.1$

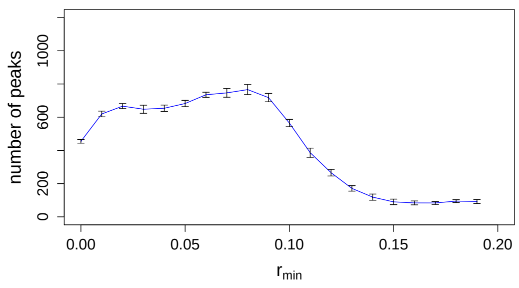


(f) Behavior of ANTCYCLE with $r_{\min} = 0.11$

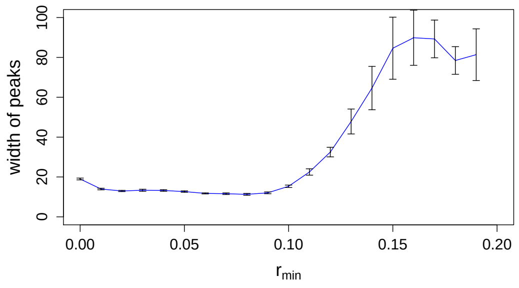
Figure 4.14: ANTCYCLE behavior for selected values of $r_{\min}$.



(a) Average system activity, peak height and valley depth



(b) Average number of peaks



(c) Average width of peaks

Figure 4.15: ANTCYCLE behavior as a function of r_{min}.

4.4 Further experiments

The experiments concerning ANTCYCLE that were presented in the previous section were aimed at giving a first indication of the way in which the proposed system works. In contrast, in this section we present a deeper experimental study which is concerned with robustness, scaling, and the search for suitable parameter settings in order to serve different application scenarios.

4.4.1 Robustness

For practical applications it is very important that a system is robust, that is, that its behavior with the same parameter settings is always the same, or at least comparable. In order to test the robustness of ANTCYCLE we applied the system 10 times with the parameter settings as given in Table 4.5. The results are presented in the form of box plots in Figure 4.16. Hereby, for each application of ANTCYCLE the corresponding box consists of exactly one value for each of the 120 sensors. Three different measures are presented. In Figure 4.16(a) a box consists of the average activity of each sensor over the whole simulation time, that is, the fraction of time steps in which a sensor has been active. Furthermore, in Figures 4.16(b) and 4.16(c) a box consists of the minimum, respectively maximum, battery level of each sensor. The information that is given by these three graphics can be summarized as follows. The low height of the boxes indicates that—within each application of ANTCYCLE—the 120 sensors behavior similarly. Moreover, the similarity of the 10 boxes per graphic indicates that ANTCYCLE behaves comparably over different applications with the same parameter settings.

4.4.2 Application Scenario: Environmental Monitoring

In Section 4.3.1 we have shown that with a variable transmission radius and a fixed probability of spontaneous activation ANTCYCLE adapts to changing energy conditions by varying the height of the activity peaks. Note that an activity peak of a height less than 1.0 means that not all sensors have been activated. This behavior may be suitable for tracking applications where sensors do not necessarily have to be active all at the same time. In tracking it is rather important that the object to be tracked does not escape unnoticed. In order to reach this goal, the whole area of deployment should be equally observed without neglecting any region. However, for that purpose it is not necessary that all sensors are active at the same time.

Nevertheless, when considering, for example, an application such as the monitoring of some environmental measure such as the temperature, it is desirable that all sensors are active at the same time, such that measurements can be taken at similar times. In other words, for an application such as environmental monitoring it would be desirable for the system to adapt to changing energy conditions by adjusting the frequency of peaks, while the height of the activity peaks is always maximal. In this context, remember that from the experiments performed in Section 4.1 we have learned the following:

- By increasing the probability of spontaneous activation p^a of the FNN model, the frequency of the activity peaks and the average system activity can be increased.

- Increasing the neighborhood size increases the height of the activity peaks as well as the average system activity.

Table 4.7: Parameter settings of ANTCYCLE for environmental monitoring.

r_{min}	r_{max}	g	p^a_{min}	p^a_{max}
0.16	0.16	0.05	0.00001	0.001

- The lower the value of parameter g, the faster the value of the variables S_i (the stimulus) will decay and the lower the average system activity.

Combining these insights, a new set of parameter values has been designed which is characterized by the fact that the probability of spontaneous activation is now variable and depending on the battery level, whereas the transmission radius is constant and the value of g is decreased in comparison to experiments presented earlier in this work. These parameter settings are outlined in Table 4.7. In Figure 4.17 the corresponding behavior of ANTCYCLE is shown. Indeed, with the new parameter settings the system adapts to changing energy conditions by adjusting the frequency of the peaks, while their height is always maximal.

In order to study the change of system behavior when changing the value of p^a_{min}, we applied ANTCYCLE with the same parameter settings and for all values of p^a_{min} from $\{0, 5 \cdot 10^{-6}, 10^{-5}, \ldots, 9.5 \cdot 10^{-5}\}$. The results are shown in graphical form in Figure 4.19 as well as in numerical form in Table 4.8. As expected, an increasing value of p^a_{min} causes a logarithmic increase in average system activity, average peak height, average valley depth, and in the average number of peaks. On the other side, after an initial increase, the average peak width decreases.

4.4.3 Scaling

So far we have only studied sensor networks of size 120, that is, networks with 120 sensor nodes. However, when changing the size of the network it is intuitively clear that at least some parameter values must be adjusted in order to maintain a functional system. Note that when changing the network size, the node density changes. Hence, it is reasonable to assume that for maintaining the shape of the activity peaks, the transmission radius and the probability of spontaneous activation should be adapted to the new network size. In the following we present a solution to this problem. With k_{new}, ${p^a}_{\text{new}}$ and r^{new} we refer to the number of nodes, the probability of spontaneous activation and the transmission radius of the new, differently sized, network. Hereby, ${p^a}_{\text{new}}$ and r^{new} correspond, respectively, to p^a and r as calculated depending on the current battery level in Equations 4.8 and 4.7. First, in order to obtain the same wake-up rates as in the case of a 120-node network, the following rule can be applied:

$${p^a}_{\text{new}} := p^a \cdot \frac{k}{k_{\text{new}}} \quad , \tag{4.13}$$

where p^a and k are the parameters from the original network. Note that this rule increases the probability of spontaneous activation of the nodes when the network population is decreased, and vice-versa when the number of nodes increases. Moreover, the average number of nodes' spontaneous activations per time unit are maintained. Next, we introduce a rule for adapting the transmission range. The basic idea is to have a constant average number of sensors being reached by a transmission. Due to the fact that the sensor nodes form a random topology at

Table 4.8: Behaviour of Antcycle with the parameter settings from Table 4.7 as a function of $p^a_{\min}$ (in numerical form).

$p^a_{\min}$	**system activity**	**num. peaks**	**peak width**		**peak height**		**valley depth**	
			width	**std**	**height**	**std**	**depth**	**std**
0.0e+00	0.000000	0.00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5.0e-06	0.117945	395.30	21.850320	12.213620	0.342251	0.119491	0.112838	0.125643
1.0e-05	0.197409	600.75	14.385550	7.268109	0.389165	0.131972	0.166201	0.145841
1.5e-05	0.250540	705.40	12.262695	6.193192	0.425596	0.139259	0.202669	0.154091
2.0e-05	0.296487	789.70	10.938980	5.528283	0.459466	0.143116	0.241002	0.161658
2.5e-05	0.332310	843.95	10.239175	5.156073	0.484357	0.146197	0.267836	0.164146
3.0e-05	0.361325	879.80	9.825499	4.903315	0.507919	0.146260	0.292590	0.166376
3.5e-05	0.386210	918.10	9.413495	4.760447	0.525886	0.147043	0.315164	0.168549
4.0e-05	0.408836	939.75	9.199885	4.647774	0.545326	0.147101	0.334959	0.169967
4.5e-05	0.425612	954.95	9.050974	4.540828	0.560289	0.147739	0.350548	0.172643
5.0e-05	0.440583	955.80	9.042987	4.471361	0.575304	0.148743	0.361146	0.174580
5.5e-05	0.454010	978.55	8.836479	4.432002	0.585238	0.147050	0.376220	0.174372
6.0e-05	0.469220	990.80	8.722034	4.339729	0.595932	0.147597	0.388996	0.173354
6.5e-05	0.478835	1000.05	8.645221	4.348797	0.604707	0.146845	0.398143	0.174319
7.0e-05	0.487128	999.45	8.649687	4.341639	0.612593	0.148669	0.404365	0.176467
7.5e-05	0.493984	1011.40	8.545203	4.263876	0.618465	0.146561	0.413119	0.174825
8.0e-05	0.500090	1004.30	8.605253	4.264081	0.624530	0.147147	0.417560	0.176420
8.5e-05	0.504807	1019.05	8.483200	4.275953	0.626718	0.147916	0.421923	0.174769
9.0e-05	0.509313	1019.95	8.474150	4.229272	0.631324	0.145016	0.427155	0.174370
9.5e-05	0.513250	1031.33	8.385169	4.187420	0.635327	0.143905	0.433020	0.174910

Table 4.9: Parameter values for networks of different sizes.

Application	k	r_{min}	r_{max}	g	p^a_{min}	p^a_{max}
Tracking	60	0.099	0.198	0.1	0.002	0.002
	120	0.07	0.14	0.1	0.001	0.001
	180	0.047	0.094	0.1	0.00067	0.00067
Monitoring	60	0.227	0.227	0.05	0.00002	0.002
	120	0.16	0.16	0.05	0.00001	0.001
	180	0.13	0.13	0.05	0.0000067	0.00067

any moment in time, the following reasoning was used. In general, the number of nodes that can be reached by the transmission of a sensor can be estimated as follows:

$$\pi \cdot r^2 \cdot \frac{k}{A} \ , \tag{4.14}$$

where k is the total number of sensors and A is the space in which the sensors reside. In our case it holds that $A = 1^2 = 1$. Therefore, Eq. (4.14) reduces to $\pi \cdot r^2 \cdot k$. As r is known for the case of 120-node networks, an adjusted transmission radius can be calculated for networks of different sizes as follows:

$$r^{\text{new}} = \sqrt{\frac{r^2 \cdot k}{k_{\text{new}}}} \ , \tag{4.15}$$

where k_{new} is the size of the new network, and r^{new} is the transmission radius for the new network. Using these two transformations, the parameter values obtained for both the tracking and the monitoring applications when considering networks with $k \in \{60, 120, 180\}$ nodes are summarized in Table 4.9. Moreover, the corresponding evolution of the average system activity of ANTCYCLE is shown in Figure 4.20. Comparing the graphics in Figures 4.20(a) and 4.20(b) with the graphic in Figure 4.13 we can state that the proposed parameter value transformations indeed preserve the behavior of ANTCYCLE in terms of the shape of the activity peaks. The difference is to be found, of course, in the amount of energy spent. Note that with decreasing network size the amount of energy spent increases. This is because for maintaining the shape of the activity peaks, the transmission radius and the probability of spontaneous activation must be increased when decreasing the network size. The same holds for the "Monitoring" application. The interested reader may verify this by comparing the graphics from Figures 4.20(c) and 4.20(d) with the one from Figure 4.17.

4.4.4 The Case of Static Networks

Currently, sensor network applications in practice are still mostly concerned with static networks. Therefore, studying the behavior of ANTCYCLE in static networks may be quite interesting for practical applications. However, note that in a static scenario, sensors may become more easily isolated in the sense that they are not reached by any transmissions of other sensor nodes. These problems may be solved with an intelligent (non-random) distribution of the nodes combined with an appropriate transmission radius. Consider, for example, the use of grid topologies.

The graphics in Figures 4.21(a) and 4.21(b) are obtained with the same parameter settings as used for obtaining the ANTCYCLE displayed in Figures 4.13 and 4.17, except for the fact that sensor nodes are now static. The similarity in system behavior lets us conclude that ANTCYCLE even works for static networks. However, note that in periods with fewer energy available the system has now more problems to synchronize. For example, in Figure 4.21(b) the activity peaks do not reach the maximum of 1.0 when the battery level drops. This is caused by the variable radius of transmission: when this radius is too low, some sensors may become isolated, and hence they are not able to stimulate or be stimulated by other sensors. However, this problem may have a simple solution. When individual nodes realize that they can not overhear any transmissions of other nodes for an extended period of time, they may individually and adaptively increase their value of $r_{\min}$.

4.4.5 A more Flexible Model for the Light Source

In all experiments that were presented so far, a periodically stable light source was considered. In other words, the evolution of the energy provided by the light source was the same for each day and for each sensor node. However, when harvesting energy from the environment sensor nodes have to deal with the uncertainty inherent to natural phenomena. For example, the amount of energy that can be harvested from sun light strongly depends on the weather. Other common natural sources for energy harvesting, such as the wind and tides, generally also show a high variability along extended periods of time. In order to show that the system proposed in this paper does not depend on a stable energy source, we consider in the following a stochastic function $s'(t)$ for the sun light intensity at time t, rather than the deterministic function $s(t)$ from Equation 4.11. This stochastic function is defined as follows:

$$s'(t) = \mathbf{min}\{0, \mathbf{max}\{\mathbf{X}(t), 1\}\} \tag{4.16}$$

where $\mathbf{X}(t)$ is a normally distributed stochastic variable with mean $s(t)$ and standard deviation σ^2. As an example we used $\sigma = 12$. The corresponding distribution is shown for three exemplary values—that is, three different values of $s(t)$—in Figure 4.22. Note that with this stochastic function for the sun light intensity, sensor nodes may perceive different light intensities even if their sensor events are located close in time.

Replacing function $s(t)$ with function $s'(t)$ we repeated the experiment that provided the results shown in Figure 4.13. The results with function $s'(t)$ are shown in Figure 4.23. When comparing the graphics in these two figures, it is obvious that the properties of the system are not altered. This was to be expected, because the average amount of energy that each sensor node is able to harvest is the same for both light intensity functions. Moreover, the system does not make a direct use of the amount of harvested energy. Instead, all actions are based on the cumulative energy that remains in the batteries. Therefore, reasonable variations of the distribution of energy over time do not directly affect the system's behavior.

4.4.6 Comparison with the Mechanism by Vigorito

Although at the beginning of this chapter we have outlined the features which make our duty-cycling mechanism different from others in the literature, we want to demonstrate how these differences affect the performance of the network. As in the rest of the chapter, we will primarily analyze the activity phases of the network to understand what are the major differences between the considered duty-cycling algorithms.

For the purpose of this comparison, we consider three algorithms. All of them are distributed and do not require any environmental data to be collected before running the algorithm. The first one, labelled RANDOM, forces each node to randomly choose at each cycle if the node should be active or inactive. This choice is made neglecting the current battery level and the decisions made by all other nodes. The algorithm accepts one parameter, the expected average activity, which in our experiment is set to 0.35. This means that a node will become active on average in 35% of all steps and inactive in the remaining 65%. Therefore, when RANDOM is used (see Figure 4.24(a)) an average amount of 35% of the nodes are active at each time step.

The second algorithm included in this comparison is the adaptive duty-cycling algorithm by Vigorito [187]. In terms of the task approached, this algorithm is one of the most similar that can be found in the literature. More in detail, it became specially attractive thanks to its ability to adapt to the changing energy conditions of the environment without requiring an environmental profile of the region of deployment of the network. The algorithm is also able to handle the fact that some regions of a network may be less adequate for energy harvesting than others, and allows each node to independently establish its average activity. For the purpose of comparison we consider the algorithm VIGORITO which is a reimplementation of the original algorithm from [187] adapted to work in our simulation framework[1]. The parameters used for the simulation of the algorithm are those suggested by the authors with the exception of the target battery level which is set to 0.65. The target battery level denotes the average battery level that we expect to find in the nodes at any moment in time. In Figure 4.24(b), we show the activity phases obtained when the network executes VIGORITO. The resulting activity phases oscillate around a line which is parallel to the average battery level line. This relation between the average battery level and the fraction of active nodes in the network is quite reasonable considering the algorithm target. VIGORITO successfully adapts the network activity to the available energy resources. However, it lacks any notion of synchronicity and all nodes behave independently of other nodes. Unfortunately, when synchronicity is omitted, the performance of applications which require communication between nodes may decrease drastically.

The third algorithm used in the comparison is ANTCYCLE with two different sets of parameters. In Figure 4.24(c) we show the behavior of the algorithm using the parameter setting for tracking applications but reducing the radius of transmission to $r_{\min} = 0.05$, whereas in Figure 4.24(d) the transmission radius is set to $r_{\min} = 0.07$. Apart from the fact that we already analyzed that choosing a higher transmission radius may result in higher energy consumption, both algorithm versions are able to correctly manage the energy resources which prevent early battery depletion. The interesting fact, which is easy to appreciate graphically, is that ANTCYCLE is not only able to adapt to the changing energy resources. It is able to do so while synchronizing the activity phases of all the nodes in the network. Moreover, this is not the only advantage of our novel algorithm. The possibility to fine-tune the parameters to obtain different activity schemes, such as for example the one proposed for monitoring applications, is a feature not available in other duty-cycling algorithms.

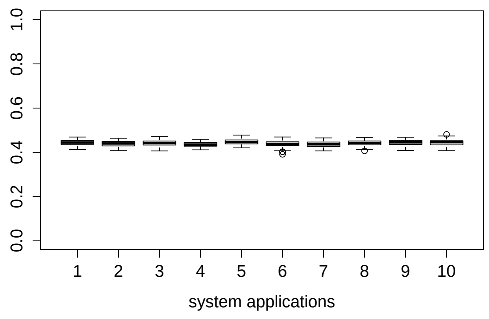
Finally, we want to emphasize that the comparison between the algorithms is fair. For this purpose, we analyze the state of the network nodes over the simulation time. All algorithms consider the same energy profile. Moreover, in all cases, the sun is simulated with the flexible

[1]The results obtained with our reimplementation of the algorithm are reasonable and in accordance with those presented in the original paper by Vigorito [187].

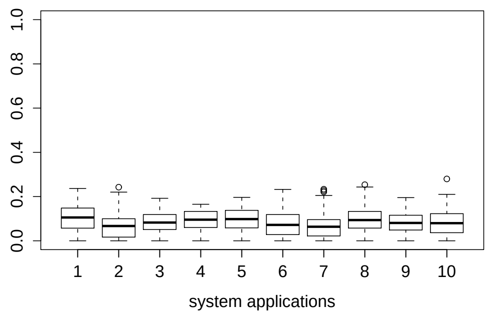
Table 4.10: Comparison of different duty-cycling mechanisms

measures	RANDOM	VIGORITO	ANTCYCLE $r_{\text{min}} = 0.05$	ANTCYCLE $r_{\text{min}} = 0.07$
activity (avg.)	0.350	0.381	0.391	0.526
battery (avg.)	0.802	0.807	0.772	0.486
dead nodes (%)	0.000	0.000	0.000	0.000
full batteries (%)	0.132	0.151	0.108	0.000

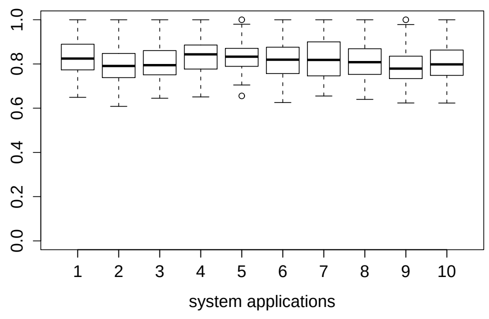
model proposed in Section 4.4.5. Table 4.10 shows aggregate results for ten days of simulation. For each algorithm we provide the average activity level, the average battery level, the percentage of steps in which a node was *dead* –the node's battery level is 0 or not enough to become active– and the percentage of steps in which the batteries of a node where full and not able to keep all the energy generated by the harvesting device. Notice that the trade-off between energy consumption and activity is comparable in all four cases. This is a reasonable result considering that the four algorithms are given the same energy model and the same amount of sun power (on average). Moreover, note that no algorithm presents significant levels of battery depletion. In addition, when the average activity is comparable –all cases except ANTCYCLE with $r_{\text{min}} = 0.07$– ANTCYCLE presents lower levels of energy wastage than RANDOM and VIGORITO. This fact is related to the better adaptation to the available energy resources.



(a) Average battery levels



(b) Minimum battery levels



(c) Maximum battery levels

Figure 4.16: Graphics that show the robustness of ANTCYCLE. The x-axis ranges over 10 repeated applications of the system. The boxes show for each of the 120 sensors a corresponding value: (a) the average activity over the simulation time, (b) the minimum battery level over all simulation steps, and (c) the maximum battery level over all simulation steps. Remember that the first day of simulation—when sensors start with full batteries—is not used for logging data.

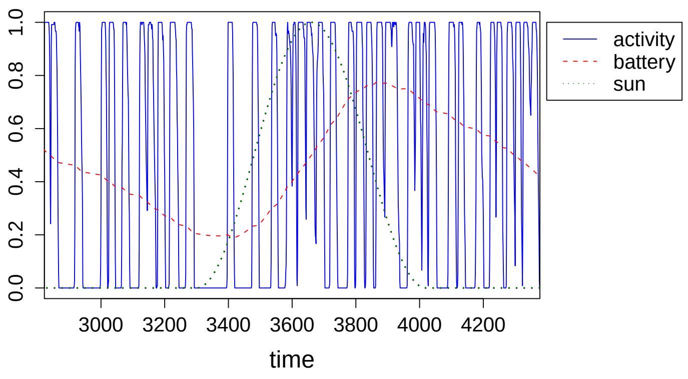
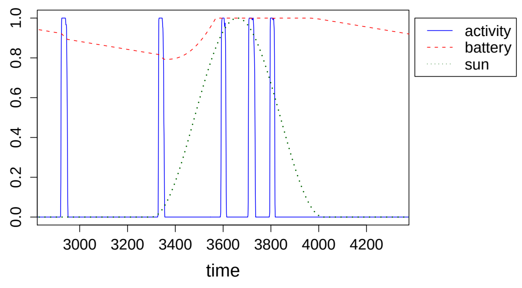
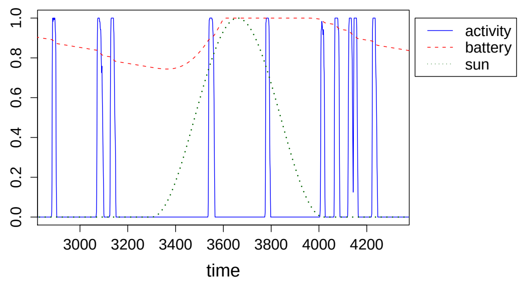


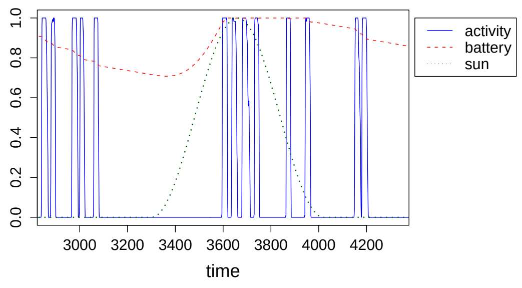
Figure 4.17: Evolution of the mean system activity of ANTCYCLE with the parameters from Table 4.7.



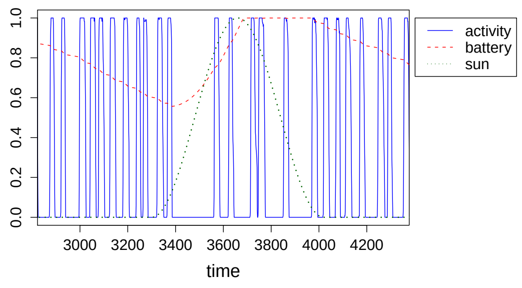
(a) Behavior of ANTCYCLE with $p^a = 0.00000025$



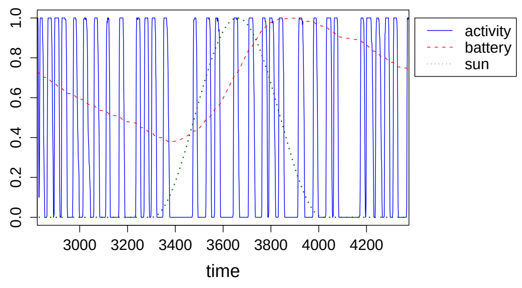
(b) Behavior of ANTCYCLE with $p^a = 0.0000005$



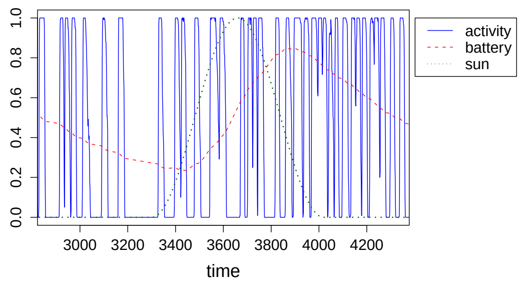
(c) Behavior of ANTCYCLE with $p^a = 0.000001$



(d) Behavior of ANTCYCLE with $p^a = 0.0000025$

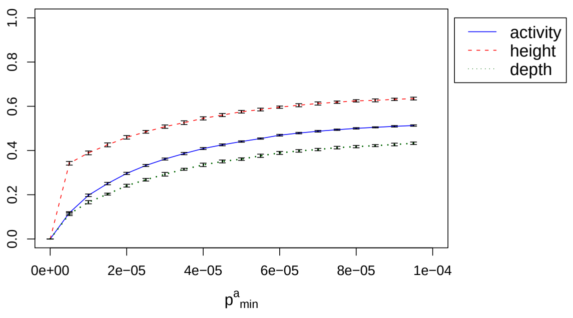


(e) Behavior of ANTCYCLE with $p^a = 0.000005$

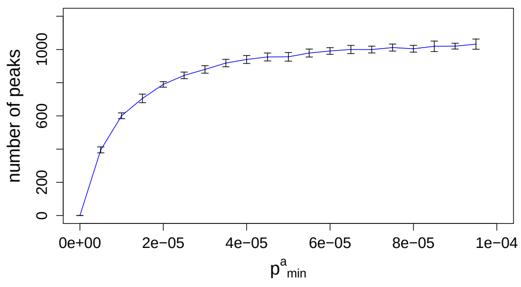


(f) Behavior of ANTCYCLE with $p^a = 0.00001$

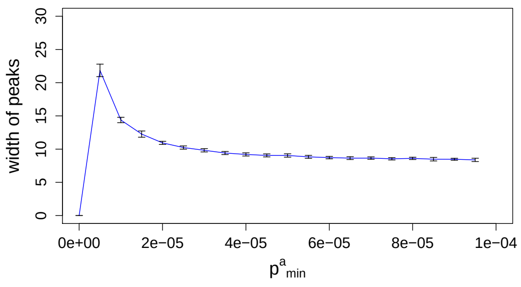
Figure 4.18: ANTCYCLE behavior for selected values of p^a_{min}.



(a) Average system activity, peak height and valley depth

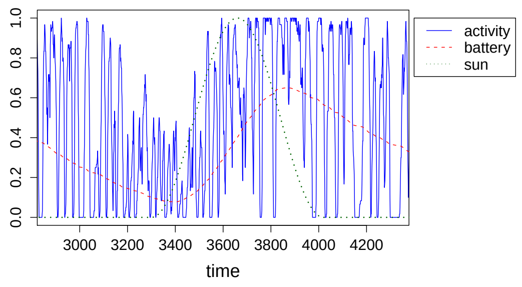


(b) Average number of peaks

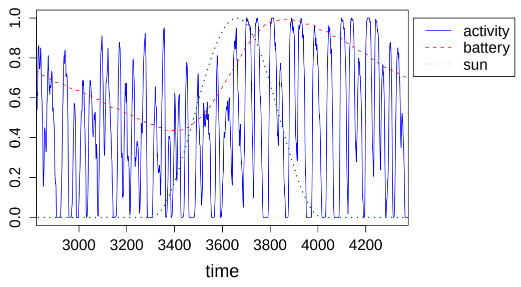


(c) Average width of peaks

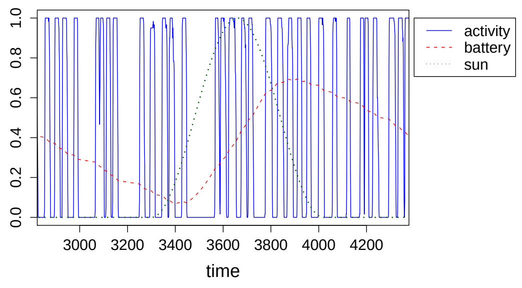
Figure 4.19: ANTCYCLE behavior as a function of p^a_{min}.



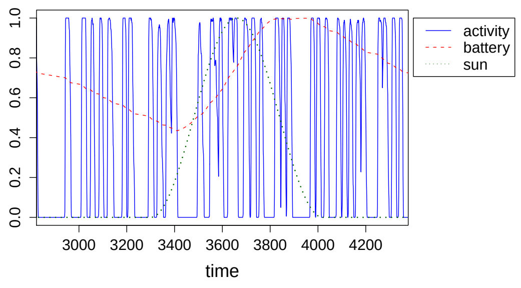
(a) Tracking, $k = 60$



(b) Tracking, $k = 180$

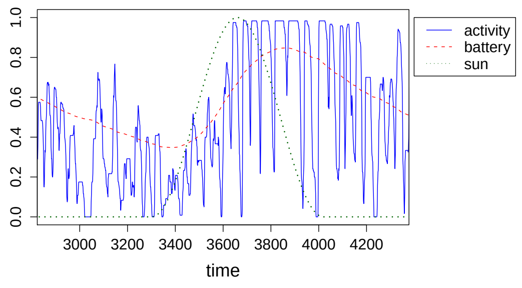


(c) Monitoring, $k = 60$

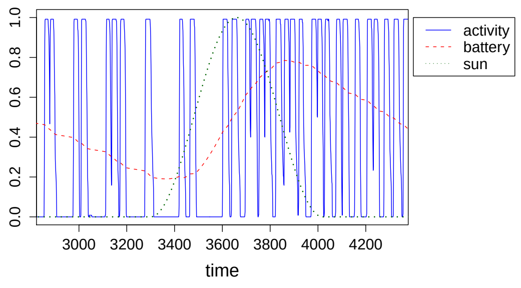


(d) Monitoring, $k = 180$

Figure 4.20: Evolution of the mean system activity of ANTCYCLE when different network sizes are concerned. Graphics (a) and (b) concern the "Tracking" setting of the parameters, while graphics (c) and (d) concern the "Monitoring" setting.



(a) Static Tracking, $k = 120$



(b) Static Monitoring, $k = 120$

Figure 4.21: Behavior of ANTCYCLE for static networks. Graphic (a) concerns the "tracking" application, whereas graphic (b) concerns the "monitoring" setting of the parameter values.

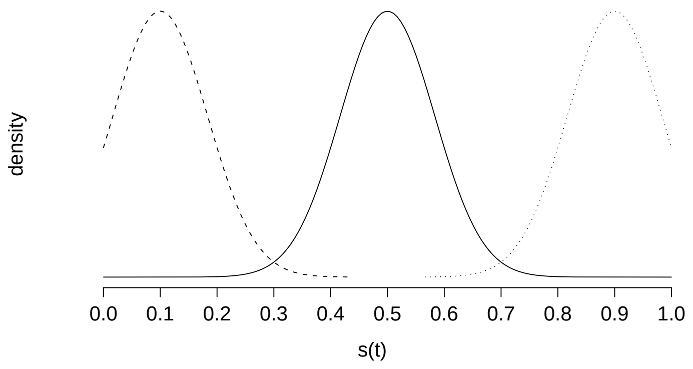


Figure 4.22: Three examples for the distribution of the sun light intensities: with mean $s(t) = 0.5$ (solid line), $s(t) = 0.1$ (slashed line), and $s(t) = 0.9$ (dotted line).

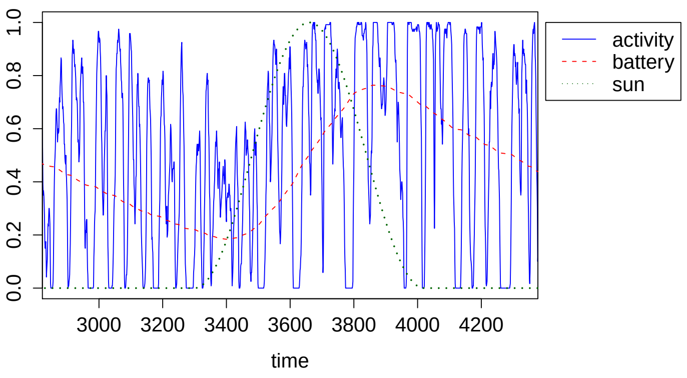
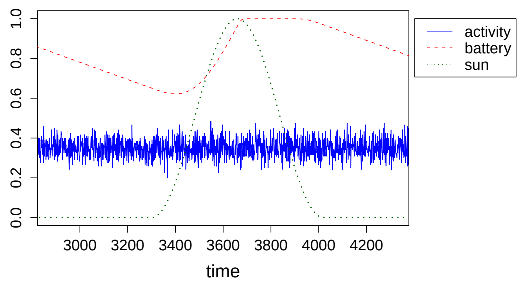
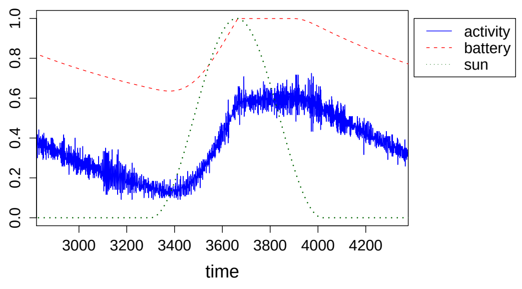


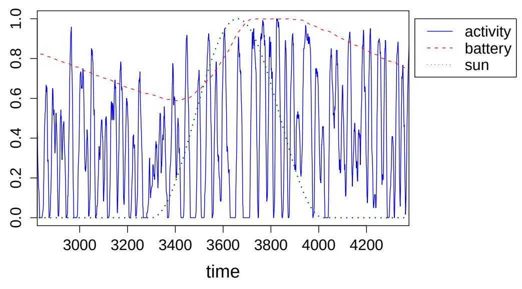
Figure 4.23: Evolution of the mean system activity of Antcycle when using the stochastic function for the sun light intensities from Equation 4.16.



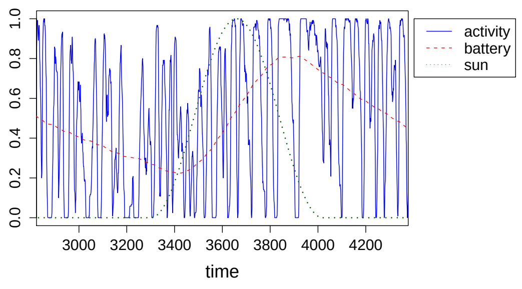
(a) Behavior of RANDOM with mean activity 0.35.



(b) Behavior of VIGORITO with target battery level 0.65.



(c) Behavior of ANTCYCLE with $r_{\min} = 0.05$ and $r_{\max} = 0.1$.



(d) Behavior of ANTCYCLE with $r_{\min} = 0.07$ and $r_{\max} = 0.14$.

Figure 4.24: Comparison of duty-cycling algorithms

4.5 Moving to Real Sensor Networks

The step of adapting ANTCYCLE for its use in real sensor networks is a rather small one. Using ANTCYCLE in real sensor networks (respectively, in sensor network simulators) requires two steps. First, a protocol must be designed that handles the duty-cycling mechanism. Second, the protocols for the remaining tasks such as routing, tracking, and monitoring, must be integrated with the duty-cycling protocol. In the following we briefly outline a possible design for the duty-cycling protocol followed by results. Note that the current version of the duty-cycling protocol assumes that there is a time synchronization algorithm executed by a lower layer of the network.

4.5.1 A Duty-Cycling Protocol

The protocol is organized in periods. Each period has a length of Δ time units (for example, seconds). Moreover, each period is divided in two phases: the first phase is concerned with actions for the management of duty-cycling, whereas the second phase is dedicated to application-specific tasks of sensor nodes. A schematic view of this organization is shown in Figure 4.25. The first phase of each period is generally very short. In this phase all nodes may receive transmissions from neighboring nodes. Moreover, each sensor node executes one duty-cycling event. The outcome of the first phase decides if a sensor node will be *active* or *inactive* for the second phase of the corresponding period. In case of being active, a sensor node is available for user-defined applications such as environmental monitoring and tracking. However, if a sensor node is inactive, the radio will be turned off and the sensor node will sleep until the start of the subsequent period.

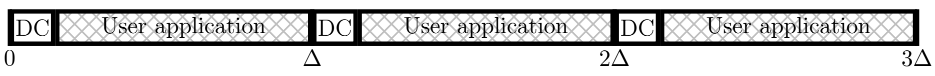


Figure 4.25: Time management between the duty-cycling mechanism and possible user applications. The protocol is organized in a sequence of time periods of length Δ. The first phase of each period is dedicated to duty-cycling (DC), whereas the second phase is concerned with user applications.

In the first phase of each period, each sensor node executes exactly one *duty-cycling event*. The exact time for the execution of this event is randomly chosen by each sensor node in each period. In addition to the state variable S_i (see Section 4.2.1), a sensor node i maintains a queue Q_i for incoming duty-cycling messages from neighboring sensor nodes. Messages $m \in Q_i$ contain a single field m_{activity}. Let us assume that m was sent by sensor node j. In this case, field m_{activity} contains the value of the state variable S_j of sensor node j at the time the message was sent. During the execution of sensor node i's duty-cycling event, the value of state variable S_i is updated depending on the messages in Q_i. Subsequently, sensor node i sends a duty-cycling message, with the new value of S_i, using a certain transmission power level which depends on the battery level. More specifically, the value of state variable S_i of a sensor node i is computed as follows:

$$S_i := tanh\left(g \cdot (S_i + \sum_{m \in Q_i} m_{\text{activity}})\right) \quad , \tag{4.17}$$

where g is again the gain parameter whose value determines how fast the value of variable S_i diminishes. After this update, all messages are deleted from Q_i, that is $Q_i := \emptyset$.

For the working of the duty-cycling mechanism, the choice of the power level for the transmission of the duty-cycling messages plays a crucial role. We assume a standard antenna model which allows sensor nodes—for each transmission—to choose from a finite set $P = \{P^1, \dots, P^n\}$ of different transmission power levels[2]. For the sake of simplicity we identify transmission power levels with the transmission radii to which they translate. In other words, P_i (for $i = 1, \dots, n$) henceforth refers to the transmission radius of the corresponding transmission power level. The choice of a transmission radius for sensor node i depends on its battery level $b_i \in [0, 1]$. In this context, note that the transmission radius r_i as defined in Equation 4.7 represents somehow an *ideal transmission radius*, which must be translated into a corresponding *real transmission radius* denoted by T_i. This may be done as follows: $T_i := P^k \in P$ such that

$$r_i \in \Big((P^{k-1} + P^k)/2, (P^k + P^{k+1})/2\Big] \tag{4.18}$$

At this point it is important to realize that the transmission radius T_i is only used for sending the duty-cycling message. For other messages that are sent in the second phase of each period, the user application is responsible for choosing transmission radii. The duty-cycling event described above is summarized in Algorithm 6.

Algorithm 6 Duty-cycling event of a sensor node i

```
1: Compute new value for state variable $S_i$ (see Eq. 4.17)
2: Calculate $a_i$ (see Eq. 4.1)
3: if $a_i = 0$ then
4:     Draw a random number $p \in [0, 1]$
5:     if $p \leq p_a$ then $S_i := S_a$ and $a_i := 1$ endif
6: end if
7: Determine transmission power level $T_i$ (see Eq. 4.18)
8: Send duty cycling message $m$ with $m_{\text{activity}} := S_i$ with transmission power level $T_i$
```

As mentioned above, the battery level of the sensor nodes is responsible for their choice of a transmission power level for sending the duty-cycling message. Therefore, the battery level of course affects the communication topology in the context of the duty-cycling mechanism. As an example, consider a network of 120 nodes randomly located with static positions in a region of one square kilometer. Suppose that the minimum transmission power level P_1 reaches a distance of 100 m and the maximum transmission power level reaches a distance of 200 m. In Figure 4.26(c) we show the communication topologies that result from fully charged batteries. Over time nodes will consume energy and, especially during night, battery levels will start to decrease. This decrease in the amount of available energy causes changes in the communication topology due to the effect of Equation 4.7. The topology obtained with the same node locations as in the previous example but with batteries filled just to half of their maximum capacity is shown in Figure 4.26(b). Finally, Figure 4.26(a) shows the induced topology when batteries are nearly empty.

[2]Current sensor hardware such as iSense nodes or SunSPOTs are equipped with such antennas.

Table 4.11: Energy parameters.

Tx/Rx (4 bytes)	7.43μC
Radio On	12.8mA
Radio Off	0.025mA
Battery capacity	2600μC
Energy harvesting (f)	1.6W
Max. Tx Power	500m

4.5.2 Experimental Evaluation

For the experimental evaluation we used the sensor network simulator *Shawn* [68], which is a discrete event simulator with a very high level of parameterization. The user can easily run experiments simulating the behavior of different sensor nodes and also add own sensor node specifications. A peculiarity of Shawn is the fact that packet collisions are not explicitly considered. Instead, Shawn simulates these collisions and the consequent packet loss under different constraints and in different scenarios. Thus, any packet-loss model can be implemented. Moreover, we did not implement directly the algorithm in Shawn, but decided to contribute with this algorithm to the *Wiselib* [13] (a general purpose cross-platform library for WSNs). Algorithms implemented in Wiselib can be used in Shawn and also executed in different kinds of sensor networks with independence of many technical specifications, manufacturers or the operating systems used in each node.

We decided to experiment with *iSense* sensor node hardware from Coalesenses GmbH [40]. For this purpose, the specification of *iSense* nodes was added to Shawn. Note that this hardware is currently used by two of the largest European projects on sensor networks: WISEBED [67] and FRONTS [66]. These sensor nodes use a Jennic JN5139 chip, a solution that combines the controller and the wireless communication transceiver in a single chip. The controller has a 32-bit RISC architecture and runs at 16Mhz. It is equipped with 96kb of RAM and 128kb of serial flash. The maximum transmission power level of iSense nodes reaches a distance of about 500m in all directions in open air conditions. In our simulations, iSense nodes were equipped with solar panels. According to their documentation, iSense nodes require $0.025mA$ to work without using any additional peripheral such as the radio or the sensing devices. The state in which the radio is also turned on requires a power supply of $12.8mA$. Additionally, to receive or send a message with 4 bytes of information, as required by duty-cycling messages, implies a consumption of $7.43\mu C$. The batteries have a maximum capacity of $2600\mu C$. Energy harvesting by solar panels can reach a maximum nominal value of 1.6W. This information is summarized in Table 4.11. Finally, let us mention that iSense nodes offer 6 possible transmission power levels, in addition to the state in which the radio is turned off. As an example, the corresponding five transmission radii other than the maximum one are obtained by reducing the maximum transmission radius by $\frac{1}{6}$, $\frac{2}{6}$, $\frac{3}{6}$, $\frac{4}{6}$, and $\frac{5}{6}$.

As no specific user application is considered for the second phase of each period, the energy consumption of phase two of the proposed protocol has to be simulated. This is done by removing an amount e_{awake} of energy from the battery for each execution of phase two. In order to compare the results of the system presented in Section 4.3 with the duty-cycling protocol executed by Shawn, we repeated the simulations for static networks of 120 nodes (see

Table 4.12: Distribution of the energy consumption in the duty-cycling protocol

	Task	Energy (%)
Duty-cycling	Tx	0.757
	Rx	18.591
	Idle	0.001
	Active	0.035
User application		80.616

Section 4.4.4) with iSense sensor nodes simulated by Shawn. The chosen parameter values are the same as the ones presented in Table 4.5. It is important to note that the information which refers to the power profile of the iSense nodes is obtained by properly rescaling the values from Table 4.11 to the $[0, 1]$ range. Apart from that, the length of a period was set to 60 seconds, and the length of the first phase was set to 0.05 seconds.

In Figure 4.27 we show the behavior of ANTCYLE obtained when simulating the duty-cycling protocol with Shawn. For this initial experiment no packet loss was considered. Note that the system behavior is very similar to the one of ANTCYCLE when executed by a discrete event simulator (see Figure 4.21(a)). This result can be seen as a proof of concept for the adaptation of ANTCYCLE to real sensor networks.

With the next experiment we aim at studying the robustness of the system with respect to communication failures. The experiment consists in simulating the duty-cycling protocol under different packet loss rates. A packet loss rate of $p_{loss} \in [0, 1]$ means that the probability of correctly receiving a message is $1 - p_{loss}$. We repeated the experiment as outlined above for all packet loss rates between 0 and 1, in steps of 0.01. The results are shown in terms of the obtained *mean system activity* for each considered packet loss rate in Figure 4.29(a). It can be observed that the behavior of the system does not visibly change until a packet loss rate of about 0.3. This means that the proposed system is quite robust against packet loss. Only for packet loss rates greater than about 0.3 the system behavior degrades.

4.5.3 Additional Realistic Experiments with the Shawn Simulator

In the following, we present a study of how the energy of the sensor network is spent. Table 4.12 presents the energy consumed by the user application against the energy consumed by the duty-cycling mechanism over a whole simulation of 43200 periods (that is, 30 days). The energy spent by duty-cycling is hereby split into the *idle* and *active* states as well as the energy spent for transmitting the duty-cycling messages (Tx) and receiving duty-cycling messages (Rx). Concerning duty-cycling, note that message reception is the task which consumes most of the energy. In total, the duty-cycling mechanism consumes approximately 20% of the total amount of spent energy. This may seem quite high at first. However, consider that this percentage is strongly dependent on the value of e_{app}, which we have set to a very moderate value of 0.001. Increasing this value will obviously cause a decrease in the percentage of energy spent by duty-cycling.

After these initial studies we will now test the duty-cycling mechanism in two adversarial scenarios. In first place, it is shown how the system responds to situations with communication failures. In second place, the behavior of the system is studied in scenarios where energy harvesting is limited, for example, due to cloudy weather. Finally, we present a mechanism for the automatic parameter adaptation of the system for what concerns different network sizes. This is an important aspect for an algorithm included in a generic algorithm library such as Wiselib.

Effect of Packet Loss

With the next experiment we aim at studying the robustness of the system with respect to communication failures. The experiment consists in simulating the duty-cycling protocol under different packet loss rates. A packet loss rate of $p_{loss} \in [0, 1]$ means that the probability of correctly transmitting any message is $1 - p_{loss}$. We repeated the initial experiment as outlined above for all packet loss rates between 0 and 1, in steps of 0.01. The results are shown in terms of the obtained *mean system activity* for each considered packet loss rate in Figure 4.29(a). It can be observed that the behavior of the system does not visibly change until a packet loss rate of about 0.3 is considered. This means that the proposed system is surprisingly robust against packet loss. Only for packet loss rates greater than about 0.3 the system behavior degrades.

