MONOGRAFIES DE L'INSTITUT D'INVESTIGACIÓ
EN INTEL·LIGÈNCIA ARTIFICIAL
Number 49

Bio-inspired Mechanisms for Self-organising Systems

Bio-inspired Mechanisms for Self-organising Systems

Jose Luis Fernandez-Marquez

Foreword by Josep Lluis Arcos

2012 Consell Superior d'Investigacions Científiques
Institut d'Investigació en Intel·ligència Artificial
Bellaterra, Catalonia, Spain.

Series Editor
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Foreword by
Josep Lluis Arcos
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

Volume Author
Jose Luis Fernandez-Marquez
Institut d'Investigació en Intel·ligència Artificial
Consell Superior d'Investigacions Científiques

ISBN: 978-84-00-09602-1
ISBN online: 978-84-00-09603-8
NIPO: 723-12-167-7
NIPO online: 723-12-168-2
DL: B.31910-2012

Ordering Information: Text orders should be addressed to the Library of the IIIA, Institut d'Investigació en Intel·ligència Artificial, Campus de la Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona, Spain.

To my family

Contents

Foreword **xiii**

Abstract **xv**

1 Introduction **1**
1.1 Motivation 1
1.2 Contributions 5
1.3 Book structure 7

2 Bio-inspired design patterns **9**
2.1 A Model to Describe Bio-Inspired Design Patterns 10
2.2 Pattern Scheme 13
2.3 Basic Patterns 15
2.3.1 Spreading Pattern 15
2.3.2 Replication Pattern 17
2.3.3 Aggregation Pattern 18
2.3.4 Evaporation Pattern 20
2.3.5 Repulsion Pattern 22
2.4 Composed Patterns 25
2.4.1 Gradient Pattern 25
2.4.2 Digital Pheromone Pattern 27
2.4.3 Gossip Pattern 29
2.5 Top Layer Patterns 30
2.5.1 Morphogenesis Pattern 30
2.5.2 Quorum Sensing Pattern 32
2.5.3 Chemotaxis Pattern 33
2.5.4 Flocking Pattern 35
2.5.5 Foraging Pattern 37
2.6 Summary 39

3 Dynamic Optimisation **41**
3.1 Background 42
3.1.1 Particle Swarm Optimisation 43
3.1.2 PSO in Dynamic Environments 44

3.1.3 PSO in Noisy functions . . . 45
3.2 Evaporation Mechanism . . . 46
3.2.1 Evaporation Mechanism . . . 47
3.2.2 mQSOE . . . 48
3.2.3 Dynamic Evaporation . . . 50
3.3 Experiments . . . 51
3.3.1 Experimental framework . . . 51
3.3.2 Determining the evaporation factor . . . 53
3.3.3 Subtraction versus multiplication as evaporation operator 54
3.3.4 Independence of peak heights and peak shifts . . . 55
3.3.5 mQSO versus mQSOE . . . 56
3.3.6 Filtering noise in mQSO . . . 57
3.3.7 Dynamic Evaporation . . . 59
3.4 Conclusions . . . 62

4 Hovering Information in Spatial Computing 67
4.1 Introduction . . . 68
4.2 Background . . . 69
4.3 Hovering Information Concept . . . 71
4.3.1 Hovering Information . . . 71
4.3.2 Anchor Areas . . . 71
4.3.3 Assumptions . . . 73
4.4 Hovering Information Algorithms . . . 73
4.4.1 Replication with Broadcast . . . 73
4.4.2 Replication with Attractor Point . . . 74
4.4.3 Cleaning . . . 74
4.4.4 Repulsion . . . 74
4.4.5 Broadcast Repulsion . . . 76
4.4.6 Attractor Point Repulsion . . . 78
4.5 Simulation Results . . . 78
4.5.1 Metrics . . . 79
4.5.2 Indoor Scenario . . . 81
4.5.3 Outdoor Scenario . . . 83
4.5.4 Analysis of algorithms . . . 83
4.6 Conclusions . . . 91

5 Detecting Diffuse Event Sources in Noisy WSN Environments 93
5.1 Introduction . . . 93
5.2 Related Work . . . 95
5.3 Sleep/Wake Modes . . . 96
5.4 Our Approach . . . 97
5.4.1 Sensors . . . 98
5.4.2 Mobile Agents . . . 99
5.5 Experiments . . . 101
5.5.1 Varying the number of sensors in WSN . . . 103
5.5.2 Quality of Convergence . . . 103

5.5.3 Varying the Noise Factor 104
5.5.4 Varying Local Exploration 104
5.5.5 Varying Global Exploration 105
5.5.6 The Exploration Cost 106
5.5.7 Tolerance to WSN failures 106
5.6 Conclusions 107

6 Conclusions and Future Work **109**
6.1 Publications related to this research 111

A Hovering Information **123**

List of Figures

2.1 Design Patterns 10
2.2 Relevant entities of the biological and computational models. . . 11
2.3 Model 12
2.4 Diffusion in science 23
2.5 Repulsion 24
2.6 Chemotaxis Pattern 35
2.7 Metric distance Model - Movements 37

3.1 mQSOE algorithm (mQSO extended with the evaporation mechanism) 49
3.2 Slow Adaptation Behavior 50
3.3 mQSODE algorithm (mQSO extended with dynamic evaporation equations) 52
3.4 Varying Evaporation Factor 54
3.5 Multiplicative versus subtractive evaporation 55
3.6 Evaporation performance when decreasing the peak height 56
3.7 mQSOE performance when changing severity 57
3.8 Comparing the offline error when introducing different noise thresholds 59
3.9 T1 - Noise free Environment 60
3.10 Average Maxima - Noise-free environment 61
3.11 Average Means - Noise-free environment 62
3.12 Average Minima - Noise-free environment 63
3.13 Standard Deviation - Noise-free environment 64
3.14 T1 - Noisy Environment 64
3.15 Average Minima - Noisy Environment 65
3.16 Average Maxima - Noisy Environment 65
3.17 Average Means - Noisy Environment 66
3.18 Standard Deviation - Noisy Environment 66

4.1 Amorphous Areas 72
4.2 Repulsion 75
4.3 Broadcast Repulsion Steps 77
4.4 Survivability and Accessibility - Indoor scenario 81
4.5 Survivability and Accessibility STD - Indoor scenario 82

4.6 Messages and Memory - Indoor scenario 82
4.7 Messages and Memory STD - Indoor scenario 83
4.8 Survivability and Accessibility - Outdoor scenario 83
4.9 Messages and Memory - Outdoor scenario 84
4.10 Accessibility and Memory - Scalability Indoor scenario 85
4.11 Messages - Scalability in Indoor scenario 85
4.12 Accessibility - Varying the repulsion interval 86
4.13 Memory and Messages - Varying the repulsion interval 86
4.14 Accessibility - steps 0 to 1000 - Indoor Scenario 88
4.15 Accessibility with zoom - Indoor Scenario 88
4.16 Accessibility - steps 0 to 1000 - Outdoor Scenario 89
4.17 Accessibility with zoom - Outdoor Scenario 89
4.18 Faults - Initialisation Phase 90

5.1 Low Power Listening (taken from [Na et al., 2008]) 97
5.2 Snapshot of a noisy scenario 102
5.3 Performance Results 104

6.1 Design Patterns 110

A.1 Broadcast, Attractor Point, Broadcast Repulsion and Attractor Point Repulsion 125
A.2 Attractor Point Repulsion - Convergence - Steps 126
A.3 Attractor Point Repulsion - nodes in area fail - Steps 127
A.4 Broadcast - convergence - Steps 128
A.5 Broadcast - Nodes in Area Fail - Steps 129
A.6 Amorphous Shape 5 130
A.7 Broadcast Simulation Steps in scenario '5' 131
A.8 Broadcast Repulsion Simulation Steps in scenario '5' 132