Limited Energy Harvesting

Another interesting question concerns the possible change in system behavior when cloud densities greater than zero are considered; in other words: when energy harvesting is limited by bad weather. Therefore, we repeated the initial experiment with a range of different cloud densities between 0 and 1. Figure 4.29(b) shows the evolution of the obtained mean system activity when moving from low to high cloud densities. As expected, with increasing cloud density the mean system activity decreases. Interestingly, the relation between cloud density and the mean system activity is linear.

Adapting to Different Network Sizes

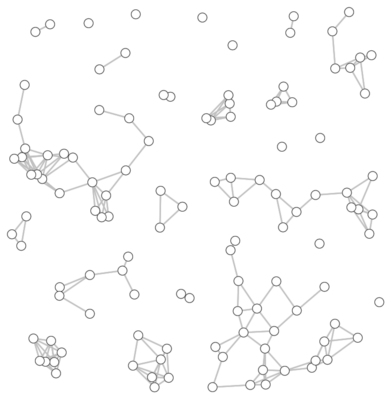
In Figure 4.29(c) we show the average activity for networks with a number of nodes $k \in [1..300]$. The average activity increases with the number of nodes until the network has between 90 and 100 nodes. In this region the system reaches a 60% of average activity that is preserved when the number of nodes is further increased. Although the scaling mechanism is used, it is difficult to obtain the same average activity when the size of the network is reduced. In the present case, this results in the average activity decreasing linearly when the number of nodes decreases. This behavior could be expected, less nodes means that to obtain the same amount of stimulus you need to receive many more messages per cycle. Our scaling algorithm does not consider any changes in the rate of messages sent. However, the scaling mechanism still allows the system to self-synchronize. When the number of nodes is increased, scaling the parameters is enough to reduce the activity in the network and, as a consequence, the number of messages sent. Therefore is not strange that the system is able to reach the average activity of the initial case with 120 nodes.

Monitoring Applications

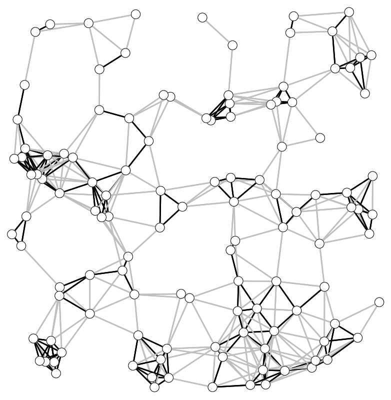
In this section, we have only experienced with the set of parameters used for tracking applications. However, the parameters for monitoring applications may, of course, also be used by the simulator. Figure 4.30 shows the activity scheme generated by ANTCYCLE for monitoring applications. The behavior observed is quite similar to that obtained in previous simulations (see Figure 4.17) that considered monitoring schemes.

Adding Mobility

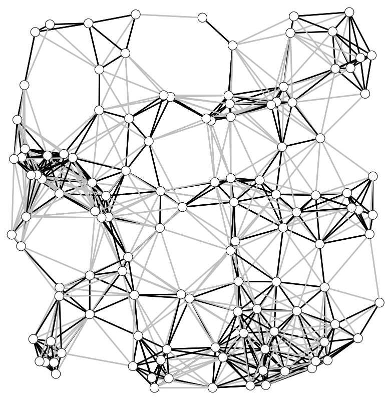
Shawn is mainly aimed at working with static WSNs. This is a reasonable assumption for most WSNs as mobility is generally difficult to achieve. However, in some scenarios such as nodes deployed on vehicles or buoys, mobility must be taken into account. Previously, in Section 4.4, we showed that ANTCYCLE was able to work both with mobile and static nodes. The following experiments are performed using the same version of the RDM mobility model as explained in Section 4.2.1. Note that RDM and some of its variants are included in the Shawn simulator. In Figure 4.31(a), respectively Figure 4.31(b), we show the behavior of ANTCYCLE in Shawn when mobile nodes are used for tracking, respectively monitoring, applications. The results obtained show that the algorithm is still able to adapt its performance to the energy available in the system by dynamically adapting the radius of transmission of the antennas or the probability of spontaneous awakening depending on the parameters set.



(a) Edges in gray show the communication topology when the transmission power levels are such that distances of up to 100 m are reached.



(b) Edges in black are the edges that were shown in gray in (a). Edges in gray show the new communication links obtained when the transmission power level is increased such that distances of up to 150 m are reached.



(c) Edges in black are all edges from (b). Edges in gray show the new communication links obtained when the transmission power level is increased such that distances of up to 200 m are reached.

Figure 4.26: Different communication topologies in a network with 120 randomly located nodes when different transmission power levels are considered.

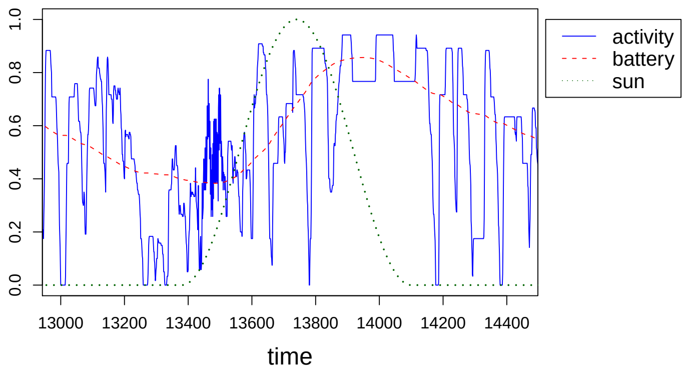
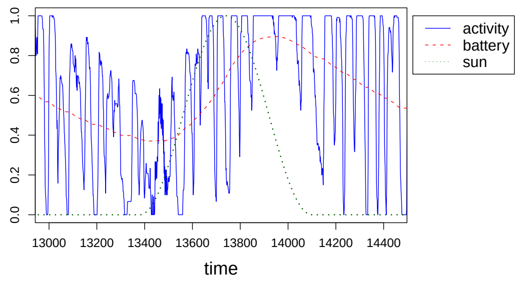
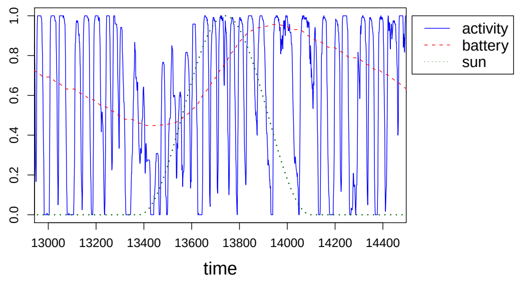


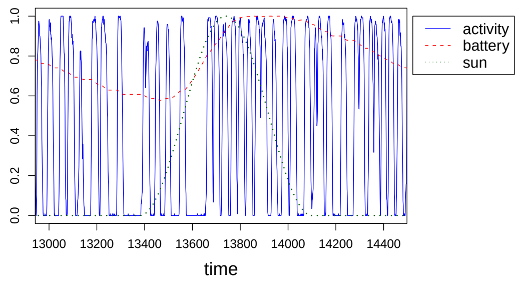
Figure 4.27: Evolution of the mean system activity of ANTCYCLE when run on iSense sensor nodes simulated by the Shawn sensor network simulator



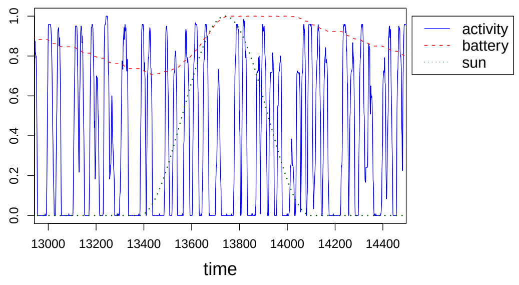
(a) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.15$



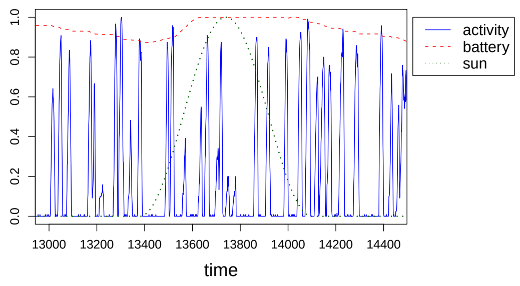
(b) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.30$



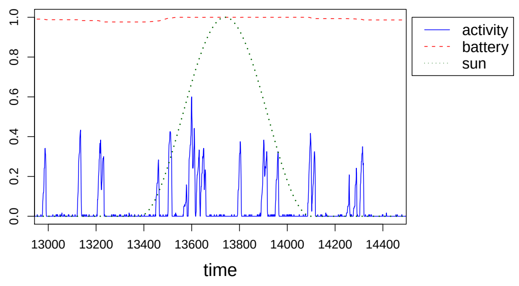
(c) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.45$



(d) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.60$

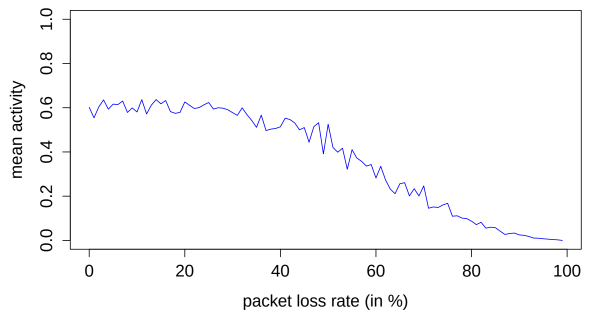


(e) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.75$

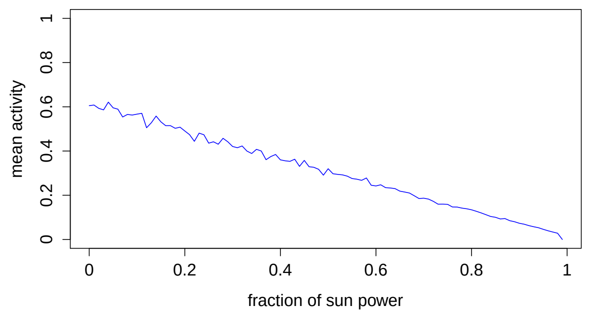


(f) Evolution of the mean system activity of ANTCYCLE with $p_{loss} = 0.90$

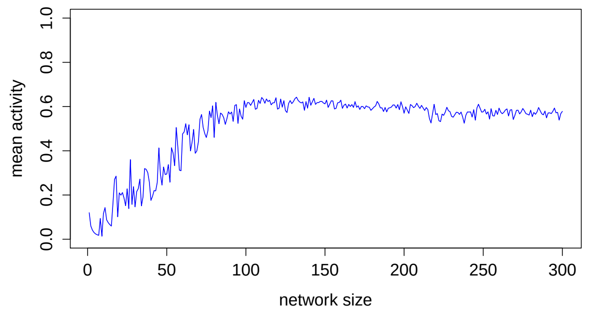
Figure 4.28: ANTCYCLE behavior for selected values of p_{loss}.



(a) Results for different packet loss rates.



(b) Results for different cloud densities.



(c) Results for different network sizes.

Figure 4.29: Behavior of the duty-cycling mechanism under varying conditions

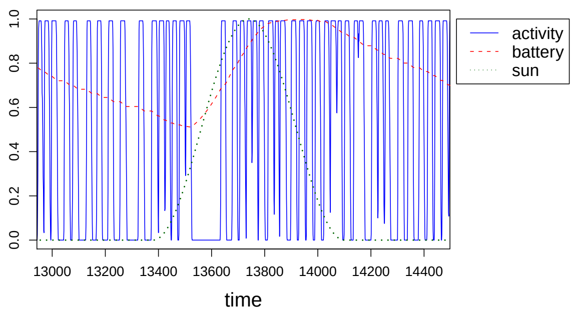
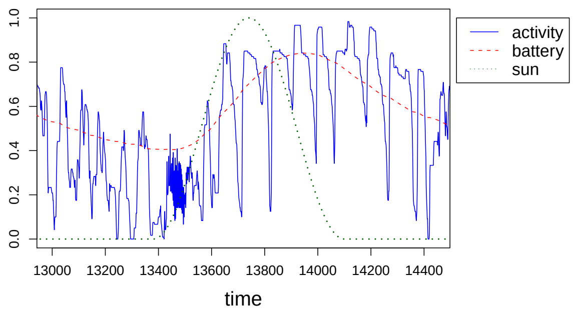
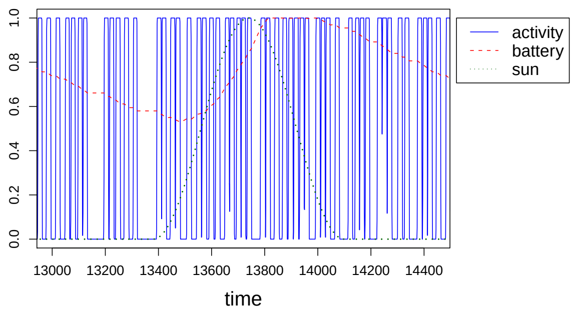


Figure 4.30: Behavior of the duty-cycling mechanism for monitoring applications



(a) Behavior of ANTCYCLE for tracking applications using mobile nodes.



(b) Behavior of ANTCYCLE for monitoring applications using mobile nodes.

Figure 4.31: Behavior of ANTCYCLE when mobile networks are concerned.

4.6 Conclusions and Future Work

In this chapter we have presented an in-depth study of the properties of a self-synchronized duty-cycling mechanism, called ANTCYCLE, that is aimed for the use in sensor networks with energy-harvesting capabilities. For this purpose, we first studied the properties of the fluid neural network model on which ANTCYCLE is based. The study of this model allowed us to design different parameter settings for potentially different applications such as tracking and monitoring. We also studied ANTCYLE with respect to robustness and scaling, as well as its behavior when applied to static networks. Finally, we showed an implementation of ANTCYCLE for real sensor networks and presented results that have shown that ANTCYCLE is very robust with respect to packet loss.

The key aspects of ANTCYCLE are as follows. It is a fully de-centralized algorithm that can adapt the way in which duty-cycling is performed to changing energy availabilities. That is, when little energy is available in the batteries of the individual nodes, duty-cycling is adapted such that either the frequency or the length of the awake-phases are reduced. On the opposite, when a lot of energy is available, the system automatically increases either the frequency or the length of the awake-phases. Moreover, no prior knowledge about the conditions for energy harvesting are needed. The system is able to adapt to changing conditions on the fly.

In the last section of this chapter we have used Shawn, a sensor network simulator, which is prepared to access the Wiselib to test our algorithm against different scenarios in a more realistic and trustable way. The results show that our algorithm is robust and, even when the scenario where the sensor network is deployed presents adversarial conditions, the algorithm is still able to obtain a good performance.

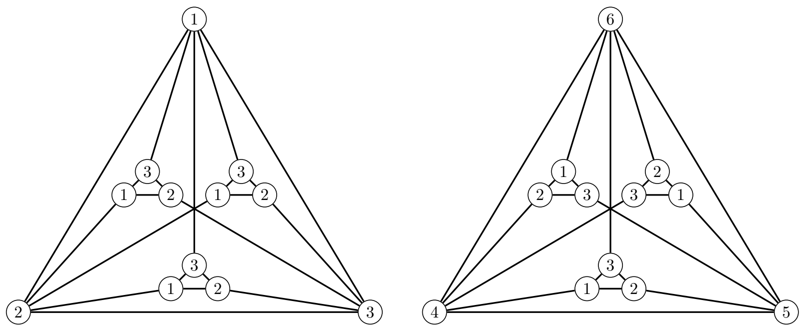
Further studies may focus on analyzing the performance improvements from applications perspective depending on whether ANTCYCLE is in use or not. Remember that duty-cycling algorithms do not aim at performing network or user tasks. The goal of duty-cycling is improving the performance of applications thanks to a better management of energy resources. In the years to come, when sensor networks become even more popular, many new applications will be developed for sensor networks. The aim of this work is to enable such applications to be run on networks whose nodes perform duty-cycled state changes to extend network lifetime. Although we have offered enough facts that sustain the benefits of using ANTCYCLE in such cases, the importance of duty-cycling algorithms in WSNs will not be recognized until this and similar algorithms start enhancing the performance of real applications. Moreover, we expect that the effort of including this algorithm in the Wiselib encourages developers to explore the benefits of including ANTCYCLE in their applications.

Chapter 5

Distributed Graph Coloring

Given an undirected graph $G = (V, E)$, where V is the node set and E is the edge set, and a number $k > 0$ of colors, a valid *k-coloring* of the graph is the assignment of exactly one color to each node such that adjacent nodes (that is, nodes that are connected by an edge) do not share the same color. Formally, we say that a k-coloring of an undirected graph $G = (V, E)$ is a function $c : V \to \{1, 2, \ldots, k\}$ such that $c(u) \neq c(v)$ for each edge $(u, v) \in E$. The optimization version of the *graph coloring problem* (GCP), which is NP-hard [110], consists in finding the minimum number k^* of colors such that a valid k^*-coloring can be found. This number is called the *chromatic number* of graph G and is denoted by $\chi(G)$. The GCP is quite generic. Practical applications originate especially from problems that can be modeled by networks and graphs; for example, communication networks. Several tasks in modern wireless ad-hoc networks, such as sensor networks, are related to graph coloring. Examples include TDMA slot assignment [88], detection of mobile objects and reduction of signaling actuators [210], distributed MAC layer management [79], energy-efficient coverage [32], delay efficient sleep scheduling [132] or wakeup scheduling [115]. Due to the distributed nature of these networks, algorithms for solving problems related to graph coloring are generally also required to be distributed [136]. Such algorithms make an exclusive use of local information for deciding the color of the nodes, that is, they are characterized by the absence of any central control mechanism. The goal of this chapter is to devise an algorithm for generating valid colorings in a distributed manner.

The distributed conception of an algorithm is generally beneficial for its scalability. Moreover, in comparison to centralized approaches; it is generally much easier to adapt a distributed algorithm to dynamic changes during execution. Unfortunately, the exclusive use of local information is often not sufficient to completely capture the internal structure of certain graphs or networks. The following example helps to illustrate the tradeoff between generating colorings from a local and a global perspective. Figure 5.1 shows a graph that has been constructed using four different triangles, that is, complete graphs of three nodes. Here, we distinguish between three *inner triangles* (the three groups of nodes that are close together) and one *outer triangle.* The three inner triangles are connected to the outer triangle such that each node of a specific inner triangle is connected to a different node of the outer triangle. Even in a distributed manner it is fairly easy to obtain optimal colorings for each of the inner triangles. Depending on the specific color assignment concerning the three inner triangles, the outer triangle may be colored with the same three colors (as in Figure 5.1(a)) or with three additional colors (as in Figure 5.1(b)). Unfortunately, when coloring the three inner triangles independently of each other, the probability for the latter case to happen is quite high, especially when the complexity of the graph is increased by adding more inner triangles. As mentioned already above, one of the key difficulties when coloring graphs in a distributed manner is that each node is only provided with local information and, therefore, it is unable to detect situations such as the one from Figure 5.1(b).



(a) 3-colored composition of triangles (b) 6-colored composition of triangles

Figure 5.1: Simple graph topology (composed of three inner triangles and one outer triangle). (a) shows an optimal 3-coloring, while (b) shows a sub-optimal 6-coloring. Distributed algorithms provide most often a 6-colored solution, because global knowledge is necessary to capture the graph structure.

Our Contribution

In this chapter we propose a distributed algorithm for graph coloring based on the calling behavior exhibited by male Japanese tree frogs (see Figure 5.2) for the attraction of females. Several researchers have observed that male Japanese tree frogs decouple their calls [193]. This property has evolved because females can only localize the males when their calling is not too close in time. In [3] Aihara et al. proposed a theoretical model for simulating the behavior of these frogs. The authors describe an oscillator system, where each oscillator has a phase $\theta \in [0, 2\pi]$ that changes over time with frequency ω (where 2π is the time interval between two calls of the same frog). When the phase reaches 2π, the oscillator fires and returns to the baseline phase ($\theta = 0$). The proposed system works such that oscillators try to maximize the distance between their phases. This model works nicely for the desynchronization of two oscillators. However, when more than two oscillators are involved, the model does not accurately reflect the real behavior of the frogs. A subsequent work [2] mentions some potential applications of this model in artificial life and robotics. In both works the author(s) mention the limitations of the systems when operating with groups of more than two coupled oscillators. In fact, already with three oscillators the final solution (and its stability) strongly depends on the initial variable settings.

The desynchronization of the frogs' calls is achieved in a self-organized way. Therefore, the proposed algorithm, which is inspired by this self-desynchronization mechanism, can be

Figure 5.2: Photograph of a Japanese tree frog (*Hyla japonica*). This photograph was released into the public domain by Masaki Ikeda. No licenses apply. The image is also available from [198].

regarded as a *swarm intelligence* approach [22, 20]. Swarm intelligence (see also Section 3 of this thesis) is a field of computer science which is inspired by the collective behavior of social animals and other self-organizing processes from nature. Successful examples from the literature include *particle swarm optimization* (PSO) [113], which is an algorithm for optimization inspired by bird flocking and fish schooling, and *ant colony optimization* (ACO) [59], which is inspired by the foraging behavior of ant colonies. One of the distinguishing properties of a swarm intelligence approach is the fact that the problem at hand is solved from a local perspective. Moreover, problem solving is based on the cooperation of rather simple entities. Instead of each entity trying to solve the problem by itself, they perform simple tasks from a local perspective. The global problem is solved as a result of cooperation. Therefore, swarm intelligence principles are well suited for their use in distributed algorithms.

Apart from some minor modifications, it makes use of a model of frogs' desynchronization introduced by Aihara et al. in [3]. Moreover, for the purpose of detecting convergence and to store the best solution found, the proposed algorithm makes use of an overlay tree structure rooted in a so-called master node. Such a mechanism is common, for example, in the context of distributed local search algorithms [212]. The algorithm can be easily implemented, for example, in sensor networks. In addition to competitive results it comes with several advantages, such as, for example, low energy consumption or the potential to adapt to changing network topology. However, as mentioned before, the main goal of the algorithm is to obtain valid colorings that use as few colors as possible, while attempting to minimize the number of iterations needed to achieve these results. An extensive experimental evaluation shows that the results of the algorithm are comparable or better than state-of-the-art algorithms in terms of the number of colors used. In particular, the good performance of our algorithm for grid graphs of any size is remarkable. On the downside, the results also show that our algorithm may sometimes require a slightly higher number of communication rounds than the baseline algorithm chosen for comparison.

Before writing this chapter, all the contributions have been made public to other scientists both in international conferences and journals. The resulting techniques have been improved with the advice of referees and colleagues in order to obtain a more refined work. The following list summarizes the publications related to the novel algorithms appearing in this chapter. Remember that a more extended explanation of the publications is available in Section 1.2. Also note that this chapter is based on the articles *Distributed Graph Coloring: An Approach Based on the Calling Behavior of Japanese Tree Frogs* [94] and *FrogSim: Distributed Graph Coloring in Wireless Ad Hoc Networks – An Algorithm Inspired by the Calling Behavior of Japanese Tree Frogs* [95]:

- Hugo Hernández and Christian Blum. Implementing a Model of Japanese Tree Frogs' Calling Behavior in Sensor Networks: a Study of Possible Improvements. In *GECCO BIS-WSN 2011 – Proceedings of the Workshop on Bio-Inspired Solutions for Wireless Sensor Networks.* Pages 615–622. ACM Press, New York, 2011.

- Hugo Hernández and Christian Blum. Distributed Graph Coloring in Wireless Ad Hoc Networks: A Light-weight Algorithm Based on Japanese Tree Frogs' Calling Behaviour. In *WMNC 2011 – Proceedings of IFIP/IEEE Wireless and Mobile Networking Conference.* Pages 1–7. Best Paper Award. IEEE Press, 2011.

- Hugo Hernández, Christian Blum. Distributed Graph Coloring: An Approach Based on the Calling Behavior of Japanese Tree Frogs. Swarm Intelligence, 2012. In press.

- Hugo Hernández, Christian Blum. FrogSim: Distributed Graph Coloring in Wireless Ad Hoc Networks – An Algorithm Inspired by the Calling Behavior of Japanese Tree Frogs. Telecommunication Systems, 2012. In press.

Prior Work on Graph Coloring

When reviewing previous work, a distinction must be made between centralized and distributed algorithms. Concerning centralized algorithms, the literature offers both exact approaches that guarantee to find an optimal solution in bounded time and approximate algorithms. A recent survey can be found in [139]. An example of an exact algorithm is [138], although—due to the intractable nature of the GCP—larger problem instances can only be tackled efficiently by heuristic approaches. Especially effective are the tabu search algorithm from [16], the evolutionary algorithm from [137], the hybrid approach combining tabu search and evolutionary algorithms from [133], and the variable space search technique from [98]. These algorithms are currently the best centralized metaheuristics for solving the GCP. Earlier remarkable works are the hybrid evolutionary algorithm from [75], the genetic local search algorithm from [60] and the variable neighborhood search in [10].

When considering distributed algorithms, it is very difficult (if not impossible) to narrow the state of the art down to a small set of algorithms. This is because distributed algorithms may be designed with very different goals. These goals may concern, for example, the performance for particular topologies, the minimization of execution time (or communication rounds), the generation of the best colorings possible, or the performance for dynamically changing topologies. In addition, a general problem is that most proposals are not evaluated on publicly available sets of benchmark instances. Moreover, results are generally not shown per instance, making it difficult to compare to the proposed algorithms. In the following, we focus on algorithms that generate valid solutions and compare their simplicity, solution quality and time complexity.[1] It must also be noted that many of the proposed distributed algorithms were developed for applications in networks of devices with scarce resources. For this reason authors often study the message load the algorithm implies and try to minimize the amount of calculation required by the algorithm. Typically, these algorithms are meant to work on a lower layer of the network in parallel with the applications or information flows that the user may require to send. In [72], Fraigniaud et al. study the effect of the amount of information shared between the nodes on the quality of the obtained colorings.

One of the most general works was presented by Finocchi et al. in [69]. The authors introduced three versions of a distributed algorithm and studied its behavior under various conditions. The authors considered both the problem of obtaining $O(\Delta + 1)$-colorings in as few communication rounds as possible, as well as the problem of generating the best possible colorings without any limit on the number of communication rounds. Here, Δ refers to the maximum degree of a graph. The authors provide extensive experimental results for both cases. Most of their experimentation is based on random graphs, which are not publicly available. However, they also present results on a well-known set of publicly available instances from the DIMACS challenge [71]. As the algorithm proposed in [69] was shown to outperform the state of the art, we chose this algorithm for comparison. Another remarkable work from the field of distributed algorithms for graph coloring is the *distributed largest-first* (DLF) algorithm introduced in [120]. DLF is an improvement over an algorithm from the literature consisting in randomly ordering the nodes and using this order to allow nodes to choose colors with the only rule that colors which are already assigned to neighbors cannot be used. DLF differs from the original algorithm version in the order of the nodes. In DLF, the higher the degree of a node the earlier the node will be able to choose a color. The random order is only used to break ties between nodes which have the same degree. DLF achieves better results than the original algorithm on random graphs.

Concerning distributed algorithms based on swarm intelligence principles, the literature offers, for example, a method inspired by the synchronous flashing of fireflies (see [124]). This algorithm, which allows a simple implementation, reaches valid colorings fast, in a constant number of communication rounds, regardless of the size of the network. However, this work does not focus on minimizing the number of colors. The first intent to use the calling behavior of frogs for graph coloring was based on a system of coupled oscillators and presented in [126]. Valid colorings are obtained by assigning a color to each phase used by the nodes (that is, the oscillators). Therefore, if two nodes are synchronized to exactly the same phase, they will be sharing a common color (the authors consider a function $f : [0, 2\pi] \rightarrow (R, G, B)$, where 2π is the time frame between two callings of the same frog). The main drawback of this approach is that nodes with very near phases will be colored with different colors. As such small deviations usually occur when the number of nodes in the system increases, the algorithm does not obtain competitive results. This work was further extended by adding a parameter for setting the number of allowed phases a priori [125]. Experimentation shows that the system is able to

[1]In the scope of this paper the time complexity is, as usual, measured in terms of *communication rounds*. A communication round is the unit of time in which each node is allowed to send at most one message.

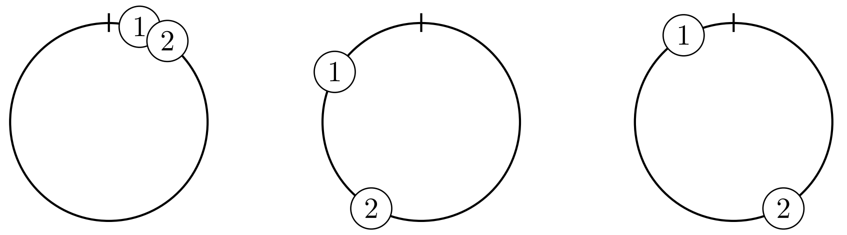
find optimal solutions for small topologies, given that the optimal number of colors is known. Note that in contrast to these works, the algorithm that we propose aims to minimize the number of colors used without any prior knowledge about the optimal solution.

The graph coloring problem has also been studied in the context of belief propagation methods [154] from the field of artificial intelligence. These methods, which belong to the family of message passing algorithms, define ways to adequately propagate and fuse information flows in networks. The goal of belief propagation is that each node—once a stable state is reached—should be able to provide a measure of belief about the satisfaction of a certain predicate. However, belief propagation techniques are generally applied to infere properties of large problem instances (for example, about the distribution of solutions in the search space) rather than to find actual solutions to specific problem instances. Some works about graph coloring from this perspective are those that study the decisional version of the Δ-coloring problem. In [24], the authors provide a survey propagation [140] method—one of the categories of belief propagation methods—for finding valid colorings and in [119] belief propagation is used for counting solution clusters in the search space.

The literature also offers many works that consider distributed graph coloring from a theoretical point of view. Most provide upper bounds for the coloring quality as well as the time complexity under different constraints. Hansen et al. [83] studied the DLF algorithm, which runs in $O(\Delta^2 \log n)$ communication rounds for arbitrary graphs and that was proven to provide good upper bounds for specific topologies. This algorithm was based on the *largest-first* approach which consists in giving priority for choosing a color to the nodes with the highest degree (Δ). This work was further extended by Kosowski and Kuszner [117], who reduced the time complexity to $O(\Delta \log n \log \Delta)$. These authors also proved that some other approaches, like *smallest-last* or *dynamic-saturation*, are not suitable for distributed environments. Later, in [148] Moscibroda and Wattenhöfer introduced an algorithm for obtaining $O(\Delta)$-colorings in $O(\Delta \log n)$ time when considering random geometric graphs and other well-known models for wireless multi-hop networks (no results are given for other topologies). Other theoretical works of interest for the development of new algorithms are: the game theoretic approach for efficient graph coloring from Panagopoulou and Spirakis [152]; the work by Kuhn and Wattenhöfer [121], which introduces a new lower bound on the number of colors used by algorithms that are restricted to one single communication round and a new lower bound on the time complexity of obtaining a $O(\Delta)$-coloring of a graph; the article by Barenboim and Elkin [11], which introduces the first polylogarithmic time algorithm able to color a graph with less than $O(\Delta^2)$ colors; and the work by Gavoille et al. [77], proposing and studying the complexity of two new problems strongly related to the distributed GCP.

Chapter Organization

The rest of this chapter is organized as follows. Section 5.1 describes the behavior of frogs in nature, which has inspired our algorithm. Moreover, existing models are outlined. In Section 5.2 an algorithm for desynchronization of nodes in sensor networks is explained. Additionally, several extensions are discussed and experimentally compared. In Section 5.3 the algorithm for graph coloring is introduced. An extensive experimental evaluation of the proposed algorithm is presented in Section 5.4. Finally, Section 5.5 is dedicated to conclusions and the outline of future work.



(a) Fictitious initial situation with two frogs calling close in time.

(b) The system after some iterations. The system has managed to increase the distance between the calls of the two frogs.

(c) Final situation. The two frogs call in perfect anti-phase.

Figure 5.3: Graphical illustration of the working of a system of two coupled oscillators. The circle in all three drawings represents the time frame between two calls of the same frog (2π), the calling period. The nodes marked by integer numbers 1 and 2 indicate the phase of the corresponding frogs, that is, the moment of time in which they call. (a) shows a fictitious initial situation. (b) shows the situation after some iterations. Clearly the system tries to put some distance between the calling of frogs 1 and 2. (c) shows an optimal final situation in which the frogs (or oscillators) are in perfect anti-phase, that is, their respective calls have reached the maximum distance in time (half a circle).

5.1 Modeling the Calling Behavior of Japanese Tree Frogs

Different studies (see, for example, [193]) have shown that male Japanese tree frogs use their call to attract females. Apparently, females of this family of frogs can recognize the source of the call to determine the current location of the corresponding male. A problem arises when two males are too close in space and communicate at the same time. In this case, females are not able to detect where the calls came from, unless the males desynchronize their sounds in time. The way males manage to desynchronize is an example of self-organization in nature.

More recently, Aihara et al. [3] introduced a formal model based on a set of coupled oscillators, each one simulating the phase change in the calling period of a single frog. As oscillators are associated to frogs, we will use both terms in the following with the same meaning. The basic operation of this model is illustrated graphically in Figure 5.3. The circle represents—in all three drawings—the time frame between two calls of the same frog (2π), the calling period. The nodes marked by numbers 1 and 2 indicate the phase of the corresponding frogs, that is, the moment in time in which they call. Note that the oscillators are not able to reach perfect anti-phase in a single step. In general, an indefinite number of steps is needed before reaching the stable situation corresponding to perfect anti-phase. Moreover, the difficulty of reaching the optimal configuration tends to increase with an increasing number of frogs and also with an increasing degree of interaction between them (note that two frogs that can not hear each other do not influence each other).

The system introduced by Aihara et al. [3] works as follows. Each oscillator i has a phase $\theta_i \in [0, 2\pi]$ that changes over time with frequency ω_i (where 2π is the time interval between two calls of the same frog, the calling period). When the phase reaches 2π, the oscillator fires and returns to the baseline. In addition, oscillators may be coupled with other oscillators. In case an oscillator j is coupled to an oscillator i, when oscillator i fires oscillator j receives a boost and changes the frequency of firing in the next round depending on the gap $\Delta_{ji} \in [0, 2\pi]$ (see below) between both oscillators. These changes do not happen instantly upon receiving the stimulus. The corresponding oscillator rather waits until it fires. The model can be summarized in the following equations. First, the behavior of an isolated oscillator i is modelled as follows:

$$\frac{d\theta_i}{dt} = \omega_i. \tag{5.1}$$

Assuming that oscillators j and i are coupled, the gap between their (current) phases is defined as:

$$\Delta_{ji} = \theta_j - \theta_i. \tag{5.2}$$

Now, the change in the behavior of oscillator j as influenced by oscillator i can be described as follows:

$$\frac{d\theta_j}{dt} = \omega_j + g(\Delta_{ji}), \tag{5.3}$$

where $g(\cdot)$ is the phase shift function which is responsible for changing the phase of the frogs that are influenced by other frogs. In [3], the authors suggest the use of the following phase shift function:

$$g(x) = -\alpha \sin(x). \tag{5.4}$$

When only two oscillators are concerned, the oscillator system is said to be in a *stable situation* when the following two conditions are satisfied:

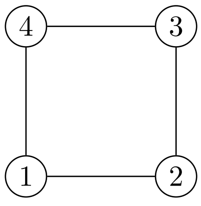
$$\begin{aligned} \Delta_{12} &= \Delta_{21}, & (5.5)\\ g(\Delta_{12}) &= 0. & (5.6) \end{aligned}$$

In fact, the system presented in [3] is always able to reach this stable situation in the case of only two coupled oscillators, independently of the initial settings of θ_1 and θ_2. Unfortunately, it is rather hard to characterize a *stable situation* for sytems with more than two oscillators. This is because the optimization goal of the system is not explicitly stated.

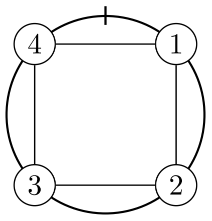
In the following, we model a coupled oscillator system by means of an undirected graph $G = (V, E)$. Each node of G corresponds to exactly one oscillator. Therefore, the terms *node* and *oscillator* will henceforth be used synonymously. Two oscillators are coupled if and only if their corresponding nodes are connected by an edge. In our experience, the implicit optimization goal of the system presented in [3] can be stated as follows:

$$\max \left\{ \sum_{\{(i,j)\in E\}} \min\{|\Delta_{ij}|, 2\pi - |\Delta_{ij}|\} \right\}. \tag{5.7}$$

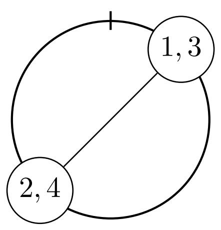
In other words, the system tries to maximize the sum of the differences between the θ-values of coupled oscillators. Figure 5.4 shows two examples of problems that may arise when more than two coupled oscillators are considered. Depending on the initial phases of the oscillators, for both topologies shown in Figures 5.4(a) and 5.4(d) it is possible to reach suboptimal desynchronizations with respect to the implicit optimization goal as stated in Eq. 5.7. These



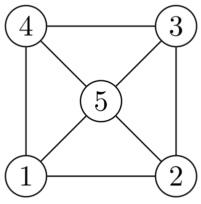
(a) Topology 1



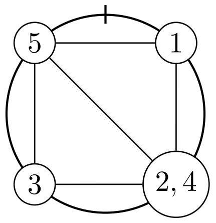
(b) Suboptimal desynchronization of Topology 1 (with 4 different phases)



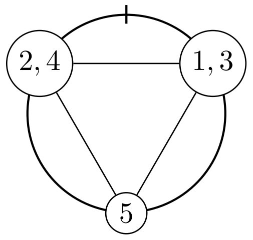
(c) Optimal desynchronization of Topology 1 (with 2 different phases)



(d) Topology 2



(e) Suboptimal desynchronization of Topology 2 (with 4 different phases)



(f) Optimal desynchronization of Topology 2 (with three different phases)

Figure 5.4: Two examples of graph topologies (graphics (a) and (d)) that may cause problems for the desynchronization as performed by the model proposed in [3]. Graphics (b) and (e) show suboptimal desynchronizations (corresponding to stable attractors of the system) for both topologies. In contrast, graphics (c) and (f) show optimal desynchronizations.

configurations are shown in Figures 5.4(b) and 5.4(e). The corresponding optimal desynchronizations are shown in Figures 5.4(c) and 5.4(f). In [3] the authors provide analytical results for using three oscillators and show that there is high sensitivity to the initial phases (only a small subset of the possible initial settings leads to an optimal solution).

The initial model by Aihara et al. [3] was later extended by Mutazono et al. [150]. They used an extended model of anti-phase synchronization for the purpose of collision-free transmission scheduling in sensor networks. To make the system applicable to larger topologies (sensor networks may consist of hundreds of nodes), they introduced weights that regulate the coupling between each pair of oscillators. The resulting phase shift function as introduced in [150] can be described as follows:

$$\delta(x) = \min\{x, 2\pi - x\}, \tag{5.8}$$

$$g(x) = \alpha \cdot \sin(x) \cdot e^{-\delta(x)}. \tag{5.9}$$

Algorithm 7 Sensor event of node i in the desynchronization algorithm

```
1: $\theta_i$ :=recalculateTheta()
2: sendMessage()
3: clearMessageQueue()
```

Due to these weights, the system reaches stable solutions more easily, especially when small values of α are used. The authors experimented with topologies of up to 20 nodes and although the system still exhibited some difficulties in reaching stable solutions, the sensitivity to initial conditions was decreased significantly.

Mutazono et al. [150] compared the results of their system to another mechanism for coupled oscillator desynchronization proposed in [49]. Note that the mechanism from [49] is not based on the calling behavior of Japanese tree frogs. The main difference to frog-inspired systems is the fact that the phase change of a node is made on the basis of only two other nodes. The phase values allow to order all the nodes sequentially from small to large phase values. The nodes whose phase values are used to change the phase value of a node are determined as the predecessor and the successor in this (cyclic) sequence. As shown in [150], both systems achieve similar results although no extensive experimentation is made on a broad-enough set of network topologies: mostly random geometric graphs and hand-made instances with at most eight nodes were used.

Another extension of the system by Aihara et al. [3] was introduced in [126]. The changes concern the use of different weights for the phase shift function and the introduction of a so-called frustration parameter which reduces the coupling between each pair of nodes. The authors show that their system is able to obtain better solutions than the original model for many different topologies as, for example, k-partite graphs, grids or platonic solids. Moreover, the authors observe that the difficulty of desynchronizing a given number of oscillators is mostly determined by the topology which is obtained by their positions and their connections. In comparison, the number of oscillators does not seem to have a high impact on the difficulty.

5.2 Implementation of the original and the extended model in Sensor Networks

First, we provide the implementation of the original model by Aihara et al. in sensor networks. In this implementation, communication rounds are the basic units of time. A communication round corresponds to the calling period (2π) as known from the models presented in the previous section. The only difference is that the length of a communication round is considered to be one time unit. Therefore, the numerical length of a communication round is denoted by 1, instead of 2π. Each sensor node executes exactly one sensor event in each communication round. The moment in time when a sensor node $i \in V$ executes its sensor event is denoted by $\theta_i \in [0, 1)$. Note that θ_i corresponds to the phase of an oscillator from the model(s) presented in the previous section. Moreover, a sensor event includes the sending of exactly one message. Therefore, each sensor node i maintains a message queue M_i for sensor event messages received from other sensor nodes since the last execution of its own sensor event. The pseudo-code of a sensor event is shown in Algorithm 7. In the following we give a rough description of the algorithm. A detailed technical explanation of the functions of Algorithm 7 will be provided later on.

The working of the algorithm requires an a priori organization of the sensor network in form of a rooted tree. The root node of this tree (henceforth also called *master node*) will have some additional functionalities in comparison to the rest of the nodes. Once the tree is available, the master node runs a protocol to calculate the height of the tree, that is, the distance in hops (communication rounds) from the master node to the farthest node in the network. In order to produce a tree with a low height, the distributed method for generating spanning trees with minimum diameter as presented in [28] may be used. Next, the master node triggers the start of the main algorithm by means of a broadcast message. In this message, the height of the tree is communicated to the rest of the nodes as well. Later on it will be described how this overlay tree structure is used to calculate the state of convergence which will be used to stop the algorithm.

As mentioned above, in each communication round, a node i executes its sensor event at time θ_i. First, node i will examine its message queue M_i. If M_i contains more than one message from the same sender node, all these messages apart from the last one are deleted. In general, a sensor event message $m \in M_i$ contains only one real number:

$$m =< \mathrm{theta}_m > \quad , \tag{5.10}$$

where $\mathrm{theta}_m \in [0.1)$ contains the θ-value of the emitter. Based on the messages in M_i, function recalculateTheta() recalculates a new value for θ_i:

$$\theta_i := \theta_i - \alpha_i \sum_{m \in M_i} \frac{\mathbf{sin}(2\pi \cdot (\theta_m - \theta_i))}{2\pi} \quad , \tag{5.11}$$

where $\alpha_i \in [0, 1]$ is a parameter used to control the convergence of the system. In general, the lower the value of α_i the smaller the change applied to θ_i. Note that the multiplication of $(\theta_m - \theta_i)$ by 2π and the division of the result of the sinus function by 2π is necessary for the transformation of the length of a calling period from $[0, 2\pi]$ to $[0, 1]$. Finally, node i sends the following message m (see function sendMessage()):

$$m =< \mathrm{theta}_m := \theta_i > \tag{5.12}$$

To conclude a sensor event, node i deletes all messages from its queue M_i (see function clearMessageQueue()), that is, $M_i = \emptyset$.

It remains to provide a description of the algorithms' stopping mechanism. In this context, recall the rooted tree that is generated before the execution of the main algorithm. The key characteristic of this tree is its height, henceforth referred to by h. It determines the maximum number of communication rounds necessary for the master node to broadcast a message to all other nodes. In turn, it also determines the maximum number of communication rounds necessary to pass information from all the nodes to the master node. The main goal of the tree is to efficiently alert the nodes about when and how to stop executing the sensor events for desynchronization. In the following we assume that the master node knows the size of the network. At each communication round, each node i must communicate the following information to its parent node in the tree: (1) a real number corresponding to the sum of the distances between the old theta values and the new ones concerning all nodes included in the subtree rooted at node i, (2) an integer number that indicates the corresponding communication round. In fact, these values can easily be added to the body of the sensor event messages used by our algorithm. In other words, no additional messages are required.

Note that, for example, in the first communication round only the leaves of the tree will report the differences between their old and new theta values to their parents. This is because the leaves are the only nodes without children. In the second communication round, the parents of the leaves will be able to add these values to their own distances between the old and new theta values from the first communication round and report the aggregated data to their respective parents. Given the height h of the tree, it takes h communication rounds until all the information regarding a specific communication round has reached the master node. This means that the sensor nodes must store the differences between their old and new theta values during h communication rounds. Once all the necessary information reaches the master node, it divides the value by the size of the network and obtains in this way the average change of the theta values in the corresponding communication round. In case this average change is below a certain threshold value (in our case we always used 0.001), the master node broadcasts a stopping message to all nodes, which terminates the algorithm.

5.2.1 Model Extensions

As in the original model, in each communication round a node i executes its sensor event at time θ_i. However, a message $m \in M_i$ has now the following format:

$$m =< \text{theta}_m, \text{relevance}_m > \quad , \tag{5.13}$$

where, as before, $\text{theta}_m \in [0.1)$ contains the θ-value of the emitter and relevance_m is a parameter that depends on the number of messages received by the emitter during the last communication round. This parameter controls the weight that is given by node i to the corresponding message m. In particular, less weight is given to messages that were emitted by nodes that are influenced by many other nodes. The intuition for this definition of the weights is that the θ-values of nodes that are little influenced by other nodes should converge first. This may facilitate the convergence of the θ-values of highly-influenced nodes, which in turn may facilitate that the system reaches a stable situation, a term which refers to a situation in which the θ-values do not change anymore.