Foreword

This monograph reports several contributions on the design of self-organizing systems able to work in open, dynamic, and partially known environments. Specifically, the approach presented in this monograph explores the capabilities of bio-inspired mechanisms to deal with open scenarios. Bio-inspired mechanisms have been successfully applied to a variety of problems, but there was a lack of a uniform description framework able to systematize the existing mechanisms. I want also to stress that the research presented in this monograph is the result of an international collaboration with the University of London and the University of Geneva.

The work presented raises a number of challenging topics. First, it proposes a computational model to describe the existing literature in a uniform framework. Moreover, existing mechanisms are described as software design patterns. The use of design patterns provides a way to distinguish the building blocks of each mechanism, to identify those that are common in different mechanisms, and to facilitate their re-usability in new combinations. Moreover, this monograph reports the use of some of these building blocks to provide solutions to open problems such as the optimization in dynamic and noisy environments, algorithms for Spatial Computing, or sensor networks. Regarding optimization problems in dynamic and noisy environments, it contributes by improving the performance of an state of the art algorithm by introducing evaporation-based mechanisms. Concerning spatial computing, repulsion and replication mechanisms are proposed to improve algorithms for Hovering Information. Finally, spreading and repulsion mechanisms are applied in sensor networks to localize dynamically changing diffuse events.

Bellaterra, November 2012

Josep Lluis Arcos
IIIA - CSIC

Abstract

Nowadays, emergent technologies are providing new communication devices (e.g. mobile phones, PDS's, smart sensors, laptops) that form complex infrastructures that are not widely exploited due to their requirements such scalability, real-time responses, or failure tolerance. To deal with these features, a new software tendency is to provide entities in the system with autonomy and pro-activity and to increment the interaction between them. This betting on incrementing interaction and decentralising responsibilities over entities, so-called self-organisation, provides systems with better scalability, robustness, and reduces the computation requirements of each entity.

Biological systems have been adopted as a source of inspiration for Self-Organising systems. Since long, Self-organisation has been studied in biology showing a rich variety of collaborative behaviours, presenting interesting characteristic such as, scalability or failure tolerance. Nowadays, self-organising systems are applied to Multi-Agent Systems (MAS). A variety of self-organising, bio-inspired mechanisms have been applied in different domains, achieving results that go beyond traditional approaches. However, researchers usually apply these mechanisms in an ad-hoc manner. In this way, their interpretation, definition, boundary and implementation typically vary among the existing literature, thus preventing these mechanisms from being applied clearly and systematically to solve recurrent problems.

This book provides a complete catalog of bio-inspired mechanisms for self-organising systems. The mechanisms presented are described using a software design pattern structure identifying when and how to use each pattern and describing the relation between the different mechanisms. This catalog of mechanisms is a step forward to engineering self-organising systems providing a systematic way to develop self-organisation systems. The effectiveness and generalisation the mechanisms presented in this book are demonstrated in three different domains: Dynamic Optimisation, Spatial Computing, and Sensor networks.

Acknowledgements

I would like to especially thank Giovanna Di Marzo for being available when I needed some help and for assisting me during my research stays in London. I would also like to thank the following: Josep Lluis Arcos, my thesis supervisor, thanks a lot for guiding me to light and giving me the chance to make my thesis; Juan Antonio Rodriguez for his assistance throughout my research; all my colleagues at the IIIA for always collaborating and offering me their assistance; and last but not least, thanks also to Arminder Deol, for helping me with the biological issues and my English presentations.

Quiero agradecer también a toda mi familia, a quienes dedico este libro, especialmente a mis padres por su apoyo incondicional en todos mis planes y por estar ahí siempre que los necesito.