Based on the messages in M_i, function recalculateTheta() recalculates a new value for θ_i:

$$\theta_i := \theta_i + \alpha_i \sum_{m \in M_i} \text{relevance}_m * inc[\theta_m - \theta_i] \quad , \tag{5.14}$$

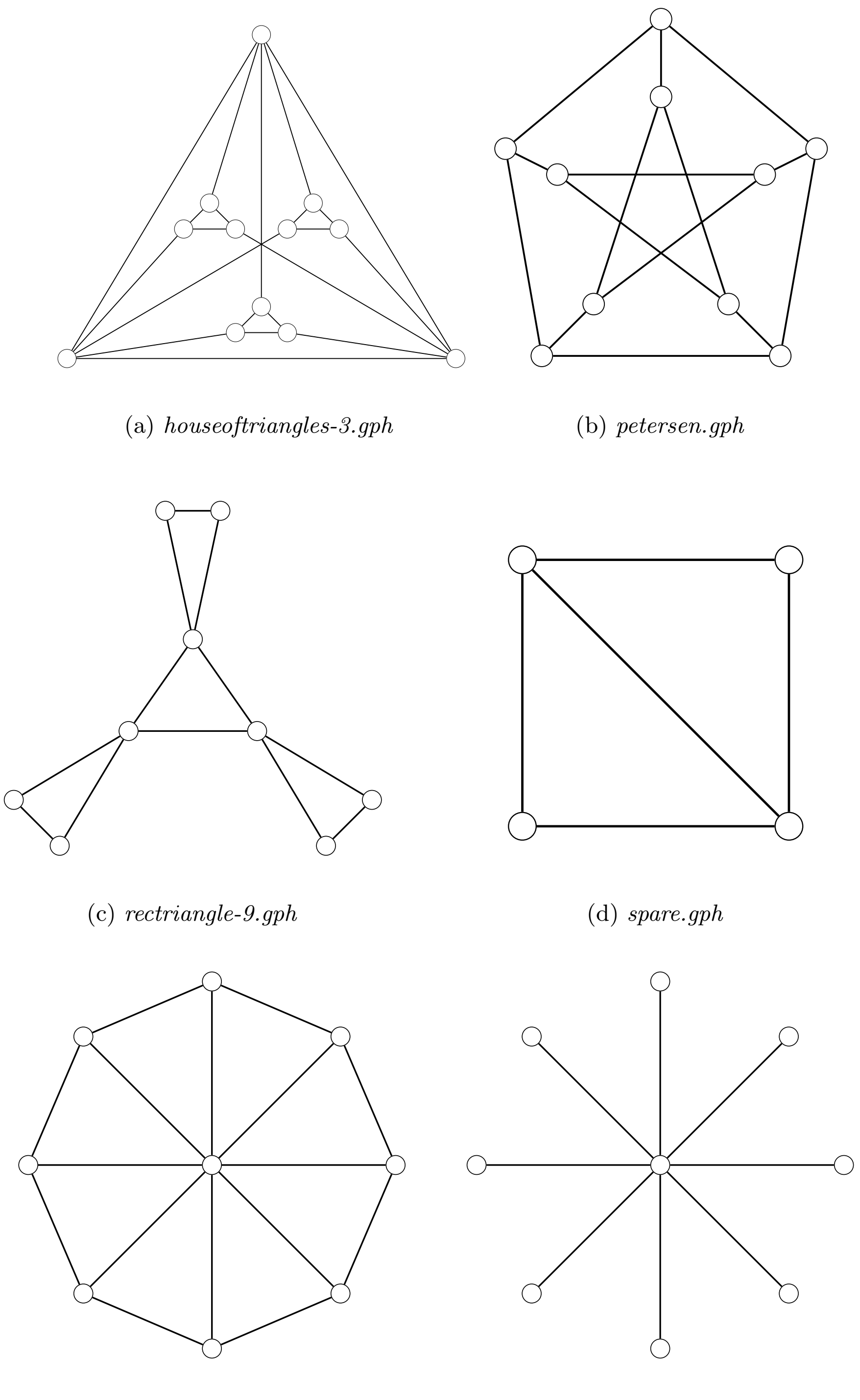
where $\alpha_i \in [0, 1]$ is, again, a parameter used to control the convergence of the system. Moreover, inc[$\cdot$] is a new function that replaces the phase shift function of Equation 5.4. This new function is defined as follows:

$$inc[x] = \begin{cases} x - 0.5 & \text{if } x \geq 0 \\ x + 0.5 & \text{if } x < 0 \end{cases} \tag{5.15}$$

The hope is to achieve a similar, or even better, convergence behavior with this much simpler function. Finally, node i sends the following message m (see function sendMessage()):

$$m =< \text{theta}_m := \theta_i, \text{relevance}_m := \frac{1}{|M_i|} > \tag{5.16}$$

Note that when $M_i = \emptyset$, then relevance_m is, of course, set to 1. As described before, to conclude a sensor event, node i deletes all messages from its queue M_i (see function clearMessageQueue()), that is, $M_i = \emptyset$.



(a) *houseoftriangles-3.gph* (b) *petersen.gph*

(c) *rectriangle-9.gph* (d) *spare.gph*

(e) *wheel-8.gph* (f) *star-8.gph*

Figure 5.5: Sensor network topologies used in this work.

5.2.2 Experiments on the desynchronization of sensor networks

We applied four versions of the presented algorithm to 12 small sensor network topologies. The four algorithm versions are defined as follows: (1) The original model, (2) the original model with the relevance term, (3) the new model (that is, using the simplified phase shift function) without the relevance term, and (4) the new model with the relevance term. Each of these four algorithm versions was applied 100 times to each of the following 12 sensor network topologies:

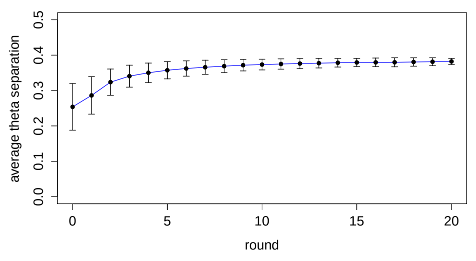
- *line-n.gph*: n connected nodes (for $n \in \{2, 3, 10\}$)
- *cycle-n.gph*: n cyclicly connected nodes (for $n \in \{3, 4, 10\}$)
- *houseoftriangles-3.gph*: see Figure 5.5(a)
- *petersen.gph*: see Figure 5.5(b)
- *rectriangle-9.gph*: see Figure 5.5(c)
- *spare.gph*: see Figure 5.5(d)
- *wheel-8.gph*: see Figure 5.5(e)
- *star-8.gph*: see Figure 5.5(f)

The last six of these topologies are shown in Figure 5.5. Note that all these topologies must be interpreted as follows. For each two nodes that are connected by a link we assume that the corresponding nodes receive the transmissions of each other. The optimal distribution of the theta values can be determined easily in each of the above described topologies. In this context, the theta values are considered to be optimally distributed if they are maximally separated from each other. For example, in the case of *line-2.gph* this value is 0.5. Another example concerns *spare.gph* where this value is $0.\overline{3}$.

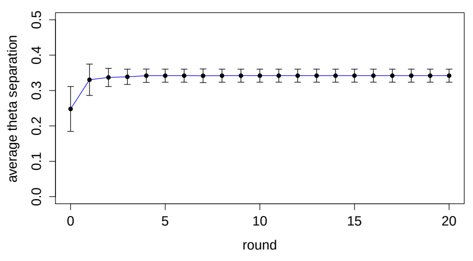
Results are shown in Tables 5.1 and 5.2. For each of the four algorithm versions we indicate, for each of the 12 sensor network topologies, the number of times (out of 100 applications) in which an optimal distribution of the theta values was reached. Hereby, Table 5.1 provides these numbers concerning an error margin of 5%, whereas the numbers given in Table 5.2 are calculated on the basis of an error margin of 15%. For each algorithm version there are four columns of numbers. Each of these columns corresponds to a different value of α.

The results can be interpreted as follows. First, the relevance term seems, unfortunately, rather not useful. Both the behavior of the original model and the one of the model with the alternative phase shift function deteriorate when using the relevance term. Second, the results of the model with the new phase shift function are—apart from only few exceptions—better than the results obtained with the original model. This applies in particular to more difficult cases such as *spare.gph*, *wheel-8.gph*, and *houseoftriangles-3.gph*. Moreover, the results indicate that a rather high setting of α, that is $\alpha \in \{0.75, 1.0\}$ seems to be best for both model versions. The results of Table 5.2, however, indicate that consistent results are rather achieved with smaller values for α, that is, $\alpha \in \{0.5, 0.75\}$. Finally, for the sake of completeness, Table 5.3 shows the average distances from the optimal distribution of the theta values.

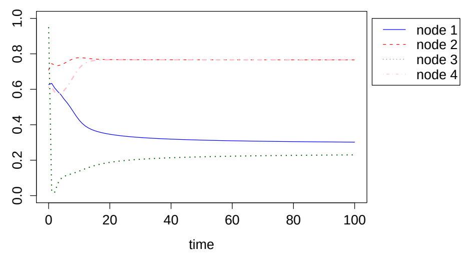
Lastly, it is also interesting to study the average number of communication rounds needed by the different algorithm versions before reaching the stopping condition. This information is given in Table 5.4. Note that the model using the new phase shift function needs significantly less communication rounds than the system using the original phase shift function. The faster convergence caused by the new phase shift function can also be observed graphically



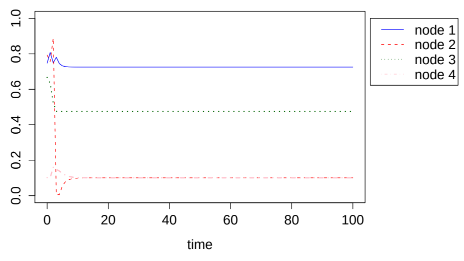
(a) Original model by Aihara et al.



(b) Model using the new phase shift function



(c) Original model by Aihara et al.



(d) Model using the new phase shift function

Figure 5.6: Experimental results concerning topology *spare.gph*. (a) and (b) show the evolution of the average distance between the theta values of connected nodes. While (a) refers to the original model, (b) concerns the model using the new phase shift function. (c) and (d) show—for a representative run of the system—the evolution of the four theta values. The graphic in (c) represents the behavior of the original model, while (d) shows the behavior of the model using the new phase shift function.

in Figure 5.6. More specifically, Figures 5.6(a) and 5.6(b) show the evolution of the average distance between the theta values of nodes connected by a link for graph *spare.gph*. Hereby, subfigure (a) presents the behavior of the original model, whereas subfigure (b) concerns the behavior of the model using the new phase shift function. Finally, Figures 5.6(c) and 5.6(d) show the evolution of the four theta values for a representative run. While subfigure (c) presents this evolution for the original model, sugfigure (d) concerns the evolution of the theta values as obtained from the model using the new phase shift function. Observe that the separation between the theta values is clearly better in subfigure (d).

On the basis of the results obtained in this section, we next devised a distributed algorithm for graph coloring which is outlined in the following section.

Table 5.1: Success rates (in terms of the number of successful applications out of 100) calculated on the basis of an error margin of 5%.

Instance	Model using the new phase shift function								Original model by Aihara et al.							
	No relevance				Relevance				No relevance				Relevance			
	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
cycle-10.gph	7	65	70	66	0	8	54	68	4	79	72	77	0	3	70	78
cycle-3.gph	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
cycle-4.gph	89	94	98	100	46	81	90	96	100	100	100	100	48	99	98	100
houseoftriangles-3.gph	18	13	9	10	22	30	15	16	0	0	0	0	27	8	1	0
line-10.gph	8	52	100	100	0	6	27	41	14	45	100	100	0	4	25	29
line-2.gph	100	100	100	100	100	100	100	100	100	100	100	100	99	100	100	100
line-3.gph	99	100	100	100	62	100	100	98	100	99	100	98	41	99	100	97
petersen.gph	8	2	1	0	1	1	5	6	0	0	0	0	2	0	0	0
rectriangle-9.gph	100	100	100	100	76	100	100	100	100	100	100	100	69	95	100	99
spare.gph	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
star-8.gph	99	100	99	100	0	0	0	93	98	97	99	100	0	0	0	84
wheel-8.gph	60	70	66	67	32	23	15	26	16	12	11	7	29	12	10	10
averages	57.333	66.333	**70.250**	**70.250**	36.583	45.750	50.500	62.000	52.667	61.000	**65.167**	**65.167**	34.583	43.333	50.333	58.083

Table 5.2: Success rates (in terms of the number of successful applications out of 100) calculated on the basis of an error margin of 15%.

Instance	Model using the new phase shift function								Original model by Aihara et al.							
	No relevance				Relevance				No relevance				Relevance			
	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
cycle-10.gph	55	65	70	66	20	58	54	68	67	79	72	77	29	67	70	78
cycle-3.gph	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
cycle-4.gph	89	94	98	100	65	81	90	96	100	100	100	100	89	99	98	100
houseoftriangles-3.gph	50	37	31	24	68	68	54	56	0	0	0	0	73	32	11	1
line-10.gph	75	100	100	100	20	52	91	100	76	100	100	100	35	66	97	100
line-2.gph	100	100	100	100	100	100	100	100	100	100	100	100	99	100	100	100
line-3.gph	99	100	100	100	99	100	100	98	100	100	100	98	99	100	100	99
petersen.gph	10	2	1	0	22	10	8	6	0	0	0	0	14	0	0	0
rectriangle-9.gph	100	100	100	100	100	100	100	100	100	100	100	100	93	100	100	100
spare.gph	80	88	90	93	23	2	0	0	0	0	0	0	0	0	0	0
star-8.gph	99	100	100	100	0	100	99	94	98	98	99	100	0	79	97	98
wheel-8.gph	95	98	98	100	80	87	82	87	93	99	100	100	85	49	28	19
averages	79.333	82.000	**82.333**	81.917	58.083	71.500	73.167	75.417	69.500	**73.000**	72.583	72.917	59.667	66.000	66.750	66.250

Table 5.3: Average distance to the optimal distribution of the theta values

Instance	Model using the new phase shift function								Original model by Aihara et al.							
	No relevance				Relevance				No relevance				Relevance			
	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
cycle-10.gph	0.055	0.039	0.032	0.035	0.073	0.052	0.051	0.036	0.045	0.026	0.030	0.024	0.063	0.045	0.037	0.027
cycle-3.gph	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
cycle-4.gph	0.032	0.016	0.005	0.000	0.095	0.051	0.027	0.011	0.005	0.001	0.000	0.000	0.039	0.007	0.007	0.001
houseoftriangles-3.gph	0.028	0.032	0.036	0.042	0.022	0.021	0.025	0.024	0.064	0.070	0.070	0.071	0.019	0.032	0.044	0.050
line-10.gph	0.028	0.012	0.006	0.004	0.062	0.035	0.022	0.014	0.028	0.013	0.007	0.004	0.048	0.032	0.021	0.015
line-2.gph	0.002	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.007	0.000	0.000	0.000
line-3.gph	0.006	0.001	0.000	0.000	0.013	0.004	0.002	0.002	0.004	0.002	0.000	0.005	0.015	0.005	0.002	0.002
petersen.gph	0.039	0.044	0.045	0.046	0.035	0.039	0.040	0.041	0.047	0.050	0.050	0.050	0.034	0.046	0.047	0.048
rectriangle-9.gph	0.000	0.000	0.000	0.000	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.001	0.000	0.000
spare.gph	0.020	0.019	0.018	0.018	0.029	0.029	0.028	0.027	0.036	0.042	0.045	0.048	0.043	0.053	0.057	0.060
star-8.gph	0.004	0.001	0.001	0.000	0.061	0.025	0.016	0.019	0.006	0.003	0.001	0.000	0.093	0.036	0.019	0.014
wheel-8.gph	0.010	0.009	0.009	0.009	0.016	0.016	0.017	0.016	0.019	0.020	0.020	0.020	0.016	0.022	0.027	0.030
averages	0.019	0.014	**0.013**	0.013	0.034	0.023	0.019	0.016	0.021	0.019	0.019	**0.019**	0.032	0.023	0.022	0.021

Table 5.4: Average number of communication rounds executed before reaching the stopping condition.

Instance	Model using the new phase shift function								Original model by Aihara et al.							
	No relevance				Relevance				No relevance				Relevance			
	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
cycle-10.gph	37.230	24.170	15.910	11.470	46.990	35.940	29.490	22.990	45.580	28.260	19.950	15.050	52.640	45.090	34.550	28.940
cycle-3.gph	10.740	6.300	4.680	4.850	17.490	10.860	8.100	6.340	23.020	14.170	10.040	8.670	31.300	20.940	16.460	14.470
cycle-4.gph	13.740	8.060	5.350	2.260	20.190	13.750	10.120	8.000	21.030	11.770	7.740	5.650	31.050	19.420	13.830	11.140
houseoftriangles-3.gph	22.500	13.830	10.290	7.880	34.540	30.340	27.450	23.130	55.210	36.520	23.370	19.060	46.950	51.070	51.380	49.060
line-10.gph	49.460	38.040	30.750	24.330	51.500	53.260	49.550	37.540	51.000	44.050	33.190	27.700	49.580	51.190	47.260	44.360
line-2.gph	8.750	5.350	3.740	2.000	8.520	5.170	3.610	2.000	10.200	6.030	4.400	3.330	9.980	6.050	4.450	3.330
line-3.gph	12.840	7.600	5.130	2.490	19.960	13.160	9.360	7.190	16.320	9.600	6.920	4.740	24.810	14.550	11.490	8.850
petersen.gph	18.430	10.910	7.860	6.690	34.230	23.690	18.550	15.040	33.290	24.430	19.820	19.480	48.520	38.320	33.030	29.530
rectriangle-9.gph	30.180	18.440	14.280	9.910	32.340	35.350	36.010	32.930	41.210	37.310	27.560	25.270	32.190	25.950	27.900	29.500
spare.gph	12.150	7.540	5.480	5.090	23.410	15.290	11.840	10.090	26.780	19.120	16.910	14.640	43.780	33.390	26.980	23.230
star-8.gph	16.240	9.060	5.920	3.090	55.230	41.250	32.030	25.340	22.360	12.990	9.170	7.110	63.430	57.220	43.260	36.430
wheel-8.gph	18.040	10.730	8.110	6.920	37.740	30.200	26.200	21.330	51.910	37.710	29.130	20.250	48.780	50.020	43.990	39.850
averages	20.858	13.336	9.792	**7.248**	31.845	25.688	21.859	17.660	33.159	23.497	17.350	**14.246**	40.251	34.434	29.548	26.558

5.3 FrogSim: An Algorithm for Distributed Graph Coloring

Although the FrogSim algorithm will be described in the context of its application in static sensor networks, it can be applied with very few modifications to any other communication network. The operation of the algorithm requires an a priori organization of the sensor network in form of a rooted tree. The root node of this tree (henceforth also called the *master node*) will have some additional functionality in comparison to the rest of the sensor nodes. Once the tree is available, the master node runs a protocol to calculate the height of the tree, that is, the distance in hops (communication rounds) from the master node to the farthest node in the network. In order to produce a tree with a low height, the distributed method for generating spanning trees with minimum diameter as presented in [28] may be used. Next, the master node triggers the start of the main algorithm (see below) by means of a broadcast message. In this message, the height of the tree is communicated to the rest of the sensor nodes as well. This overlay tree will serve two purposes: first, it will be used to communicate and store the best coloring found by the algorithm. Second, it will be used to communicate the state of the algorithm's convergence to the master node.

For the main algorithm, which works iteratively using communication rounds, we assume the sensor nodes to be time-synchronized. A communication round corresponds to the calling period (2π) as known from the models presented in the previous section. The only difference is that the length of a communication round is considered to be one time unit. Therefore, the numerical length of a communication round is denoted by 1, instead of 2π. Each sensor node executes exactly one sensor event in each communication round. Assuming that the current communication round has started at time t, the moment in time when a sensor node $i \in V$ executes its sensor event is determined by $t + \theta_i \in [0, 1)$. Note that θ_i corresponds to the phase of an oscillator in the models presented in the previous sections. Apart from θ_i, a sensor node i also stores its current color, denoted by $c_i \in \mathbb{N}$. For simplicity and without loss of generality, we assume that each color is uniquely identified by a natural number. Therefore, in the following we will use natural numbers greater than zero to refer to colors. Moreover, a sensor event includes the sending of exactly one message. Therefore, each sensor node i maintains a message queue M_i for sensor event messages received from other sensor nodes since the last execution of its own sensor event. The pseudo-code of a sensor event is shown in Algorithm 8. In the following we give a rough description of the algorithm. A detailed technical description of the functions of Algorithm 8 will be provided later on.

The simulation of the FrogSim algorithm is composed of two distinct phases. The first phase (called phase I; see lines 2–4 of Algorithm 8) makes use of the model for the desynchronization of frog calling as introduced by Aihara et al. [3], with only a few modifications. The main difference to other distributed graph coloring algorithms inspired by this model is as follows. The θ_i-values are used for determining the order in which the nodes are allowed to choose colors, whereas in previous algorithms these values were directly associated to specific colors. The second phase (called phase II, see lines 6–14 of Algorithm 8), which is initiated after phase I has finished, serves to improve the current coloring by means of a refinement technique, similar to distributed local search.

Phases I and II of FrogSim will be described in detail in Sections 5.3.1 and 5.3.2. Moreover, the way in which the initially computed tree structure will be used to communicate and store the best coloring found by the algorithm and to detect convergence in phase I is presented in Section 5.3.3.

Algorithm 8 Sensor event of node i

```
1: if not yet notified by the master node about the end of phase I then
2:    $\theta_i$ := recalculateTheta()
3:    $c_i$ := minimumColorNotUsed()
4:    sendColoringMessage()
5: else
6:    if first communication round of Phase II then
7:       if ($c_i = 1$) then $p_i$ := randomPositiveInteger()
8:       else $p_i := 0$ end if
9:    end if
10:   if $\exists m \in M_i \mid (\text{power}_m \geq p_i)$ then
11:      $c_i$ := minimumColorNotUsedByNeighborsWithHigherPower()
12:      $p_i$ := adoptPowerFromStrongestNode()
13:   end if
14:   sendRefinementMessage()
15: end if
16: clearMessageQueue()
```

5.3.1 Phase I of FrogSim

Sensor nodes will execute phase I of the sensor event until they are notified by the master node to change to phase II. During phase I, when executing its sensor event, a node i executes lines 2–4 of Algorithm 8. First, node i will examine its message queue M_i. If M_i contains more than one message from the same sender node, all these messages apart from the latest one are deleted. In general, a message $m \in M_i$ sent in this phase has the following format:

$$m =< \text{theta}_m, \text{color}_m >, \tag{5.17}$$

where $\text{theta}_m \in [0.1)$ contains the θ-value of the emitter, color_m is the color currently used by the emitter. Note that the sub-indices of theta_m and color_m denote that they are data fields belonging to message m.

Based on the messages in M_i, function recalculateTheta() recalculates a new value for θ_i:

$$\theta_i := \theta_i + \alpha_i \sum_{m \in M_i} inc[\text{theta}_m - \theta_i], \tag{5.18}$$

where α_i is a parameter used to control the convergence of the system which usually takes values in $[0, 1]$. In general, the lower the value of α_i the smaller the change applied to θ_i. Due to the experience gathered in previous work (see [92]), this parameter was set to 1 for all experiments. Moreover, the value of θ_i must be kept within $[0, 1)$. Therefore, if—after executing Eq. 5.18—the value of θ_i is negative, 1 is iteratively added to θ_i until $\theta_i \in [0, 1)$. Finally, $inc[\cdot]$ is the function defined in Eq. 5.15.

Next, node i decides for a possibly new color in function minimumColorNotUsed(). Formally, the possible color change by node i can be described as follows:

$$c_i := \min\{c \in \mathbb{N} \mid \nexists m \in M_i \text{ with color}_m = c\}. \tag{5.19}$$

In words, node i chooses among the colors that do not appear in any of the received messages $m \in M_i$, the one with the lowest identifier. Finally, node i sends the following message m (see function sendColoringMessage()):

$$m =< \text{theta}_m := \theta_i, \text{color}_m := c_i > . \tag{5.20}$$

To conclude a sensor event, node i deletes all messages from its queue M_i (see function clearMessageQueue()), that is, $M_i = \emptyset$.

5.3.2 Phase II of FrogSim

Phase II of FrogSim (which is initiated after phase I has been stopped) consists of the execution of lines 6–14 of Algorithm 8. As mentioned before, this phase is used for the distributed refinement of the current coloring, similar to a distributed local search. Note that in this phase the θ-values of the nodes are not changed anymore. Within the scope of phase II, each node i additionally maintains a so-called power parameter p_i. This parameter is initialized in the first communication round of phase II with a positive random integer for the nodes i with $c_i = 1$, and 0 for the rest of the nodes. The values of these power parameters are used to resolve conflicts that may arise during the color changes executed in phase II. In particular, in case two neighboring nodes—that is, two nodes that can communicate—have chosen the same color, the one with the higher power value is allowed to keep it. Our distributed local search starts—at the same communcation round—in all nodes with a power value which is higher than the one of their neighbors. In addition to (potentially) requiring neighbors to change color, nodes also pass their power value to their neighbors. This leads to the fact that, eventually, the node which initially had the highest power value imposes itself over all the other nodes of the network. The θ-values are again used to determine the moment in time at which a sensor node executes its sensor event. At the end of this section, a graphic example illustrates the working of phase II.

A message m sent by function sendRefinementMessage() (see line 14 of Algorithm 8) has the following format:

$$m =< \text{color}_m, \text{power}_m > . \tag{5.21}$$

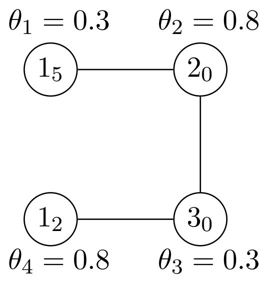
In case the current communication round is not the first communication round of phase II, node i first examines again its message queue M_i. If M_i contains more than one message from the same sender node, all these messages apart from the latest one are deleted. Then, the remaining messages are examined, and a color change only occurs if there is a message $m \in M_i$ such that $\text{color}_m = c_i$ and $\text{power}_m \geq p_i$. In words, node i only changes its color if there is an adjacent node with the same color and a higher (or equal) power value. The new color chosen by node i is the first free color that is not already in use by a node influencing node i and that has a power equal to or greater than the power value of node i. Formally, the new color c_i is chosen by the function minimumColorNotUsedByNeighborsWithHigherPower() as follows:

$$c_i := \mathsf{min}\{c \in \mathbb{N} \mid \nexists m \in M_i \text{ with } \text{color}_m = c \wedge \text{power}_m \geq p_i\}. \tag{5.22}$$

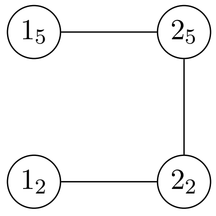
In addition, node i updates its power value using function adoptPowerFromStrongestNode() in the following way:

$$p_i := \mathsf{argmax}_{m \in M_i}\{\text{power}_m\}. \tag{5.23}$$

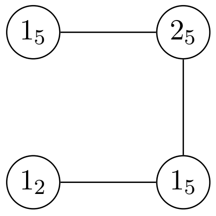
This is the highest power among the powers of the nodes that have forced node i to choose its current color. As a result, in following communication rounds node i will not be forced to change its color, because with the new power it has priority over all nodes with a lower power.



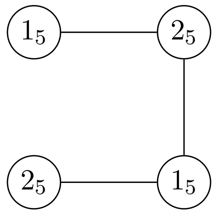
(a) Possible situation after phase I.



(b) Conflict between nodes 2 and 3.



(c) Conflict between nodes 3 and 4.



(d) After phase II all conflicts are resolved.

Figure 5.7: Example of the working of phase II of FrogSim. Nodes are labeled with their respective color. The nodes' powers are shown as sub-indices of their colors. Graphic (a) shows a possible situation after phase I. Three colors are used in the current feasible coloring. The possible θ-values, which correspond to a perfect desynchronization, are as indicated aside the nodes. Note that in phase II they will not change anymore. Initially, the nodes with color 1 receive a random power greater than 0 (in this case, 2, respectively 5), while the remaining nodes receive a power of 0. In (b), the nodes with the highest power values in their neighborhood force their neighbors to adopt their power value (a color change only happens if necessary). As a result, node 2 updates its power to 5 while node 3 updates both its color and power to 2. Next, as shown in (c), a color conflict between nodes 2 and 3 arises. The higher power of node 2 forces node 3 to change its color. This generates a new color conflict between nodes 3 and 4. Contrary to what happened in the first round, node 1 is forced to change its color and to update its power. Note that the final situation shown in (d) uses one color less than the starting configuration in (a).

Finally, node i sends a refinement message m in function sendRefinementMessage(), where m is defined as follows:

$$m =< \text{color}_m := c_i, \text{power}_m := p_i > . \tag{5.24}$$

The last action of the sensor event consists again in deleting all messages from the message queue M_i, that is, $M_i = \emptyset$. Figure 5.7 shows a small example of the kind of conflicts that phase II is supposed to resolve.

5.3.3 Use of the Overlay Tree Structure

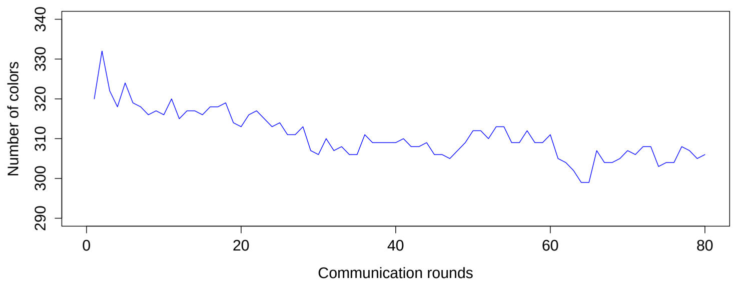
As mentioned before, the overlay tree structure which is produced before the start of the main algorithm serves two different purposes. First, for the identification of the best coloring found by the algorithm, and second, to detect the convergence of the θ-values in phase I of the algorithm.

5.3.3.1 Identifying and Storing the Best Coloring

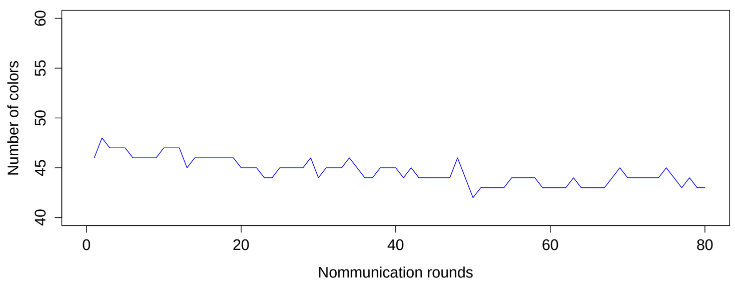
Recall that each node chooses—at each communication round—a color such that no conflicts between neighboring nodes arise. This means that FrogSim produces a valid coloring already in the first communication round. Even though the algorithm tends to improve this coloring over time, due to the self-organized desynchronization of the θ-values, the current coloring frequently becomes worse from one communication round to the next one. In other words, the color assignment of the last communication round might not correspond to the best color assignment found during the course of the algorithm. The graphic of Figure 5.8 shows the evolution of the quality of the current solution over 80 communication rounds (concerning graphs *DSJC1000.9.col* and *flat300_ 26_ 0.col* from the DIMACS challenge [71]). Even though the quality of the coloring, as expected, improves over time, it also frequently worsens.

Therefore, we found it convenient to provide the system with a mechanism—originally proposed by Zivan in [212]—to identify and store the best coloring that was found. The key characteristic of the initially produced overlay tree is its height, henceforth referred to by h. It determines the maximum number of communication rounds necessary for the root node to broadcast a message to all other nodes. The mechanism for storing the best coloring requires each node to store the color it uses in the best-found coloring in an additional variable. The main goal of the system is to efficiently alert the nodes about when and how to update this variable. The algorithm also requires each node to store the history of colors from the last $2h$ communication rounds. At each communication round, each node must communicate the maximum color used in a certain communication round by itself and its children (with respect to the tree) to its parent in the tree. Messages for this type of information passing only contain two integers: one for the maximum color used and another one to identify the corresponding communication round. In fact, these values can easily be added to the body of the coloring event messages used by FrogSim. In other words, no additional messages are required.

Note that, for example, in the first communication round only the leaves of the tree will report their colors to their parents. This is because the leaves are the only nodes without children. In the second communication round, the parents of the leaves will be able to use this information in addition to their own color information stored for the previous communication round and report the aggregated data to their respective parents. Given the height h of the tree, it takes h communication rounds until all the information regarding a specific communication round has reached the root node. Moreover, the number of colors used in a particular communication round is the maximum color identifier that reaches the root node via one of its children. In case this maximum color is lower than the number of colors used in the currently best-found coloring, the root node broadcasts a message with the corresponding communication round identifier in which this coloring was obtained. In order for this information to reach all the nodes of the network, another h communication rounds are necessary. This is why all nodes must store their colors from the last $2h$ communication rounds.



(a) Instance DSJC1000.9.col



(b) Instance flat300_26_0.col

Figure 5.8: Example of the evolution over time of the quality of the colorings generated by FrogSim. The network topologies are obtained from graphs (a) DSJC1000.9.col and (b) flat300_26_0.col of the DIMACS challenge.

5.3.3.2 Detection of Convergence in Phase I

In most distributed algorithms there is a tradeoff between the quality of the results and the consumed resources. An example of a study of this tradeoff for a similar problem in wireless radio networks can be found in [12]. The most important resource in the case of FrogSim is the number of consumed communication rounds. In principle it is possible to assign a fixed number of communication rounds to each of the two algorithm phases. In this way it would be possible to study the tradeoff between solution quality and computation time for different communication round limits. However, in the case of phase I we decided on a different option. Even though a theoretical proof does not exist, in practice the θ-values converge over time. Moreover, after having reached convergence, it does not make sense to continue with phase I, because the current coloring will not change anymore. This provides us with a natural, adaptive, stopping criterion for phase I. According to our experience, in those cases in which graph topologies are concerned that are easily colorable with local information, the θ-values converge after very few communication rounds. On the other side, in the case of graph topologies that are difficult to color in a distributed setting, more communication rounds are spent before convergence is reached. The exact criterion that was used is the following one:

$$\frac{\sum_{i\in V} \mathsf{min}(|\theta_i - \theta_i^{\mathrm{old}}|, 1 - |\theta_i - \theta_i^{\mathrm{old}}|)}{|V|} < th_{\mathrm{stop}}, \tag{5.25}$$

where V is the set of all sensor nodes, θ_i^{old} is the θ-value of node i before the update, θ_i is the θ-value of node i after the update, and th_{stop} is a threshold which we have set to 0.001 for all experiments. The convergence value as shown in Equation 5.25 is computed in the master node of the overlay tree. The remaining sensor nodes send the information that is needed for this purpose to the master node, in the same way in which the information about the current coloring is sent to the master node. This means that no additional messages are necessary. Once the master node detects convergence, it broadcasts a message to all other nodes, instructing them to start with phase II. As in the case of the identification and storing of the best coloring found, this procedure takes $2h$ communication rounds.

Concerning phase II, after tuning by hand we decided on a communication round limit of 20 for all experiments.

5.4 Experimental Results

We coded our algorithm by means of discrete event simulation, implemented from scratch in C++. For the experimental evaluation we chose a large set of different graph topologies: random geometric graphs of different densities, grid graphs of different sizes, and most of the graphs used for the DIMACS challenge [71]. All graphs that we used for the experimental evaluation can be found for download at *http://www.lsi.upc.edu/~hhernandez/graphcoloring.* Note that an edge connecting two nodes indicates that both nodes are able to communicate directly with each other via their radio antennas.

For the purpose of comparison we re-implemented one of the currently best algorithms from the literature. This algorithm was presented by Finocchi et al. in [69]. For simplicity, this algorithm will henceforth be referred to by FINOCCHI. Unfortunately, the description of this algorithm in the original article contains some ambiguities, which required us to make some decisions regarding certain aspects in the context of the re-implementation. Fortunately, our own implementation of the FINOCCHI algorithm provides generally better results than the ones reported in [69]. This can be verified by comparing the results of the original implementation with the results of our re-implementation for the graph topologies that are used both in [69] and in the present work.

In the following we present the results of three distributed algorithms: (1) FINOCCHI [69], (2) FROGSIM$^{\ominus}$, which is the FROGSIM algorithm without phase II, and (3) FROGSIM, which is the complete FROGSIM algorithm. In our opinion, the study of the results of FROGSIM$^{\ominus}$ is worthwhile, because it reflects the power of the frog-based model without any additional improvements of the refinement phase. We applied each of these three stochastic algorithms 100 times to each graph topology and report the best coloring found in all 100 runs, as well as the average quality of the best colorings found per run and the number of rounds necessary to reach these solutions. However, it is important to note that algorithms such as FINOCCHI and FROGSIM—when used in sensor networks—are generally applied continuously in a lower-level layer of the network. Therefore, the number of communication rounds necessary to reach the best solution is not overly significant. Instead our algorithm creates a valid coloring in the very first communication round and continuously tries to improve the current solution.

As mentioned above, for each of the three studied algorithms we provide—for each problem instance—the average number of communication rounds used by the respective algorithm. This is standard in the context of distributed algorithms. In contrast, we do not provide the average computation times. However, the interested reader may note that the maximum computation time needed by our algorithm for any of the considered problem instances is 2.08 seconds (on a computer equipped with an AMD64X2 4400 processor and 4 Gigabyte of memory). Moreover, for most of the tackled problem instances the computation time required by FrogSim is only a fraction of a second.

5.4.1 Resource Consumption

The time complexity of the FrogSim algorithm depends on the number of messages received at each node. Extremely large message queues can be avoided by a reasonable deployment of the network. This is usually done by topology control techniques [163, 164]. Concerning single nodes, the sensor event of each node i requires $O(|M_i|)$ time.

The FrogSim algorithm uses rather small messages of constant size. They are all composed of five data fields with a total size of $O(log\,|V|)$, which is the number of bits necessary to represent the largest integer identifier of the $|V|$ nodes in the network. Two of these data fields are needed by the main FrogSim algorithm. Two more are required for storing the best solution, and one final data field is needed to detect convergence in phase I. Nodes also store additional information such as the color used in the best coloring found so far, the node's father and its children concerning the overlay tree structure, etc. Finally, the method for transmitting aggregate data to the master through the overlay tree is the primary memory consumer within the sensor nodes. The amount of memory required by this mechanism depends on the height of the overlay tree. More precisely, nodes must remember the color chosen during the last $2h$ rounds to be able to restore the color chosen at any given round after receiving a notification of the master (see Section 5.3.3). More formally, the mechanism requires $O(h)$ space.

5.4.2 Results for Random Geometric Graphs

Random geometric graphs are popular models for sensor networks. Therefore, they are frequently used for the evaluation of algorithms developed for such networks. They are generated by randomly distributing a set of n nodes in the unit square, $[0,1]^2$. Two vertices u and v are connected by an edge, if and only if $d(u,v) \leq r$, where $d(\cdot,\cdot)$ is the Euclidean distance and $r > 0$ is a threshold.[2] More specifically, the three algorithms were applied to 40 random geometric graphs with $n \in \{20, 50, 100, 200\}$ and $r = 0.05$.

Table 5.5 presents the results obtained for this set of instances. In particular, the first column shows the instance numbers (or names) and the four columns grouped under *Properties* provide the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ of the corresponding graph, and the quality of the coloring (*cent.*) obtained for the corresponding graph with the centralized algorithm by Malaguti et al. [137]. In the cases in which the chromatic numbers are not known, a lower bound is provided (in the form $\geq$X). The following three

[2]Note that random geometric graphs are a generalization of unit disk graphs, because unit disk graphs are restricted to $r = 1$. Moreover, note that all random geometric graphs considered in this work are connected graphs. In the case of disconnected graphs, the proposed algorithm would simply color each component independently of the other ones.

groups of columns provide the results obtained by the three algorithms. For each algorithm we first give the number of colors from the best coloring found over 100 independent runs. In the second column, we show the average number of colors used by the 100 colorings obtained in 100 runs. The third column shows the average number of communication rounds necessary to reach the best solution found over the 100 rounds. In addition, for algorithms FrogSim$^{\ominus}$ and FrogSim a fourth column is added. This additional column shows the average number of communication rounds that the whole simulation lasts. In the case of the FrogSim$^{\ominus}$ algorithm, the number of consumed communication rounds is equal to the number of rounds used before reaching convergence plus the communication rounds required by the algorithm to determine convergence. In the case of FrogSim the 20 rounds used for phase II (distributed local search) are additionally counted. For ease of comparison the best performing algorithm for each instance is indicated in bold face. Hereby, the *best performing algorithm* is defined as the algorithm that finds the best coloring. Ties are broken (if possible) by the average values. The four bottom rows of the table provide a summary of the results. The first one of these rows gives averages for each column. In addition, the last three rows summarize how each algorithm has performed in comparison to the others. The first of these rows (labelled **# times best**) indicates for each algorithm the number of instances for which the corresponding algorithm was the sole winner, that is, better than the other two algorithms. The second row (labelled **# times all equal**) indicates for how many instances the results of the three algorithms were equal, whereas the last table row (labelled **# times worst**) indicates for each algorithm the number of instances for which the corresponding algorithm was the sole loser.

As expected, the results show that the smaller the size of the graph, the easier it is to find good colorings. The algorithms obtain equivalent results for 15 out of 40 instances. Although the best coloring found by Finocchi is worse than the best colorings found by the FrogSim algorithms just in one single case, it performs worse in terms of average solution quality in 25 cases. In addition, note that FrogSim$^{\ominus}$ does not obtain the worst result for any instance. FrogSim improves the results of Finocchi and FrogSim$^{\ominus}$ especially for what concerns the larger instances. It turns out to be the sole winner for 21 instances. It is interesting to note that in those cases where FrogSim is better than FrogSim$^{\ominus}$ this is due to the average solution quality. In this sense it can be said that in the context of random geometric graphs the use of phase II makes the FrogSim algorithm more robust. It is also important to note that the best colorings obtained are—for almost all instances—better than $\Delta + 1$ colors.

5.4.3 Results for DIMACS Graphs

One of the most popular sets of instances in the context of graph coloring is the one introduced for the *second DIMACS challenge* [71]. This challenge had among its objectives to establish the state-of-the-art techniques for centralized graph coloring. These graphs are generally larger and more complex than, for example, random geometric graphs. The instances originate from very different contexts, ranging from industrial problems to hand-crafted cases that were created to show the ineffectiveness of certain algorithms. This set of instances is often used as a benchmark to study the quality of new algorithms, also in the context of distributed graph coloring (see, for example, [69, 124, 139, 133]).

The results are presented in Tables 5.6 and 5.7, in the same way as in the case of random geometric graphs. Concerning the chromatic numbers, in many cases they are known. In the cases in which they are not known, we provide the lower bound obtained by [138]. Remember

Table 5.5: Results for random geometric graphs. The first column indicates the instance number. The group of columns labelled "Properties" provides for each problem instance the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ, and the quality of the coloring (column "cent.") obtained with the centralized algorithm by Malaguti et al. [137]. For each of the three algorithms FINOCCHI, FROGSIM$^{\ominus}$ and FROGSIM three measures are provided: (1) the value of the best solution found (column "colors"), (2) the average solution quality (column "avg."), and (3) the average number of communication rounds that were used (column "rounds"). In the case of FROGSIM$^{\ominus}$ and FROGSIM, the average number of communication rounds at which the algorithms converged is additionally provided (column "conv."). The best algorithm for each instance is indicated in bold face. The best algorithm is hereby defined as the one that generates the best solution. Ties are broken by the average performance.

Instance number	Properties				FINOCCHI			FROGSIM$^{\ominus}$				FROGSIM			
	$\|V\|$	Δ	χ	*cent.*	colors	avg.	rounds	colors	avg.	rounds	conv.	colors	avg.	rounds	conv.
1	20	2	3	3	**3**	3.000	3.000	**3**	3.000	5.000	8.570	**3**	3.000	5.000	32.570
2	20	2	3	3	**3**	3.000	3.000	**3**	3.000	5.000	8.490	**3**	3.000	5.000	32.490
3	20	2	2	2	2	2.250	2.250	2	2.010	5.670	7.710	**2**	2.000	5.770	31.710
4	20	3	3	3	**3**	3.000	3.000	**3**	3.000	5.000	9.010	**3**	3.000	5.000	33.010
5	20	3	3	3	**3**	3.000	3.000	**3**	3.000	7.000	13.740	**3**	3.000	7.000	39.740
6	20	1	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.300	**2**	2.000	5.000	30.300
7	20	2	3	3	**3**	3.000	3.000	**3**	3.000	5.000	8.900	**3**	3.000	5.000	32.900
8	20	2	3	3	**3**	3.000	3.000	**3**	3.000	5.000	8.570	**3**	3.000	5.000	32.570
9	20	2	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.300	**2**	2.000	5.000	30.300
10	20	1	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.000	**2**	2.000	5.000	30.000
11	50	6	≥ 4	6	**6**	6.000	6.000	**6**	6.000	5.000	10.240	**6**	6.000	5.000	34.240
12	50	3	3	3	3	3.130	3.130	**3**	3.000	7.410	13.190	**3**	3.000	7.410	39.190
13	50	4	4	4	**4**	4.000	4.000	**4**	4.000	7.000	12.150	**4**	4.000	7.000	38.150
14	50	4	3	3	3	3.430	3.430	3	3.120	5.880	11.290	**3**	3.030	6.890	35.290
15	50	4	3	3	3	3.460	3.460	3	3.090	5.690	10.180	**3**	3.040	6.280	34.180
16	50	4	≥ 3	4	**4**	4.000	4.000	**4**	4.000	5.000	10.260	**4**	4.000	5.000	34.260
17	50	6	4	4	4	4.630	4.630	4	4.280	7.950	13.870	**4**	4.020	11.990	39.870
18	50	4	3	3	3	3.367	3.367	3	3.070	5.890	9.610	**3**	3.000	6.670	33.610
19	50	3	3	3	**3**	3.000	3.000	**3**	3.000	5.000	9.010	**3**	3.000	5.000	33.010
20	50	6	5	5	5	5.140	5.140	5	5.050	5.310	14.060	**5**	5.020	5.750	38.060
21	100	8	≥ 4	5	5	5.740	5.740	5	5.470	10.010	26.390	**5**	5.280	15.270	54.390
22	100	7	4	4	4	4.740	4.740	4	4.520	10.410	29.150	**4**	4.290	17.210	57.150
23	100	7	≥ 5	6	**6**	6.000	6.000	**6**	6.000	13.000	27.950	**6**	6.000	13.000	59.950
24	100	9	≥ 4	5	5	5.730	5.730	5	5.590	26.060	56.260	**5**	5.370	38.610	100.260
25	100	7	4	4	4	4.920	4.920	4	4.730	6.290	24.090	**4**	4.330	17.080	48.090
26	100	6	≥ 5	6	**6**	6.000	6.000	**6**	6.000	35.000	62.900	**6**	6.000	35.000	116.900
27	100	6	4	4	4	4.810	4.810	4	4.530	12.150	30.660	**4**	4.100	26.510	60.660
28	100	6	4	4	4	4.640	4.640	4	4.170	10.460	22.780	**4**	4.000	14.690	50.780
29	100	7	6	6	6	6.120	6.120	**6**	6.000	21.190	44.140	**6**	6.000	21.190	84.140
30	100	8	5	5	5	5.750	5.750	5	5.500	5.870	24.420	**5**	5.380	9.250	48.420
31	200	13	8	8	8	8.750	7.790	8	8.620	35.490	63.740	**8**	8.560	39.640	115.740
32	200	12	≥ 6	8	8	8.270	7.300	**8**	8.010	21.980	55.110	**8**	8.010	21.980	95.110
33	200	12	≥ 6	7	7	8.030	7.303	7	7.690	35.950	74.300	**7**	7.520	47.890	126.300
34	200	12	≥ 7	8	8	8.590	7.750	8	8.220	32.640	64.590	**8**	8.100	40.840	114.590
35	200	17	≥ 7	8	9	9.550	8.550	8	9.120	32.840	65.270	**8**	8.940	43.630	115.270
36	200	12	≥ 7	8	8	8.210	7.330	**8**	8.000	31.610	72.990	**8**	8.000	31.610	122.990
37	200	12	6	6	6	7.250	6.670	6	7.040	33.060	71.960	**6**	6.950	40.400	121.960
38	200	11	7	7	7	7.910	7.130	7	7.810	32.590	67.050	**7**	7.700	41.400	117.050
39	200	11	7	7	7	7.810	7.320	7	7.380	29.150	66.150	**7**	7.210	41.000	112.150
40	200	13	≥ 6	8	8	8.250	7.440	8	8.020	32.160	67.890	**8**	8.010	32.920	117.890
average					4.725	5.087	4.886	4.700	4.926	14.518	30.381	4.700	4.846	17.722	63.131
# times better						0			0				21		
# times all equal						15			15				15		
# times worse						25			0				0		

that the quality of the solution provided by a centralized algorithm is also provided (in column *cent.*) and can be used as an upper bound. As a general remark before analyzing the results in depth, we would like to mention that for distributed algorithms it is very difficult, if not impossible, to capture the global structure of the DIMACS graphs in many cases. Therefore, it is not surprising that the results obtained by distributed algorithms are often far away from the chromatic numbers.