Especial agradecimiento a Martha por su paciencia y comprensión, por su ayuda y por haberme apoyado en todo momento.

Gracias a Laura por haberme realizado la portada, por su esfuerzo y por su paciencia intentando entender este trabajo. Gracias a Dani Polak por haber estado ahí dá tras día. Gracias a Nuria, por facilitarme acceso a artículos de investigación y ayudarme en la maquetación de este libro. Gracias a María por acompañarme en "el viaje", y por hacer feliz a las personas que le rodean. Gracias a Abraham, por sus valiosas opiniones y las interesantes conversaciones que hemos tenido en Londres.

Y finalmente agradecer a todos mis amigos y compañeros de escalada por los grandes momentos que hemos pasado juntos.

Chapter 1

Introduction

1.1 Motivation

Today's software applications increasingly rely on wireless devices – such as, PDAs, laptops, mobile phones or sensors – interacting each other on top of novel infrastructures (e.g. sensor networks, ad-hoc networks). As time goes by, the number of these devices in daily life increases continuously providing powerful infrastructures that are not widely exploited due to current software limitations (scalability, realtime requirements, or failure tolerance). Moreover, devices are becoming smaller, making possible new infrastructures such as smart sensor networks, smart materials, self-reconfiguring robots, collaborative unmanned aerial vehicles, or self-assembling nanostructures that were science fiction until now. These new structures allow the implementation of a wide range of new application and services, such as, rescue applications (search of survivor or ad-hoc communication infrastructures), disaster prevention (toxic clouds monitorisation, Earthquake and tsunami detection, or forest fires monitorisation) or act in favor of the environment (pollution sources detection, acoustic sources detection, oil's leaks on the oceans or traffic light control).

Such infrastructures are characterised by a great deal of *openness* (the number of entities in the system can change), *large scale* (thousands of nodes), *dynamism* (the network topologies are changing along the time), and *unpredictability* (environment changes, failures, or new entities joining), which cannot be coped with traditional or centralized approaches to system design or engineering.

To deal with these features, a new software tendency is to provide entities in the system with autonomy and pro-activity and to increment the interaction between them. This betting on incrementing interaction and decentralising responsibilities over those entities provides systems with better scalability, robustness, and reduces the computation requirements of each entity. Moreover, by increasing the interaction the system reduces the computation power required by each entity. Moreover, it also permits to reduce, even more, the devices'

sizes making possible new structures commented above. According to this new paradigm, so-called *Self-Organising Multi-Agent Systems*, the desired goal or behaviour emerges from the local interactions rather than from a centralised entity. Moreover, the intelligence resides in the interactions among the agents and not in each individual model (i.e. intelligence is collective). Thus, in spite of individual agents being simple (i.e. while they work individually), the agents can overtake the individual limitations and achieve complex tasks in a collaborative manner.

Self-Organisation in computer science is described as follows:

"A system described as self-organizing is one in which elements interact in order to achieve dynamically a global function or behavior." (Carlos Gershenson 2007)

Biological systems have been adopted as a source of inspiration for Self-Organising systems. Biological self-organising systems are present in the nature in a wide range of pattern formation processes, such as, fish swimming in coordinated schools, flocking behaviour in birds, patterns on seashells, synchronous fireflies flashing, etc.... In all of them the desired behaviour emerges from the coordination between the individuals using only local interactions, and local knowledge. A more general definition of Self-organising system is:

"Self-organization systems are physical and biological systems in which pattern and structure at the global level arise solely from interactions among the lower-level components of the system. The rules specifying interactions among the systems's components are executed using only local information, without reference to the global pattern." (Scott Camazine 2006)

Since long, Self-Organisation has been mainly discussed in physics and biology. Nowadays, self-organising systems are applied to Multi-Agent Systems (MAS) to achieve robustness, failure tolerance, scalability and adaptability. A variety of self-organising, bio-inspired mechanisms have been applied in different domains, achieving results that go beyond traditional approaches [Mamei et al., 2006]. However, researchers usually apply these mechanisms in an ad-hoc manner. In this way, their interpretation, definition, boundary, and implementation typically vary among the existing literature preventing these mechanisms from being applied clearly and systematically to solve recurrent problems.

Two main challenges appear when Self-Organising mechanisms are applied in MAS: (1) to control the emergent behaviour of the system from the local interactions and local knowledge of the environment avoiding uncontrolled emergent behaviours, and (2) engineering self-organising MAS in a systematic way. These

challenges are addressed by self-organising systems engineering.

The idea of engineering self-organising systems in computer science has attracted different researchers recently. Nagpal et al. [Nagpal, 2004] presented a set of biologically-inspired primitives that describe how organising principles from multi-cellular organisms may apply to large scale multi-agent systems. Those primitives have emerged as part of the Amorphous Computing project [Abelson et al., 2000a], which is focused on developing programming methodologies for systems composed of vast numbers of locally-interacting and identically programmed agents. That work was motivated by new emerging technologies such as MEMS (micro-electronic mechanical devices), which enable to create tiny computing and sensing elements that can be embedded into materials, structures or the environment. Some envisioned applications are: reconfigurable robots or structures composed of millions of identical modules that self-assemble into different shapes to achieve different tasks, smart environments where the sensors are embedded into the walls, or armies of ants-like robots that can achieve complex tasks in a collaborative way. That work was a first attempt towards assembling a catalog of primitives for large scale multi-agent control, where the desired behaviour emerges from the local interaction and coordination between agents. Moreover, the primitives are interesting from the application point of view and it was a step forward for engineering self-organising systems. However, those primitives are not presented together with an implementation process or by taking into consideration the different scenarios where the primitives can be applied. Thus, it is difficult to use them in a systematic way.

Mamei et al. [Mamei et al., 2006] presented a review on the state of the art of nature-inspired self-organising mechanisms in computer science and proposes a taxonomy to classify those self-organising mechanisms found in the literature. As previous works, the mechanisms are described together with the biological process they were inspired from. Moreover, they add the different domains that can be addressed with each mechanism. That work was the most complete catalog at that time, and certainly it was a step forward for engineering bio-inspired self-organising systems. Even when these descriptions can drive the implementation of the mechanisms, they are far away from being considered as set of mechanisms that can be applied in a systematic manner. However, that work motivates to go further and propose new questions:

- Which are the problems that those mechanisms can solve?
- What solution contributes each pattern?
- What are the main trade-offs to consider in the implementation?

To answer those questions and make the self-organising mechanisms applicable more systematically, different authors have focused on proposing descriptions of self-organising mechanisms under the form of software design patterns. Software design patterns was proposed by [Gamma et al., 1995], [Buschmann et al., 1996] and [Lind, 2003] for object oriented software. In software engineering, a pattern design describes a reusable solution for a

commonly occurring problem. Focusing on self-organising mechanisms, the idea of the design pattern structure makes it easy to identify the *problems* that each mechanism can solve, the specific *solution* that it brings, the *dynamics* among the entities and the *implementation*. Thus, self-organising systems can be designed more systematically.

In [Babaoglu et al., 2006], based on the existing software design patterns, they proposed a conceptual framework for transferring knowledge from biology to distributed computing. The mechanisms proposed in that paper are described using some problems that are solved with each pattern, the solution that each pattern provides and the biological process where it was inspired. Another relevant idea presented in [Babaoglu et al., 2006] paper is the distinction between basic patterns and composite patterns. However, they presented only one composite pattern (chemotaxis) and it was not really presented as a composition, it was an extension or application using one basic pattern.