First it should be emphasized that both versions of FrogSim achieve the best results for all instances except for instance queen8-12.col (see Table 5.7), where Finocchi finds a solution with one color less than the solution found by both versions of FrogSim. However, note that the average solution quality obtained by Finocchi is nevertheless worse than the one obtained by the FrogSim algorithms. Moreover, only in two further cases, Finocchi is able to match the results of the FrogSim algorithms. On the other side, for some instances the FrogSim algorithms improve remarkably upon Finocchi. Consider, for example, instance DSJC1000.9.col (see Table 5.7) where the best colorings found by the FrogSim algorithms need 300 colors, while the best coloring found by Finocchi makes use of 314 colors. Other examples of remarkable improvements over Finocchi are the six flat$*$ instances from Table 5.7. Concerning the comparison between FrogSim$^{\ominus}$ and FrogSim, we can state that the power of the algorithm can clearly be attributed to the first (frog-inspired) phase. As in the case of random geometric graphs, phase II of FrogSim basically helps to make the algorithm more robust. It should also be emphasized that, in all cases, the FrogSim colorings require a number of colors that is smaller than $\Delta + 1$. Finally, for most of the instances of type mulsol.X, myciel.X and zeroin.X, FrogSim generates optimal colorings in each run.

5.4.4 Results for Grid Topologies

Grid topologies are frequently used in various application areas of sensor networks. In theory, the coloring of grids is very simple. They all can be painted as a chessboard, requiring only two colors. For an example see Figure 5.9.

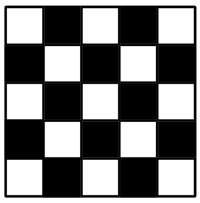


Figure 5.9: Optimal coloring of a grid topology

The way in which an optimal coloring can easily be achieved is to start the coloring process in a single node with the first color, and then proceed incrementally. The next step consists in coloring all neighbors of the starting node in the second color. All neighbors of these nodes have to be painted in the first color again, and so on. However, when considering distributed computing, nodes only have local information, where information about the position in the grid is missing. Moreover, the incremental process described above is difficult to achieve without a global control. Therefore, when coloring grids in a distributed way, the coloring process is generally initiated in several different nodes at the same time. If the coloring of these nodes does not follow the chessboard distribution of colors, eventually borders will form

Table 5.6: Results for the first set of instances from the DIMACS challenge. The first column indicates the instance name. The group of columns labelled "Properties" provides for each problem instance the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ, and the quality of the coloring (column "cent.") obtained with the centralized algorithm by Malaguti et al. [137]. For each of the three algorithms FINOCCHI, FROGSIM$^\ominus$ and FROGSIM three measures are provided: (1) the value of the best solution found (column "colors"), (2) the average solution quality (column "avg."), and (3) the average number of communication rounds that were used (column "rounds"). In the case of FROGSIM$^\ominus$ and FROGSIM, the average number of communication rounds at which the algorithms converged is additionally provided (column "conv."). The best algorithm for each instance is indicated in bold face. The best algorithm is hereby defined as the one that generates the best solution. Ties are broken by the average performance.

Instance	Properties				FINOCCHI			FROGSIM$^\ominus$				FROGSIM			
	$\|V\|$	Δ	χ	*cent.*	colors	avg.	rounds	colors	avg.	rounds	conv.	colors	avg.	rounds	conv.
DSJC1000.1.col	1000	127	≥ 5	20	30	31.790	20.040	29	29.930	20.180	40.150	**29**	29.820	23.360	64.150
DSJC1000.5.col	1000	551	≥ 13	83	125	128.550	75.220	118	120.520	45.750	69.740	**118**	120.500	46.160	93.740
DSJC1000.9.col	1000	924	≥ 51	224	314	322.590	231.520	300	308.280	42.660	56.160	**300**	308.270	43.120	80.160
DSJC125.1.col	125	23	5	5	7	8.120	6.800	7	7.770	8.020	17.330	**7**	7.640	10.140	43.330
DSJC125.5.col	125	75	17	17	24	25.600	18.790	22	24.060	12.000	25.910	**22**	23.880	15.660	49.910
DSJC125.9.col	125	120	44	44	53	56.110	45.020	51	53.880	12.630	23.570	**51**	53.670	15.640	47.570
DSJC250.1.col	250	38	≥ 6	8	12	12.990	9.990	12	12.310	10.850	23.200	**11**	12.130	14.560	49.200
DSJC250.5.col	250	147	≥ 20	28	41	42.850	29.270	37	40.290	19.680	37.630	**37**	40.130	23.460	61.630
DSJC250.9.col	250	234	≥ 71	72	93	98.060	75.560	89	93.600	21.660	34.430	**89**	93.470	23.120	58.430
DSJC500.1.col	500	68	≥ 5	12	18	19.860	13.870	18	18.650	14.870	30.960	**18**	18.460	19.960	56.960
DSJC500.5.col	500	286	≥ 16	48	70	73.520	46.750	67	69.270	30.060	52.170	**67**	69.180	32.480	76.170
DSJC500.9.col	500	471	≥ 123	126	173	178.280	132.640	165	169.730	31.110	47.740	**165**	169.680	32.030	71.740
DSJR500.1.col	500	25	12	12	13	14.550	12.550	13	14.190	34.980	50.590	**13**	14.090	40.430	102.590
DSJR500.1c.col	500	497	85	85	106	114.050	65.320	99	104.400	19.960	30.940	**99**	104.260	21.270	54.940
DSJR500.5.col	500	388	122	122	145	149.420	112.320	141	146.370	5.000	89.330	**141**	146.350	6.630	113.330
flat1000-50-0.col	1000	520	50	50	123	125.870	75.360	**116**	118.470	47.380	68.350	**116**	118.470	47.380	92.350
flat1000-60-0.col	1000	524	60	60	123	126.130	75.070	**116**	118.750	47.050	68.930	**116**	118.750	47.050	92.930
flat1000-76-0.col	1000	532	76	82	123	126.860	75.120	**117**	119.260	47.680	67.810	**117**	119.260	47.680	91.810
flat300-20-0.col	300	160	20	20	45	46.700	31.640	42	43.880	22.200	40.140	**42**	43.790	24.090	64.140
flat300-26-0.col	300	158	26	26	45	47.480	32.500	42	44.730	23.790	40.190	**42**	44.610	26.640	64.190
flat300-28-0.col	300	162	28	28	45	47.260	32.110	43	44.550	22.370	39.640	**43**	44.500	23.590	63.640
fpsol2.i.1.col	496	252	65	65	65	65.720	42.500	**65**	65.000	5.000	19.770	**65**	65.000	5.000	43.770
fpsol2.i.2.col	451	346	30	30	35	38.300	7.160	30	30.310	6.680	17.590	**30**	30.160	9.270	41.590
fpsol2.i.3.col	425	346	30	30	34	38.180	7.100	30	30.290	6.060	17.560	**30**	30.080	9.690	41.560
inithx.i.1.col	864	502	54	54	54	60.000	16.450	**54**	54.000	5.120	22.800	**54**	54.000	5.120	46.800
inithx.i.2.col	645	541	31	31	38	50.240	5.510	31	31.020	5.850	18.330	**31**	31.010	5.990	42.330
inithx.i.3.col	621	542	31	31	39	50.250	5.500	**31**	31.000	5.340	18.890	**31**	31.000	5.340	42.890
le450-15a.col	450	99	15	15	20	21.980	13.960	20	21.520	8.460	22.090	**20**	20.850	20.360	48.090
le450-15b.col	450	94	15	15	21	21.910	14.330	20	21.360	8.220	22.230	**19**	20.780	17.900	48.230
le450-15c.col	450	139	15	15	29	31.520	19.340	28	30.000	10.190	29.450	**28**	29.370	23.390	53.450
le450-15d.col	450	138	15	15	30	31.520	19.470	29	30.180	9.990	28.690	**28**	29.580	21.630	52.690
le450-25a.col	450	128	25	25	27	28.570	18.370	27	28.100	7.510	20.500	**26**	27.440	18.140	46.500
le450-25b.col	450	111	25	25	26	28.040	18.570	26	27.820	7.720	20.540	**26**	27.070	18.620	46.540
le450-25c.col	450	179	25	25	35	37.070	22.190	34	35.810	6.790	25.870	**34**	35.000	21.340	49.870
le450-25d.col	450	157	25	25	35	37.110	23.110	34	35.920	6.820	26.000	**33**	34.840	24.360	50.000
le450-5a.col	450	42	5	5	12	13.260	9.790	11	12.460	12.800	24.050	**11**	12.310	16.270	50.050
le450-5b.col	450	42	5	5	12	13.290	9.730	12	12.420	11.560	23.780	**12**	12.260	15.430	49.780
le450-5c.col	450	66	5	5	15	16.880	11.160	9	12.130	26.850	42.780	**9**	11.080	40.330	68.780
le450-5d.col	450	68	5	5	15	16.820	11.340	9	11.860	29.760	45.290	**8**	10.860	41.820	71.290
average					58.205	61.469	38.283	54.974	57.028	18.477	35.926	54.821	56.759	23.191	61.208
# times better						0			0				33		
# times all equal						0			0				0		
# times worse						39			0				0		

Table 5.7: Results for the second set of instances from the DIMACS challenge. The first column indicates the instance name. The group of columns labelled "Properties" provides for each problem instance the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ, and the quality of the coloring (column "cent.") obtained with the centralized algorithm by Malaguti et al. [137]. For each of the three algorithms FINOCCHI, FROGSIM$^{\ominus}$ and FROGSIM three measures are provided: (1) the value of the best solution found (column "colors"), (2) the average solution quality (column "avg."), and (3) the average number of communication rounds that were used (column "rounds"). In the case of FROGSIM$^{\ominus}$ and FROGSIM, the average number of communication rounds at which the algorithms converged is additionally provided (column "conv."). The best algorithm for each instance is indicated in bold face. The best algorithm is hereby defined as the one that generates the best solution. Ties are broken by the average performance.

Instance	Properties				FINOCCHI			FROGSIM$^{\ominus}$				FROGSIM			
	$\|V\|$	Δ	χ	*cent.*	colors	avg.	rounds	colors	avg.	rounds	conv.	colors	avg.	rounds	conv.
anna.col	138	71	11	11	11	11.470	6.400	11	11.120	9.350	32.820	**11**	11.010	11.720	60.820
david.col	87	82	11	11	11	12.250	6.840	11	11.570	9.620	35.910	**11**	11.180	23.080	59.910
games120.col	120	13	9	9	9	9.030	8.030	**9**	9.000	11.240	24.420	**9**	9.000	11.240	54.420
homer.col	561	99	13	?	13	14.510	7.640	13	13.820	12.470	39.050	**13**	13.210	31.830	69.050
huck.col	74	53	11	11	**11**	11.000	7.720	**11**	11.000	5.000	31.530	**11**	11.000	5.000	55.530
jean.col	80	36	10	10	10	10.450	8.360	10	10.110	7.550	27.180	**10**	10.030	9.360	53.180
miles1000.col	128	86	42	42	43	45.060	37.420	43	44.570	9.440	25.330	**42**	44.090	17.840	51.330
miles1500.col	128	106	73	73	73	74.760	64.800	73	73.970	6.280	23.520	**73**	73.640	12.450	47.520
miles250.col	128	16	8	8	8	9.440	8.440	8	8.940	24.460	46.120	**8**	8.740	34.460	88.120
miles500.col	128	38	20	20	21	22.330	19.340	20	21.770	16.340	30.360	**20**	21.390	25.770	64.360
miles750.col	128	64	31	31	32	34.050	28.630	32	33.540	11.830	27.100	**31**	33.160	20.410	57.100
mulsol.i.1.col	197	121	49	49	49	49.480	38.790	**49**	49.000	5.000	27.570	**49**	49.000	5.000	51.570
mulsol.i.2.col	188	156	31	31	31	32.450	19.190	31	31.010	6.270	22.360	**31**	31.000	6.490	46.360
mulsol.i.3.col	184	157	31	31	31	32.520	19.240	**31**	31.000	5.910	21.750	**31**	31.000	5.910	45.750
mulsol.i.4.col	185	158	31	31	31	32.610	19.290	**31**	31.000	5.900	20.530	**31**	31.000	5.900	44.530
mulsol.i.5.col	186	159	31	31	31	32.380	19.210	**31**	31.000	5.050	22.040	**31**	31.000	5.050	46.040
myciel2.col	5	2	3	3	**3**	3.000	3.000	**3**	3.000	5.000	9.480	**3**	3.000	5.000	33.480
myciel3.col	11	5	4	4	4	4.030	4.030	**4**	4.000	5.040	10.220	**4**	4.000	5.040	34.220
myciel4.col	23	11	5	5	5	5.090	4.320	**5**	5.000	5.130	12.180	**5**	5.000	5.130	36.180
myciel5.col	47	23	6	6	6	6.280	4.710	**6**	6.000	5.300	13.670	**6**	6.000	5.300	37.670
myciel6.col	95	47	7	7	7	8.010	4.480	7	7.050	5.700	17.420	**7**	7.010	6.410	41.420
myciel7.col	191	95	8	8	10	11.600	4.080	8	8.080	7.200	22.540	**8**	8.010	8.660	46.540
queen10-10.col	100	35	11	11	14	15.360	12.360	14	14.790	7.140	16.910	**14**	14.640	9.670	40.910
queen11-11.col	121	40	11	12	15	16.730	13.610	15	16.080	8.820	17.160	**15**	16.000	9.970	41.160
queen12-12.col	144	43	12	13	17	18.090	14.120	16	17.430	8.400	17.290	**16**	17.280	11.030	41.290
queen13-13.col	169	48	13	14	18	19.430	15.430	18	18.780	8.510	17.650	**18**	18.700	9.720	41.650
queen14-14.col	196	51	14	15	20	20.830	16.270	19	20.080	8.730	18.120	**19**	19.940	10.830	42.120
queen15-15.col	225	56	15	16	21	21.960	16.960	20	21.450	8.480	18.530	**20**	21.230	12.300	42.530
queen16-16.col	256	59	16	17	22	23.330	18.330	21	22.720	8.770	18.820	**21**	22.570	11.510	42.820
queen5-5.col	25	16	5	5	6	7.690	6.690	5	6.820	6.650	12.540	**5**	6.620	8.370	36.540
queen6-6.col	36	19	7	7	8	9.490	8.490	8	8.950	7.270	13.640	**8**	8.860	8.600	37.640
queen7-7.col	49	24	7	7	10	11.100	9.100	9	10.400	7.370	14.700	**9**	10.180	10.210	38.700
queen8-12.col	96	32	12	12	**13**	15.120	12.120	14	14.740	7.540	16.600	14	14.610	9.770	40.600
queen8-8.col	64	27	9	9	11	12.490	10.490	11	11.920	7.510	15.730	**11**	11.870	8.380	39.730
queen9-9.col	81	32	10	10	13	13.880	10.930	12	13.410	7.500	15.980	**12**	13.230	10.180	39.980
school1.col	385	282	14	14	41	45.200	19.570	36	38.930	23.150	44.100	**36**	38.660	28.790	70.100
school1-nsh.col	352	232	14	14	37	40.190	20.190	33	36.160	18.940	37.380	**33**	35.900	24.560	63.380
zeroin.i.1.col	211	111	49	49	49	49.980	39.850	49	49.020	5.720	23.100	**49**	49.010	5.910	47.100
zeroin.i.2.col	211	140	30	30	30	31.620	18.440	30	30.160	6.110	16.270	**30**	30.030	8.410	40.270
zeroin.i.3.col	206	140	30	30	30	31.560	18.530	30	30.130	6.410	17.690	**30**	30.030	8.170	41.690
average					20.625	21.896	15.636	20.175	20.938	8.703	22.383	20.125	20.796	11.836	47.833
# times better						1			0				29		
# times all equal						2			2				2		
# times worse						37			0				0		

where additional colors are needed in order to obtain valid colorings. An example is shown in Figure 5.10. In this context, remember that numbers correspond to colors. The process of an incremental coloring is shown starting at the top left grid and ending at the bottom right grid. The first row shows several nodes where the coloring is initiated with color 1. These wrong initial decisions lead to borders (see the gray-colored nodes in the bottom row) where additional colors are needed.

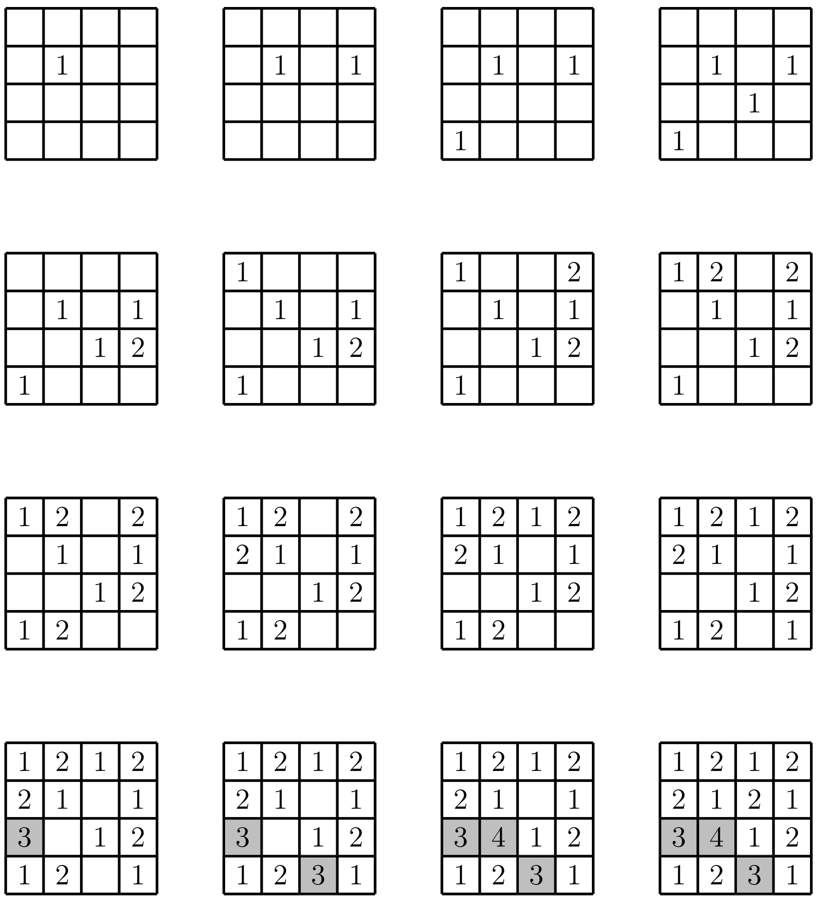


Figure 5.10: Example of the distributed incremental process of coloring a grid of 4×4 nodes. Nodes choose colors in the order as determined by their θ-values, one node at a time. For the example of this figure we have chosen a fictious node order. The process starts with the top left drawing in which the first node (from row 2 and column 2 of the grid) chooses a color. It proceeds with the second drawing from the top in which the node from row 2 and column 4 chooses a color. The process stops with the drawing at the bottom right. Due to early decisions (see the top row), the last row shows the formation of borders (see the gray-colored nodes) where additional colors are needed for achieving valid colorings.

Computational results are shown in Table 5.8. Note that in this case all chromatic numbers are known as they can be established theoretically. While small grids can basically be colored correctly by all three algorithms, both FINOCCHI and FROGSIM$^{\ominus}$ have—as expected—

Table 5.8: Results for grid (respectively, torus) topologies. The first column indicates the instance name. The group of columns labelled "Properties" provides for each problem instance the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ, and the quality of the coloring (column "cent.") obtained with the centralized algorithm by Malaguti et al. [137]. For each of the three algorithms Finocchi, FrogSim$^\ominus$ and FrogSim three measures are provided: (1) the value of the best solution found (column "colors"), (2) the average solution quality (column "avg."), and (3) the average number of communication rounds that were used (column "rounds"). In the case of FrogSim$^\ominus$ and FrogSim, the average number of communication rounds at which the algorithms converged is additionally provided (column "conv."). The best algorithm for each instance is indicated in bold face. The best algorithm is hereby defined as the one that generates the best solution. Ties are broken by the average performance.

Instance	Properties				Finocchi			FrogSim$^\ominus$				FrogSim			
	$\|V\|$	Δ	χ	*cent.*	colors	avg.	rounds	colors	avg.	rounds	conv.	colors	avg.	rounds	conv.
grid2x1	2	1	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.000	**2**	2.000	5.000	30.000
grid2x2	4	2	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.380	**2**	2.000	5.000	30.380
grid3x1	3	2	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.350	**2**	2.000	5.000	30.350
grid3x2	6	3	2	2	2	2.300	2.300	2	2.060	5.480	8.660	**2**	2.000	6.160	32.660
grid3x3	9	4	2	2	2	2.510	2.510	2	2.090	5.770	10.010	**2**	2.000	6.810	34.010
grid4x1	4	2	2	2	2	2.270	2.270	**2**	2.000	5.680	8.240	**2**	2.000	5.680	32.240
grid4x2	8	3	2	2	2	2.570	2.570	2	2.100	7.880	13.220	**2**	2.000	9.210	39.220
grid4x3	12	4	2	2	2	2.870	2.870	2	2.220	8.620	15.160	**2**	2.000	11.470	41.160
grid4x4	16	4	2	2	2	3.020	3.020	2	2.490	10.620	18.110	**2**	2.000	17.410	46.110
grid5x1	5	2	2	2	2	2.310	2.310	2	2.030	8.190	12.970	**2**	2.000	8.600	38.970
grid5x2	10	3	2	2	2	2.720	2.720	2	2.090	11.000	18.840	**2**	2.000	12.770	46.840
grid5x3	15	4	2	2	2	3.000	3.000	2	2.290	10.860	19.280	**2**	2.000	15.750	47.280
grid5x4	20	4	2	2	2	3.360	3.360	2	2.600	12.980	22.290	**2**	2.000	23.470	52.290
grid5x5	25	4	2	2	2	3.540	3.540	2	2.900	14.840	26.340	**2**	2.000	32.820	58.340
grid6x1	6	2	2	2	2	2.424	2.424	2	2.010	10.690	18.220	**2**	2.000	10.870	46.220
grid6x2	12	3	2	2	2	2.980	2.980	2	2.190	13.470	24.300	**2**	2.000	18.450	54.300
grid6x3	18	4	2	2	2	3.160	3.160	2	2.430	13.590	24.370	**2**	2.000	21.810	54.370
grid6x4	24	4	2	2	2	3.440	3.440	2	2.680	15.940	27.460	**2**	2.000	29.480	59.460
grid6x5	30	4	2	2	2	3.640	3.640	2	3.020	17.380	30.010	**2**	2.000	37.170	64.010
grid6x6	36	4	2	2	2	3.770	3.770	2	3.190	19.680	33.510	**2**	2.000	45.190	69.510
grid7x1	7	2	2	2	2	2.550	2.550	2	2.030	13.430	22.780	**2**	2.000	14.180	52.780
grid7x2	14	3	2	2	2	3.050	3.050	2	2.280	16.380	28.550	**2**	2.000	23.900	60.550
grid7x3	21	4	2	2	2	3.200	3.200	2	2.520	16.020	29.120	**2**	2.000	27.620	61.120
grid7x4	28	4	2	2	2	3.720	3.720	2	2.950	17.500	31.580	**2**	2.000	37.180	65.580
grid7x5	35	4	2	2	2	3.670	3.670	2	3.100	20.050	34.150	**2**	2.000	45.590	70.150
grid7x6	42	4	2	2	2	3.760	3.760	2	3.290	21.910	37.380	**2**	2.000	53.190	75.380
grid7x7	49	4	2	2	3	3.940	3.940	2	3.480	23.290	40.530	**2**	2.000	59.700	80.530
grid8x1	8	2	2	2	2	2.600	2.600	2	2.010	18.560	28.690	**2**	2.000	18.900	60.690
grid8x2	16	3	2	2	2	3.140	3.140	2	2.260	18.310	33.710	**2**	2.000	26.490	67.710
grid8x3	24	4	2	2	2	3.490	3.490	2	2.590	18.390	34.610	**2**	2.000	34.810	68.610
grid8x4	32	4	2	2	2	3.810	3.810	2	2.970	20.440	36.570	**2**	2.000	45.520	72.570
grid8x5	40	4	2	2	2	3.840	3.840	2	3.210	21.890	39.730	**2**	2.000	54.220	77.730
grid8x6	48	4	2	2	3	3.940	3.940	2	3.340	23.980	43.010	**2**	2.000	59.950	83.010
grid8x7	56	4	2	2	3	3.920	3.920	2	3.510	24.650	43.710	**2**	2.000	64.470	85.710
grid8x8	64	4	2	2	3	4.040	4.040	2	3.650	27.150	47.460	**2**	2.000	72.080	91.460
grid9x1	9	2	2	2	2	2.710	2.710	2	2.010	19.980	32.720	**2**	2.000	20.300	66.720
grid9x2	18	3	2	2	2	3.310	3.310	2	2.350	21.250	38.810	**2**	2.000	34.050	74.810
grid9x3	27	4	2	2	2	3.600	3.600	2	2.650	21.470	39.420	**2**	2.000	41.740	75.420
grid9x4	36	4	2	2	3	3.840	3.840	2	3.090	22.960	41.580	**2**	2.000	54.240	79.580
grid9x5	45	4	2	2	3	3.900	3.900	2	3.260	24.250	52.380	**2**	2.000	68.050	92.380
grid9x6	54	4	2	2	3	3.990	3.990	2	3.540	24.780	46.300	**2**	2.000	68.070	88.300
grid9x7	63	4	2	2	3	4.000	4.000	2	3.730	26.740	47.350	**2**	2.000	73.150	91.350
grid9x8	72	4	2	2	3	4.000	4.000	2	3.710	29.240	51.530	**2**	2.000	80.430	97.530
grid9x9	81	4	2	2	4	4.100	4.100	2	3.860	29.770	54.830	**2**	2.000	86.260	102.830
Ising32x8.col	256	4	2	2	4	4.200	4.200	4	4.010	45.790	85.720	**2**	2.000	140.340	149.720
ising32x8-torus.col	256	4	2	2	4	4.370	4.370	4	4.010	43.370	76.800	**2**	2.000	124.080	136.800
average					2.326	3.273	3.273	2.087	2.735	17.809	31.494	2.000	2.000	38.188	66.016
# times better						0			0				42		
# times all equal						3			3				3		
# times worse						43			0				0		

increasing difficulties when the grid size grows. Although this is the case, FrogSim$^{\ominus}$ has clear advantages over Finocchi. This is indicated by the average numbers given in the first of the four summarizing rows at the end of the table, and also by the fact that Finocchi is the sole loser in 43 cases, whereas FrogSim$^{\ominus}$ is never the sole loser. In contrast to the deteriorating performance of Finocchi and FrogSim$^{\ominus}$ when the grid size grows, FrogSim achieves perfect colorings in all 100 applications for all instances, which is a remarkable achievement. Even the large grids with or without periodic boundary conditions (see graphs Ising32x8.col and Ising32x8-torus.col used in [124]) do not pose any difficulty to FrogSim. In contrast, both Finocchi and FrogSim$^{\ominus}$ make use of four or more colors instead of the optimal two colors. Summarizing we can state that phase II of FrogSim is very useful when applied to grid topologies, helping the algorithm to achieve an excellent performance.

5.4.5 Results for Small Instances from [124]

Finally, we present results obtained by the three algorithms for rather small instances used by S. Lee in [124] for the evaluation of a firefly-inspired distributed graph coloring algorithm. We do not directly compare with the results presented in [124], because the algorithm proposed in [124] assumes that the number of colors required for the coloring is known a priori, that is, the algorithm must be run for a pre-determined number of colors. When graphs are large and chromatic numbers are unknown, such an algorithm is not practical. Anyway, FrogSim and the algorithm from [124] behave very similarly for most instances, with some exceptions: for *hexagon*-based instances, FrogSim is not quite able to match the average results obtained by the algorithm from [124]. Moreover, concerning *icosahedron.col*, the best solution by FrogSim uses one color more than the best one by Lee's algorithm. On the other side, concerning *4-partite-4-diff-sizes.col* and *dodecahedron.col*, FrogSim improves upon the average results obtained by Lee's algorithm.

As shown in Table 5.9, the three algorithms achieve equal results in 3 out of 14 cases. In the remaining cases both FrogSim$^{\ominus}$ and FrogSim obtain better results than Finocchi. The difference between FrogSim$^{\ominus}$ and FrogSim is again to be found in the fact that FrogSim is more robust, which is indicated by a better average solution quality.

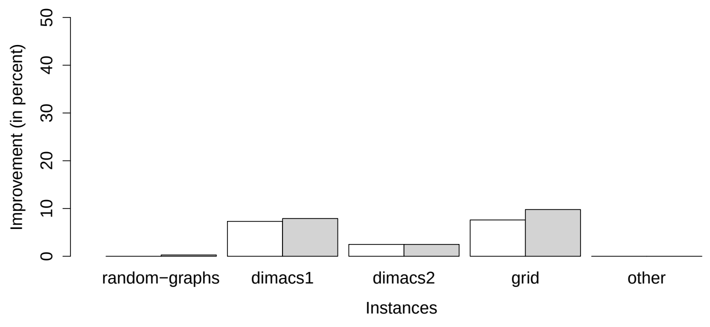
5.4.6 Summary of Results

Finally, the results from Tables 5.5–5.9 are visually presented in a summarized form in Figure 5.11. For the instances of each of the five tables (x-axis) the average improvement of FrogSim$^{\ominus}$ and FrogSim over Finocchi in terms of the best coloring (see Figure 5.11(a)) and the average solution quality (see Figure 5.11(b)) is presented. The 5 groups of instances are ordered from left to right as they appear in the text. These graphics show nicely that both versions of FrogSim gain an advantage over Finocchi in all cases. Although the best results are similar for both the FrogSim$^{\ominus}$ and FrogSim algorithms, FrogSim shows to be more consistent over multiple runs. Figure 5.11 shows that, as previously mentioned, FrogSim works especially well for grid topologies.

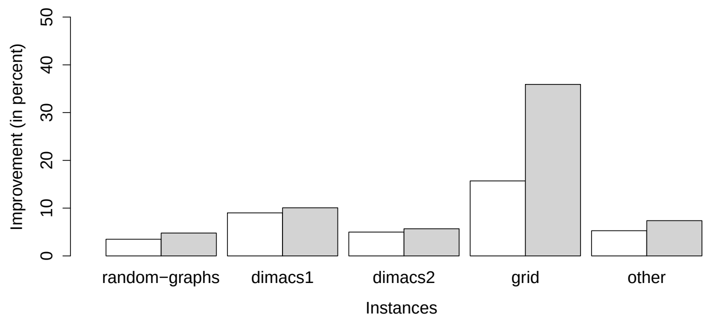
Concerning all problem instances tackled in this work, algorithm FrogSim$^{\ominus}$ needs, on average, 13.16 communication rounds to find its best solution. Algorithm Finocchi uses, on average, a comparable number of communication rounds (13.14). The average number of communication rounds that FrogSim spends on phase I before converging and switching to phase II is 19.65. Counting both phases, FrogSim requires, on average, 26.41 communication

Table 5.9: Results for the instances introduced in [124]. The first column indicates the instance name. The group of columns labelled "Properties" provides for each problem instance the number of nodes $|V|$, the maximum degree Δ, the chromatic number χ, and the quality of the coloring (column "cent.") obtained with the centralized algorithm by Malaguti et al. [137]. For each of the three algorithms Finocchi, FrogSim$^\ominus$ and FrogSim three measures are provided: (1) the value of the best solution found (column "colors"), (2) the average solution quality (column "avg."), and (3) the average number of communication rounds that were used (column "rounds"). In the case of FrogSim$^\ominus$ and FrogSim, the average number of communication rounds at which the algorithms converged is additionally provided (column "conv."). The best algorithm for each instance is indicated in bold face. The best algorithm is hereby defined as the one that generates the best solution. Ties are broken by the average performance.

Instance	Properties				Finocchi			FrogSim$^\ominus$				FrogSim			
	$\vert V\vert$	Δ	χ	*cent.*	colors	avg.	rounds	colors	avg.	rounds	conv.	colors	avg.	rounds	conv.
1hexagon-tess.col	7	6	3	3	3	3.330	3.330	3	3.070	5.430	10.040	**3**	3.050	5.640	34.040
2-partite-size6.col	12	6	2	2	**2**	2.000	2.000	**2**	2.000	5.000	6.500	**2**	2.000	5.000	30.500
2hexagon-tess.col	10	6	3	3	3	3.930	3.930	3	3.580	5.680	11.420	**3**	3.350	8.270	35.420
3-partite-3-diff-sizes.col	6	5	3	3	**3**	3.000	3.000	**3**	3.000	5.000	9.170	**3**	3.000	5.000	33.170
3-partite-size-6.col	18	12	3	3	3	3.310	2.310	**3**	3.000	5.000	11.040	**3**	3.000	5.000	35.040
3hexagon-tess.col	12	6	3	3	3	4.180	4.180	3	3.860	6.040	12.430	**3**	3.480	10.150	36.430
3partite6.col	18	12	3	3	3	3.300	2.300	**3**	3.000	5.000	11.170	**3**	3.000	5.000	35.170
4-partite-4-diff-sizes.col	10	9	4	4	**4**	4.000	4.000	**4**	4.000	5.000	10.380	**4**	4.000	5.000	34.380
4triangles	12	3	3	3	3	3.260	3.260	3	3.060	7.460	13.760	**3**	3.000	8.330	39.760
6hexagon-tess.col	19	6	3	3	3	4.640	4.640	3	4.250	8.290	16.290	**3**	3.950	12.820	42.290
7partite2.col	14	12	7	7	7	7.060	6.060	**7**	7.000	5.000	10.620	**7**	7.000	5.000	34.620
dodecahedron.col	20	3	3	3	3	3.640	3.640	3	3.460	12.000	19.580	**3**	3.430	12.660	49.580
icosahedron.col	12	5	4	4	4	4.770	4.770	4	4.480	7.880	12.990	**4**	4.400	9.010	38.990
peterson.col	10	3	3	3	3	3.150	3.150	**3**	3.020	5.340	10.620	**3**	3.020	5.340	34.620
average					3.357	3.826	3.612	3.357	3.627	6.294	11.858	3.357	3.549	7.301	36.715
# times better						0			0				7		
# times all equal						3			3				3		
# times worse						11			0				0		



(a) Improvement upon Finocchi (in percent) concerning the best solutions found



(b) Improvement upon Finocchi (in percent) concerning the average algorithm performance

Figure 5.11: Summary of results for the different groups of graph topologies. Both graphics show the performance improvement of FrogSim$^{\ominus}$ (light gray bars) and FrogSim (dark gray bars) upon Finocchi (in percent). The graphic in (a) concerns the best colorings found, whereas the graphic in (b) concerns the average algorithm performance. The five groups correspond with those in Tables 5.5–5.9.

rounds to find its best solution. It should be noted that, in the case of FrogSim these numbers do not depend so much on the size of the network, but rather on the networks' edge density.

Additionally, we use an ANCOVA model to study the statistical significance of the results. In this context, the quality of the results can be explained as a combination of the algorithm used, the number of nodes of the graph, and the degree of the graph. This will make it possible to compare Finocchi and FrogSim. The results tell us that over all instances, FrogSim uses on average 1.675 fewer colors than Finocchi. Moreover, with a confidence of 95%, we can estimate that this advantage will be in the interval $[1.331, 2.020]$. However, an important remark must be made regarding the validity of these results. First of all, ANCOVA methods generally require samples to be normally distributed. Although this assumption does probably not hold for our data set, Diehr and Lumley [54] justify that the results remain valid for large data sets. Besides, the authors mention that heterocedasticity can reduce the precision of the confidence intervals found. In our case, this heterocedasticity exists, and therefore, a small increase of the above-mentioned confidence interval of the advantages of using FrogSim over Finocchi must be assumed. Probably, performing this analysis separately for different families of graphs, homocedasticity could easily be determined and we would be able to achieve more precise intervals of confidence for each of these families of graphs. Nevertheless, the goal of showing that, with statistical significance, FrogSim is better than Finocchi has been achieved.

The same ANCOVA model can be used to check a claim which is generally assumed in the graph coloring community, namely that the amount of nodes is not decisive for determining the number of colors needed for coloring a graph. Instead, the degree of the graph (Δ) is the primary factor to determine the number of colors required. Our ANCOVA model suggests that, on average, an increase of one node in the size of a graph is translated in 0.01 colors less needed for coloring the graph. In other words, the size of the graph is almost negligible. On the contrary, for each unit that increases the degree of the graph, the amount of colors needed for coloring increases, on average, 0.24 colors. This is approximately one color for every increase of four units of the degree.

5.5 Conclusions and Future Work

Graph coloring is a classical problem of modern mathematics with more than 150 years of history. The problem has been extensively studied in theory and practice. Recently, its connection to problems that have arisen with the proliferation of wireless networks has caused a special interest in resolving the problem in a distributed manner. Due to the lack of global knowledge, in distributed algorithms the nodes have to base their color choices exclusively on information they receive from their neighbors.

The algorithm we have presented in this paper is inspired by the behavior of a family of frogs native to Japan, namely the calling behavior of Japanese tree frogs. First, we have reviewed a model for the calling behavior of Japanese tree frogs from the literature and we have used it for the desynchronization of nodes in sensor networks. Moreover, we have proposed possible extensions of this model concerning the weight that is given to different sensor nodes (relevance) and also concerning the phase shift function that is responsible for the desynchronization of the sensor nodes over time. The presented results have shown that especially the new phase shift function helps in improving the desynchronization capabilities of the system, both in quality and speed.

Second, we have presented an algorithm for graph coloring in sensor networks. The results achieved by the proposed algorithm compare very favorably with current state-of-the-art algorithms. In particular, an improved performance has been measured for about 90% of the studied instances. The benchmark set that we chose for comparison includes random geometric graphs, most of the graphs of the DIMACS challenge, and grid graphs. Apart from the favorable results, the proposed algorithm uses messages of constant size, which means that it is scalable up to very large networks. Moreover, the number of communication rounds required is comparable to that required by other algorithms that provide high quality solutions. Finally, our algorithm provides a valid coloring already in the very first communication round.

With regard to future work, we consider the use of the proposed algorithm for *time division* *multiplexing* (TDM), a mechanism for collision-free communication in wireless networks that is strongly related to graph coloring. Finally, due to its adaptive nature, our algorithm might also be interesting for mobile networks, or any dynamically changing network.

Chapter 6

Minimum Energy Broadcasting with Realistic Antenna Model

As already mentioned in the introduction of this thesis, sensor networks are wireless ad hoc networks that are being used in practical scenarios such as the monitoring of certain events [167]. Sensor nodes are generally equipped with omni-directional antennas for sending and receiving information. They have a packet-forwarding capability in order to communicate via shared and limited radio channels. In order to transmit information, a sender node must adjust its transmission power in order to reach the desired receiver node. As network lifetime is limited by batteries, energy saving is critical. A fundamental problem in sensor networks arises when one node is required to transmit data to all other nodes of the network. This scenario is known as *broadcasting*. Obviously, for broadcasting to be energy-efficient, the transmission powers of the sensor nodes should be adjusted such that the sum of the energy spent by all nodes is minimized. This problem is known as the *minimum energy broadcast* (MEB) in the literature [195].

Consider, for example, the two different patterns for a broadcast transmission of node 1 shown in Figure 6.1. Although both communication schemes reach the same nodes, the second one makes a more efficient use of energy. The MEB problem aims at obtaining energy savings by choosing appropriate communication patterns for each situation. Note that the light gray node is the source node.

To our knowledge, most—if not all—works from the literature use an antenna model where transmission powers can be adjusted to any real value between zero and the maximum transmission power level. However, available hardware such as SunSPOTs (see `http://www.sunspotworld.com/`) or iSense sensor nodes (see `http://www.coalesenses.com/`) are equipped with antennas that offer a limited set of different transmission powers; 201 in the case of SunSPOTs and seven in the case of the iSense hardware. Note that SunSPOTs are among the most widely used sensor hardware, while iSense nodes are used by two of the currently largest European projects on sensor networks, FRONTS [66] and WISEBED [67].

Related Work

The classical *minimum energy broadcast* (MEB) problem, assuming omni-directional antennas whose transmission power can be adjusted to any desired real value between zero and the maximum transmission power, has mainly been tackled with centralized heuristics. Only very

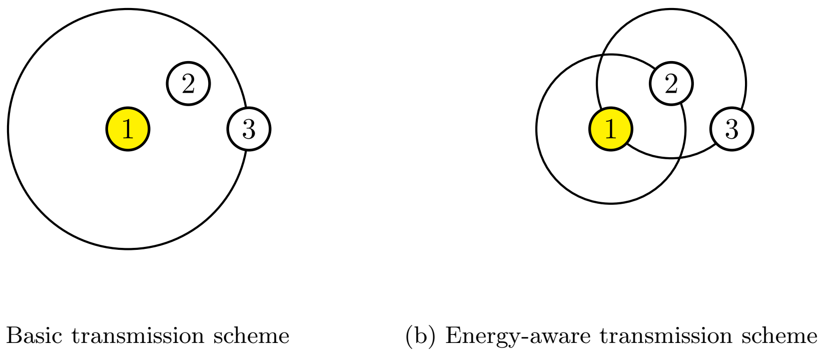

1
2
3
(a) Basic transmission scheme
1
2
3
(b) Energy-aware transmission scheme

Figure 6.1: Energy benefits by the usage of relay nodes. (a) shows a configuration one-to-all, and (b) shows an energy-aware configuration using intermediate nodes.

few distributed approaches can be found in the literature (see [196, 36]). Due to the fact that the ACO approach initially proposed in this chapter is centralized, we focus in the following on other centralized proposals.

Centralized heuristics include the ones presented in [195, 189, 130]. The most popular constructive technique is the broadcast incremental power (BIP) algorithm proposed in [195]. Moreover, local search methods including tree-based methods such as [128, 80] and power-based methods such as [47] have been developed. More recently, the MEB problem was also tackled by metaheuristics [46, 108, 6, 204]. The latest approaches are an ant colony optimization (ACO) approach proposed in [90] and the hybrid genetic algorithm presented in [174]. Even though no direct comparison is available between ACO and the hybrid genetic algorithm, a rough comparison can be made based on the results obtained for a set of benchmark instances from the literature. In particular, for benchmark instances with 50 nodes, ACO finds on average for each instance an optimal solution in 29.7 out of 30 trials. On the contrary, the hybrid genetic algorithm on average only finds in 22.9 trials an optimal solution for the same problem instances. Moreover, the results of ACO are achieved in less computational time. Therefore, the ACO algorithm from [90] can currently be regarded to be a state-of-the-art approach among the centralized techniques.

A comprehensive survey of existing work for the MEB problem and other related problems can be found in [81].

Our Contribution

This work offers three contributions. The first one concerns a re-formulation of the classical MEB problem for an antenna model that is frequently encountered, for example, in antennas of sensor nodes such as SunSPOTs and iSense nodes. While the classical MEB problem is formulated for antennas where the transmission power can be adjusted to any desired real-value between 0 and the maximum transmission power, our re-formulation considers an antenna model where transmission powers must be chosen from a finite set of discrete values, as it is the case for the antennas found in SunSPOTs and iSense sensor nodes.

The second contribution of this article concerns the adaptation of the current state-of-the-art algorithm for the classical MEB problem to the re-formulated problem. The obtained results show that the advantage of this algorithm over classical heuristics even grows when the number of possible transmission power levels decreases.

The third contribution of this work is the adaptation of the ACO algorithm for the MEBRA to work in sensor networks and other distributed scenarios. Although important changes are necessary to run the algorithm in a distributed environment, we show that using a new type of local heuristic information the algorithm obtains competitive results.

The contributions of this chapter have been previously shared with the research community. The advice given by the referees has been used to improve the algorithms introduced in this chapter. The following list contains all our publications related to novel techniques introduced in this chapter. Remember that a more extended description of the publications is available in Section 1.2:

- Hugo Hernández and Christian Blum. Ant Colony Optimization for Broadcasting in Sensor Networks under a Realistic Antenna Model. In Bogdan Filipic and Jurij Silc editors, *BIOMA 2010 – Proceedings of 4th International Conference on Bio-Inspired Optimization Methods and their Applications*. Pages 153-162. Published by Jozef Stefan Institute, Ljubljana, Slovenia, 2010. ISBN 978-961-264-017-0.