Following the idea proposed by [Babaoglu et al., 2006], i.e. some patterns are composed by basic ones. Gardelli et al. [Gardelli et al., 2007] propose a set of basic design patterns for Self-Organising Multi-Agent Systems that can be combined to create a well known mechanism. The idea related with the basic patterns is to get a deeper understanding of the systems dynamics and also improve the controlability. The basic patterns properly combined produce new complex patterns and make easy to adapt the existing patterns to a new problems. As far as we know, [Gardelli et al., 2007] was the first to decompose a complex pattern into basic ones. Specifically, the Stigmergy Pattern was decomposed in evaporation, aggregation, and diffusion patterns. The stigmergy pattern is a coordination mechanism, based on indirect communication where the agents are mobile and communicate each other by modifing variables that are located in the environment. The name of stigmergy comes from the greek stigma (mark, sign) ergon (work, action) and contains the idea that the agents leave the marks in the environment that stimulate the behaviour of other agents. The stigmergy was first time used to describe the ants behaviour and is in the Ants Colony Optimisation (ACO) algorithm where the stigmergy has been more fruitful.

The decomposition process consists on identifying the internal mechanisms existing in a complex mechanism and on observing the contribution that these internal mechanisms provide to the complex one. In this example the stigmergy pattern is decomposed in evaporation, diffusion, and aggregation patterns. In [Gardelli et al., 2007], they keep the idea to use design patterns as conceptual framework to describe self-organising mechanisms.

The patterns proposed in [Gardelli et al., 2007] paper are all related with the ant colonies behaviour. The model provided presents too many constraints to be generalised and the examples of usage are not related to engineered self-organising systems. Thus, it is still an open question if basic patterns can be used isolated in self-organising systems and if basic patterns can be used to compose other self-organising patterns different than the pattern where they come from.

Another interesting work where a set of mechanisms are presented as design patterns is presented in [De Wolf and Holvoet, 2007], where they discussed an extended catalogue of mechanisms as design patterns for self-organising emergent applications. The patterns are presented in detail and can be used to systematically apply them for engineering self-organising systems. However, relations among the patterns are missed, i.e. the authors do not describe how patterns can be combined to create new patterns or adapted to tackle different problems. Moreover, in the paper one of the mechanisms proposed is applied to a case study, "A packet delivery service", but not compared with existing approaches.

Based on the set of mechanisms proposed in [Mamei et al., 2006], Sudeikat et al. [Sudeikat and Renz, 2008] discuss how intended MAS dynamics can be modeled and refined to decentralised MAS designs, proposing a systematic design procedure that is exemplified in a case study. Until now the only research focus to decompose complex patterns in basic ones was done by [Gardelli et al., 2007] as we commented before. The decomposition of complex patterns has not widely exploited and it is still an open issue. Thus, some of the questions that this book tries to answer are:

1. Can other complex patterns be decomposed?
2. Can the basic patterns be used isolated and make contributions in known problems?
3. Can the basic patterns be used to compose several complex patterns?

None of those papers have implemented the patterns to solve existing problems and have demonstrated the contribution of the patterns in different existing problems. Although, Self-Organising mechanisms described as design patterns provide useful descriptions that help to clarify the definitions of these mechanisms. These efforts are still fragmented: no clear catalogue of these patterns is provided, interpretations vary among authors, or the relations among patterns and their precise boundaries are not described. Moreover, they are far away for being applied systematically.

1.2 Contributions

In this book, we focus on modeling bio-inspired mechanisms for engineering self-organising systems by design patterns, arguing that some mechanisms can be described in terms of basic ones, i.e. fundamental mechanisms that can be used alone or as a part of more complex patterns. This decomposition of the self-organising mechanisms permits the identification of their exact boundaries, the relations they have with basic mechanisms, and to use them to compose new mechanisms or adapt them to acquire the desired behaviour. Every pattern is provided with a detailed description of the problem that it focus on, the corresponding solution that each pattern provides, and their behaviors (interaction dynamics and algorithmic behavior). Unlike the existing works, the basic patterns proposed in this book have demonstrated to be able to compose more than

one pattern allowing, in this way, to easily create new composed patterns in a systematic way. On the other hand, this book presents the generality of some of these patterns by applying them to different areas such as, dynamic optimisation, spatial computing, or sensor networks. The use of these basic patterns has improved the performance in the different areas compared with the existing techniques, demonstrating their general purpose.