- Hugo Hernández and Christian Blum. Minimum Energy Broadcasting in Wireless Sensor Networks: An Ant Colony Optimization Approach For a Realistic Antenna Model. *Applied Soft Computing*, 11(8):5684-5694, 2011.

- Hugo Hernández and Christian Blum. Distributed Ant Colony Optimization for Minimum Energy Broadcasting in Sensor Networks with Realistic Antennas. *Computers & Mathematics with Applications*, 2012. In press.

Chapter Organization

Section 6.1 is devoted to the description of the minimum energy broadcast problem with realistic antennas. Then, Sections 6.2 and 6.3 describe the adapted BIP algorithm, respectively the adapted ant colony optimization algorithm. In Section 6.4 we provide an experimental evaluation of the presented algorithms. Finally, Section 6.5 introduces a distributed version of the ACO algorithm for the MEBRA, whereas Section 6.6 will offer conclusions and an outlook to future work.

6.1 Minimum Energy Broadcasting With Realistic Antennas

Broadcasting is a usual communication pattern in *wireless ad hoc networks* (WANETs) and, in particular, in sensor networks. However, in contrast to traditional wired networks, the role played by energy in the quality of the communication is much more important. As explained before, energy is a scarce resource in wireless ad hoc networks. In many algorithms it arises as a critical constraint. Moreover, batteries may be difficult to recharge or replace in specific scenarios. When batteries begin to deplete, the network consistency will be threatened, dramatically decreasing network performance and partitioning the network into smaller independent networks. In contrast with the slow increase of the capacity of batteries over the last years, the popularity of energy saving procedures capable of extending wireless ad hoc networks' lifetime, consistency and performance has increased notably.

The main energy consumer in WSNs and WANETs is communication. Therefore, the selection of relay nodes, radio frequency (RF) transmission power and correct beam width and beam direction are some of the major considerations for the design of broadcast and multicast routing algorithms.

For the following explanation, we consider the nodes equipped with omnidirectional antennas. However, the ideas are easily extensible to the case of directional antenna networks. When a source node starts a broadcast communication it must decide its transmission power. The transmission power will be directly related to the area covered by the signal. Receivers of the transmission must act as relay nodes in order for the message to reach all its destination nodes. Each relay node must also choose a proper communication power before transmitting. In other words, in order to establish a broadcast tree, each node must be assigned a transmission power. With the antenna model used, if u, v are nodes, p_v is the transmission power of node v and $d(v, u)$ is the Euclidean distance from v to u, the signal emitted by v reaches u if $p_v \geq d(v, u)^\alpha$. When the above inequality holds for an emitting node v and any other node u, we say that v is connected to u through a direct link. The only alternative way of establishing a communication path is by means of intermediate nodes that will act as relay nodes. In this scheme, a chain of pairs of nodes directly linked (i.e. fulfilling the above inequality) must exist in order to guarantee the existence of a communication path from v to u.

Note that in case v and u are located such that $p_v \geq d(v, u)^\alpha$ then for all nodes w with $d(v, w) \leq d(v, u)$, $p_v \geq d(v, w)^\alpha$ also holds. In words, this means that creating a direct link from v to u will also assure that v will establish a direct link to all nodes which are not farther from v than u. In wireless scenarios, communication must no longer be understood as a set of links but as an area of coverage that surrounds each node. The recipients of node v, are those located inside its area of coverage. This also means that communication trees in wireless scenarios might not be symmetric, a property which is quite common for wired links. Hence, a node v can transmit a message to nodes $u_1, u_2, \ldots, u_k$, in only one transmission using the maximum of the powers that would be required for reaching each node separately (p_{uv} is the power required for a transmission from node u to v):

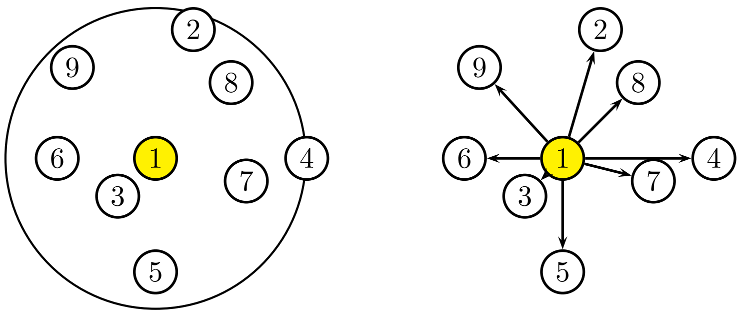
$$p_v^{wireless} = \max\{p_{vu_i} \mid 1 \leq i \leq k\} \tag{6.1}$$

In contrast, wired networks require v to make a different transmission to each node, and hence, the powers required for transmitting from node v to nodes $u_1, u_2, \ldots, u_k$ will be the sum of the transmission powers required for the connection of v to each host:

$$p_v^{wired} = \Sigma_{i=1}^{k} p_{vu_i} \tag{6.2}$$

Figure 6.2 shows the different communication schemes and infrastructure needed for two networks with the same topology, one using wireless networking (Figure 6.2(a)) and the other wired networking (Figure 6.2(b)).

The related literature generally uses the term *Wireless Multicast Advantage* [195] (or WMA) for referring to the ability of wireless networks of broadcasting to local neighborhoods with a single transmission. The WMA is inherent to the wireless network model. This means that it cannot be disabled. Using the air as a communication channel, any node in the area of influence of a transmission can receive and process the signal. Therefore, any one-to-one communication in wireless networks must be implemented logically with the use of an additional protocol or algorithm. On the contrary, the nature of wired networks is to consider one-to-one connections and to use switches to produce multicast transmissions. Although transmitting over the air is more expensive than using a solid channel, an adequate use of WMA reduces the energy consumed by wireless networks. As a result, it is quite important to find optimal transmission trees in wireless networks. In fact, the MEB problem has been classified as NP-hard [30, 130], while a similar problem in wired networks can be easily solved by creating a minimum spanning tree (MST), which is a polynomially solvable problem. The goal of MEB is no other than to take profit from the benefits of the WMA.



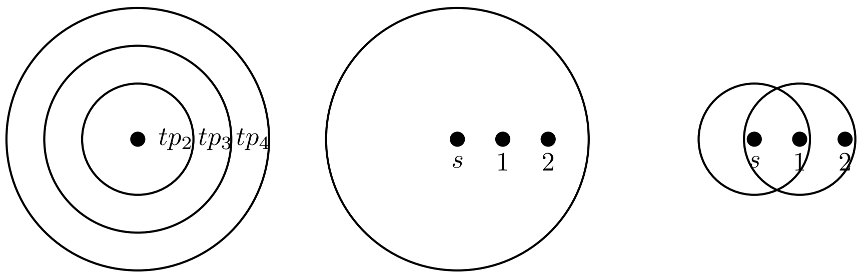
(a) Wireless Network (b) Wired network

Figure 6.2: *Wireless Multicast Advantage* of omnidirectional antennas.

Formulation of the MEBRA

Given a set of sensor nodes V, we assume that each node $i \in V$ can choose a transmission power level p_i such that $p_i \in P = \{tp_1, \ldots, tp_m\}$, where P is a finite set of m different transmission powers such that $tp_1 = 0$ and $tp_l < tp_{l+1}$ for all $l = 1, \ldots, m-1$. Moreover, we assume that signal power diminishes at a rate proportional to $r^{-\alpha}$, where r is the distance to the signal source, and α is a parameter that, depending on the environment, takes typically values between 2 and 4. In our work we choose $\alpha = 2$, as in most other works (see, for example, [195]). A sender node i is able to successfully transmit a signal to a receiver node j if $p_i \geq k \cdot d(i,j)^\alpha$, where $d(i,j)$ is the Euclidean distance between i and j, and k is the receiving node's power threshold for signal detection which is usually normalized to 1.

The minimum energy broadcast problem with realistic antennas (MEBRA), as introduced in the following, is NP-hard. This easily follows from being a generalization of the standard MEB problem as defined in the literature [30]. It can be stated as follows. V is a given set of nodes with fixed positions in a 2-dimensional area. Introducing a directed link (i,j) between all (ordered) pairs $i \neq j$ of nodes such that $d(i,j)^\alpha \leq tp_m$, where $d(i,j)$ is the Euclidean distance between i and j, induces a directed network $G = (V, E)$. Given a source node $s \in V$, one must find transmission powers for all nodes such that a broadcast from s to all other nodes is possible, and such that the sum of all transmission powers is minimal. This corresponds to



(a) Antenna with four transmission power levels (remember that $tp_1 = 0$)

(b) Solution T_1 with edge set $\{(s,1),(s,2)\}$. Translates to $p_s = tp_4$, $p_1 = 0$, and $p_2 = 0$

(c) Solution T_2 with edge set $\{(s,1),(1,2)\}$. Translates to $p_s = p_1 = tp_2$, and $p_2 = 0$

Figure 6.3: An example for the MEBRA problem and different solutions. We assume that each sensor node has an antenna as shown in (a), that is, with four transmission power levels. The first one, tp_1, is always zero. The three other transmission power levels—tp_2, tp_3, and tp_4—are shown as concentric circles of the area they cover. In this example let us assume that $2 \cdot tp_2 < tp_4$. (b) and (c) each show a solution for a problem instance with three sensor nodes: the source node s and nodes 1 and 2. Note that because $2 \cdot tp_2 < tp_4$, the solution in (c) is better than the one from (b). In fact, (c) is the optimal solution for this small example.

finding a directed spanning tree $T = (V, E_T)$ with root node s in G such that function $f(\cdot)$, whose definition is given in the following, is minimized:

$$f(T) := \sum_{i \in V} \max_{(i,j) \in E_T} p_{ij} \tag{6.3}$$

where p_{ij} is the minimum transmission power level with which node j can receive the signal sent by node i. Technically, p_{ij} can be defined as follows:

$$p_{ij} := \min \{tp_l \in P \mid l \in \{2, \ldots, m\}, tp_l \geq d(i,j)^\alpha, tp_{l-1} < d(i,j)^\alpha\} \tag{6.4}$$

Note that a solution T is easily converted to a transmission power level p_i for each node $i \in V$ as follows. For all leaf nodes of T we choose $p_i = 0$, and for all other nodes $p_i := \max_{(i,j) \in E_T} p_{ij}$. An example of the MEBRA problem is given in Figure 6.3.

6.2 BIP-Algorithm for the MEBRA Problem

As previously explained, the classical heuristic for the original MEB problem is the broadcast incremental power (BIP) algorithm proposed in [195]. In the following we outline the adaptation of this algorithm for the MEBRA problem, for two reasons. First, the way of constructing solutions that is employed by the BIP algorithm is also used by the ACO algorithm that will be outlined in Section 6.3. Second, the adapted BIP algorithm will be used as a benchmark method for comparison in Section 6.4.

The BIP algorithm starts from a partial solution $T = (V_T, E_T)$, where $V_T := \{s\}$ and $E_T := \emptyset$, which is iteratively increased until all nodes in the network are included in the

Algorithm 9 The BIP algorithm for the MEBRA problem

```
1: INPUT: a network $G = (V, E)$ and a source node $s \in V$
2: $T := (\{s\}, \emptyset)$
3: while $V_T \neq V$ do
4:     $e = (i, j) := \mathsf{argmin}_{e' \in \mathcal{N}_T}\{\eta(e')\}$
5:     $T := T \oplus (i, j)$
6: end while
7: OUTPUT: $T$
```

solution. In other words, T initially includes only the source node s. Henceforth we denote by $\overline{V_T}$ the set of sensor nodes which are not included in the current partial solution, that is, $\overline{V_T} := V \setminus V_T$. At each construction step, one edge is added to the current partial solution. The set $\mathcal{N}_T$ of potential links that can be added to T is defined as follows:

$$\mathcal{N}_T := \{(i, j) \in E \mid i \in V_T, j \in \overline{V_T}\} \ , \tag{6.5}$$

where E is the edge set of the directed network G as defined in Section 6.2. More specifically, $\mathcal{N}_T$ consists of those links whose source node is in T, whereas the goal node is not in T. The choice of a link from $\mathcal{N}_T$ is done by means of a greedy function $\eta(\cdot)$ that assigns a value to each $e \in \mathcal{N}_T$. The BIP algorithm uses the following greedy function[1]:

$$\eta(e) := f(T^\star) - f(T) \ , e \in \mathcal{N}_T. \tag{6.6}$$

Hereby, $T^\star$ is defined as follows:

$$T^\star := T \oplus \{e\} \oplus \{(u, v) \mid u \in V_T \wedge v \in \overline{V_T} \wedge f(T \oplus e \oplus \{u, v\}) = f(T \oplus e)\} \tag{6.7}$$

The operation $\oplus$ adds an edge e to a tree T. In addition it adds to T the endnode of e which is not yet part of T. Concerning Equation 6.7, $T^\star$ is obtained by extending T with edge $e = (i, j)$ and additionally with all the other edges from $\mathcal{N}_T$ (if any) that can be added to T without any further increase of the global transmission power. This concerns all links $e' = (i, k) \in \mathcal{N}_T$ with $d(i, k) \leq p_{ij}$.

In other words, the greedy function $\eta(\cdot)$ used by BIP accounts for the increase of transmission power caused by adding a link e. At each step of the BIP algorithm, the link $e \in \mathcal{N}_T$ with minimal greedy value is chosen. Finally, note that the solution construction stops when $\overline{V_T} = \emptyset$. The pseudo-code of BIP is provided in Algorithm 9.

6.3 The Ant Colony Optimization Algorithm

In the following we present the adaptation of the ACO algorithm proposed for the classical MEB problem [90] to the MEBRA problem. This algorithm is a so-called $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) in the Hyper-Cube Framework [19], which is an iterative algorithm that works roughly as follows. First, at each iteration n_a solutions to the problem are constructed in a probabilistic way based on pheromone information and heuristic information. Remember that solutions in this context are spanning trees rooted in source node s. Second, improvement procedures are applied to each of these trees. The pheromone model $\mathcal{T}$ used by our

[1]Remember that $f(\cdot)$ denotes the objective function of the MEBRA problem as defined in Equation 6.3.

Algorithm 10 ACO for the MEBRA problem

```
1: INPUT: the network $G = (V, E)$ and a source node $s \in V$
2: $T^{bs}$ := NULL, $T^{rb}$ := NULL, $cf := 0$, bs_update := FALSE
3: forall $e \in E$ do $\tau_e := 0.5$ end forall
4: while termination conditions not satisfied do
5:    for $j = 1$ to $n_a$ do
6:       $T^j$ := ConstructBroadcastTree($G$,$s$)
7:       $T^j$ := LocalSearch($T^j$)
8:    end for
9:    $T^{ib}$ := argmin$\{f(T^1), \ldots, f(T^{n_a})\}$
10:   Update($T^{ib}$,$T^{rb}$,$T^{bs}$)
11:   ApplyPheromoneValueUpdate($cf$,bs_update,$\mathcal{T}$,$T^{ib}$,$T^{rb}$,$T^{bs}$)
12:   $cf$ := ComputeConvergenceFactor($\mathcal{T}$, $T^{rb}$, $T^{bs}$)
13:   if $cf \geq 0.99$ then
14:      if bs_update = TRUE then
15:         forall $e \in E$ do $\tau_e := 0.5$ end forall
16:         $T^{rb}$ := NULL, bs_update := FALSE
17:      else
18:         bs_update := TRUE
19:      end if
20:   end if
21: end while
22: OUTPUT: $T^{bs}$
```

ACO algorithm consists of a pheromone value τ_e for each link $e \in E$. After the initialization of the variables T^{bs} (i.e., the best-so-far solution), T^{rb} (i.e., the restart-best solution), and cf (i.e., the convergence factor), all the pheromone values are set to 0.5. At each iteration, after the generation of the n_a solutions, some of them are used for updating the pheromone values. The details of the algorithmic framework shown in Algorithm 10 are explained in the following.

ConstructBroadcastTree(G,s): The construction of a solution in ACO is basically done in the same way as explained in the context of the BIP algorithm in Section 6.2. However, there are two exceptions. First, note that $\mathcal{N}_T$ may contain a lot of bad-quality links, especially at the beginning of the solution construction process. Therefore, we considered the following restriction of $\mathcal{N}_T$:

$$\mathcal{N}_T^r := \{(*, l) \in \mathcal{N}_T\} \quad , \tag{6.8}$$

where l is the *best node* in $\overline{V_T}$, which is defined as follows. A node l is called the best node in $\overline{V_T}$ if and only if exists a link $e = (k, l) \in \mathcal{N}_T$ such that $\eta(e) < \eta(e')$ for all $e' \in \mathcal{N}_T$. Hereby, $\eta(\cdot)$ is the greedy function as defined in Equation 6.6. After this reduction, set $\mathcal{N}_T^r$ may still contain quite a lot of links. Therefore, we additionally applied a so-called candidate list strategy, that is, $\mathcal{N}_T^r$ is restricted to c elements with the greatest heuristic information values. After tuning by hand we decided on a setting of $c = 8$ for all experiments.

The second aspect of the solution construction that is different to the BIP algorithm is the fact that a link from $\mathcal{N}_T^r$ is chosen in a probabilistic way, rather than deterministically. More

Algorithm 11 Variable neighborhood descent (VND)

```
1: INPUT: the network $G = (V, E)$, a source node $s \in V$, a solution $T = (V, E_T)$, a parameter $r_{\max}$
2: $r := 1$
3: while $r \leq r_{\max}$ do
4:     $T' := r$-shrink$(T)$
5:     if $f(T') < f(T)$ then $T := T'$ and $r := 1$ else $r := r + 1$
6: end while
7: OUTPUT: a (possibly) improved solution $T$
```

in detail, a link from $\mathcal{N}_T^r$ is chosen according to the following probabilities:

$$\mathbf{p}(e) := \frac{\tau_e \cdot \eta(e)^{-1}}{\sum_{e' \in \mathcal{N}_T^r} \tau_{e'} \cdot \eta(e')^{-1}} \ , e \in \mathcal{N}_T^r \tag{6.9}$$

LocalSearch(T^j): Note that solutions constructed by the ants may contain nodes whose transmission power level can be reduced without destroying the broadcast property of the solution. Therefore, we first apply the so-called SWEEP procedure (see [195]) in order to detect and fix these cases. Afterwards the variable neighborhood descent (VND) algorithm [84] outlined in Algorithm 11 is applied to further improve the given solution. VND is based on the local search procedure r-shrink, which was developed by Das et al. in [47] for the original MEB problem. The only difference between our implementation and the one by Das et al. is the re-definition of the transmission power p_{ij} necessary to reach node j from node i. While for the original MEB problem, p_{ij} is defined as $d(i,j)^\alpha$, in the context of the MEBRA problem this transmission power is defined as in Section 6.1. In Section 6.4 we justify the setting of parameter $r_{\max}$ from Algorithm 11 to $|V| - 2$ in all our experiments.

Update(T^{ib},T^{rb},T^{bs}): In this procedure T^{rb} and T^{bs} are set to T^{ib} (i.e., the iteration-best solution), if $f(T^{ib}) < f(T^{rb})$ and $f(T^{ib}) < f(T^{bs})$.

ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,T^{ib},T^{rb},T^{bs}): As it is the case for all $\mathcal{MAX}$–$\mathcal{MIN}$ Ant Systems implemented in the Hyper-Cube Framework, the proposed ACO algorithm may use three different solutions for updating the pheromone values: (i) the iteration-best solution T^{ib}, (ii) the restart-best solution T^{rb}, and (iii) the best-so-far solution T^{bs}. The influence of each of these three solutions depends on the so-called *convergence factor cf*, which provides an estimate on the state of convergence of the algorithm. The pheromone update is then performed as follows. First, an update value ξ_e for each link $e \in E$ is computed:

$$\xi_e := \kappa_{ib} \cdot \delta(T^{ib}, e) + \kappa_{rb} \cdot \delta(T^{rb}, e) + \kappa_{bs} \cdot \delta(T^{bs}, e) \ , \tag{6.10}$$

where κ_{ib} is the weight of solution T^{ib}, κ_{rb} is the weight of T^{rb}, and κ_{bs} is the weight of T^{bs}. Hereby, the three parameters must be chosen such that $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1.0$. Moreover, in Equation 6.10 the δ-function is the characteristic function of the set of links in a tree T, that is, $\delta(T, e) = 1$ if $e \in E_T$, and $\delta(T, e) = 0$ otherwise. Given the ξ_e values for all $e \in E$, the following update rule is applied to all pheromone values τ_e:

$$\tau_e := \min\left\{\max\{\tau_{\min}, \tau_e + \rho \cdot (\xi_e - \tau_e)\}, \tau_{\max}\right\} \ , \tag{6.11}$$

Table 6.1: The schedule used for values κ_{ib}, κ_{rb} and κ_{bs} depending on *cf* (the convergence factor) and the Boolean control variable bs_update.

	bs_update = FALSE			bs_update = TRUE
	$cf < 0.7$	$cf \in [0.7, 0.9)$	$cf \geq 0.9$	
κ_{ib}	2/3	1/3	0	0
κ_{rb}	1/3	2/3	1	0
κ_{bs}	0	0	0	1

where $\rho \in (0, 1]$ is the learning rate, which we have set to 0.1. The upper and lower bounds $\tau_{\max} = 0.99$ and $\tau_{\min} = 0.01$ keep the pheromone values always in the range $(\tau_{\min}, \tau_{\max})$, thus preventing the algorithm from converging to a solution. After tuning, the values for κ_{ib}, κ_{rb} and κ_{bs} are chosen as shown in Table 6.1.

ComputeConvergenceFactor($\mathcal{T}, T^{rb}, T^{bs}$): This function computes, at each iteration, the convergence factor as

$$cf := \frac{\sum_{e \in E(T^{rb})} \tau_e}{(|E(T^{rb})| - 1) \cdot \tau_{\max}} \text{ ,respectively } cf := \frac{\sum_{e \in E(T^{bs})} \tau_e}{(|E(T^{bs})| - 1) \cdot \tau_{\max}} \text{ ,} \tag{6.12}$$

if bs_update = FALSE, respectively if bs_update = TRUE. Here, $\tau_{\max}$ is the upper limit for the pheromone values. The convergence factor *cf* can therefore only assume values between 0 and 1. The closer *cf* is to 1, the higher is the probability to produce the solution T^{rb} (or T^{bs} analogously).

Table 6.2: Summary of the ACO parameters.

Parameter	Value	Description
n_a	10	Number of ants per iteration
c	8	Number of elements on the candidate list
ρ	0.1	Learning or evaporation rate
$\tau_{\min}$	0.01	Minimum pheromone saturation
$\tau_{\max}$	0.99	Maximum pheromone saturation
$r_{\max}$	$\lvert V\rvert - 2$	Highest r-shrink applied

6.4 Experimental Evaluation

The adapted BIP algorithm from Section 6.2 and the proposed ACO algorithm from Section 6.3 were implemented in ANSI C++. Moreover, GCC 3.2.2 was used for compiling the software. All experiments were executed on a PC with an AMD64X2 4400 processor and 4 GB of memory. Two sets of problem instances were used for the experimental evaluation. The first set, consisting of 30 problem instances with 50 nodes each was introduced in [6, 204] for the original MEB problem. The second set, which was introduced in [90], contains 30 problem instances with 100 nodes each.

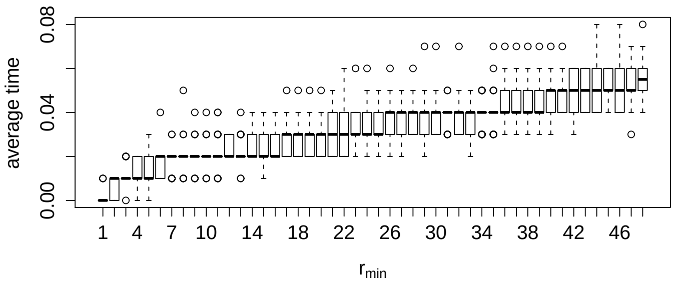
As an example, we decided to work with antennas whose maximum transmission power level reaches a distance of 500 space units. Moreover, we assume that antennas offer x transmission power levels that are equidistantly distributed in $[0, 500]$. For example, when $x = 3$ the distances reached with the three different transmission power levels are $\{0, 250, 500\}$. With the motivation of testing two antennas that are used in practice, we performed tests with $x = 7$ (corresponding to the antennas of iSense nodes), and $x = 201$ (corresponding to the antennas of SunSPOTs)[2].

Three algorithms were applied to all problem instances: (1) the adapted BIP algorithm, (2) BIP+VND, which is the BIP algorithm with the subsequent application of VND as outlined in Section 6.3, and (3) the ACO algorithm. All algorithms were applied considering antennas with seven transmission power levels, and also considering antennas with 201 transmission power levels. As BIP and BIP+VND are deterministic algorithms they were applied to each problem instance exactly once (per antenna version). On the other side, ACO was applied 30 times to each problem instance (per antenna version). Hereby, the following computation time limits were used. Concerning the instances with 50 nodes, a computation time limit of 20 seconds per run was used, whereas 100 seconds were used as a computation time limit for the larger instances with 100 nodes.

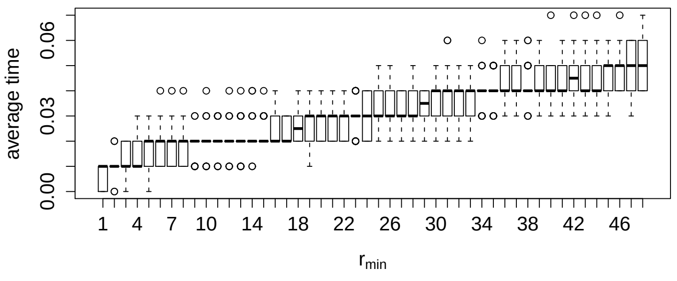
To analyze the cost of the $r_{\max}$ parameter of the VND algorithm, we have conducted some experimentation (see Figure 6.4). More specifically, we show the execution times for the BIP+VND algorithm for values of $r_{\max} \in \{1, \ldots, |V|\text{-}2\}$. Each box plot is built with the results obtained over the 30 different instances available for each network size. In Figure 6.4(a), respectively Figure 6.4(b), we show the execution times for the networks with 50 nodes and 7 transmission power levels, respectively 201 levels, whereas in Figure 6.4(c), respectively Figure 6.4(d), we show the results for networks of 100 nodes and 7 transmission power levels, respectively 201 transmission power levels. In all cases we can observe that time consumption linearly increases in relation with the $r_{\max}$ value. However, the highest consumption time observed is still relatively low (around 0.4 seconds) and affordable. Therefore, we have decided to use $r_{\max} = |V| - 2$ in all our experimentation.

Detailed numerical results are shown in Tables 6.3 to 6.6. The structure of these tables is as follows. The first column of each table provides the instance name. Afterwards, the results of the three algorithms are provided. Concerning BIP, which is a deterministic constructive heuristic, the result and the computation time are provided. Note that a computation time of 0.00 means that the algorithm was faster than 0.01 seconds. The same holds for BIP+VND. In addition, the percentage improvement (labelled deviation) over the results of BIP are provided. Note that negative percentages indicate an improvement over BIP. Finally, the results of ACO are given in four columns. More specifically, the provided information consists of the best result obtained within 30 runs (see column **best**), the average result over 30 runs (see column **average**), and the average computation time for reaching the best solution of each run (see column **time (s)**). As in the case of BIP+VND, the percentage improvement of ACO over BIP is also shown (see column **deviation**). Note that the calculation of this deviation was based on the average results of ACO, rather than on the best result. Finally, the last row of each table provides averages of all measures over all instances considered in the

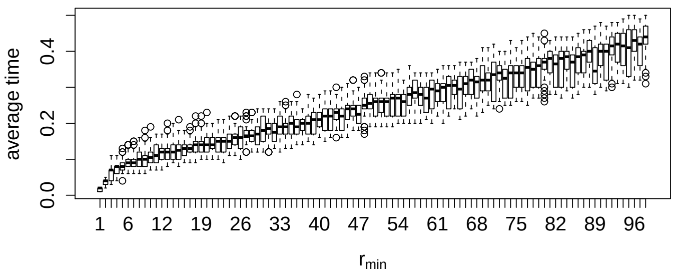
[2]Note that $x = 7$ means that the antennas of iSense nodes (`http://www.coalesenses.com/`) have six transmission power levels greater than zero. The same for SunSPOTs (`http://www.sunspotworld.com/`): $x = 201$ means that SunSPOT antennas have 200 transmission power levels greater than zero.



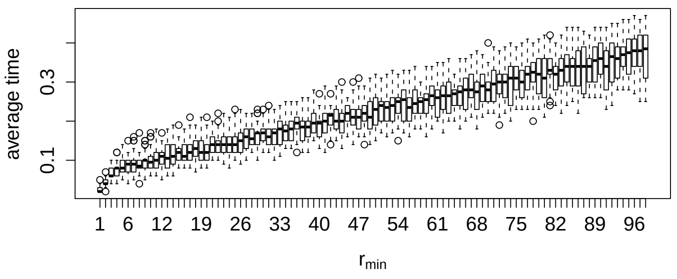
(a) $|V| = 50$, $x = 7$



(b) $|V| = 50$, $x = 7$



(c) $|V| = 50$, $x = 7$



(d) $|V| = 50$, $x = 7$

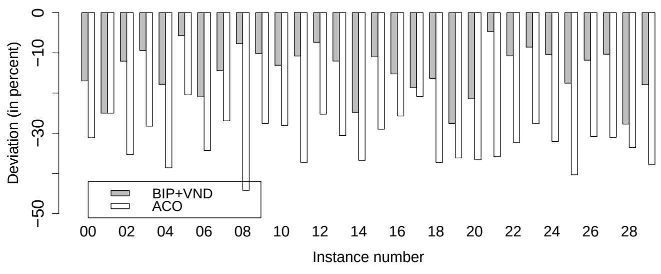
Figure 6.4: Graphical presentation of the computation times of the BIP+VND algorithm for values of $r_{\max} \in$ [1,|V|-2]

respective table. In order to improve the readability of the results, the percentage deviations of BIP+VND and ACO over BIP are presented in graphical form in Figure 6.5. Hereby, each of the four subfigures of Figure 6.5 corresponds to one of the four tables.

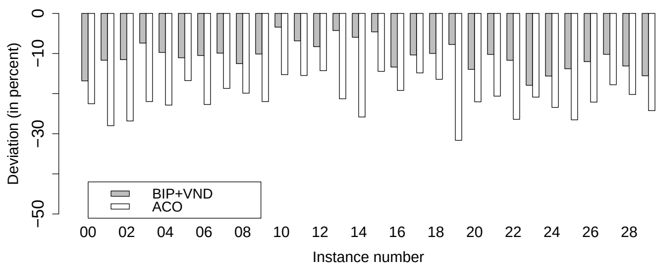
The results show, first, that both BIP+VND and ACO are—for each single instance and antenna version—better than BIP. Moreover, except for one single case where both algorithms achieve the same result, ACO always improves quite substantially upon BIP+VND. The average percentages of improvement, which are to be found in the last table rows, are summarized in Tables 6.6(a) and 6.6(b). In addition to the results of BIP+VND and ACO for the antenna versions with seven, respectively 201, transmission power levels, these tables also provide the results of BIP+VND and ACO for the original MEB problem (as taken from [90]). There are several interesting observations that we want to point out. First, the results of BIP+VND and ACO are very similar for the original MEB problem and the MEBRA problem with antennas of 201 transmission power levels. This can easily be explained by the fact that the MEBRA problem becomes more similar to the original MEB problem when the number of transmission power levels grows. Second, the advantage of BIP+VND and ACO over BIP seems to grow when the number of transmission power levels decreases. This is indicated by the average percentages of deviation over BIP as summarized in Tables 6.6(a) and 6.6(b). The advantage of BIP+VND over BIP increases from around -10.50% (concerning the original MEB problem and the MEBRA problem with antennas of 201 transmission power levels) to an advantage of around -15.00%, both for instances of 50 nodes and instances of 100 nodes. In the case of ACO, this improvement over BIP is even more pronounced. The advantage increases from around -20.00% to more than -30.00%.

From a statistical point of view, we provide the confidence intervals of the advantage (in percent) of ACO over BIP for similar instances. For the networks with 50 nodes with 7 transmission power levels, the interval is $[-34.12, -29.70]$ with a confidence level of 95%. When the number of transmission power levels is increased to 201, the confidence interval is $[-22.82, -19.43]$. For the larger instances with 100 nodes, the confidence intervals are $[-38.65, -35.08]$ and $[-20.97, 18.96]$, respectively when 7 or 201 levels of transmission powers are considered. Moreover, all differences are statistically significant. These intervals are obtained using Student's t-tests with R. The normality of the samples used is obtained from the Central Limit Theorem, which can be applied in this case thanks to the use of a large enough set of samples per instance (30).

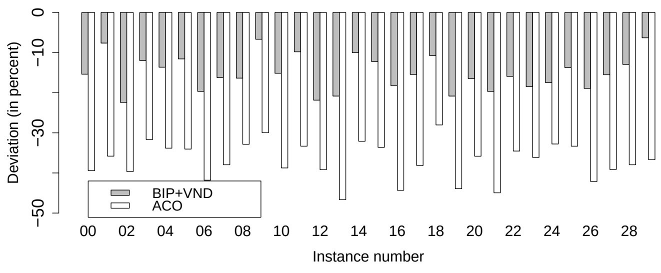
In order to study the reasons for the increasing advantage of ACO over BIP when the number of transmission power levels decreases, we exemplary provide the solutions of BIP and ACO for problem instances p50.25 (one of the problem instances with 50 nodes) in graphical form; see Figure 6.8 for seven transmission power levels, and Figure 6.9 for 201 transmission power levels. In the following we first focus on the solutions for antennas with 201 transmission power levels. In this case, the disadvantage of BIP seems to be based on the fact that many nodes use an intermediate transmission power level (neither small nor large), whereas in the solution of ACO only one node uses a large transmission power level and very few additional nodes use rather small transmission power levels. With this in mind let us have a look now at the solutions of BIP and ACO for antennas with only seven transmission power levels (see Figure 6.8). The solution of BIP seems to suffer from the same disadvantage as in the case of 201 transmission power levels. However, we can also observe an additional disadvantage. The rings painted in light-gray indicate wasted transmission power, which occurs when the furthest neighbor that is reached by the chosen transmission power level of a node is quite far



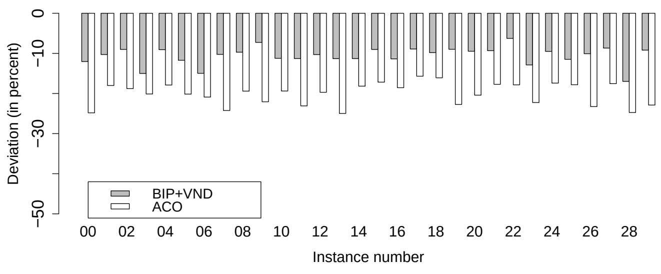
(a) 50 node instances, seven transmission power levels



(b) 50 node instances, 201 transmission power levels



(c) 100 node instances, seven transmission power levels



(d) 100 node instances, 201 transmission power levels

Figure 6.5: Graphical presentation of the results. The x-axes range over the problem instances, whereas the y-axes show the percent deviation from BIP.

Problem version	BIP + VND	ACO$^{\ominus}$	ACO
MEBRA (7 levels)	-14.61 %	-31.71 %	-31.88 %
MEBRA (201 levels)	-10.67 %	-20.57 %	-21.17 %
Original MEB	-10.61 %	-21.24 %	-21.47 %

(a) 50 node instances, advantage over BIP

Problem version	BIP + VND	ACO$^{\ominus}$	ACO
MEBRA (7 levels)	-15.07 %	-35.37 %	-36.95 %
MEBRA (201 levels)	-10.56 %	-17.55 %	-20.13 %
Original MEB	-10.31 %	-18.40 %	-19.69 %

(b) 100 node instances, advantage over BIP

Problem version	BIP + VND	ACO$^{\ominus}$	ACO
MEBRA (7 levels)	0.06 sec	4.07 sec	0.76 sec
MEBRA (201 levels)	0.05 sec	7.57 sec	0.92 sec
Original MEB	0.07 sec	8.42 sec	1.51 sec

(c) 50 node instances, computation time

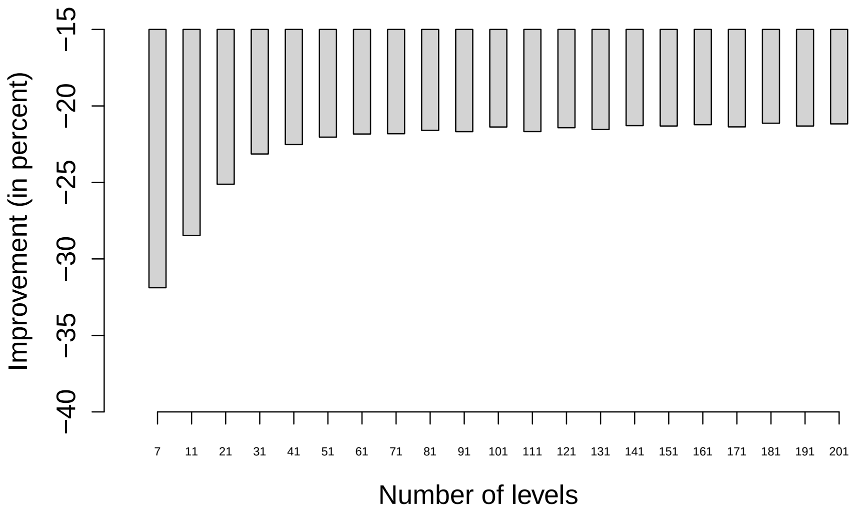
Problem version	BIP + VND	ACO$^{\ominus}$	ACO
MEBRA (7 levels)	0.45 sec	41.42 sec	26.74 sec
MEBRA (201 levels)	0.40 sec	61.02 sec	29.11 sec
Original MEB	0.47 sec	63.83 sec	28.70 sec

(d) 100 node instances, computation time

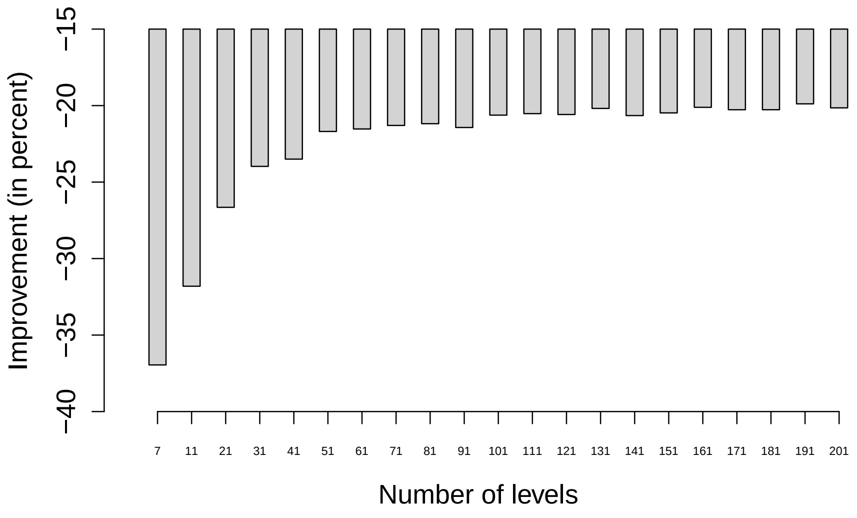
Figure 6.6: Summarized algorithm behaviour for different problem versions.

away from the transmission power range of this level. The BIP solution seems to suffer quite a lot from this phenomenon, which may explain the growing advantage of ACO over BIP when the number of transmission power levels decreases (see Figure 6.7).

Concerning computation time we observe that the type of problem (MEB versus MEBRA with different transmission power levels) does not seem to play an important role for the computation times, that is, they stay in the same order of magnitude. The average computation times of BIP+VND and ACO are given in Table 6.6(c) (for instances with 50 nodes) and Table 6.6(d) (for instances with 100 nodes). In this context note that both the algorithms for the original MEB problem from [90] and the adapted algorithms from this chapter have been run with the same computation time limits on the same processors. Therefore, the computation times can be directly compared.

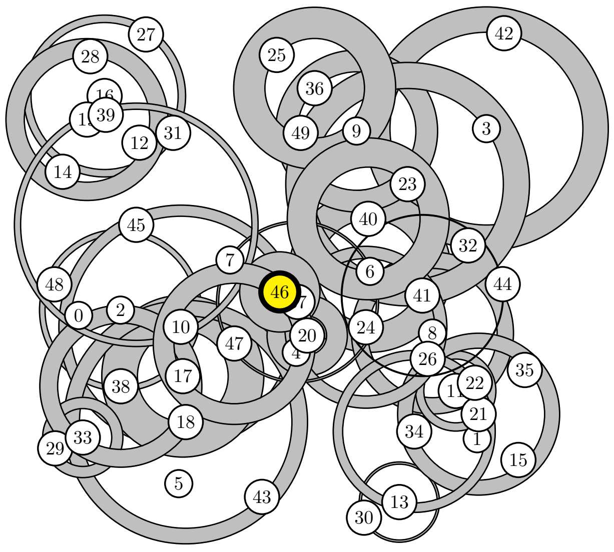


(a) 50 nodes instances

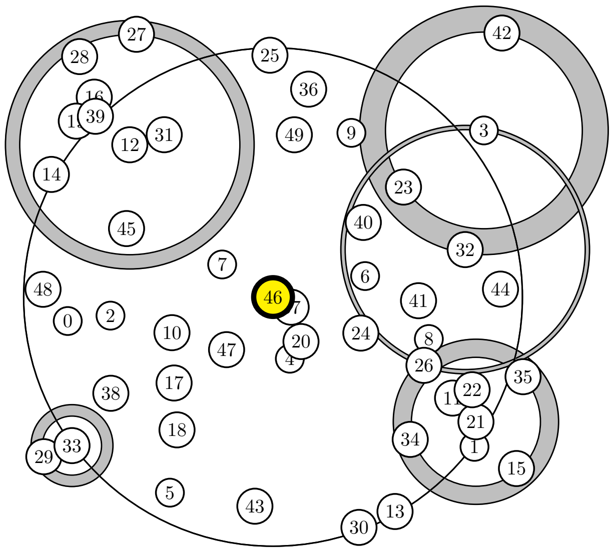


(b) 100 nodes instances

Figure 6.7: Improvement of the ACO algorithm solutions over the basic BIP algorithm when considering different number of transmission levels.

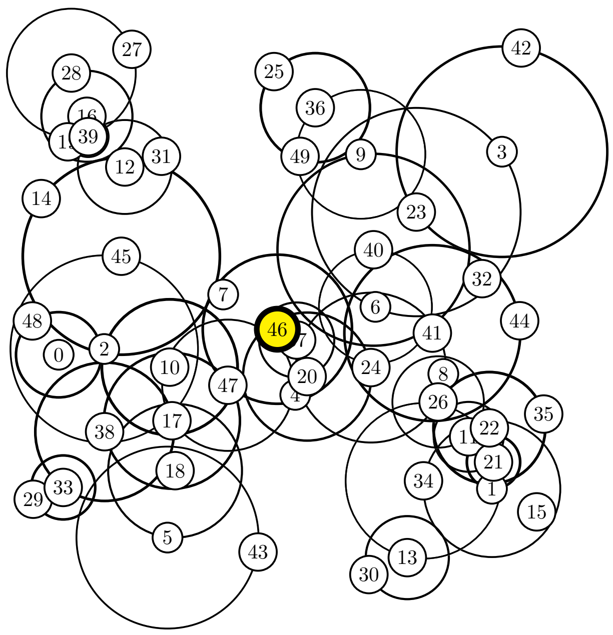


(a) Solution of BIP. Objective function value: 791666.67

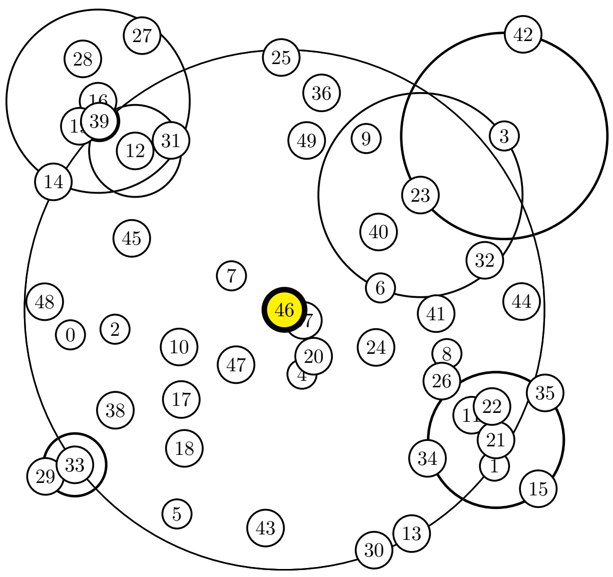


(b) Best solution of ACO. Objection function value: 472222.22

Figure 6.8: Solutions for problem instance p50.25 using seven transmission power levels. Source node is node 46. The circles show the transmission power levels of the individual sensor nodes. Gray shaded rings show the *wasted energy* that is due to transmitting with more power than necessary in order reach the furthest neighbor node.



(a) Solution of BIP. Objective function value: 533143.75



(b) Best solution of ACO. Objective function value: 391675.00

Figure 6.9: Solutions for problem instance p50.25 using 201 transmission power levels. Source node is node 46. The circles show the transmission power levels of the individual sensor nodes. Gray shaded rings (hardly visible, because they are very thin) show the *wasted energy* that is due to transmitting with more power than necessary in order reach the furthest neighbor node.

Table 6.3: Results for the 30 instances with 50 nodes when using antennas with seven transmission power levels.