Structuring mechanisms as design patterns allows a better support to create new mechanisms and to adapt existing ones to solve new problems. Moreover, this structure also allows a clear identification and separation of the mechanisms appropriate to each pattern.

To evaluate the performance of the proposed patterns, this book focuses on three different domains where the patterns are applied: (1) Dynamic and Noisy Optimisation, (2) Spatial Computing, and (3) Sensor Networks. The goal of this book is to propose general-purpose self-organising patterns and to show how these patterns can be applied to existing problems making contributions to different fields.

The main contributions of this book are the following:

- A set of complex bio-inspired self-organising mechanisms are presented as design patterns for engineering self-organising systems. These patterns are generalised and classified from their existing applications in the literature.
- Complex bio-inspired design patterns are decomposed in basic patterns. Basic patterns are presented also as design patterns that can be combined to create new patterns or to adapt the existing patterns to resolve new problems.
- We propose a model where bio-inspired patterns can be defined. The model covers a wide set of applications found in the literature.
- Some of the patterns proposed in this book have been implemented, making contribution in different communities. The communities and their contributions based on the pattern are:
  - Dynamic Optimisation: Adaptation of Particle Swarm Optimisation for working in noisy and dynamic environments. The existing Particle Swarm Optimisation algorithm has been extended with the Evaporation Pattern improving its performance in dynamic and noisy optimisation.
  - Spatial Computing: New set of algorithms for infrastructureless spatial storage of information. These new algorithms exploit mobile devices located in the environment. In these algorithms the agents decide when to replicate and collaborate between them to ensure the information coverage in a specified area.
  - Sensor Networks: Multi Mobile Agent approach to locate diffuse event sources using a Wireless Sensor Network infrastructure. Locating and tracking of diffuse event sources problem is proposed as a new

interesting problem with important application in real word domains. The mechanism used to tackle this new problem has demonstrated to achieve a good performance and presents good tolerance in front of sensor network failures and noisy and dynamic environments

1.3 Book structure

This book is organised as follows:

- Chapter 2: Bio-inspired design patterns.

  The goal of this chapter is to provide a complete catalog of bio-inspired mechanisms existing in the literature, their relation, their boundaries and the problem that each mechanism is focused on. Specifically, we analyze bio-inspired self-organising mechanisms existing in the literature and present these mechanisms as design patterns that can be applied in a systematic manner to engineer self-organising systems. The patterns are classified and the relations between them are identified. To describe the dynamics and interactions between the entities participating in each pattern, a computational model is proposed.

- Chapter 3: Dynamic Optimisation.

  In this chapter the Evaporation Pattern, proposed in Section 2.3.4, is incorporated to the Particle Swarm Optimisation (PSO) algorithm to deal with dynamic and noisy optimisation problems. The new algorithm proposed is compared to the existing version without evaporation. Benchmark results are provided, demonstrating that the evaporation mechanism allows the PSO algorithm to improve its performance in dynamic optimisation problems, mainly, when the fitness function is subjet to noise.

- Chapter 4: Hovering Information in Spatial Computing.

  In this chapter we define and analyse a collection of algorithms based on the Replication Pattern (Section 2.3.2) and the Repulsion Pattern (Section 2.3.5), for persistent storage of information at specific geographical zones exploiting the resources of mobile devices located in these areas. This proposed application is an example of problem difficult to trackle with traditional approaches due to the large number of nodes and real time requirements. Performed experiments and study of algorithms' parameter are analysed.

- Chapter 5: Detecting Dynamically Changing Diffuse Event Sources in Noisy WSN Environments.