Instance	**BIP**		**BIP + VND**			**ACO**			
	obj. fun	**time (s)**	**deviation**	**obj. fun**	**time (s)**	**best**	**deviation**	**average**	**time (s)**
p50.00	736111.11	0.00	-16.98%	611111.11	0.06	506944.44	-31.13%	506944.44	0.14
p50.01	611111.11	0.00	-25.00%	458333.33	0.06	458333.33	-25.00%	458333.33	0.18
p50.02	805555.56	0.00	-12.07%	708333.33	0.06	520833.33	-35.34%	520833.33	0.62
p50.03	590277.78	0.00	-9.41%	534722.22	0.05	423611.11	-28.24%	423611.11	0.68
p50.04	701388.89	0.00	-17.82%	576388.89	0.06	430555.56	-38.61%	430555.56	0.18
p50.05	611111.11	0.00	-5.68%	576388.89	0.06	486111.11	-20.45%	486111.11	0.44
p50.06	729166.67	0.00	-20.95%	576388.89	0.06	479166.67	-34.29%	479166.67	0.15
p50.07	722222.22	0.00	-14.42%	618055.56	0.05	527777.78	-26.92%	527777.78	0.44
p50.08	722222.22	0.00	-7.69%	666666.67	0.05	402777.78	-44.23%	402777.78	0.43
p50.09	680555.56	0.00	-10.20%	611111.11	0.06	493055.56	-27.55%	493055.56	1.79
p50.10	743055.56	0.00	-13.08%	645833.33	0.05	534722.22	-28.04%	534722.22	0.33
p50.11	708333.33	0.00	-10.78%	631944.44	0.06	444444.44	-37.25%	444444.44	0.08
p50.12	659722.22	0.00	-7.37%	611111.11	0.06	493055.56	-25.26%	493055.56	1.55
p50.13	750000.00	0.00	-12.04%	659722.22	0.05	520833.33	-30.56%	520833.33	0.66
p50.14	812500.00	0.00	-24.79%	611111.11	0.08	513888.89	-36.75%	513888.89	0.25
p50.15	694444.44	0.00	-11.00%	618055.56	0.06	493055.56	-29.00%	493055.56	1.99
p50.16	729166.67	0.00	-15.24%	618055.56	0.06	541666.67	-25.71%	541666.67	1.22
p50.17	631944.44	0.00	-18.68%	513888.89	0.05	500000.00	-20.88%	500000.00	0.17
p50.18	763888.89	0.00	-16.36%	638888.89	0.05	479166.67	-37.27%	479166.67	0.21
p50.19	680555.56	0.00	-27.55%	493055.56	0.06	430555.56	-36.16%	434490.74	3.11
p50.20	777777.78	0.00	-21.43%	611111.11	0.06	493055.56	-36.61%	493055.56	0.11
p50.21	736111.11	0.00	-4.72%	701388.89	0.05	472222.22	-35.85%	472222.22	0.43
p50.22	645833.33	0.00	-10.75%	576388.89	0.06	437500.00	-32.26%	437500.00	0.09
p50.23	729166.67	0.00	-8.57%	666666.67	0.05	527777.78	-27.62%	527777.78	0.60
p50.24	736111.11	0.00	-10.38%	659722.22	0.06	500000.00	-32.08%	500000.00	0.50
p50.25	791666.67	0.00	-17.54%	652777.78	0.05	472222.22	-40.35%	472222.22	0.28
p50.26	763888.89	0.00	-11.82%	673611.11	0.06	527777.78	-30.82%	528472.22	0.62
p50.27	805555.56	0.00	-10.34%	722222.22	0.06	555555.56	-31.03%	555555.56	0.82
p50.28	826388.89	0.00	-27.73%	597222.22	0.08	548611.11	-33.53%	549305.56	4.05
p50.29	736111.11	0.00	-17.92%	604166.67	0.05	458333.33	-37.74%	458333.33	0.59
	721064.82	0.00	-14.61%	614814.82	0.06	489120.37	-31.88%	489297.84	0.76

Table 6.4: Results for the 30 instances with 50 nodes when using antennas with 201 transmission power levels.

Instance	**BIP**		**BIP + VND**			**ACO**			
	obj. fun	**time (s)**	**deviation**	**obj. fun**	**time (s)**	**best**	**deviation**	**average**	**time (s)**
p50.00	523075.00	0.00	-16.82%	435093.75	0.08	405175.00	-22.52%	405272.08	0.66
p50.01	526025.00	0.00	-11.67%	464662.50	0.06	378718.75	-28.00%	378718.75	2.10
p50.02	544862.50	0.00	-11.52%	482112.50	0.05	398787.50	-26.81%	398787.50	1.01
p50.03	412675.00	0.00	-7.40%	382137.50	0.06	321968.75	-21.98%	321968.75	0.41
p50.04	429493.75	0.00	-9.73%	387712.50	0.05	331293.75	-22.86%	331293.75	1.60
p50.05	471518.75	0.00	-11.06%	419375.00	0.06	392512.50	-16.76%	392512.50	0.36
p50.06	507443.75	0.00	-10.50%	454175.00	0.05	392218.75	-22.71%	392218.75	0.30
p50.07	502312.50	0.00	-9.90%	452568.75	0.06	408406.25	-18.69%	408406.25	0.48
p50.08	433562.50	0.00	-12.53%	379250.00	0.05	347325.00	-19.89%	347325.00	0.96
p50.09	452750.00	0.00	-10.12%	406925.00	0.05	353212.50	-21.99%	353212.50	1.11
p50.10	499337.50	0.00	-3.42%	482281.25	0.06	423068.75	-15.27%	423068.75	0.49
p50.11	442775.00	0.00	-6.84%	412481.25	0.05	374225.00	-15.48%	374225.00	3.42
p50.12	465137.50	0.00	-8.30%	426518.75	0.06	398731.25	-14.28%	398731.25	0.29
p50.13	516387.50	0.00	-4.27%	494343.75	0.04	406487.50	-21.28%	406487.50	0.71
p50.14	566643.75	0.00	-5.95%	532931.25	0.04	420356.25	-25.82%	420356.25	0.39
p50.15	438243.75	0.00	-4.60%	418093.75	0.05	374937.50	-14.45%	374937.50	0.74
p50.16	520862.50	0.00	-13.37%	451212.50	0.06	420850.00	-19.20%	420850.00	0.33
p50.17	425031.25	0.00	-10.36%	381000.00	0.06	361987.50	-14.83%	361987.50	0.47
p50.18	458956.25	0.00	-9.98%	413137.50	0.05	383562.50	-16.43%	383562.50	0.29
p50.19	497393.75	0.00	-7.75%	458837.50	0.04	340006.25	-31.64%	340006.25	0.41
p50.20	538018.75	0.00	-13.95%	462962.50	0.06	419450.00	-22.04%	419450.00	0.47
p50.21	463031.25	0.00	-10.22%	415712.50	0.05	367487.50	-20.63%	367487.50	0.23
p50.22	477400.00	0.00	-11.69%	421600.00	0.05	351343.75	-26.40%	351343.75	3.49
p50.23	492225.00	0.00	-17.92%	404031.25	0.06	389487.50	-20.87%	389487.50	0.52
p50.24	534443.75	0.00	-15.62%	450975.00	0.06	409250.00	-23.43%	409250.00	0.59
p50.25	533143.75	0.00	-13.80%	459568.75	0.06	391675.00	-26.53%	391675.00	3.19
p50.26	528281.25	0.00	-12.02%	464756.25	0.05	411450.00	-22.12%	411450.00	0.80
p50.27	557493.75	0.00	-10.20%	500656.25	0.06	458293.75	-17.79%	458293.75	1.06
p50.28	529150.00	0.00	-13.11%	459762.50	0.04	422187.50	-20.21%	422187.50	0.44
p50.29	525537.50	0.00	-15.55%	443825.00	0.05	398131.25	-24.24%	398131.25	0.14
	493773.75	0.00	-10.67%	440623.33	0.05	388419.58	-21.17%	388422.82	0.92

Table 6.5: Results for the 30 instances with 100 nodes when using antennas with seven transmission power levels.

Instance	BIP		BIP + VND			ACO			
	obj. fun	time (s)	deviation	obj. fun	time (s)	best	deviation	average	time (s)
p100.00	722222.22	0.00	-15.38%	611111.11	0.44	437500.00	-39.42%	437500.00	13.32
p100.01	729166.67	0.00	-7.62%	673611.11	0.42	465277.78	-35.81%	468055.56	32.55
p100.02	805555.56	0.00	-22.41%	625000.00	0.48	486111.11	-39.66%	486111.11	21.01
p100.03	694444.44	0.00	-12.00%	611111.11	0.42	472222.22	-31.67%	474537.04	35.26
p100.04	763888.89	0.00	-13.64%	659722.22	0.44	500000.00	-33.82%	505555.56	41.86
p100.05	840277.78	0.00	-11.57%	743055.56	0.42	541666.67	-34.05%	554166.67	37.91
p100.06	847222.22	0.00	-19.67%	680555.56	0.46	493055.56	-41.80%	493055.56	23.68
p100.07	770833.33	0.00	-16.22%	645833.33	0.46	472222.22	-37.96%	478240.74	12.10
p100.08	763888.89	0.00	-16.36%	638888.89	0.52	506944.44	-32.85%	512962.96	36.60
p100.09	729166.67	0.00	-6.67%	680555.56	0.42	506944.44	-29.97%	510648.15	31.61
p100.10	687500.00	0.00	-15.15%	583333.33	0.43	409722.22	-38.75%	421064.81	35.02
p100.11	708333.33	0.00	-9.80%	638888.89	0.46	472222.22	-33.33%	472222.22	25.26
p100.12	826388.89	0.00	-21.85%	645833.33	0.44	500000.00	-39.19%	502546.30	25.60
p100.13	833333.33	0.00	-20.83%	659722.22	0.51	444444.44	-46.67%	444444.44	7.44
p100.14	694444.44	0.00	-10.00%	625000.00	0.42	465277.78	-32.10%	471527.78	30.60
p100.15	680555.56	0.00	-12.24%	597222.22	0.44	451388.89	-33.61%	451851.85	17.56
p100.16	798611.11	0.00	-18.26%	652777.78	0.46	437500.00	-44.32%	444675.93	35.96
p100.17	763888.89	0.00	-15.45%	645833.33	0.43	472222.22	-38.15%	472453.70	27.16
p100.18	645833.33	0.00	-10.75%	576388.89	0.44	458333.33	-28.03%	464814.81	34.36
p100.19	833333.33	0.00	-20.83%	659722.22	0.43	465277.78	-43.92%	467361.11	19.28
p100.20	715277.78	0.00	-16.50%	597222.22	0.42	458333.33	-35.83%	459027.78	28.54
p100.21	847222.22	0.00	-19.67%	680555.56	0.46	465277.78	-44.95%	466435.19	34.01
p100.22	784722.22	0.00	-15.93%	659722.22	0.49	506944.44	-34.54%	513657.41	35.27
p100.23	826388.89	0.00	-18.49%	673611.11	0.41	527777.78	-36.13%	527777.78	18.59
p100.24	715277.78	0.00	-17.48%	590277.78	0.41	479166.67	-32.78%	480787.04	31.25
p100.25	708333.33	0.00	-13.73%	611111.11	0.47	472222.22	-33.33%	472222.22	9.55
p100.26	770833.33	0.00	-18.92%	625000.00	0.44	444444.44	-42.13%	446064.81	28.26
p100.27	805555.56	0.00	-15.52%	680555.56	0.45	486111.11	-39.14%	490277.78	27.12
p100.28	750000.00	0.00	-12.96%	652777.78	0.47	465277.78	-37.96%	465277.78	12.58
p100.29	770833.33	0.00	-6.31%	722222.22	0.42	486111.11	-36.70%	487962.96	32.92
	761111.11	0.00	-15.07%	644907.41	0.45	475000.00	-36.95%	478109.57	26.74

Table 6.6: Results for the 30 instances with 100 nodes when using antennas with 201 transmission power levels.

Instance	**BIP**		**BIP + VND**			**ACO**			
	obj. fun	**time (s)**	**deviation**	**obj. fun**	**time (s)**	**best**	**deviation**	**average**	**time (s)**
p100.00	459212.50	0.00	-12.04%	403943.75	0.40	345231.25	-24.82%	345231.25	11.33
p100.01	441043.75	0.00	-10.29%	395675.00	0.37	361331.25	-18.01%	361623.75	40.57
p100.02	474512.50	0.00	-9.01%	431768.75	0.36	385406.25	-18.78%	385411.67	9.36
p100.03	458281.25	0.00	-15.01%	389487.50	0.42	365556.25	-20.12%	366090.62	23.86
p100.04	479650.00	0.00	-9.08%	436093.75	0.40	393343.75	-17.89%	393829.58	15.68
p100.05	531850.00	0.00	-11.72%	469525.00	0.48	424700.00	-20.15%	424700.00	18.91
p100.06	491356.25	0.00	-14.97%	417781.25	0.36	385681.25	-20.89%	388715.83	39.89
p100.07	463506.25	0.00	-10.25%	415993.75	0.36	350681.25	-24.24%	351135.42	14.84
p100.08	472868.75	0.00	-9.71%	426931.25	0.38	380725.00	-19.41%	381064.58	43.65
p100.09	480850.00	0.00	-7.26%	445956.25	0.38	374731.25	-22.07%	374731.25	4.91
p100.10	424231.25	0.00	-11.23%	376575.00	0.38	342006.25	-19.38%	342006.25	3.86
p100.11	471656.25	0.00	-11.29%	418425.00	0.41	362737.50	-23.09%	362737.50	9.21
p100.12	500500.00	0.00	-10.30%	448943.75	0.38	401350.00	-19.70%	401879.17	43.62
p100.13	451593.75	0.00	-11.31%	400525.00	0.45	338781.25	-24.98%	338783.96	39.31
p100.14	429406.25	0.00	-11.30%	380875.00	0.41	351075.00	-18.17%	351387.92	42.73
p100.15	432450.00	0.00	-9.02%	393431.25	0.38	358181.25	-17.17%	358181.25	26.72
p100.16	424568.75	0.00	-11.37%	376300.00	0.36	344987.50	-18.56%	345751.67	32.89
p100.17	458856.25	0.00	-8.91%	417956.25	0.38	378987.50	-15.69%	386868.33	50.71
p100.18	408706.25	0.00	-9.82%	368581.25	0.38	338337.50	-16.09%	342941.04	25.93
p100.19	479712.50	0.00	-8.98%	436618.75	0.44	370693.75	-22.73%	370693.75	10.19
p100.20	454231.25	0.00	-9.47%	411225.00	0.40	360156.25	-20.42%	361480.62	49.89
p100.21	450000.00	0.00	-9.33%	408031.25	0.41	369750.00	-17.71%	370295.62	22.70
p100.22	454412.50	0.00	-6.26%	425981.25	0.32	373293.75	-17.84%	373324.58	24.32
p100.23	537300.00	0.00	-12.88%	468118.75	0.40	417356.25	-22.25%	417749.38	48.98
p100.24	442118.75	0.00	-9.51%	400068.75	0.43	364725.00	-17.41%	365165.00	47.77
p100.25	445268.75	0.00	-11.48%	394168.75	0.46	365462.50	-17.82%	365938.96	44.36
p100.26	465362.50	0.00	-10.09%	418418.75	0.42	357131.25	-23.26%	357131.25	17.85
p100.27	459843.75	0.00	-8.68%	419950.00	0.37	377900.00	-17.55%	379123.96	32.73
p100.28	473806.25	0.00	-16.98%	393337.50	0.45	356293.75	-24.74%	356566.46	49.76
p100.29	470712.50	0.00	-9.19%	427462.50	0.37	363006.25	-22.88%	363006.25	26.78
	462928.96	0.00	-10.56%	413938.33	0.40	368653.33	-20.13%	369451.56	29.11

6.5 A distributed ACO for the MEBRA

In contrast to centralized approaches, work concerning distributed approaches for tackling the MEB problem is quite rare. The best one among these approaches is a distributed version of the BIP algorithm (known as DBIP) which was introduced by Wieselthier et al in [196]. A more recent work by Chen et al. [37] also considers broadcasting. However, the authors focus on a slightly different problem which concerns the reduction of the broadcast time. Although the authors of [37] also give importance to the efficient use of power resources, they show that their algorithm consumes much more energy than DBIP.

In this section we introduce a distributed ant colony optimization algorithm for solving the MEBRA problem. The working of this algorithm is based on the classical BIP heuristic (see Section 6.2). However, in order to be able to use the mechanism of BIP, we introduce a new localized criterion for extending partial solutions during solution construction. The obtained results show that the distributed ant colony optimization algorithm even outperforms the classical (centralized) BIP heuristic. This is in contrast to the distributed version of BIP (labelled DBIP), which performs significantly worse than BIP.

6.5.1 Modification of BIP's Greedy Function

Due to its global character, the original greedy function of BIP can hardly be used within a distributed algorithm in which each sensor node has to choose a transmission power level without any knowledge about the shape of the current partial solution. With this in mind, we subsequently propose a modification of the original greedy function of BIP. This is done with two aims:

1. The modified greedy function should also be applicable in a distributed environment
2. Ideally, the modified greedy function should also be beneficial for the centralized BIP algorithm.

The central idea for a modified greedy function is quite simple. Instead of only taking into account the global increase in transmission power, the modified greedy function should also consider the number of new nodes which will be added to the current solution at each step. In other words, the increase in transmission power should be weighted by the number of added nodes. This idea gave rise to the following modified greedy function:

$$\eta^{+}(e) := \frac{f(T^{\star}) - f(T)}{|V_{T^{\star}}| - |V_T|}, \tag{6.13}$$

here $f(\cdot)$, T and $T^{\star}$ are defined as in the BIP algorithm for the MEBRA (see Section 6.2). In the following we will refer to the version of BIP using this new greedy function as BIP^{+}. Obviously, this greedy function still has a global character. However, it may be used in a distributed setting by replacing the numerator of the definition of $\eta^{+}(\cdot)$ by the transmission power that is necessary to include the goal-node of the edge under consideration.

Although the change in the greedy function is rather small, the structure of the solutions generated by BIP and BIP^{+} is quite different. An example is shown in Figures 6.10 and 6.11 for a small network of seven sensor nodes with node 1 being the source node. The concentric circles around node 1 indicate the six different transmission power levels (remember that $tp_1 = 0$). Moreover, we assume that all sensor nodes are equipped with equivalent antennas.

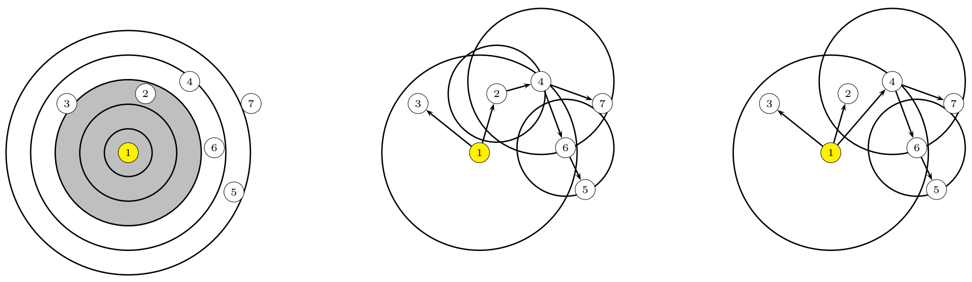
Figures 6.10(a) and 6.11(a) show the initial situation of BIP and BIP$^+$, respectively. The gray-shaded area indicates the transmission power level for node 1 chosen by the two algorithms in the first step. While BIP chooses the smallest transmission power level such that at least one new sensor node is covered, BIP$^+$ makes use of the greedy function η^+ and chooses tp_5. The final solutions of BIP and BIP$^+$ are shown in Figures 6.10(b) and 6.11(b). Due to the solution construction mechanism of BIP it might be that—at the end of a solution construction—some sensor nodes may choose a lower transmission power level without destroying the broadcast property of the solution. For example, in the final solution of BIP (see Figure 6.10(b)) sensor node 2 could even be switched off, because sensor node 4 is also covered by the transmission of the source node 1. In fact, the literature offers a procedure, called SWEEP [197], which takes care of this. The result after applying SWEEP is shown in Figure 6.10(c). Unfortunately, SWEEP cannot be easily applied in a distributed setting. Moreover, in our example, the solution of BIP$^+$ is still better than the one of BIP after the application of SWEEP. This example can, of course, not be seen as a proof that BIP$^+$ consistently outperforms BIP. However, it shows that BIP$^+$ might have advantages over BIP under certain conditions. An experimental evaluation of BIP versus BIP$^+$ is presented in Section 6.5.3 of this work.

6.5.2 The Distributed Ant Colony Optimization Algorithm

In the following we outline the distributed ACO algorithm that we developed on the basis of the centralized state-of-the-art ACO algorithm introduced in Section 6.3. The original centralized ACO algorithm is a so-called $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System (MMAS) implemented in the Hyper-Cube Framework (HCF) [19], which is an iterative algorithm working roughly as follows. First, at each iteration n_a solutions to the problem are constructed in a probabilistic way based on pheromone information and heuristic information. Remember that solutions in this context are spanning trees rooted in the source node s. Second, improvement procedures based on local search are applied to each of these trees. The pheromone model $\mathcal{T}$ consists of a pheromone value τ_e for each link $e \in E$ (for the definition of E see Section 6.1). After the initialization of the variables T^{bs} (i.e., the best-so-far solution), T^{rb} (i.e., the restart-best solution), and cf (i.e., the convergence factor), all the pheromone values are set to 0.5. At each iteration, after the generation of n_a solutions, some of them are used for updating the pheromone values.

DistACO uses the same algorithmic framework. Nevertheless, major design and implementation issues—as outlined in the following—arise. DistACO is devised to be executed directly on the sensor network whose transmissions power levels are to be optimized. The algorithm is controlled by a master node that governs the algorithmic flow described above. This role is played by the source node s. The tasks of master node s consist in triggering actions of other nodes of the network and collect their results. A high-level description of the work flow of master node s is given in Algorithm 12. The actions to be triggered—that is, the construction of a solution and the pheromone update—are described in detail in Sections 6.5.2.1 and 6.5.2.2. Note that, in addition to triggering these actions, the master node itself has to execute the operations described in these sections.

An important remark must be made with regards to other parallel or distributed implementations of ACO algorithms. A considerable amount of work has been made for the parallelization of ACO. The goal of those works was mainly to reduce computation time by means of enabling ants to construct solutions and update the pheromone matrix in parallel. Some examples are the parallel implementation of ACO with OpenMP from [52], the paral-

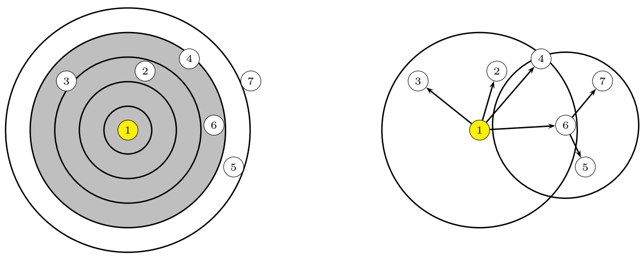


(a) BIP initial state ($V_T = \{1\}$)

(b) BIP final solution

(c) BIP+ SWEEP final solution

Figure 6.10: An example for the working of BIP in the case of a simple sensor network with seven nodes. Node 1 is the source node. All sensor nodes are equipped with the same antennas providing six different transmission power levels (shown as concentric circles around source node 1). (a) shows the initial situation, (b) shows the final solution of BIP, and (c) shows the final solution of BIP after the application of SWEEP.



(a) BIP$^+$ initial state ($V_T = \{1\}$)

(b) BIP$^+$ final solution ($V_T = V$)

Figure 6.11: An example for the working of BIP$^+$ in the case of a simple sensor network with seven nodes. Node 1 is the source node. All sensor nodes are equipped with the same antennas providing six different transmission power levels (shown as concentric circles around source node 1). (a) shows the initial situation, and (b) shows the final solution of BIP$^+$.

Algorithm 12 Task of master node s in DistACO for the MEBRA problem

```
1: INPUT: the number of solutions to be constructed at each iteration ($n_a$), and an iteration
   limit ($\text{iter}_{\text{limit}}$)
2: $it = 0$
3: while $it < \text{iter}_{\text{limit}}$ do
4:     for $j = 1$ to $n_a$ do
5:         $T^j :=$ TriggerSolutionConstruction()
6:     end for
7:     $T^{ib} := \text{argmin}\{f(T^1), \ldots, f(T^{n_a})\}$
8:     TriggerPheromoneUpdate()
9:     $it = it + 1$
10: end while
11: OUTPUT: $T^{bs}$
```

lelization strategies for running ACO in GPUs from [33], and the approach based on sharing a unique pheromone matrix from [135]. In [184], a survey of parallelized multi-colony algorithms is introduced. However, our work differs from these in a very significant point. Our scenario is distributed, and we do not aim at running our centralized algorithm faster, but at being able to run the algorithm directly in the network where broadcasting is required. With this purpose, we cannot consider the amount of CPUs a variable, although this is generally done when analyzing distributed or parallel algorithms. Instead, our number of CPUs in DistACO is exactly the same as the number of nodes in the network. Some important constraints of this approach are that the size and number of transmitted messages must be kept under control. This being especially important in WSNs where energy might be a scarce resource. Besides, differences also appear in the distribution of work among nodes. Generally, parallel algorithms are based on tasks that can be performed, for some time, independently.

Therefore, in the case of ACO this is generally translated to solutions constructed in different CPUs. In our algorithm, all nodes are necessary for constructing a solution. All nodes must be involved in the construction of each solution. Or, in other words, each ant must visit all the nodes before finding a valid solution. As a last remark, notice that the problem that we tackle is quite particular, in the sense that the scenario where the problem arises can also be used to solve the problem itself.

One of the most notable differences with respect to the centralized ACO algorithm concerns the solution components and the pheromone values. For constructing a solution, the centralized algorithm globally maintains a partial solution in form of a directed tree. This partial solution is iteratively extended until the solution is complete. Moreover, solution components are the edges of E, and a pheromone value τ_e is assigned to each solution component (edge) $e \in E$. Just like partial solutions, these pheromone values are maintained globally by the algorithm. In contrast, in DistACO each node is responsible for a set of solution components. More specifically, the m different transmission power levels of a node are now labelled as solution components. Moreover, each node locally maintains a pheromone value τ_{tp_l} for each transmission power level $tp_l \in P$. In DistACO, the task of a node during a solution construction consists in choosing one of the available transmission power levels. This means that nodes do not have any knowledge about the global structure of a solution.

In the following we roughly describe the network behavior while the algorithm is being executed. Later in this section we provide more technical details. DistACO is an iterative algorithm. From a global point of view each iteration consists in first constructing a predefined number of solutions. Second, the quality of these solutions is used to modify the search space for the next iteration. In this context we must mention that DistACO uses a slightly modified objective function. Due to the distributed solution construction mechanism, it may happen that solutions do not span the whole network. In order to *penalize* these (non-valid) solutions, it is assumed that network nodes which are not included in a solution make use of the maximal transmission power level.

Each node in the network has two roles: in first place, a sensor node is responsible for choosing a transmission power level for its antenna, and, in second place, it is responsible for transmitting information to its neighbors guaranteeing the correct working of the algorithm. Both actions are triggered by messages received from neighboring sensor nodes. Sensor nodes use a unique identifier (labelled id) in order to be distinguishable from other sensor nodes. Note that all messages sent include the identifier of the sender and the target nodes. Although the only communication scheme that we use is broadcasting to the local neighborhood, receiver nodes only process those messages whose target identifier matches their own id. The wildcard identifier $*$ is used in case a message is supposed to be processed by all possible receiver nodes.

In order to allow a parallel construction of the n_a solutions per iteration, the master node s associates a sol_id to each solution whose construction it triggers. All messages related to the construction of a particular solution include this sol_id. This enables sensor nodes to recognize the particular solution which is related to each message or request. With this mechanism, the master node can trigger the construction of the n_a solutions per iteration one after the other without having to wait for the end of a triggered solution construction. Once the master node detects that n_a solutions have been constructed, it triggers the update of the pheromone values. This pheromone update is performed locally at each sensor node. The information which is necessary to perform this update is either stored in the sensor nodes themselves or transmitted by the request messages. The respective acknowledgement messages are used to aggregate information about the state of convergence of the algorithm in the master node. Finally, after performing $\mathsf{iter}_{\mathsf{limit}}$ iterations, the master node stops the algorithm and notifies all sensor nodes about the fact that from now on the best-so-far solution should be used for broadcast transmissions having originated from source node s.

6.5.2.1 Construction of Broadcast Trees

In this section we describe the way in which nodes cooperate for the construction of a solution. The program executed by each node for this purpose is shown in Algorithm 13, which is described in detail in the following. All messages related to the construction of a certain solution are identified by a unique key called sol_id. As mentioned before, solution identification allows the network to manage the construction of more than one solution in parallel. During solution construction, the action of a sensor node always depends on its state with respect to this solution construction. The state of construction concerning a solution with key sol_id is stored in state[sol_id], where state[] is an array stored within the sensor nodes. Variables state[sol_id] are initially set to NULL for all sol_id $\in \{1, \ldots, n_a\}$.

A solution construction—from the point of view of an individual sensor node—starts with the arrival of a solution construction request (see lines 3–11 of Algorithm 13). The message m^{req} requesting the construction of a broadcast tree has the following format:

Algorithm 13 Protocol of a sensor node id handling messages concerning solution construction

```
1: $\forall\, i \in \{1,\dots,n_a\}$ do $\mathsf{state_{ini}}[i] := \mathsf{state}[i]$
2: for all $m \in M$ do
3:   if $\mathsf{type}_m = \mathrm{SC}^{\mathrm{REQ}}$ and $\mathsf{state[sol\_id}_m] = \textsc{null}$ then
4:     $\mathsf{father[sol\_id}_m] = \mathsf{origin}_m$
5:     $m^{\mathsf{acc}} :=< \mathsf{id}, \mathsf{father[sol\_id}_m], \mathrm{SC}^{\mathrm{ACC}}, \mathsf{sol\_id}_m >$
6:     $\mathsf{broadcastMessage}(m^{\mathsf{acc}}, p_{\mathsf{max}})$
7:     $\mathsf{power[sol\_id}_m] = \mathsf{chooseTransmissionPowerLevel}()$
8:     $m^{\mathsf{req}_2} :=< \mathsf{id}, *, \mathrm{SC}^{\mathrm{REQ}}, \mathsf{sol\_id}_m >$
9:     $\mathsf{broadcastMessage}(m^{\mathsf{req}_2}, p[\mathsf{sol\_id}_m])$
10:    $\mathsf{accepted[sol\_id}_m] := 0$
11:    $\mathsf{state[sol\_id}_m] := \mathsf{COUNTING\_CHILDREN}$
12:  else if $\mathsf{type}_m = \mathrm{SC}^{\mathrm{ACC}}$ and $\mathsf{state[sol\_id}_m] = \mathsf{COUNTING\_CHILDREN}$ then
13:    $\mathsf{accepted[sol\_id}_m] := \mathsf{accepted[sol\_id}_m] + 1$
14:  else if $\mathsf{type}_m = \mathrm{SC}^{\mathrm{FIN}}$ and $\mathsf{state[sol\_id}_m] = \mathsf{WAITING\_FINMSGS}$ then
15:    $\mathsf{power\_subtree[sol\_id}_m] := \mathsf{power\_subtree[sol\_id}_m] + \mathsf{power\_subtree}_m$
16:    $\mathsf{size\_subtree[sol\_id}_m] := \mathsf{size\_subtree[sol\_id}_m] + \mathsf{size\_subtree}_m$
17:    $\mathsf{accepted[sol\_id}_m] := \mathsf{accepted[sol\_id}_m] - 1$
18:    if $\mathsf{accepted[sol\_id}_m] = 0$ then
19:      $m^{\mathsf{fin}} :=< \mathsf{id}, \mathsf{father[sol\_id}_m], \mathrm{SC}^{\mathrm{FIN}}, \mathsf{sol\_id}_m, \mathsf{power\_subtree[sol\_id}_m],$
                 $\mathsf{size\_subtree[sol\_id}_m] >$
20:      $\mathsf{broadcastMessage}(m^{\mathsf{fin}}, p_{\mathsf{max}})$
21:      $\mathsf{state[sol\_id}_m] := \mathsf{FINISHED}$
22:    end if
23:  end if
24:  Remove $m$ from message queue
25: end for
26: if $\mathsf{state_{ini}[sol\_id}_m] = \mathsf{COUNTING\_CHILDREN}$ then
27:   if $\mathsf{accepted[sol\_id}_m] = 0$ then
28:     $\mathsf{power[sol\_id}_m] := 0$
29:     $\mathsf{power\_subtree[sol\_id}_m] := 0$
30:     $\mathsf{size\_subtree[sol\_id}_m] := 1$
31:     $m^{\mathsf{fin}} :=< \mathsf{id}, \mathsf{father[sol\_id}_m], \mathrm{SC}^{\mathrm{FIN}}, \mathsf{sol\_id}_m, \mathsf{power[sol\_id}_m], 1 >$
32:     $\mathsf{broadcastMessage}(m^{\mathsf{fin}}, p_{\mathsf{max}})$
33:     $\mathsf{state[sol\_id}_m] := \mathsf{FINISHED}$
34:   else
35:     $\mathsf{power\_subtree[sol\_id}_m] := \mathsf{power[sol\_id}_m]$
36:     $\mathsf{size\_subtree[sol\_id}_m] := 1$
37:     $\mathsf{state[sol\_id}_m] := \mathsf{WAITING\_FINMSGS}$
38:   end if
39: end if
```

$$\begin{aligned} m^{\mathsf{req}} =< \mathsf{origin}_{m^{\mathsf{req}}} & \;, \\ \mathsf{target}_{m^{\mathsf{req}}} & = *, \\ \mathsf{type}_{m^{\mathsf{req}}} & = \mathrm{SC}^{\mathrm{REQ}}, \\ \mathsf{sol\_id}_{m^{\mathsf{req}}} & > \;, \end{aligned}$$

where $\mathsf{type}_{m^{\mathsf{req}}}$ is set to $\mathrm{SC}^{\mathrm{REQ}}$ (a unique constant used to identify the message type) and $\mathsf{sol\_id}_{m^{\mathsf{req}}} \in \{1, \ldots, n_a\}$ is the key of the solution that is constructed.

After receiving such a request message, sensor nodes check their state variable concerning solution $\mathsf{sol\_id}_{m^{\mathsf{req}}}$. Only in case $\mathsf{state}[\mathsf{sol\_id}_{m^{\mathsf{req}}}] = \textsc{null}$ the sensor node responds to the sender node $\mathsf{origin}_{m^{\mathsf{req}}}$ by means of a message m^{acc} (see below). Moreover, the sensor node becomes the child of node $\mathsf{origin}_{m^{\mathsf{req}}}$ for what concerns the construction of solution $\mathsf{sol\_id}_{m^{\mathsf{req}}}$, and it stores the sender node $\mathsf{origin}_{m^{\mathsf{req}}}$ as its father in array $\mathsf{father}[]$: $\mathsf{father}[\mathsf{sol\_id}_{m^{\mathsf{req}}}] := \mathsf{origin}_{m^{\mathsf{req}}}$. Message m^{acc} has the following format:

$$\begin{aligned} m^{\mathsf{acc}} =< \mathsf{origin}_{m^{\mathsf{acc}}} & := \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{acc}}} & := \mathsf{father}[\mathsf{sol\_id}_{m^{\mathsf{req}}}], \\ \mathsf{type}_{m^{\mathsf{acc}}} & := \mathrm{SC}^{\mathrm{ACC}}, \\ \mathsf{sol\_id}_{m^{\mathsf{acc}}} & := \mathsf{sol\_id}_{m^{\mathsf{req}}} > \end{aligned}$$

After sending this message the sensor node has to choose a transmission power level in order to contribute to the construction of solution $\mathsf{sol\_id}_{m^{\mathsf{req}}}$. This is done probabilistically, based on the local pheromone values and heuristic information. The heuristic information is, in essence, derived from the idea of the greedy function of BIP^+ (see Section 6.5.1). In this context we consider that each sensor node is aware of the number of neighboring nodes it can reach with each of its transmission power levels. This information can be derived—before the execution of $\mathsf{DistACO}$ is started—from a simple two-step protocol consisting in sending a discovery message with each of the available transmission power levels and counting the number of replies. In the following let n_{tp_l} be the number of neighboring sensor nodes reachable with transmission power level tp_l. Then, a transmission power level from P is chosen according to the following probabilities:

$$\mathbf{p}(tp_l) := \frac{\tau_{tp_l} \cdot \frac{n_{tp_l}}{tp_l}}{\sum_{tp_l' \in P} \tau_{tp_l'} \cdot \frac{n_{tp_l'}}{tp_l'}} \quad , tp_l \in P \tag{6.14}$$

That is, preference is given to transmission power levels with a high pheromone value, weighted by the number of neighboring sensor nodes that may be reached with the corresponding transmission power level. The chosen transmission power level is stored in array $\mathsf{power}[]$: $\mathsf{power}[\mathsf{sol\_id}_{m^{\mathsf{req}}}]$. Moreover, the chosen transmission power level is used to send message m^{req_2}, which has the following format:

$$\begin{aligned} m^{\mathsf{req}_2} =< \mathsf{origin}_{m^{\mathsf{req}_2}} & := \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{req}}} & := *, \\ \mathsf{type}_{m^{\mathsf{req}_2}} & := \mathrm{SC}^{\mathrm{REQ}}, \\ \mathsf{sol\_id}_{m^{\mathsf{req}_2}} & := \mathsf{sol\_id}_{m^{\mathsf{req}}} > \end{aligned}$$

Then, variable $\mathsf{accepted}[\mathsf{sol\_id}_{m^{\mathsf{req}}}]$ of array $\mathsf{accepted}[]$ is initialized to 0. This variable is used to count the number of acknowledgement messages received in response to message m^{req_2}. Note that the sensor nodes which respond to this message adopt the sender node as father for the construction of solution $\mathsf{sol\_id}_{m^{\mathsf{req}}}$. Finally, after sending the m^{req_2} message, the sensor node changes its state to $\mathsf{COUNTING\_CHILDREN}$ and discards the initial m^{req} message.

When the state of a sensor node id concerning solution $\mathsf{sol\_id}$ is $\mathsf{COUNTING\_CHILDREN}$, the node processes messages of type $\mathrm{SC}^{\mathrm{ACC}}$ with $\mathsf{sol\_id}_{m^{\mathsf{acc}}} = \mathsf{sol\_id}$ from its message queue (see lines 12–13 of Algorithm 13). For each of these messages $\mathsf{accepted}[\mathsf{sol\_id}]$ is incremented by one unit. After handling all $\mathrm{SC}^{\mathrm{ACC}}$-messages with $\mathsf{sol\_id}_{m^{\mathsf{acc}}} = \mathsf{sol\_id}$, the value of $\mathsf{accepted}[\mathsf{sol\_id}]$ is checked. In case $\mathsf{accepted}[\mathsf{sol\_id}] = 0$, no neighboring sensor node has accepted to be the child of sensor node id. Therefore, sensor node id may savely be switched off, that is, its transmission power level for $\mathsf{sol\_id}$ is set to 0. Moreover, it directly notfies $\mathsf{father}[\mathsf{sol\_id}]$ by means of a $\mathrm{SC}^{\mathrm{FIN}}$ message (see lines 27–33 of Algorithm 13):

$$\begin{aligned} m^{\mathsf{fin}} :=< \mathsf{origin}_{m^{\mathsf{fin}}} &:= \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{fin}}} &:= \mathsf{father}[\mathsf{sol\_id}], \\ \mathsf{type}_{m^{\mathsf{fin}}} &:= \mathrm{SC}^{\mathrm{FIN}}, \\ \mathsf{sol\_id}_{m^{\mathsf{fin}}} &:= \mathsf{sol\_id}, \\ \mathsf{power\_subtree}_{m^{\mathsf{fin}}} &:= \mathsf{power}[\mathsf{sol\_id}], \\ \mathsf{size\_subtree}_{m^{\mathsf{fin}}} &:= 1 > \end{aligned}$$

Hereby, $\mathsf{power\_subtree}_{m^{\mathsf{fin}}}$ stands for the sum of the transmission power levels of all the nodes in the subtree rooted in sensor node id, which is sending the $\mathrm{SC}^{\mathrm{FIN}}$ message. Moreover, $\mathsf{size\_subtree}_{m^{\mathsf{fin}}}$ contains the size (in terms of the number of sensor nodes) of this subtree.

In case $\mathsf{accepted}[\mathsf{sol\_id}] > 0$, the sensor node updates the following variables (see lines 34–37 of Algorithm 13):

$$(i)\ \ \mathsf{power\_subtree}[\mathsf{sol\_id}] := \mathsf{power}[\mathsf{sol\_id}] \tag{6.15}$$

$$(ii)\ \ \mathsf{size\_subtree}[\mathsf{sol\_id}] := 1 \tag{6.16}$$

Hereby, $\mathsf{power\_subtree}[]$ and $\mathsf{size\_subtree}[]$ are arrays which are used by the sensor node to compute the sum of the transmission power levels, respectively the size, of the subtree for which it acts as root. Moreover, the sensor node changes its state to $\mathsf{WAITING\_FINMSGS}$. As a result, only those neighboring sensor nodes who successfully notify sensor node id during the same communication round will be counted as children.

Being in state $\mathsf{WAITING\_FINMSGS}$ concerning a solution with identifier $\mathsf{sol\_id}$, a sensor node id handles messages of type $\mathrm{SC}^{\mathrm{FIN}}$ in which $\mathsf{sol\_id}_{m^{\mathsf{fin}}} = \mathsf{sol\_id}$ (see lines 14–22 of Algorithm 13). These messages contain aggregated information about the quality of the solution corresponding to $\mathsf{sol\_id}$. These $\mathrm{SC}^{\mathrm{FIN}}$ messages have the following format:

$$\begin{aligned} m^{\mathsf{fin}} =< \mathsf{origin}_{m^{\mathsf{fin}}}&, \\ \mathsf{target}_{m^{\mathsf{req}}} &= \mathsf{id}, \\ \mathsf{type}_{m^{\mathsf{fin}}} &= \mathrm{SC}^{\mathrm{FIN}}, \\ \mathsf{sol\_id}_{m^{\mathsf{fin}}} &= \mathsf{sol\_id}, \\ \mathsf{power\_subtree}_{m^{\mathsf{fin}}}&, \\ \mathsf{size\_subtree}_{m^{\mathsf{fin}}} &> \end{aligned}$$

where $\mathsf{power\_subtree}_{m^{\mathsf{fin}}}$ and $\mathsf{size\_subtree}_{m^{\mathsf{fin}}}$ contain the sum of the transmission power levels, respectively the size, of the subtree for which the sender node $\mathsf{origin}_{m^{\mathsf{fin}}}$ of the SC$^{\mathrm{FIN}}$-message acts as root. Each of these messages received by sensor node id is handled as follows:

$$(i)\quad \mathsf{power\_subtree}[\mathsf{sol\_id}] := \mathsf{power\_subtree}[\mathsf{sol\_id}] + \mathsf{power\_subtree}_{m^{\mathsf{fin}}} \tag{6.17}$$
$$(ii)\quad \mathsf{size\_subtree}[\mathsf{sol\_id}] := \mathsf{size\_subtree}[\mathsf{sol\_id}] + \mathsf{size\_subtree}_{m^{\mathsf{fin}}} \tag{6.18}$$
$$(iii)\quad \mathsf{accepted}[\mathsf{sol\_id}] := \mathsf{accepted}[\mathsf{sol\_id}] - 1 \tag{6.19}$$

Once $\mathsf{accepted}[\mathsf{sol\_id}] = 0$, no further SC$^{\mathrm{FIN}}$-messages are expected concerning $\mathsf{sol\_id}$. Therefore, sensor node id notifies its own parent $\mathsf{father}[\mathsf{sol\_id}]$ about the end of the construction process by sending itself a SC$^{\mathrm{FIN}}$-message in which the content variables are set as follows:

$$\begin{aligned} m^{\mathsf{fin}} :=< \mathsf{origin}_{m^{\mathsf{fin}}} &:= \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{fin}}} &:= \mathsf{father}[\mathsf{sol\_id}], \\ \mathsf{type}_{m^{\mathsf{fin}}} &:= \mathrm{SC}^{\mathrm{FIN}}, \\ \mathsf{sol\_id}_{m^{\mathsf{fin}}} &:= \mathsf{sol\_id}, \\ \mathsf{power\_subtree}_{m^{\mathsf{fin}}} &:= \mathsf{power\_subtree}[\mathsf{sol\_id}], \\ \mathsf{size\_subtree}_{m^{\mathsf{fin}}} &:= \mathsf{size\_subtree}[\mathsf{sol\_id}] > \end{aligned}$$

In general, sensor nodes delete messages from their message queue directly after processing them, regardless of being in a state in which they are considered or not. Therefore, at the end of each communication round, the message queue of a sensor node is always empty. In Table 6.7 we summarize the message flow of a sensor node for the construction of a single solution.

6.5.2.2 Updating the Pheromone Values

After receiving the aggregated information for all initialized solution constructions, the master node s identifies the best one of these solutions and triggers the pheromone update by means of a pheromone update request message (see below). In the following we outline the protocol which is followed by all sensor nodes when receiving messages related to the pheromone update. In this context, sensor nodes keep variables $\mathsf{power}_{\mathsf{rb}}$ and $\mathsf{obj\_funct\_val}_{\mathsf{rb}}$ to store the transmission power level they use in the restart-best solution and the objective function value of the restart-best solution, as well as variables $\mathsf{power}_{\mathsf{gb}}$ and $\mathsf{obj\_funct\_val}_{\mathsf{gb}}$ for storing the transmission power level they use in the best-so-far solution and the objective function value

Table 6.7: Summary of the messages handled by a sensor node id for the construction of a solution with identifier $\mathsf{sol\_id}$.

Order	Type	Sent/Received	Amount	Origin	Target
1	$\mathrm{SC}^{\mathrm{REQ}}$	Received	1	$\mathsf{father[sol\_id]}$	$*$
2	$\mathrm{SC}^{\mathrm{ACC}}$	Sent	1	id	$\mathsf{father[sol\_id]}$
3	$\mathrm{SC}^{\mathrm{REQ}}$	Sent	1	id	$*$
4	$\mathrm{SC}^{\mathrm{ACC}}$	Received	$\mathsf{accepted[sol\_id]}$	children	id
5	$\mathrm{SC}^{\mathrm{FIN}}$	Received	$\mathsf{accepted[sol\_id]}$	children	id
6	$\mathrm{SC}^{\mathrm{FIN}}$	Sent	1	id	$\mathsf{father[sol\_id]}$

of the best-so-far solution. Note that $\mathsf{obj\_funct\_val_{rb}}$ and $\mathsf{obj\_funct\_val_{gb}}$ are set to $|V|\cdot p_{max}$ at the start of $\mathsf{DistACO}$. Moreover, the pheromone update state variable $\mathsf{state_{ph}}$ is initially set to NULL. The protocol of each sensor node is summarized in Algorithm 14. In the following we provide a technical description of all aspects of this protocol.

Sensor nodes engage in the pheromone update upon receiving a request message m^{req} of the following type (see lines 3–19 of Algorithm 14):

$$\begin{aligned} m^{\mathsf{req}} =< \mathsf{origin}_{m^{\mathsf{req}}} &\;, \\ \mathsf{target}_{m^{\mathsf{req}}} &= *, \\ \mathsf{type}_{m^{\mathsf{req}}} &= \mathrm{PH}^{\mathrm{REQ}}, \\ \mathsf{best\_sol\_id}_{m^{\mathsf{req}}} &\;, \\ \mathsf{obj\_funct\_val}_{m^{\mathsf{req}}} &\;, \\ \mathsf{conv\_fact}_{m^{\mathsf{req}}} &\quad > \;, \end{aligned}$$

where $\mathrm{PH}^{\mathrm{REQ}}$ is a constant that identifies the pheromone update request, $\mathsf{best\_sol\_id}_{m^{\mathsf{req}}}$ and $\mathsf{obj\_funct\_val}_{m^{\mathsf{req}}}$ are the identifier and the objective function value of the best solution among the n_a solutions constructed previously. This solution is henceforth referred to as the iteration-best solution. Moreover, $\mathsf{conv\_fact}_{m^{\mathsf{req}}}$ is the value of the convergence factor which was calculated (as explained below) in the previous pheromone update. The sensor node id receiving message m^{req} stores the sender $\mathsf{origin}_{m^{\mathsf{req}}}$ of this message as its father in variable $\mathsf{father_{ph}}$, that is, $\mathsf{father_{ph}} := \mathsf{origin}_{m^{\mathsf{req}}}$. Moreover, sensor node id responds with the following message:

$$\begin{aligned} m^{\mathsf{acc}} =< \mathsf{origin}_{m^{\mathsf{acc}}} &:= \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{acc}}} &:= \mathsf{father_{ph}}, \\ \mathsf{type}_{m^{\mathsf{acc}}} &:= \mathrm{PH}^{\mathrm{ACC}} > \;, \end{aligned}$$

where $\mathrm{PH}^{\mathrm{ACC}}$ is a unique identifier for a message, which has the purpose of accepting a pheromone update request.

The next step consists in the local pheromone update within sensor node id. First, variables $\mathsf{power_{rb}}$, $\mathsf{obj\_funct\_val_{rb}}$, $\mathsf{power_{gb}}$ and $\mathsf{obj\_funct\_val_{gb}}$ are updated in case $\mathsf{obj\_funct\_val}_{m^{\mathsf{req}}} <$

Algorithm 14 Protocol of a sensor node id handling messages concerning the pheromone update

```
 1: $\mathsf{state_{ph}}^{\mathsf{ini}} := \mathsf{state_{ph}}$
 2: for all $m \in M$ do
 3:   if $\mathsf{type}_m = \mathrm{PH}^{\mathrm{REQ}}$ and $\mathsf{state_{ph}} = \textsc{null}$ then
 4:     $\mathsf{father_{ph}} = \mathsf{origin}_m$
 5:     $m^{\mathsf{acc}} :=<\mathsf{id}, \mathsf{father_{ph}}, \mathrm{PH}^{\mathrm{ACC}}>$
 6:     $\mathsf{broadcastMessage}(m^{\mathsf{acc}}, p_{\mathsf{max}})$
 7:     $\mathsf{updateBestSolutions}()$
 8:     $\mathsf{applyPheromoneUpdate}(\mathcal{T})$
 9:     if $\mathsf{conv\_fact}_m \geq 0.98$ then
10:       if $\mathsf{bs\_update} = \textsc{true}$ then
11:         forall $tp_l \in P$ do $\tau_{tp_l} := 0.5$ end forall
12:         $\mathsf{obj\_funct\_val_{rb}} := 0$, $\mathsf{bs\_update} := \textsc{false}$
13:       else
14:         $\mathsf{bs\_update} := \textsc{true}$
15:       end if
16:     end if
17:     $m^{\mathsf{req}_2} :=<\mathsf{id}, *, \mathrm{PH}^{\mathrm{REQ}}, \mathsf{best\_sol\_id}_m, \mathsf{obj\_funct\_val}_m, \mathsf{conv\_fact}_m>$
18:     $\mathsf{broadcastMessage}(m^{\mathsf{req}_2}, p_{\mathsf{max}})$
19:     $\mathsf{accepted_{ph}} := 0$, $\mathsf{state_{ph}} := \mathsf{COUNTING\_CHILDREN}$
20:   else if $\mathsf{type}_m = \mathrm{PH}^{\mathrm{ACC}}$ then
21:     if $\mathsf{state_{ph}} = \mathsf{COUNTING\_CHILDREN}$ then $\mathsf{accepted_{ph}} := \mathsf{accepted_{ph}} + 1$ end if
22:   else if $\mathsf{type}_m = \mathrm{PH}^{\mathrm{FIN}}$ and $\mathsf{state_{ph}} = \mathsf{WAITING\_FINMSGS}$ then
23:     $\mathsf{conv\_fact_{local}} := \mathsf{conv\_fact_{local}} + \mathsf{conv\_fact}_m$
24:     $\mathsf{size\_subtree_{ph}} := \mathsf{size\_subtree_{ph}} + \mathsf{size\_subtree}_m$
25:     $\mathsf{accepted_{ph}} := \mathsf{accepted_{ph}} - 1$
26:     if $\mathsf{accepted_{ph}} = 0$ then
27:       $m^{\mathsf{fin}} :=<\mathsf{id}, \mathsf{father_{ph}}, \mathrm{PH}^{\mathrm{FIN}}, \mathsf{conv\_fact_{local}}, \mathsf{size\_subtree_{ph}}>$
28:       $\mathsf{broadcastMessage}(m^{\mathsf{fin}}, p_{\mathsf{max}})$
29:       $\mathsf{state_{ph}} := \mathsf{FINISHED}$
30:     end if
31:   end if
32:   Remove message $m$ from message queue
33: end for
34: if $\mathsf{state_{ph}}^{\mathsf{ini}} = \mathsf{COUNTING\_CHILDREN}$ then
35:   if $\mathsf{accepted_{ph}} = 0$ then
36:     $m^{\mathsf{fin}} :=<\mathsf{id}, \mathsf{father_{ph}}, \mathrm{PH}^{\mathrm{FIN}}, \mathsf{conv\_fact_{local}}, 1>$
37:     $\mathsf{broadcastMessage}(m^{\mathsf{fin}}, p_{\mathsf{max}})$
38:     $\mathsf{state_{ph}} := \mathsf{FINISHED}$
39:   else
40:     Calculate $\mathsf{conv\_fact_{local}}$, $\mathsf{size\_subtree_{ph}} := 1$,
        $\mathsf{state_{ph}} := \mathsf{WAITING\_FINMSGS}$
41:   end if
42: end if
```

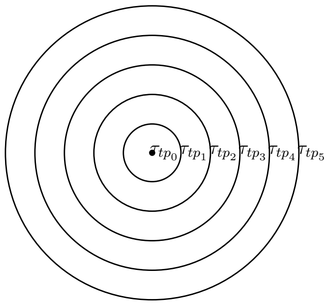


Figure 6.12: Graphical illustration of the pheromone values related to the transmission power levels. The antenna in this example has six different transmission power levels.

$\mathsf{obj\_funct\_val}_{\mathsf{rb}}$, respectively $\mathsf{obj\_funct\_val}_{m^{\mathsf{req}}} < \mathsf{obj\_funct\_val}_{\mathsf{gb}}$. This is done in function updateBestSolutions() of Algorithm 14. Then, the standard pheromone update of a $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System implemented in the Hyper-Cube Framework is performed. This update is based on the objective function values of the iteration-best, restart-best and best-so-far solutions. The influence of each of these three solutions depends on the so-called convergence factor, which provides an estimation of the state of convergence of the algorithm, and on variable bs_update, which is a Boolean control variable. In this context, remember that the current value of the convergence factor is provided by the initial pheromone update request message ($\mathsf{conv\_fact}_{m^{\mathsf{req}}}$), while bs_update is maintained by the sensor nodes as a local variable. Concerning the pheromone values, remember that each sensor node maintains a pheromone value $\tau_{tp_l} \in \mathcal{T}$ for each transmission power level $tp_l \in P$ (see Figure 6.12 for a graphical illustration). The pheromone update is then performed as follows. First, sensor node id calculates an update term ξ_{tp_l} for each transmission power level $tp_l \in P$:

$$\xi_{tp_l} := \kappa_{ib} \cdot \mathbb{1}_{\mathsf{power[best\_sol\_id]}=tp_l} + \kappa_{rb} \cdot \mathbb{1}_{\mathsf{power}_{\mathsf{rb}}=tp_l} + \kappa_{bs} \cdot \mathbb{1}_{\mathsf{power}_{\mathsf{gb}}=tp_l} \quad , \tag{6.20}$$

The indicator function $\mathbb{1}_X$ is defined as 1 if the predicate X is true and 0 otherwise. Moreover, κ_{ib} is the weight of the iteration-best solution, κ_{rb} is the weight of the restart-best solution, and κ_{bs} is the weight of the best-so-far solution. Hereby, the three parameters must be chosen such that $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1.0$. Moreover, in Equation 6.20, the indicator function guarantees that only those solutions in which the transmission power level tp_l is used in sensor node id, contribute to value ξ_{tp_l}. Given the ξ_{tp_l}-values for all $tp_l \in P$, the following update rule is applied to all pheromone values $\tau_{tp_l} \in \mathcal{T}$:

$$\tau_{tp_l} := \min\left\{\max\{\tau_{\min}, \tau_{tp_l} + \rho \cdot \left(\xi_{tp_l} - \tau_{tp_l}\right)\}, \tau_{\max}\right\} \quad , \tag{6.21}$$

where $\rho \in (0, 1]$ is the learning rate, which we have set—as it is standard in ACO algorithms—to 0.1. The upper and lower bounds $\tau_{\max} = 0.99$ and $\tau_{\min} = 0.01$ keep the pheromone values always in the range $(\tau_{\min}, \tau_{\max})$, thus preventing the algorithm from converging to a solution. As in the centralized version of the ACO algorithm presented in [93], the values for κ_{ib}, κ_{rb} and κ_{bs} are chosen depending on the current value of the convergence factor ($\mathsf{conv\_fact}_{m^{\mathsf{req}}}$) and the Boolean control variable bs_update as shown in Table 6.8.

The next step consists in checking if a rescheduling of the κ_* values or a restart is necessary. This depends on the value of $\mathsf{conv\_fact}_{m^{\mathsf{req}}}$ as provided by the request message. Finally, sensor node id sends a pheromone update request message m^{req_2}, which has the following format:

Table 6.8: The schedule used for values κ_{ib}, κ_{rb} and κ_{bs} depending on $\mathsf{conv\_fact}_{m^{\mathsf{req}}}$ (the convergence factor) and the Boolean control variable bs_update.

	$\mathsf{bs\_update} = \text{FALSE}$			$\mathsf{bs\_update} =$
	$\mathsf{conv\_fact}_{m^{\mathsf{req}}} < 0.7$	$\mathsf{conv\_fact}_{m^{\mathsf{req}}} \in [0.7, 0.9)$	$\mathsf{conv\_fact}_{m^{\mathsf{req}}} \geq 0.9$	TRUE
κ_{ib}	2/3	1/3	0	0
κ_{rb}	1/3	2/3	1	0
κ_{bs}	0	0	0	1

$$\begin{aligned} m^{\mathsf{req}_2} =< \mathsf{origin}_{m^{\mathsf{req}_2}} &:= \mathsf{id}, \\ \mathsf{target}_{m^{\mathsf{req}_2}} &:= *, \\ \mathsf{type}_{m^{\mathsf{req}_2}} &:= \mathrm{PH}^{\mathrm{REQ}}, \\ \mathsf{best\_sol\_id}_{m^{\mathsf{req}_2}} &:= \mathsf{best\_sol\_id}_{m^{\mathsf{req}}}, \\ \mathsf{obj\_funct\_val}_{m^{\mathsf{req}_2}} &:= \mathsf{obj\_funct\_val}_{m^{\mathsf{req}}} \\ \mathsf{conv\_fact}_{m^{\mathsf{req}_2}} &:= \mathsf{conv\_fact}_{m^{\mathsf{req}}} > \end{aligned}$$

The handling of the initially received request message m^{req} finishes by setting $\mathsf{accepted}_{\mathsf{ph}}$ to 0. This variable is used to count the number of acknowledgement messages received in response to message m^{req_2}. Note that the sensor nodes which respond to this message adopt sender node id as father for the pheromone update. Finally, after sending the m^{req_2} message, sensor node id changes its state to COUNTING_CHILDREN and discards the initial m^{req} message.

Sensor nodes in the COUNTING_CHILDREN state wait for messages of type $\mathrm{PH}^{\mathrm{ACC}}$ with $\mathsf{target} = \mathsf{id}$ (see lines 20–21 of Algorithm 14). For each of these messages $\mathsf{accepted}_{\mathsf{ph}}$ is incremented by one unit. Moreover, sensor node id's contribution to the calculation of the new convergence factor in the master node ($\mathsf{conv\_fact}_{\mathsf{local}}$) is computed, and the size of the subtree rooted in id (for the aggregation of the information for the pheromone update) is initialized:

$$(i)\quad \mathsf{conv\_fact}_{\mathsf{local}} := \begin{cases} \frac{\tau_{\mathsf{rb}}}{\tau_{\max}} & \text{if } \mathsf{bs\_update} = \text{TRUE} \\ \frac{\tau_{\mathsf{gb}}}{\tau_{\max}} & \text{if } \mathsf{bs\_update} = \text{FALSE} \end{cases} \tag{6.22}$$

$$(ii)\quad \mathsf{size\_subtree}_{\mathsf{ph}} := 1 \tag{6.23}$$

After handling all $\mathrm{PH}^{\mathrm{ACC}}$-messages with $\mathsf{target} = \mathsf{id}$, the value of $\mathsf{accepted}_{\mathsf{ph}}$ is checked. In case $\mathsf{accepted}_{\mathsf{ph}} = 0$, no neighboring node has accepted to be a child of sensor node id. Moreover, it directly notfies sensor node $\mathsf{father}_{\mathsf{ph}}$ by means of a $\mathrm{PH}^{\mathrm{FIN}}$ message (see lines 35–38 of Algorithm 14):

$$
\begin{aligned}
m^{\mathsf{fin}} =< \mathsf{origin}_{m^{\mathsf{fin}}} &:= \mathsf{id},\\
\mathsf{target}_{m^{\mathsf{fin}}} &:= \mathsf{father}_{\mathsf{ph}},\\
\mathsf{type}_{m^{\mathsf{fin}}} &:= \mathrm{PH}^{\mathrm{FIN}},\\
\mathsf{conv\_fact}_{m^{\mathsf{fin}}} &:= \mathsf{conv\_fact}_{\mathsf{local}},\\
\mathsf{size\_subtree}_{m^{\mathsf{fin}}} &:= 1 >
\end{aligned}
$$

Then, sensor node id changes to the FINISHED state. If $\mathsf{accepted}_{\mathsf{ph}} > 0$, sensor node id changes to the WAITING_FINMSGS state. Only in the WAITING_FINMSGS state, $\mathrm{PH}^{\mathrm{FIN}}$-messages are processed (see lines 22–30 of Algorithm 14). These messages have the the following format:

$$
\begin{aligned}
m^{\mathsf{fin}} =< \mathsf{origin}_{m^{\mathsf{fin}}} &\ ,\\
\mathsf{target}_{m^{\mathsf{fin}}} &= \mathsf{id},\\
\mathsf{type}_{m^{\mathsf{fin}}} &= \mathrm{PH}^{\mathrm{FIN}},\\
\mathsf{conv\_fact}_{m^{\mathsf{fin}}} &\ ,\\
\mathsf{size\_subtree}_{m^{\mathsf{fin}}} &>
\end{aligned}
$$

where $\mathsf{conv\_fact}_{m^{\mathsf{fin}}}$ and $\mathsf{size\_subtree}_{m^{\mathsf{fin}}}$ contain the sum of the local convergence factor contributions, respectively the size, of the subtree for which the sender node $\mathsf{origin}_{m^{\mathsf{fin}}}$ of the m^{fin}-message acts as root. Processing a message of type $\mathrm{PH}^{\mathrm{FIN}}$ consists in the following three operations:

$$(i)\quad \mathsf{conv\_fact}_{\mathsf{local}} := \mathsf{conv\_fact}_{\mathsf{local}} + \mathsf{conv\_fact}_{m^{\mathsf{fin}}} \tag{6.24}$$

$$(ii)\quad \mathsf{size\_subtree}_{\mathsf{ph}} := \mathsf{size\_subtree}_{\mathsf{ph}} + \mathsf{size\_subtree}_{m^{\mathsf{fin}}} \tag{6.25}$$

$$(iii)\quad \mathsf{accepted}_{\mathsf{ph}} := \mathsf{accepted}_{\mathsf{ph}} - 1 \tag{6.26}$$

Once $\mathsf{accepted}_{\mathsf{ph}} = 0$, no further $\mathrm{PH}^{\mathrm{FIN}}$-messages are expected. Therefore, sensor node id notifies its own $\mathsf{father}_{\mathsf{ph}}$ about the end of the pheromone value update in its subtree by sending a $\mathrm{PH}^{\mathrm{FIN}}$-message in which the content variables are set as follows:

$$
\begin{aligned}
m^{\mathsf{fin}} =< \mathsf{origin}_{m^{\mathsf{fin}}} &:= \mathsf{id},\\
\mathsf{target}_{m^{\mathsf{fin}}} &:= \mathsf{father}_{\mathsf{ph}},\\
\mathsf{type}_{m^{\mathsf{fin}}} &:= \mathrm{PH}^{\mathrm{FIN}},\\
\mathsf{conv\_fact}_{m^{\mathsf{fin}}} &:= \mathsf{conv\_fact}_{\mathsf{local}},\\
\mathsf{size\_subtree}_{m^{\mathsf{fin}}} &:= \mathsf{size\_subtree}_{\mathsf{ph}} >
\end{aligned}
$$

Finally, sensor node id changes to the FINISHED state. No further messages are processed until the $\mathsf{state}_{\mathsf{ph}}$ is set again to NULL. In Table 6.9 we summarize the message flow of a sensor node for the pheromone update.

Table 6.9: Summary of the messages handled by a sensor node id for the pheromone update.

Order	Type	Sent/Received	Amount	Origin	Target
1	PH^{REQ}	Received	1	$\mathsf{father}_{\mathsf{ph}}$	$*$
2	PH^{ACC}	Sent	1	id	$\mathsf{father}_{\mathsf{ph}}$
3	PH^{REQ}	Sent	1	id	$*$
4	PH^{ACC}	Received	$\mathsf{accepted}_{\mathsf{ph}}$	children	id
5	PH^{FIN}	Received	$\mathsf{accepted}_{\mathsf{ph}}$	children	id
6	PH^{FIN}	Sent	1	id	$\mathsf{father}_{\mathsf{ph}}$

6.5.2.3 Complexity Issues

The data aggregation mechanism used by DistACO for the solution construction and the pheromone update allows to keep the size of the messages quite low. For example, the transmission of a complete solution through the network would require messages of an excessive size of $O(n)$, where n is the number of nodes in the network. In contrast, our mechanism of distributedly storing and updating solution components and pheromone values results in messages of size $O(\log n)$, and avoids the transmission of complete solutions. The amount of information stored at each node is $O(n + n_a \cdot \log |P|)$. This is the space required to store the set of sensor nodes reached with each transmission power level, and the space required to store the index of the transmission power level used at each of the n_a tree constructions per iteration.

In the process of constructing a single solution, each node sends three messages. Moreover, the pheromone update mechanism also requires three messages per node and iteration. The only exception is the master node, which sends only one message for both solution construction and pheromone update. Therefore, the total amount of messages required for a single iteration of the algorithm is $(3n - 2) * (n_a + 1)$, where n_a is the number of solutions constructed per iteration. Finally, the amount of messages processed by each node strongly depends on the network topology.

6.5.3 Experimental Results

We implemented all algorithms considered in this work—namely, BIP, BIP$^+$ and DistACO—in C++ and compiled them with GNU GCC version 4. Hereby, the DistACO algorithm was simulated by means of discrete event simulation. The only external libraries used were those of the STL (standard template library). All the simulations were performed on a cluster with 6 quadcore Intel Xeon X3230 at 2667 MHz (64 bits). Each machine is equipped with 8GB of RAM memory.

Concerning the benchmark instances, we tested the algorithms on a set of 30 publicly available instances introduced in [6, 204] for the case of networks with 50 nodes, and on further 30 instances provided in [90] for networks with 100 nodes. All these networks were randomly generated.

Due to the fact that BIP and BIP$^+$ are both deterministic algorithms, we applied each of them exactly once to each of the 30 instances with 50 nodes and the 30 instances with 100 nodes. These results are used to study the benefits of using the modified greedy function of

BIP$^+$, in comparison to the original greedy function of BIP. Moreover, these results are also used as reference results for evaluating the quality of DistACO. As already mentioned in the introduction, there exists a distributed version of BIP. However, as will be shown below, the results of DistACO compare favorably over those obtained by the centralized BIP algorithm, which—in turn—is significantly better than the distributed version of the BIP algorithm.

Following the characteristics of two industry standard sensor network nodes—namely, *iSense* sensor nodes and *SunSpots*—we test all implemented algorithms in two different scenarios: (1) considering antennas equipped with 7 different transmission power levels, and (2) considering antennnas provided with 201 transmission power levels. The first scenario corresponds to iSense sensor nodes, while the second one is inspired by SunSpots. We fixed the maximum radius of transmission to 500 space units. Moreover, the radius of transmission of the transmission power levels increases proportionally from one level to the next, while the energy consumed increases quadratically. More in detail, considering an antenna with l transmission power levels, the transmission powers are determined as follows:

$$P = \left\{0, \left(1 \cdot \frac{500}{l-1}\right)^2, \left(2 \cdot \frac{500}{l-1}\right)^2, \ldots, \left((l-2)\frac{500}{l-1}\right)^2, 500^2\right\} \tag{6.27}$$

Experimental Setup of DistACO

Concerning the simulation of DistACO, 10 solutions are constructed at each iteration, that is, $n_a = 10$. Moreover, the iteration limit was fixed to 10000 for the 50-node instances and to 50000 for the larger 100-node instances. In the scenario with antennas that offer 201 transmission power levels, a candidate list strategy (as in Section 6.3.) with 8 candidates is used. This means that when a sensor node chooses a transmission power level during the construction of a solution (see Equation 6.14), instead of considering all possibibilities, the set of candidates is reduced to the best 8 candidates as determined by the heuristic information. In the case of antennas with 7 transmission power levels this technique is not used, because the set of candidates is already small enough. As DistACO is a distributed algorithm, we provide the average number of iterations that it used to find its best solution of a run.

Results

Results are presented in tables and reported for each instance separately. Apart from the first column which indicates the instance name, each table consists of four sets of columns, presenting the results of BIP, BIP$^+$, DistACO and the centralized version of ACO, respectively. First is shown the objective function value generated by BIP. The second group of columns presents the results of BIP$^+$. The first one of these columns indicates the deviation (in %) of the results of BIP$^+$ from the results of BIP. Note that a negative deviation indicates that the result of BIP$^+$ is better than the corresponding result of BIP. The second column provides the absolute objective function values generated by BIP$^+$. The third group of columns presents the results obtained by DistACO. The columns contain the best solutions found over 30 independent trials, the average deviation from the results of BIP, the average objective function values obtained in 30 independent runs, and the average number of iterations needed to find the best solution in each run. Finally, the fourth group of columns presents information about the results of the centralized ACO approach (see Section 6.3). In the order of appearance, the columns contain the objective function value of the best solutions found (over 30 independent runs), the average deviation from the results of BIP, and the average quality of the best solutions found in 30 independent trials. Additionally, the last row of each table contains the results averaged over the whole table.

Table 6.10 presents the results for instances composed of 50 nodes considering the scenario with 7 different transmission power levels. First of all, note that only in the case of three instances BIP performs better than BIP^+. Moreover, the solutions provided by BIP^+ never require more than an additional 1.05% of the transmission power consumed by the solution of BIP. However, in those cases in which BIP^+performs better than BIP, energy savings are typically greater than 15% and even reach 36.36% in the case of instance *p50.18*. On average, the results obtained by BIP^+ are 18.08% better than those of BIP. DistACO performs generally better than both BIP and BIP^+. In fact, only in the case of two instances the results of BIP^+ improve over the results obtained by DistACO. On average, the solutions obtained by DistACO consume 25.16% less energy than the ones generated by BIP. Surprisingly, the quality of the solutions produced by DistACO is quite close to the quality of the solutions obtained by the centralized version of ACO. While the results of DistACO are, on average, 25.16% better than the ones of BIP, the results of the centralized ACO version are, on average, 31.88% better than the ones of BIP. This is despite the fact that the centralized ACO version uses global information for constructing solutions and makes use of sophisticated local search procedures.

When the second scenario is considered—that is, antennas with 201 transmission power levels—the differences between BIP, BIP^+ and DistACO are much more subtle. Table 6.11 shows the results obtained for instances consisting of 50 nodes equipped with antennas of 201 different transmission power levels. However, even in this scenario, DistACO performs, on average, slightly better than BIP and BIP^+. It is important to realize that in this scenario the performance of these three algorithms seems to depend strongly on the instance structure. Finally, the centralized ACO algorithm from Section 6.3 performs much better than the other three algorithms. On average, its results are 21.17% better than those of BIP, while the results of DistACO are, on average, only 1.39% better than those of BIP. This might partially be due to the fact that 201 transmission power levels generate a much bigger search space than only 7 transmission power levels.

Considering the larger instances of 100 nodes, in combination with the scenario of 7 transmission power levels (see Table 6.12), the results of BIP^+ and DistACO are again much better than those of BIP. In fact, the difference between BIP^+ and BIP, respectively between DistACO and BIP, is now even bigger than in the case of the 50-node instances. Although the relative increase in performance of DistACO is not as significant as in the case of the BIP^+ algorithm, it still performs better than both BIP and BIP^+. On average, BIP^+ obtains transmission power reductions of 26.11% when compared to the results of BIP. In the case of DistACO, the deviation is even of 27.39%. As expected, the centralized ACO algorithm (characterized by a 36.95% deviation from BIP) again outperforms the other algorithms.

In the scenario in which the nodes of the larger instances composed of 100 nodes are equipped with antennas of 201 different transmission power levels (see Table 6.13), the performance of BIP^+ and DistACO downgrades similar to the case of the 50-node instances. Moreover, BIP^+ and DistACO perform, on average, slightly worse than BIP. This clearly indicates that the number of nodes and number of transmission power levels has strong implications for the complexity of the problem instances. When the number of nodes and the number of transmission power levels grow, the importance of global information seems to grow as well.

Moreover, remember that DistACO is the only one of the four algorithms which is designed to be executed on the network itself and, therefore, the only one capable of generating solutions even in the case of dynamically changing networks.

From a statistical point of view, we provide the confidence intervals of the advantage of ACO over BIP. For instances with 50 nodes with 7 transmission power levels, the interval is $[-27.89, -22.51]$ with a confidence level of 95%. When the number of transmission power levels is increased to 201, the confidence interval is $[-4.61, 1.95]$. For the larger instances with 100 nodes, the confidence intervals are $[-29.75, -24.87]$ and $[5.94, 11.95]$, respectively when 7 or 201 levels of transmission powers are considered. Moreover, all differences are statistically significant. These intervals are obtained using Student's t-tests with R. As in the centralized case, normality can be assumed using the Central Limit Theorem and the fact that we have 30 independent runs per instance.

Finally, in Figure 6.13 we show sample solutions generated by BIP, BIP$^+$, DistACO and the centralized ACO algorithm for instance *p50.26* in the case of the scenario with antennas equipped with 7 transmission power levels. Note that depending on the algorithm used, the structure of the solution may change significantly. More in detail, we observe that the solution provided by BIP seems to waste more energy (as indicated by the gray-shaded areas). Moreover, the structure of the solutions provided by BIP$^+$ and DistACO is quite similar, which may be due to the use of the same heuristic information for the construction of solutions.

Table 6.10: Results for the 30 instances with 50 nodes and antennas with seven transmission power levels.

Instance	**BIP**	BIP$^+$		DistACO				**ACO (centralized)**		
	obj. fun	**deviation**	**obj. fun**	**best**	**deviation**	**average**	**iters.**	**best**	**deviation**	**average**
p50.00	736111.11	-18.87%	597222.22	520833.33	-24.15%	558333.33	3504.47	506944.44	-31.13%	506944.44
p50.01	611111.11	0.00%	611111.11	465277.78	-16.33%	511342.59	4227.63	458333.33	-25.00%	458333.33
p50.02	805555.56	-18.97%	652777.78	527777.78	-32.13%	546759.26	2958.50	520833.33	-35.34%	520833.33
p50.03	590277.78	-10.59%	527777.78	430555.56	-23.84%	449537.04	4621.50	423611.11	-28.24%	423611.11
p50.04	701388.89	-26.73%	513888.89	430555.56	-34.52%	459259.26	2304.53	430555.56	-38.61%	430555.56
p50.05	611111.11	-9.09%	555555.56	506944.44	-14.55%	522222.22	3319.80	486111.11	-20.45%	486111.11
p50.06	729166.67	-18.10%	597222.22	479166.67	-25.87%	540509.26	4149.27	479166.67	-34.29%	479166.67
p50.07	722222.22	-17.31%	597222.22	562500.00	-19.13%	584027.78	3872.90	527777.78	-26.92%	527777.78
p50.08	722222.22	-35.58%	465277.78	402777.78	-35.29%	467361.11	3261.27	402777.78	-44.23%	402777.78
p50.09	680555.56	-17.35%	562500.00	500000.00	-20.07%	543981.48	3146.67	493055.56	-27.55%	493055.56
p50.10	743055.56	-14.95%	631944.44	555555.56	-21.46%	583564.81	3784.87	534722.22	-28.04%	534722.22
p50.11	708333.33	-27.45%	513888.89	458333.33	-31.83%	482870.37	2141.97	444444.44	-37.25%	444444.44
p50.12	659722.22	1.05%	666666.67	506944.44	-15.82%	555324.07	3385.93	493055.56	-25.26%	493055.56
p50.13	750000.00	-17.59%	618055.56	534722.22	-23.40%	574537.04	5119.57	520833.33	-30.56%	520833.33
p50.14	812500.00	-26.50%	597222.22	520833.33	-34.13%	535185.19	3285.27	513888.89	-36.75%	513888.89
p50.15	694444.44	1.00%	701388.89	548611.11	-16.60%	579166.67	3623.10	493055.56	-29.00%	493055.56
p50.16	729166.67	0.95%	736111.11	569444.44	-17.43%	602083.33	4639.93	541666.67	-25.71%	541666.67
p50.17	631944.44	-5.49%	597222.22	500000.00	-15.53%	533796.30	3108.43	500000.00	-20.88%	500000.00
p50.18	763888.89	-36.36%	486111.11	506944.44	-27.33%	555092.59	5321.83	479166.67	-37.27%	479166.67
p50.19	680555.56	-29.59%	479166.67	444444.44	-31.87%	463657.41	3252.70	430555.56	-36.16%	434490.74
p50.20	777777.78	-25.00%	583333.33	548611.11	-26.07%	575000.00	2851.00	493055.56	-36.61%	493055.56
p50.21	736111.11	-26.42%	541666.67	472222.22	-26.45%	541435.19	4871.57	472222.22	-35.85%	472222.22
p50.22	645833.33	-21.51%	506944.44	437500.00	-29.10%	457870.37	3310.13	437500.00	-32.26%	437500.00
p50.23	729166.67	-5.71%	687500.00	541666.67	-18.16%	596759.26	3839.30	527777.78	-27.62%	527777.78
p50.24	736111.11	-20.75%	583333.33	513888.89	-24.50%	555787.04	2204.30	500000.00	-32.08%	500000.00
p50.25	791666.67	-35.09%	513888.89	472222.22	-38.07%	490277.78	2860.33	472222.22	-40.35%	472222.22
p50.26	763888.89	-14.55%	652777.78	576388.89	-22.18%	594444.44	2668.07	527777.78	-30.82%	528472.22
p50.27	805555.56	-18.10%	659722.22	555555.56	-24.86%	605324.07	3317.23	555555.56	-31.03%	555555.56
p50.28	826388.89	-17.65%	680555.56	562500.00	-26.64%	606250.00	4522.57	548611.11	-33.53%	549305.56
p50.29	736111.11	-30.19%	513888.89	458333.33	-37.61%	459259.26	1204.87	458333.33	-37.74%	458333.33
	721064.82	-18.08%	587731.48	503703.70	-25.16%	537700.62	3489.32	489120.37	-31.88%	489297.84

Table 6.11: Results for the 30 instances with 50 nodes and antennas with 201 transmission power levels.

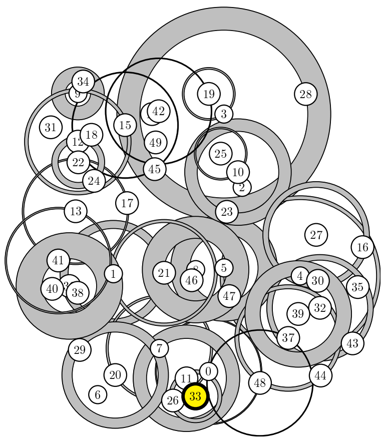
Instance	BIP	BIP$^+$		DistACO				ACO (centralized)		
	obj. fun	deviation	obj. fun	best	deviation	average	iters.	best	deviation	average
p50.00	523075.00	7.34%	561468.75	461612.50	-3.12%	506745.00	4203.60	405175.00	-22.52%	405272.08
p50.01	526025.00	-3.06%	509950.00	444581.25	-11.09%	467673.33	6524.00	378718.75	-28.00%	378718.75
p50.02	544862.50	-13.47%	471462.50	450568.75	-5.24%	516297.50	4266.43	398787.50	-26.81%	398787.50
p50.03	412675.00	-8.77%	376493.75	376718.75	-5.81%	388711.46	3459.93	321968.75	-21.98%	321968.75
p50.04	429493.75	-16.14%	360162.50	380162.50	-8.95%	391071.67	5116.93	331293.75	-22.86%	331293.75
p50.05	471518.75	8.58%	511987.50	433156.25	-6.87%	439120.21	5162.03	392512.50	-16.76%	392512.50
p50.06	507443.75	-3.03%	492075.00	440987.50	-2.21%	496245.62	6160.03	392218.75	-22.71%	392218.75
p50.07	502312.50	-0.89%	497843.75	469506.25	6.04%	532659.58	6169.77	408406.25	-18.69%	408406.25
p50.08	433562.50	-12.17%	380818.75	403987.50	1.73%	441066.67	4904.20	347325.00	-19.89%	347325.00
p50.09	452750.00	-2.63%	440856.25	446681.25	5.47%	477537.29	4779.83	353212.50	-21.99%	353212.50
p50.10	499337.50	3.39%	516256.25	519050.00	7.67%	537661.46	5929.30	423068.75	-15.27%	423068.75
p50.11	442775.00	-6.36%	414606.25	432756.25	4.72%	463659.17	5321.23	374225.00	-15.48%	374225.00
p50.12	465137.50	12.45%	523068.75	449443.75	8.94%	506708.12	5359.87	398731.25	-14.28%	398731.25
p50.13	516387.50	9.58%	565843.75	519650.00	4.75%	540938.12	5024.97	406487.50	-21.28%	406487.50
p50.14	566643.75	3.15%	584506.25	519993.75	0.24%	568008.75	5797.97	420356.25	-25.82%	420356.25
p50.15	438243.75	9.41%	479500.00	396762.50	8.05%	473526.25	6090.67	374937.50	-14.45%	374937.50
p50.16	520862.50	1.39%	528112.50	488506.25	-1.04%	515459.79	5685.50	420850.00	-19.20%	420850.00
p50.17	425031.25	11.52%	473987.50	387250.00	3.42%	439565.83	4342.03	361987.50	-14.83%	361987.50
p50.18	458956.25	-6.25%	430281.25	432131.25	2.06%	468417.08	6498.70	383562.50	-16.43%	383562.50
p50.19	497393.75	-16.61%	414793.75	371037.50	-16.66%	414506.46	5483.80	340006.25	-31.64%	340006.25
p50.20	538018.75	-7.81%	496006.25	609862.50	22.51%	659105.00	3301.20	419450.00	-22.04%	419450.00
p50.21	463031.25	-0.91%	458825.00	393668.75	-6.29%	433892.50	6080.93	367487.50	-20.63%	367487.50
p50.22	477400.00	-20.38%	380125.00	396625.00	-14.73%	407063.54	2847.80	351343.75	-26.40%	351343.75
p50.23	492225.00	8.47%	533900.00	477493.75	5.92%	521357.29	6466.40	389487.50	-20.87%	389487.50
p50.24	534443.75	31.56%	703125.00	480856.25	-4.36%	511135.21	4576.17	409250.00	-23.43%	409250.00
p50.25	533143.75	-23.05%	410237.50	450618.75	-10.53%	476979.79	6184.70	391675.00	-26.53%	391675.00
p50.26	528281.25	-11.93%	465275.00	452525.00	-6.10%	496031.25	3937.40	411450.00	-22.12%	411450.00
p50.27	557493.75	4.08%	580250.00	481868.75	-10.18%	500721.88	4504.63	458293.75	-17.79%	458293.75
p50.28	529150.00	19.83%	634081.25	483787.50	1.64%	537813.96	5213.77	422187.50	-20.21%	422187.50
p50.29	525537.50	-11.52%	465006.25	439312.50	-11.68%	464155.62	6233.00	398131.25	-24.24%	398131.25
	493773.75	-1.14%	488696.88	449705.42	-1.39%	486461.18	5187.56	388419.58	-21.17%	388422.82

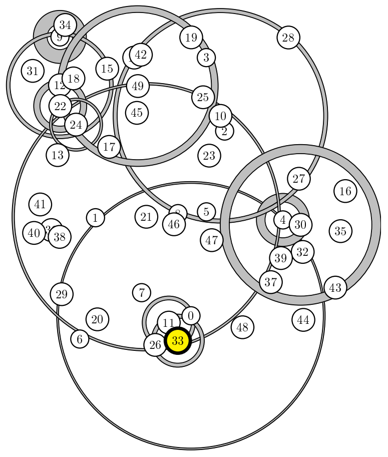
Table 6.12: Results for the 30 instances with 100 nodes and antennas with seven transmission power levels..

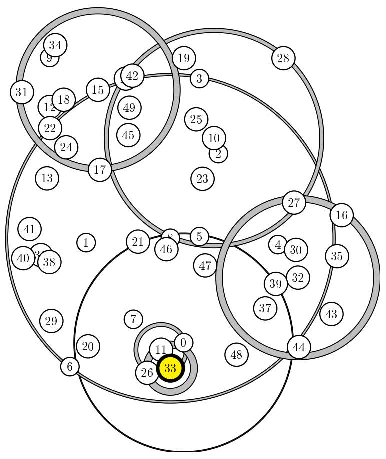
Instance	**BIP**	BIP$^+$		DistACO				**ACO (centralized)**		
	obj. fun	**deviation**	**obj. fun**	**best**	**deviation**	**average**	**iters.**	**best**	**deviation**	**average**
p100.00	722222.22	-16.35%	604166.67	472222.22	-29.71%	507638.89	13441.30	437500.00	-39.42%	437500.00
p100.01	729166.67	-20.00%	583333.33	479166.67	-22.83%	562731.48	21517.90	465277.78	-35.81%	468055.56
p100.02	805555.56	-22.41%	625000.00	506944.44	-33.85%	532870.37	10486.23	486111.11	-39.66%	486111.11
p100.03	694444.44	-26.00%	513888.89	500000.00	-21.67%	543981.48	20945.73	472222.22	-31.67%	474537.04
p100.04	763888.89	-23.64%	583333.33	527777.78	-20.09%	610416.67	17046.27	500000.00	-33.82%	505555.56
p100.05	840277.78	-24.79%	631944.44	541666.67	-27.99%	605092.59	13385.63	541666.67	-34.05%	554166.67
p100.06	847222.22	-31.97%	576388.89	513888.89	-34.54%	554629.63	16601.50	493055.56	-41.80%	493055.56
p100.07	770833.33	-32.43%	520833.33	527777.78	-27.63%	557870.37	21788.63	472222.22	-37.96%	478240.74
p100.08	763888.89	-17.27%	631944.44	534722.22	-26.85%	558796.30	21351.50	506944.44	-32.85%	512962.96
p100.09	729166.67	-20.00%	583333.33	527777.78	-19.05%	590277.78	18675.20	506944.44	-29.97%	510648.15
p100.10	687500.00	-26.26%	506944.44	444444.44	-27.61%	497685.19	16892.20	409722.22	-38.75%	421064.81
p100.11	708333.33	-19.61%	569444.44	493055.56	-26.96%	517361.11	11678.90	472222.22	-33.33%	472222.22
p100.12	826388.89	-21.01%	652777.78	583333.33	-22.66%	639120.37	21443.60	500000.00	-39.19%	502546.30
p100.13	833333.33	-37.50%	520833.33	451388.89	-37.61%	519907.41	17121.43	444444.44	-46.67%	444444.44
p100.14	694444.44	-28.00%	500000.00	500000.00	-24.17%	526620.37	8171.57	465277.78	-32.10%	471527.78
p100.15	680555.56	-29.59%	479166.67	465277.78	-26.46%	500462.96	13650.63	451388.89	-33.61%	451851.85
p100.16	798611.11	-35.65%	513888.89	479166.67	-37.39%	500000.00	11625.37	437500.00	-44.32%	444675.93
p100.17	763888.89	-20.91%	604166.67	513888.89	-22.39%	592824.07	23599.27	472222.22	-38.15%	472453.70
p100.18	645833.33	-13.98%	555555.56	493055.56	-18.39%	527083.33	19341.67	458333.33	-28.03%	464814.81
p100.19	833333.33	-29.17%	590277.78	500000.00	-39.39%	505092.59	6521.87	465277.78	-43.92%	467361.11
p100.20	715277.78	-18.45%	583333.33	527777.78	-19.35%	576851.85	25879.30	458333.33	-35.83%	459027.78
p100.21	847222.22	-44.26%	472222.22	500000.00	-36.34%	539351.85	24670.20	465277.78	-44.95%	466435.19
p100.22	784722.22	-22.12%	611111.11	548611.11	-21.36%	617129.63	19458.07	506944.44	-34.54%	513657.41
p100.23	826388.89	-27.73%	597222.22	569444.44	-23.84%	629398.15	19842.73	527777.78	-36.13%	527777.78
p100.24	715277.78	-24.27%	541666.67	486111.11	-22.46%	554629.63	19582.57	479166.67	-32.78%	480787.04
p100.25	708333.33	-27.45%	513888.89	472222.22	-21.37%	556944.44	17766.83	472222.22	-33.33%	472222.22
p100.26	770833.33	-26.13%	569444.44	444444.44	-37.66%	480555.56	14782.57	444444.44	-42.13%	446064.81
p100.27	805555.56	-35.34%	520833.33	506944.44	-32.07%	547222.22	13912.53	486111.11	-39.14%	490277.78
p100.28	750000.00	-28.70%	534722.22	486111.11	-29.01%	532407.41	18549.63	465277.78	-37.96%	465277.78
p100.29	770833.33	-32.43%	520833.33	500000.00	-31.02%	531712.96	10822.00	486111.11	-36.70%	487962.96
	761111.11	-26.11%	560416.67	503240.74	-27.39%	550555.56	17018.43	475000.00	-36.95%	478109.57

Table 6.13: Results for the 30 instances with 100 nodes and antennas with 201 transmission power levels.

Instance	**BIP**	BIP$^+$		DistACO				**ACO (centralized)**		
	obj. fun	**deviation**	**obj. fun**	**best**	**deviation**	**average**	**iters.**	**best**	**deviation**	**average**
p100.00	459212.50	20.51%	553418.75	427987.50	9.46%	502632.50	31104.17	345231.25	-24.82%	345231.25
p100.01	441043.75	-1.83%	432962.50	446475.00	8.28%	477561.46	35948.73	361331.25	-18.01%	361623.75
p100.02	474512.50	-0.05%	474287.50	458506.25	12.08%	531841.46	34998.23	385406.25	-18.78%	385411.67
p100.03	458281.25	-4.52%	437575.00	421975.00	7.59%	493063.75	30045.07	365556.25	-20.12%	366090.62
p100.04	479650.00	0.46%	481837.50	501156.25	18.76%	569651.25	34910.47	393343.75	-17.89%	393829.58
p100.05	531850.00	5.87%	563062.50	539875.00	9.40%	581866.67	35639.13	424700.00	-20.15%	424700.00
p100.06	491356.25	-2.57%	478706.25	456325.00	0.47%	493685.42	30643.30	385681.25	-20.89%	388715.83
p100.07	463506.25	-0.33%	461993.75	449725.00	10.39%	511657.50	35996.93	350681.25	-24.24%	351135.42
p100.08	472868.75	12.46%	531787.50	497393.75	12.98%	534237.08	35687.17	380725.00	-19.41%	381064.58
p100.09	480850.00	-3.88%	462181.25	449750.00	3.97%	499927.71	32664.03	374731.25	-22.07%	374731.25
p100.10	424231.25	-3.75%	408331.25	465037.50	18.08%	500937.08	35938.43	342006.25	-19.38%	342006.25
p100.11	471656.25	-6.57%	440656.25	419650.00	-7.40%	436731.04	23662.83	362737.50	-23.09%	362737.50
p100.12	500500.00	12.84%	564768.75	474075.00	6.46%	532809.38	37494.23	401350.00	-19.70%	401879.17
p100.13	451593.75	-6.84%	420718.75	396512.50	16.50%	526127.50	33125.40	338781.25	-24.98%	338783.96
p100.14	429406.25	-0.32%	428050.00	417456.25	12.98%	485138.54	33157.83	351075.00	-18.17%	351387.92
p100.15	432450.00	5.73%	457218.75	393050.00	0.11%	432940.00	24793.07	358181.25	-17.17%	358181.25
p100.16	424568.75	-9.08%	386031.25	403012.50	7.56%	456667.50	35510.87	344987.50	-18.56%	345751.67
p100.17	458856.25	-2.59%	446987.50	482043.75	17.15%	537536.46	35722.37	378987.50	-15.69%	386868.33
p100.18	408706.25	3.97%	424943.75	432081.25	13.16%	462508.96	35715.07	338337.50	-16.09%	342941.04
p100.19	479712.50	6.89%	512781.25	441818.75	7.24%	514463.75	34162.07	370693.75	-22.73%	370693.75
p100.20	454231.25	17.73%	534787.50	470781.25	13.48%	515482.50	38139.67	360156.25	-20.42%	361480.62
p100.21	450000.00	-2.29%	439712.50	440475.00	7.37%	483158.54	30296.20	369750.00	-17.71%	370295.62
p100.22	454412.50	10.19%	500731.25	499243.75	20.61%	548047.71	31138.57	373293.75	-17.84%	373324.58
p100.23	537300.00	-7.86%	495075.00	488875.00	1.44%	545028.54	36687.47	417356.25	-22.25%	417749.38
p100.24	442118.75	12.06%	495418.75	465737.50	15.44%	510383.96	34717.83	364725.00	-17.41%	365165.00
p100.25	445268.75	6.92%	476093.75	423037.50	4.29%	464386.04	26163.67	365462.50	-17.82%	365938.96
p100.26	465362.50	-12.19%	408625.00	391437.50	-14.08%	399827.08	35453.83	357131.25	-23.26%	357131.25
p100.27	459843.75	7.19%	492912.50	463725.00	13.78%	523203.33	33381.10	377900.00	-17.55%	379123.96
p100.28	473806.25	-10.25%	425225.00	455043.75	4.19%	493639.79	31688.00	356293.75	-24.74%	356566.46
p100.29	470712.50	-8.67%	429918.75	487862.50	17.11%	551254.58	28794.63	363006.25	-22.88%	363006.25
	462928.96	1.31%	468893.33	452004.17	8.96%	503879.90	33112.68	368653.33	-20.13%	369451.56


(a) BIP. Obj. func. value: 763888.89


(b) BIP$^+$. Obj. func. value: 652777.78


(c) DistACO. Obj. func. value: 611111.11

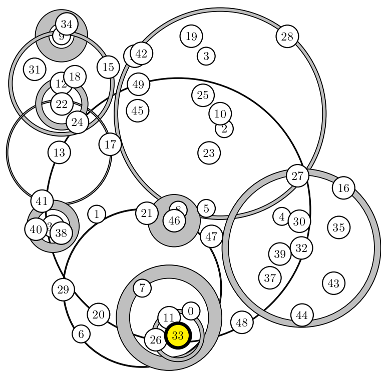

(d) Centralized ACO. Obj. func. value: 527777.78

Figure 6.13: Solutions provided by BIP, BIP$^+$, DistACO and the centralized ACO algorithm for instance *p50.26*. The scenario in which nodes are equipped with antennas of 7 different transmission power levels is considered. The gray shaded parts of the transmissions indicate the waste of energy.

6.6 Conclusions

In this chapter we have first introduced a generalization of the classical minimum energy broadcast problem in wireless ad hoc networks such as sensor networks. The introduced generalization concerns the fact that in many radio antennas used in real sensor network hardware, the choice of a transmission power level is limited to a discrete set of values, rather than being unlimited. As a second contribution we adapted ant colony optimization, a current state-of-the-art algorithm for the classical minimum energy broadcast problem, to the problem generalization. The experimental results have shown that the advantage of this algorithm over classical heuristics even grows when the number of available transmission power levels decreases.

Our third contribution has been to develop a distributed version of the ant colony optimization algorithm for minimum energy broadcasting with realistic antennas. Note that the use of a distributed algorithm is always necessary when the structure of the deployed network is unknown, which might be the case in rather many applications of sensor networks. Due to the fact that the heuristic information used in the centralized algorithm is not available in a distributed environment, one of the main challenges for the development of a distributed algorithm was to define a suitable heuristic information. The experimental evaluation of the proposed algorithm shows that it generally improves over the centralized version of the classical BIP heuristic. Moreover, depending on the exact antenna model used, the results of the distributed ant colony optimization algorithm are very close to the results of the centralized algorithm version.

In this chapter we have first introduced a generalization of the classical minimum energy broadcast problem in wireless ad hoc networks such as sensor networks. The introduced generalization, minimum energy broadcast problem in sensor networks with realistic antennas (MEBRA), concerns the fact that in many radio antennas used in real sensor network hardware, the choice of a transmission power level is limited to a discrete set of values, rather than being unlimited. As a second contribution we adapted ant colony optimization, a current state-of-the-art algorithm for the classical minimum energy broadcast problem, to the problem generalization. The experimental results have shown that the advantage of this algorithm over classical heuristics even grows when the number of available transmission power levels decreases.

The third contribution of this chapter consists in a modified greedy function for the classical broadcast incremental power (BIP) heuristic. The fourth contribution is a distributed version of an ant colony optimization (ACO) algorithm proposed in the literature for the MEBRA problem. The proposed distributed ACO algorithm makes use of localized greedy information for constructing solutions. Moreover, the source node of the broadcast transmission is the pacemaker of the algorithm. Information about constructed solutions and the pheromone update is aggregated in the source node, which, in turn, triggers actions such as the construction of solutions and the update of the pheromone values.

Concerning the results, we were able to observe that the distributed ACO algorithm generally outperforms the centralized classical BIP heuristic. This is a remarkable achievement, because the distributed version of BIP is significantly worse than the centralized version of BIP. When sensor nodes with few transmission power levels are considered, the results of the distributed ACO algorithm are even close to the results of the centralized ACO version. This changes when the number of available transmission power levels grows. In this case, the performance of the centralized ACO algorithm is significantly better than the one of the distributed ACO version. It is also worth to point out that our distributed ACO algorithm uses a relatively low number of messages at each iteration, while keeping a rather low message size. Finally, the main advantage of the proposed distributed ACO algorithm over the centralized ACO version is the fact that it can be applied even when the structure of the network is unknown. This may be the case, for example, in outdoor applications concerning inhospitable environments.

Part III

Conclusions

Chapter 7

Conclusions and further work

In the last decade, wireless sensor networks have attracted the attention of many researchers mainly due to their low infrastructure cost and ease of use. WSNs are a particular kind of wireless ad hoc networks which aim at monitoring large areas of terrain for extended periods of time by means of small devices with limited capabilities. Sensing is performed thanks to different sensors which are generally rather small and cheap. However, WSNs are still not very common in real applications and appear, mostly, as feasibility studies in the context of certain problems. The main barrier for generalizing the use of sensor networks is the lack of standards for their development and the high dependency on the manufacturers. This situation results in a high heterogeneity that may render the integration of different nodes in a single network and the communication between neighboring networks difficult. As a consequence, many of the studies and works that are presented every year face some of the issues that prevent the popularization of sensor networks.

In addition to these problems, research in the field of WSNs is focused mainly on two topics. First, the improvement of the quality, size and capability of the hardware. Second, the development of new algorithms that make a more efficient use of the limited resources of sensor nodes, such as, for instance, the CPU, batteries or communication channels. In particular, WSNs are generally envisaged for remote regions where an unlimited energy source might not be available. The combination of energy being a scarce resource and the low increase of battery capacity during the last years forces devices, algorithms and protocols to control adequately their energetic cost as the overall performance of the network will depend on how energy is managed.

The main contributions of this thesis are to be located in the domain of energy-aware algorithms and protocols. More in detail, we have tackled general problems and tasks that occur in most WSNs applications. When developers implement a new solution for a WSN, they wish to approach the problem from a logical perspective. That is, they want to work from the logical perspective of considering devices as logical units which are able to perform certain tasks by default. Thanks to the tools and libraries provided by some manufacturers and cross-platform projects such as Wiselib [13] this dream comes closer to reality each day. With such tools, developers of new applications can simply obviate the energetic footmark of those tasks, as they are not developing them in detail. However, while some of these algorithms are energy-aware, many others are only focused on speed. In addition to energy-aware algorithms for general tasks, libraries such as Wiselib may also contain algorithms which only aim at improving the management of energetic resources from an application perspective, i.e., minimizing the effect of energy constraints on the performance of the network application. This is the case of the first problem considered in this thesis, duty-cycling.

The first contribution of this work is ANTCYCLE, a self-synchronized duty-cycling mechanism for WSNs which aims at improving the performance of networks with energy-harvesting capabilities (Chapter 4). In addition to a detailed study of the behavior of the algorithm, we provided a robust implementation of ANTCYCLE for real sensor networks. The key aspects of ANTCYCLE are as follows. It is a fully de-centralized algorithm that can adapt the way in which duty-cycling is performed to changing energy availabilities. That is, when little energy is available in the batteries of the individual nodes, duty-cycling is adapted such that either the frequency or the length of the awake-phases are reduced. On the contrary, when a lot of energy is available, the system automatically increases either the frequency or the length of the awake-phases. Moreover, no prior knowledge about the conditions for energy harvesting are needed. The system is able to adapt to changing conditions on the fly.

The second problem studied is graph coloring (Chapter 5), a classical problem of modern mathematics with more than 150 years of history. The problem has been extensively studied in theory and practice. However, its relation to problems that arose with the proliferation of wireless networks has sparked a special interest in resolving the problem in a distributed manner. In distributed algorithms—due to the lack of global knowledge—the nodes have to base their color choices exclusively on information they receive from their direct neighborhood. Common tasks in WSNs related to distributed graph coloring are TDMA slot assignment or the optimization of resources for covering certain regions. The novel algorithm introduced in this work, named FROGSIM, makes use of a desynchronization mechanism for solving the coloring problem. Results show that the resulting lightweight algorithm compares favorably to state-of-the-art techniques.

The last one of the problems considered (Chapter 6) is a generalization of the classical minimum energy broadcast problem in wireless ad hoc networks. The introduced generalization concerns the fact that in many radio antennas used in real sensor network hardware, the choice of a transmission power level is limited to a discrete set of values, rather than being unlimited. We show that an extension of a successful metaheuristic used for the original problem version can be used in centralized scenarios. In fact, the experimental results show that the advantage of this algorithm over classical heuristics even grows when the number of available transmission power levels decreases. Moreover, due to the fact that the use of centralized algorithms in WSN applications is highly restricted, we show that a competitive distributed version of the algorithm is also possible.

All above-mentioned problems have been tackled from the common perspective of swarm intelligence. Swarm intelligence algorithms and mechanisms are inspired by the behavior of social animals and insects which are able to solve complex problems thanks to cooperation. The collective behavior that can be observed in animal societies is the inspiring idea for applying such techniques to a distributed scenario such as WSNs. Moreover, the instinctive behaviors observed in nature can be generally translated to lightweight solutions which are computationally not very intensive and, therefore, favor energy savings. More in detail, the duty-cycling algorithm from Chapter 4 is inspired by the resting phases of ant colonies, the distributed graph coloring algorithm from Chapter 5 is based on the anti-phase synchronization behavior of the calling of Japanese tree frogs, and the foraging behavior of ant colonies inspired the ant colony optimization metaheuristic used to solve the minimum energy broadcast problem with realistic antennas (both from a centralized and a distributed perspective) in Chapter 6.

All contributions of this thesis have been presented to the scientific community before writing this thesis. More specifically, contributions have been presented in international conferences and journals. The reports provided by the referees have been read carefully. The concerns and comments of the referees have been handled and resolved in order to improve the quality of this work. Moreover, our ideas have evolved thanks to the advice and concerns of many colleagues from the scientific community. At every conference and workshop that we attended, we have received invaluable comments which have caused our novel algorithms to mature. A thorough and detailed description of the techniques developed for each problem has been published in internationally recognized journals.

Future work has been proposed at the end of each of the chapters. However, future work can mostly be summarized in two words: *real applications*. First of all, this work demonstrates the feasibility of the introduced algorithms from a swarm intelligence perspective. However, in our aim to prove the ability of all solutions to work in real scenarios, we provided implementations for real simulators and published our results also in conferences and journals specialized on WSNs and general wireless networking. In this sense, the work provides evidence about the maturity of the introduced algorithms. We believe that it will be possible to deploy them in real scenarios, independently of whether they are real or simulated. From the very beginning, our goal has been to provide real solutions for developers working on real problems, with real networks and sensors.

The natural continuation of this work is to implement our novel algorithms in real networks. In this line, the duty-cycling algorithm has been included in the Wiselib and thus is publicly available for the users of this general purpose algorithmic library for WSNs. In fact, in a preliminary and promising work by Hernández and Baumgartner [89], the authors used the Wiselib implementation of the duty-cycling algorithm from Chapter 4 in a real sensor network composed of iSense nodes which is installed in a corridor of the Technical University of Braunschweig. This network lacks of energy harvesting capabilities. However, the authors used the duty-cycling algorithm to adapt the activity of the nodes to the amount of people crossing the corridor.

The ideas proposed in this work can be used to reduce the energy consumption in sensor networks. In particular, we have shown that there is still room for more efficient energy-aware algorithms. In this sense, some advances may come from carefully reviewing traditional approaches. However, as demonstrated in this thesis, innovative computational paradigms such as swarm intelligence should not be disregard. This is because the distributed and autonomous organization of WSNs strongly resembles the one of swarm intelligence systems. As previously mentioned, swarm intelligence is based on the collective behavior of social insects and animals. While working on this thesis, we have observed over and over again, with astonishment and satisfaction, how complex global results have emerged from simple local behaviors, just like the way in which ants and frogs function in real life.

Bibliography

[1] N. Abramson. The ALOHA System: Another alternative for computer communications. In *Proceedings of the ACM Fall Joint Computer Conference*, pages 281–285. ACM, 1970.

[2] I. Aihara. Modeling synchronized calling behavior of Japanese tree frogs. *Physical Review E*, 80(1):11–18, 2009.

[3] I. Aihara, H. Kitahata, K. Yoshikawa, and K. Aihara. Mathematical modeling of frogsćalling behavior and its possible application to artificial life and robotics. *Artificial Life and Robotics*, 12(1):29–32, 2008.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[6] S. Al-Shihabi, P. Merz, and S. Wolf. Nested partitioning for the minimum energy broadcast. In V. Maniezzo, R. Battiti, and J.P. Watson, editors, *LION 2007 – Proceedings of the Workshop on Learning and Intelligent Optimization*, volume 5313 of *Lecture Notes in Computer Science*, pages 1–11. Springer Verlag, Berlin, Germany, 2007.

[7] C. Álvarez, I. Chatzigiannakis, A. Duch, J. Gabarro, O. Michail, M. Serna, and P. G. Spirakis. Computational models for networks of tiny artifacts: A survey. *Computer Science Review*, 5(1):7 – 25, 2011.

[8] C. Álvarez, A. Duch, J. Gabarro, and M. Serna. Sensor field: A computational model. In Shlomi Dolev, editor, *ALGOSENSORS 2009 – Proceedings of the 5th International Workshop in Algorithmic Aspects of Wireless Sensor Networks*, volume 5804 of *Lecture Notes in Computer Science*, pages 3–14. Springer Berlin / Heidelberg, 2009.

[9] K. Arisha, M. Youssef, and M. Younis. *System-Level Power Optimization for Wireless Multimedia Communication*, chapter Energy-aware TDMA-based MAC for sensor networks, pages 21–40. Springer, 2002.

[10] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379 – 388, 2003.

[11] L. Barenboim and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *ACM SIGACT-SIGOPS 2010 – Proceedings of the 29th Symposium on Principles of Distributed Computing*, pages 410–419. ACM, New York, NY, USA, 2010.

[12] R. Battiti, A. A. Bertossi, and M. A. Brunato. Distributed Saturation Degree Methods for Code Assignment in Multihop Radio Networks. In *WSDAAL 00 – Proceedings of the* 5^{th} *Workshop Su Sistemi Distribuiti: Algoritmi, Architetture e Linguaggi*, 2000.

[13] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, and A. Pyrgelis. Wiselib: A generic algorithm library for heterogeneous sensor networks. In J. Silva, B. Krishnamachari, and F. Boavida, editors, *Wireless Sensor Networks*, volume 5970 of *Lecture Notes in Computer Science*, pages 162–177. Springer Berlin / Heidelberg, 2010.

[14] C. Bettstetter and C. Hartmann. Connectivity of wireless multihop networks in a shadow fading environment. *Wireless Networks*, 11(5):571–579, 2005.

[15] S. Bird and X. Li. Adaptively choosing niching parameters in a pso. In *GECCO 2006 – Proceedings of the 8th Annual Conference in Genetic and Evolutionary Computation, LNCS*, pages 3–10. Springer, July 8–12, 2006.

[16] I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960–975, 2008.

[17] C. Blum. *Theoretical and practical aspects of ant colony optimization*, volume 282. IOS Press, 2004.

[18] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

[19] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man anc Cybernetics – Part B*, 34(2):1161–1172, 2004.

[20] C. Blum and D. Merkle, editors. *Swarm Intelligence: Introduction and Applications.* Natural Computing. Springer Verlag, Berlin, Germany, 2008.

[21] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[22] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, NY, 1999.

[23] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Fixed response thresholds and the regulation of division of labor in insect societies. *Bulletin of Mathematical Biology*, 60(4):753–807, 1998.

[24] A. Braunstein, R. Mulet, A. Pagnani, M. Weigt, and R. Zecchina. Polynomial iterative algorithms for coloring and analyzing random graphs. *Physical Review E*, 68(3), 2003.

[25] R. Brits, A. P. Engelbrecht, and F. Van den Bergh. A niching particle swarm optimizer. In *SEAL 2002 – Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 692–696, 2002.

[26] J. Buck. Synchronous rhythmic flashing of fireflies. II. *Quarterly Review of Biology*, 63(3):265–289, 1988.

[27] J. Buck and E. Buck. Synchronous fireflies. *Scientific American*, 234(5):74–85, 1976.

[28] M. Bui, F. Butelle, and C. Lavault. A distributed algorithm for constructing a minimum diameter spanning tree. *Journal of Parallel and Distributed Computing*, 64(5):571–577, 2004.

[29] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: a computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[30] M. Cagalj, J. P. Hubaux, and C. Enz. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. In *ACM MobiCom 2002 – Proccedings of The Annual International Conference on Mobile Computing and Networking*, pages 172–182. ACM press, 2002.

[31] M. Campos, E. Bonabeau, G. Theraulaz, and J. L. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(2):83–95, 2000.

[32] M. Cardei, D. MacCallum, and X. Cheng. Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks*, 3(4):213–229, 2002.

[33] J.M. Cecilia, J.M. Garcia, M. Ujaldon, A. Nisbet, and M. Amos. Parallelization strategies for ant colony optimization on gpus. In *IPDPSW 2011 – Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops*, pages 339–346, 2011.

[34] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[35] A. Chandrakasan, R. Amirtharajah, S. H. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang. Design considerations for distributed microsensor systems. In *Proceedings of CICC 1999 – IEEE Custom Integrated Circuits Conference*, pages 279–286. IEEE Press, 1999.

[36] L. S. Chen, J. H. Chen, and H. C. Wang. An optimum branching-based efficient distributed broadcast scheme for wireless ad hoc networks. In *AH-ICI 2009 – Proceedings of the 1st Asian Himalayas International Conference on Internet*, pages 1–6. IEEE press, 2009.

[37] Li-Sheng Chen, Jung-Hsien Chen, and Hwang-Cheng Wang. An optimum branching-based efficient distributed broadcast scheme for wireless ad hoc networks. In *AH-ICI 2009 – Proceedings of First Asian Himalayas International Conference on Internet.*, pages 1–6. IEEE Press, 2009.

[38] M. Clerc. *Particle swarm optimization.* ISTE Ltd, UK, 2006.

[39] M. Clerc and J. Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[40] Coalsesenses GmbH. http://www.coaelesenses.com.

[41] B. J. Cole. The social behavior of Leptothorax allardycei (Hymenoptera, Formicidae): time budgets and the evolution of worker reproduction. *Behavioral Ecology and Sociobiology*, 18(3):165–173, 1986.

[42] B. J. Cole. Is animal behaviour chaotic? Evidence from the activity of ants. In *Proceedings of the Royal Society*, volume 244, pages 253–259. The Royal Society, 1991.

[43] B. J. Cole. Short-Term Activity Cycles in Ants: Generation of Periodicity by Worker Interaction. *American Naturalist*, 137(2):244, 1991.

[44] T. C. Collier and C. Taylor. Self-organization in sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):866–873, 2004.

[45] C. Cutts and J. Speakman. Energy savings in formation flight of pink-footed geese. *Journal of experimental biology*, 189(1):251–261, 1994.

[46] A. K. Das, R. J. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray. The minimum power broadcast problem in wireless networks: an ant colony system approach. In *IEEE CAS 2002 – Proceedings of the Workshop on Wireless Communications and Networking*, pages 1001–1010. IEEE Press, 2002.

[47] A. K. Das, R. J. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray. r-shrink: A heuristic for improving minimum power broadcast trees in wireless networks. In *IEEE GLOBECOM 2003 – Proceedings of the Global Communications Conference*, pages 523–527. IEEE press, 2003.

[48] L. Davis. *Handbook of genetic algorithms.* Van Nostrand Reinhold, New York, NY, USA, 1991.

[49] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In Sven Brueckner, Paul Robertson, and Umesh Bellur, editors, *IEEE SASO 2008 – Proceedings of the 2nd International Conference Self-Adaptive and Self-Organizing Systems*, pages 129–138. IEEE Press, 2008.

[50] J. Delgado and R. V. Solé. Mean-field theory of fluid neural networks. *Phys. Rev. E*, 57:2204–2211, 1998.

[51] J. Delgado and R. V. Solé. Self-synchronization and Task Fulfilment in Ant Colonies. *Journal of Theoretical Biology*, 205(3):433–441, 2000.

[52] P. Delisle, M. Krajecki, M. Gravel, and C. Gagné. Parallel implementation of an ant colony optimization metaheuristic with OpenMP. In *EWOMP 2001 – Proceedings of the 3rd European Workshop on OpenMP*, pages 8–12, 2001.

[53] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[54] P. Diehr and T. Lumley. The importance of the normality assumption in large public health data sets. *Annu Rev Public Health*, 23:151–169, 2002.

[55] M. Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[56] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the travelingsalesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[57] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, Dipartimento di Elettronica e Informatica, Politecnico di Milano, 1991.

[58] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(1):29–41, 1996.

[59] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[60] R. Dorne and J. K. Hao. A new genetic local search algorithm for graph coloring. In Agoston Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *PPSN '98 – Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 745–755. Springer Berlin / Heidelberg, 1998.

[61] F. Dressler. *Organic Computing*, chapter Bio-Inspired Networking—Self-Organizing Networked Embedded Systems, pages 261–284. Understanding Complex Systems. Springer Verlag, Berlin, Germany, 2008.

[62] R. C. Eberhart and R. W. Dobbins, editors. *Neural Network PC Tools: a Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[63] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *IEEE CEC 2000 – Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 84–88. IEEE Press, 2000.

[64] J. T. Enright. Temporal precision in circadian systems: a reliable neuronal clock from unreliable components? *Science*, 209(4464):1542, 1980.

[65] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Proceedings of ICASSP 2001 – IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2033–2036. IEEE Press, 2001.

[66] EU. FRONTS Project Website. http://www.fronts.eu.

[67] EU. WISEBED Project Website. http://www.wisebed.eu.

[68] S. Fekete, A. Kröller, S. Fischer, and D. Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *INSS 2007 – Proceedings of the 4th International Conference on Networked Sensing Systems*, page page 299. IEEE Press, Piscataway, NJ, 2007.

[69] I. Finocchi, A. Panconesi, and R. Silvestri. An experimental analysis of simple, distributed vertex coloring algorithms. *Algorithmica*, 41(1):1–23, 2005.

[70] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[71] Center for Discrete Mathematics and Theoretical Computer Science. Dimacs implementation challenges, 2006. http://dimacs.rutgers.edu/Challenges/.

[72] P. Fraigniaud, C. Gavoille, D. Ilcinkas, and A. Pelc. Distributed computing with advice: Information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, 2009.

[73] N. R. Franks and S. Bryant. Rhythmical patterns of activity within the nest of ants. *Chemistry and Biology of Social Insects*, 7(2):122–123, 1987.

[74] N. R. Franks, S. Bryant, R. Griffiths, and L. Hemerik. Synchronization of the behaviour within nests of the ant Leptothorax acervorum (fabricius)-I. Discovering the phenomenon and its relation to the level of starvation. *Bulletin of Mathematical Biology*, 52(5):597–612, 1990.

[75] P. Galinier and J.-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[76] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of SenSys 2003 – 1st ACM Conference on Embedded Networked Sensor Systems*, pages 138–149. ACM Press, 2003.

[77] C. Gavoille, R. Klasing, A. Kosowski, L. Kuszner, and A. Navarra. On the complexity of distributed graph coloring with local minimality constraints. *Networks*, 54(1):12–19, 2009.

[78] P. P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez les Bellicositermes Natalenis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, 1959.

[79] C. Guo, L. C. Zhong, and J. M. Rabaey. Low power distributed mac for ad hoc sensor radio networks. In *IEEE GLOBECOM 2001 – Proceedings of the Global Telecommunications Conference*, volume 5, pages 2944 –2948, 2001.

[80] S. Guo and O. Yang. A dynamic multicast tree reconstruction algorithm for minimum-energy multicasting in wireless ad hoc networks. In *IEEE IPCCC 2004 – Proceedings of the International Performance Computing and Communications Conference*, pages 637–642. IEEE press, 2004.

[81] S. Guo and O. Yang. Energy-aware multicasting in wireless ad hoc networks: A survey and discussion. *Computer Communications*, 30:2129–2148, 2007.

[82] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[83] J. Hansen, M. Kubale, Ł. Kuszner, and A. Nadolski. Distributed largest-first algorithm for graph coloring. In Marco Danelutto, Marco Vanneschi, and Domenico Laforenza, editors, *Euro-Par 2004 Parallel Processing – Proceedings of the* 10th *International Euro-Par Conference*, pages 804–811. Springer Berlin / Heidelberg, 2004.

[84] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[85] F. E. Hanson. Comparative studies of firefly pacemakers. *Federation Proceedings*, 37(8):2158–2164, 1978.

[86] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. *The Ubiquity of Chaos*, page 233, 1990.

[87] J. M. Herhers. Social organisation in Leptothorax ants: within-and between-species patterns. *Psyche: A Journal of Entomology*, 90(4):361–386, 1983.

[88] T. Herman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. In Sotiris Nikoletseas and José D. P. Rolim, editors, *ALGOSENSORS 2004 – Proceedings of* 1^{st} *International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 45–58. Springer, 2004.

[89] H. Hernández and T. Baumgartner. Perada research exchange: Real experimentation with a self-synchronized duty-cycling mechanism. Technical report, Available as a PerAda exchange report in `http://www.perada.eu/documents/2010/exchange-reports/hugo-hernandez.pdf`, 2010.

[90] H. Hernández and C. Blum. Ant colony optimization for multicasting in static wireless ad-hoc networks. *Swarm Intelligence*, 3(2):125–148, 2009.

[91] H. Hernández and C. Blum. Foundations of antcycle: Self-synchronized duty-cycling in mobile sensor networks. *The Computer Journal*, 54(9):1427–1448, 2011.

[92] H. Hernández and C. Blum. Implementing a model of Japanese tree frogs' calling behavior in sensor networks: A study of possible improvements. In *BIS-WSN 2009 – Proceedings of the 1st International Workshop on Bio-Inspired Solutions for Wireless Sensor Networks.* ACM press, New York, NY, 2011. In press.

[93] H. Hernández and C. Blum. Minimum energy broadcasting in wireless sensor networks: An ant colony optimization approach for a realistic antenna model. *Applied Soft Computing*, 11(8):5684–5694, 2011.

[94] H. Hernández and C. Blum. Distributed graph coloring: an approach based on the calling behavior of japanese tree frogs. *Swarm Intelligence*, 2012. In press.

[95] H. Hernández and C. Blum. Frogsim: Distributed graph coloring in wireless ad hoc networks – an algorithm inspired by the calling behavior of japanese tree frogs. *Telecommunication Systems*, 2012. In press.

[96] H. Hernández, C. Blum, M. Blesa, T. Baumgartner, S. Fekete, and A. Kröller. Self-synchronized duty-cycling for sensor networks with energy harvesting capabilities: Implementation in wiselib. In *MSN 2010 – Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 134–139. IEEE Press, 2010.

[97] H. Hernández, C. Blum, M. Middendorf, K. Ramsch, and A. Scheidler. Self-synchronized duty-cycling for mobile sensor networks with energy harvesting capabilities: A swarm intelligence study. In Y. Shi, editor, *IEEE SIS 2009 – Proceedings of the Swarm Intelligence Symposium*, pages 153–159. IEEE press, Piscataway, NJ, 2009.

[98] A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[99] J. Honerkamp. The heart as a system of coupled nonlinear oscillators. *Journal of Mathematical Biology*, 18(1):69–88, 1983.

[100] Y. W. Hong, L. F. Cheow, and A. Scaglione. A simple method to reach detection consensus in massively distributed sensor networks. In *Proceedings of ISIT 2004 – IEEE International Symposium on Information Theory*, page 251. IEEE Press, 2004.

[101] Y. W. Hong and A. Scaglione. Time synchronization and reach-back communications with pulse-coupled oscillators for UWB wireless ad hoc networks. In *Proceedings of UWBST 2003 – IEEE Conference on Ultra Wideband Systems and Technologies*, pages 190–194. IEEE Press, 2003.

[102] J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan. Adaptive duty cycling for energy harvesting systems. In *ISLPED 2006 – Proceedings of the International Symposium on Low Power Electronics and Design*, pages 180–185. IEEE Press, Piscataway, NJ, 2006.

[103] A. Iyer, S. S. Kulkarni, V. Mhatre, and C. P. Rosenberg. *Wireless Sensor Networks and Applications*, chapter A Taxonomy-based Approach to Design of Large-scale Sensor Networks, pages 3–33. Springer, 2008.

[104] J. Jalife. Mutual entrainment and electrical coupling as mechanisms for synchronous firing of rabbit sino-atrial pace-maker cells. *The Journal of Physiology*, 356(1):221–243, 1984.

[105] P. Janacik, T. Heimfarth, and F. Rammig. Emergent Topology Control Based on Division of Labour in Ants. In *AINA 2006 – Proceedings of the International Conference on Advanced Information Networking and Applications*, pages 733–740. IEEE Computer Society, Los Alamitos, CA, USA, 2006.

[106] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1272–1282, 2005.

[107] T. R. Jensen. *Graph coloring problems.* John Wiley & Sons, 1994.

[108] I. Kang and R. Poovendran. Iterated local optimization for minimum energy broadcast. In *WiOpt 2005 – Proceedings of the 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 332–341. IEEE press, 2005.

[109] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *Transactions on Embedded Computing Systems*, 6(4):Article 32, 38 pp, 2007.

[110] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Communications – Proceedings of a Symposium on the Complexity of Computer Computations*, page 85, 1972.

[111] J. Kennedy. Bare bones particle swarms. In *IEEE SIS'03 – Proceedings of the 2003 Swarm Intelligence Symposium*, pages 80–87. IEEE Press, 2003.

[112] J. Kennedy. *Handbook of Nature-Inspired and Innovative Computing*, chapter Swarm intelligence, pages 187–219. Springer, 2006.

[113] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of ICNN 1995 – IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press, 1995.

[114] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence.* The Morgan Kaufmann Series in Networking. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[115] A. Keshavarzian, H. Lee, and L. Venkatraman. Wakeup scheduling in wireless sensor networks. In *ACM MobiHoc 06 – Proceedings of the* 7th *International Symposium on Mobile Ad-Hoc Networking and Computing*, pages 322–333. ACM, New York, NY, USA, 2006.

[116] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[117] A. Kosowski and Ł. Kuszner. On greedy graph coloring in the distributed model. In Wolfgang Nagel, Wolfgang Walter, and Wolfgang Lehner, editors, *Euro-Par 2006 – Proceedings of the* 12th *International Euro-Par Conference*, pages 592–601. Springer Berlin / Heidelberg, 2006.

[118] G. T. Kovacs. *Micromachined Transducers Sourcebook.* McGraw-Hill Science, 1998.

[119] L. Kroc, A. Sabharwal, and B. Selman. Counting solution clusters in graph coloring problems using belief propagation. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS 2008 – Proceedings of 22nd Annual Conference on Neural Information Processing Systems*, pages 873–880. MIT Press, Cambridge, MA, 2009.

[120] M. Kubale and L. Kuszner. A better practical algorithm for distributed graph coloring. In *PARELEC 02 – Proceedings of the International Conference on Parallel Computing in Electrical Engineering*, pages 72–75. IEEE Computer Society, Washington, DC, USA, 2002.

[121] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *ACM PODC 2006 – Proceedings of the* 25th *Annual Symposium on Principles of Distributed Computing*, page 15. ACM, 2006.

[122] S. Kumar, V. S. Raghavan, and J. Deng. Medium access control protocols for ad hoc wireless networks: A survey. *Ad Hoc Networks*, 4(3):326–358, 2006.

[123] T. Labella and F. Dressler. A bio-inspired architecture for division of labour in sanets. In F. Dressler and I. Carreras, editors, *Advances in Biologically Inspired Information Systems*, volume 69 of *Studies in Computational Intelligence*, pages 211–230. Springer Berlin / Heidelberg, 2007.

[124] S. A. Lee. Firefly Inspired Distributed Graph Coloring Algorithms. In Hamid R. Arabnia and Youngsong Mun, editors, *Proceedings of PDPTA 2008 – International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 211–217. CSREA Press, 2008.

[125] S. A. Lee. k-Phase Oscillator Synchronization for Graph Coloring. *Mathematics in Computer Science*, 3(1):61–72, 2010.

[126] S. A. Lee and R. Lister. Experiments in the dynamics of phase coupled oscillators when applied to graph coloring. In *ACSC 2008 – Proceedings of the 31st Australasian Conference on Computer Science*, pages 83–89. Australian Computer Society, Inc., 2008.

[127] F. L. Lewis. *Wireless Sensor Networks*, pages 11–46. John Wiley & Sons, Inc., 2005.

[128] F. Li and I. Nikolaidis. On minimum-energy broadcasting in all-wireless networks. In *IEEE LCN 2001 – Proceedings of the Conference on Local Computer Networks*, pages 14–16. IEEE press, 2001.

[129] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In K. Deb, editor, *GECCO 2004 – Proceedings of the 6th Annual Conference in Genetic and Evolutionary Computation, LNCS 3102*, pages 105–116. Springer, 2004.

[130] W. Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *ACM MobiHoc 2002 – Proceedings of the 3rd International Symposium on Mobile Ad hoc Networking & Computing*, pages 112–122. ACM Press New York, NY, USA, 2002.

[131] W. H. Liao, Y. Kao, and C. M. Fan. Data aggregation in wireless sensor networks using ant colony algorithm. *Journal of Network and Computer Applications*, 31(4):387–401, 2008.

[132] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel. Delay efficient sleep scheduling in wireless sensor networks. In K. Makki and E. Knightly, editors, *IEEE INFOCOM 2005 – Proceedings of the* 24^{th} *International Conference on Computer Communications*, pages 2470–2481. IEEE, 2005.

[133] Z. Lü and J. K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[134] D. Lucarelli and I. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *Proceedings of SenSys 2004 – 2nd ACM Conference on Embedded Networked Sensor Systems*, pages 62–68. ACM, 2004.

[135] Q. Lv, X. Xia, and P. Qian. A parallel aco approach based on one pheromone matrix. In Marco Dorigo, Luca Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, editors, *Ant Colony Optimization and Swarm Intelligence*, volume 4150 of *Lecture Notes in Computer Science*, pages 332–339. Springer Berlin / Heidelberg, 2006.

[136] N. A. Lynch. *Distributed Algorithms.* Elsevier, 2009.

[137] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *Journal on Computing*, 20(2):302–316, 2008.

[138] E. Malaguti, M. Monaci, and P. Toth. An exact approach for the Vertex Coloring Problem. *Discrete Optimization*, 8(2):174 – 190, 2010.

[139] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

[140] E. Maneva, E. Mossel, and M. J. Wainwright. A new look at survey propagation and its generalizations. *Journal of the Association for Computing Machinery*, 54(4):1089–1098, 2007.

[141] K. Martinez, J. K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50–56, 2004.

[142] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204 – 210, 2004.

[143] D. Merkle, M. Middendorf, and A. Scheidler. Self-organized task allocation for computing systems with reconfigurable components. In *IPDPS 2006 – Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*. IEEE Press, 2006.

[144] K. L. Mills. A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(7):823–834, 2009.

[145] O. Miramontes. *Complexity and Behaviour in* Leptothorax *Ants*. CopIt ArXives, Washington, DC, 2007.

[146] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[147] G. Molina. *Optimization Techniques for Wireless Sensor Networks*. PhD thesis, Universidad de Málaga, 2010.

[148] T. Moscibroda and R. Wattenhofer. Coloring unstructured radio networks. *Distributed Computing*, 21(4):271–284, 2008.

[149] S. D. Muruganathan, D. C. Ma, R. I. Bhasin, and A. O. Fapojuwo. A centralized energy-efficient routing protocol for wireless sensor networks. *IEEE Communications Magazine*, 43(3):S8–13, 2005.

[150] A. Mutazono, M. Sugano, and M. Murata. Frog Call-Inspired Self-Organizing Anti-Phase Synchronization for Wireless Sensor Networks. In *INDS 09 – Proceedings of the* 2nd *International Workshop on Nonlinear Dynamics and Synchronization*, pages 81 – 88. IEEE Press, 2009.

[151] S. Okdem and D. Karaboga. Routing in wireless sensor networks using ant colony optimization. In *Proceedings of AHS 2006 – 1st NASA/ESA Conference on Adaptive Hardware and Systems*, pages 401–404. IEEE Press, 2006.

[152] P. Panagopoulou and P. Spirakis. A game theoretic approach for efficient graph coloring. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *ISAAC 2008 – Proceedings of the* 19th *International Symposium on Algorithms and Computation*, pages 183–195. Springer Berlin / Heidelberg, 2008.

[153] T. Pavlidis. *Biological oscillators: their mathematical analysis.* Academic Press, New York, 1973.

[154] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[155] C. S. Peskin. *Mathematical aspects of heart physiology.* Courant Institute of Mathematical Sciences, New York University, 1975.

[156] R. A. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, 1995.

[157] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *IPSN 2005 – Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 457–462. ACM Press, New York, NY, 2005.

[158] M. Rahimi, H. Shah, G. S. Sukhatme, J. Heideman, and D. Estrin. Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network. In *IEEE ICRA 2003 – Proceedings of the International Conference on Robotics and Automation*, volume 1, pages 19–24. IEEE press, Piscataway, NJ, 2003.

[159] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87 – Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 25–34, New York, NY, USA, 1987. ACM Press.

[160] T. J. Richer and T. Blackwell. The Lévy particle swarm. In *IEEE CEC 2006 – Proceedings of the 2006 Congress on Evolutionary Computation*, pages 808–815. IEEE Press, 2006.

[161] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer – the ARPSO. *Technical Report 2002-02*, 2002.

[162] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice Hall Englewood Cliffs, NJ, 1995.

[163] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computer Surveys*, 37:164–194, 2005.

[164] P. Santi. *Topology control in wireless ad hoc and sensor networks.* John Wiley & Sons, San Francisco, CA, 2005.

[165] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Effective methods for scalable and continuous media streaming on peer-to-peer networks. *European Transactions on Telecommunications*, 15(6):549–558, 2004.

[166] C. Savarese, J. M. Rabaey, and J. Beutel. Location in distributed ad-hoc wireless sensor networks. In *Proceedings of ICASSP 2001 – IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2037–2040, 2001.

[167] F. Schulz. Modeling sensor and ad hoc networks. In D. Wagner and R. Wattenhofer, editors, *Advanced Lectures – Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*, pages 21–36. Springer Verlag, Berlin, Germany, 2007.

[168] C. Schurgers. *Wireless Sensor Networks and Applications*, chapter Wakeup strategies in Wireless Sensor Networks, pages 195–217. Signals and Communication Technology. Springer Verlag, Berlin, Germany, 2008.

[169] M. Schwartz and N. Abramson. The AlohaNet-Surfing for Wireless Data. *Communications Magazine*, 47(12):21–25, 2009.

[170] R. C. Shah and J. M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *IEEE WCNC 2002 – Proceedings of the Wireless Communications and Networking Conference*, volume 1, pages 350–355. IEEE Press, Piscataway, NJ, 2002.

[171] S. Shakkottai, R. Srikant, and N. B. Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. *Ad Hoc Networks*, 3(6):702–716, 2005.

[172] http://shawn.sf.net.

[173] E. Shih, S. H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *Proceedings of SIGMOBILE 2001 – 7th Annual International Conference on Mobile Computing and Networking*, pages 272–287. ACM, 2001.

[174] A. Singh and W. N. Bhukya. A hybrid genetic algorithm for the minimum energy broadcast problem in wireless ad hoc networks. *Applied Soft Computing*, 11(1):667 – 674, 2011.

[175] G. Singh, S. Das, S. Gosavi, and S. Pujar. Ant colony algorithms for Steiner trees: an application to routing in sensor networks. *Recent Developments in Biologically Inspired Computing*, pages 181–206, 2004.

[176] H.M. Smith. Synchronous flashing of fireflies. *Science*, 82(2120):151, 1935.

[177] T. Stützle. Parallelization strategies for ant colony optimization. In *PPSN V – Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, LNCS 1498*, pages 722–731. Springer, 1998.

[178] T. Stützle and H. H. Hoos. MAX-MIN Ant system. *Future Generation Computer Systems*, 16(9):889–914, 2000.

[179] G. Theraulaz, E. Bonabeau, and J. N. Denuebourg. Response threshold reinforcements and division of labour in insect societies. *Proceedings: Biological Sciences*, 265(1393):327, 1998.

[180] J. Tillett, R. Rao, and F. Sahin. Cluster-head identification in ad hoc sensor networks using particle swarm optimization. In *Proceedings of ICPWC 2002 – IEEE International Conference on Personal Wireless Communications*, pages 201–205. IEEE Press, 2002.

[181] V. Torre. A theory of synchronization of heart pace-maker cells. *Journal of Theoretical Biology*, 61(1):55–71, 1976.

[182] R. D. Traub, R. Miles, and R. K. Wong. Model of the origin of rhythmic population oscillations in the hippocampal slice. *Science*, 243(4896):1319, 1989.

[183] M. Tubaishat and S. Madria. Sensor networks: an overview. *IEEE Potentials*, 22(2):20–23, 2003.

[184] C. Twomey, T. Stützle, M. Dorigo, M. Manfrin, and M. Birattari. An analysis of communication policies for homogeneous multi-colony aco algorithms. *Information Sciences*, 180(12):2390 – 2404, 2010.

[185] J. Van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *Proceedings of WSNA 2003 – 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19. ACM Press, 2003.

[186] K. Veeramachaneni, T. Peram, C. Mohan, and L. Osadciw. Optimization using particle swarms with near neighbor interactions. In *GECCO 2003 – Proceedings of the 5th Annual Conference in Genetic and Evolutionary Computation, LNCS 2723*, pages 110–121. Springer, 2003.

[187] C. M. Vigorito, D. Ganesan, and A. G. Barto. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *IEEE SECON 2007 – Proceedings of the 4th Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 21–30. IEEE Press, Piscataway, NJ, 2007.

[188] C. von der Malsburg. *Organic Computing*, chapter The Organic Future of Information Technology. Understanding Complex Systems. Springer Berlin, 2008.

[189] P. J. Wan, G. Calinescu, X. Y. Li, and O. Frieder. Minimum-energy broadcasting in static ad hoc wireless networks. *Wireless Networks*, 8(6):607–617, 2002.

[190] L. Wang and Y. Xiao. A survey of energy-efficient scheduling mechanisms in sensor networks. *Mobile Networks and Applications*, 11(5):723–740, 2006.

[191] X. Wang, S. Wang, and J. J. Ma. An improved co-evolutionary particle swarm optimization for wireless sensor networks with dynamic deployment. *Sensors*, 7(3):354–370, 2007.

[192] H. Weimerskirch, J. Martin, Y. Clerquin, P. Alexandre, and S. Jiraskova. Energy saving in flight formation. *Nature*, 413(6857):697–698, 2001.

[193] K. D. Wells. The social behaviour of anuran amphibians. *Animal Behaviour*, 25:666–693, 1977.

[194] A. M. Wenner. Sound production during the waggle dance of the honey bee. *Animal Behaviour*, 10(1-2):79 – 95, 1962.

[195] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. *Proceedings of INFOCOM 2000 – 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2:585–594, 2000.

[196] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. The energy efficiency of distributed algorithms for broadcasting in ad hoc networks. In G. L. Stüber et al., editor, *WPMC 2002 – Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications*, volume 2, pages 499–503. IEEE press, 2002.

[197] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. *Mobile Networks and Applications*, 7(6):481–492, 2002.

[198] Transistor count. Wikipedia. http://en.wikipedia.org/wiki/Transistor_count.

[199] E. O. Wilson. The relation between caste ratios and division of labor in the ant genus pheidole (hymenoptera: Formicidae). *Behavioral Ecology and Sociobiology*, 16:89–98, 1984.

[200] T. Wimalajeewa and S.K. Jayaweera. PSO for constrained optimization: optimal power scheduling for correlated data fusion in wireless sensor networks. In *Proceedings of PIMRC 2007 – IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications.*, pages 1–5. IEEE Press, 2007.

[201] A. T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 16(1):15–42, 1967.

[202] A. T. Winfree. *The timing of biological clocks.* Scientific American Books New York, 1987.

[203] A. T. Winfree. *The geometry of biological time.* Springer Verlag, 2001.

[204] S. Wolf and P. Merz. Evolutionary local search for the minimum energy broadcast problem. In C. Cotta and J. van Hemert, editors, *VOCOP 2008 – Proceedings of the 8th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 4972 of *Lecture Notes in Computer Science*, pages 61–72. Springer Verlag, Berlin, 2008.

[205] X. Wu, J. Cho, B. J. d'Auriol, and S. Lee. Mobility-assisted relocation for self-deployment in wireless sensor networks. *IEICE Transactions on Communications*, 90(8):2056–2069, 2007.

[206] X. Wu, Y. Niu, L. Shu, J. Cho, Y. Lee, and S. Lee. Relay shift based self-deployment for mobility limited sensor networks. In *Proceedings of UIC 2006 – 3rd International Conference on Ubiquitous Intelligence and Computing*, pages 556–564. Springer Berlin, 2006.

[207] W. Xiaoling, S. Lei, Y. Jie, X. Hui, J. Cho, and S. Lee. Swarm based sensor deployment optimization in ad hoc sensor networks. In *Proceedings of ICESS 2005 – 2nd International Conference on Embedded Software and Systems*, pages 533–541. Springer Berlin, 2005.

[208] X.F. Xie, W.J. Zhang, and Z.L. Yang. Dissipative particle swarm optimization. In *CEC 2002 – Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1456–1461. IEEE Press, 2002.

[209] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[210] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.

[211] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

[212] R. Zivan. Anytime local search for distributed constraint optimization. In *AAMAS 08 – Proceedings of the* 7^{th} *International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1449–1452. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008.