  This chapter proposes and evaluates the use of the Chemotaxis Pattern applied in sensor networks for localizing dynamically changing diffuse events. Aimed in reducing the power consumption of the sensors, mobile agents collaborate to find diffuse event sources in dynamic and noise environments.

The goal is to locate the diffuse event sources using a minimum number of messages and sensors' reads. Conducted experiments demonstrate that the proposed approach is able to find the sources even in presence of noise, using an aceptable number of messages and sensors' read.

- Chapter 6: Conclusions and Future Work. Main conclusions and open research lines are detailed in this Chapter.

Chapter 2

Bio-inspired design patterns

The goal of this chapter is to analyze bio-inspired self-organising mechanisms existing in the literature and present these mechanisms as design patterns that can be applied in a systematic manner in engineering self-organising systems. Many bio-inspired self-organising mechanisms have been proposed in the literature making important contributions in decentralised and distributed application domains. However, the knowledge and experience on how and when to use them is spread across the corresponding literature, becoming very difficult to apply those mechanisms in a systematic manner.

In order to provide a way to use systematically those mechanisms, in this chapter the self-organising mechanisms are presented using a software design pattern structure. In software engineering, a design pattern describes a reusable solution for a commonly recurring problem. The design pattern structure mainly provides when to apply, how to apply and what are the results and trade-off of applying the pattern.

We analyze the relation between the mechanisms existing in the literature to understand how they work and facilitate their adaptation or extension to tackle new problems. As result, we have classified the patterns in three layers. In the bottom layer are the basic mechanisms that can be used individually or to compose complex patterns. In the middle layer there are the mechanisms composed by combinations of bottom layer mechanisms. Composed Patterns involve a single agent creating a distributed structure. The top layer contains complex patterns that show different ways to exploit the basic and composed mechanisms proposed in the bottom and middle layer. The exploitation is carried out by adding a set of policies (i.e. set of rules), thus, top level patterns can be described as a mechanism + a set of policies. In this book the policies are explained together with the pattern dynamics.

Figure 2.1 shows the different patterns described in this book and their relation. The arrows indicate how the patterns are composed. A discontinuous arrow indicates that it is optional (e.g. the Gradient Pattern can use evaporation, but the evaporation is not necessary to implement gradients).

In order to make easy to understand how the patterns work, we propose a

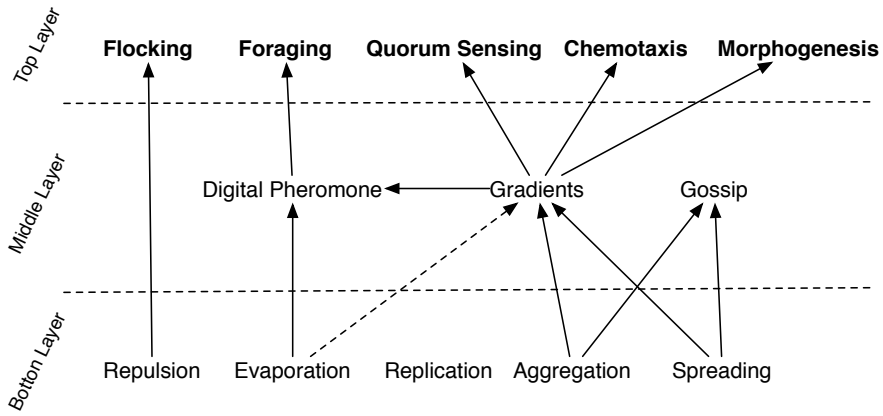


Figure 2.1: Design Patterns

model that covers a wide range of self-organising systems. Each pattern behaviour (dynamics and relation between the entities) is described based on the proposed model.

2.1 A Model to Describe Bio-Inspired Design Patterns

This section presents the computational model we use to describe the dynamics of the patterns and the relations between the different entities involved in each pattern. The model proposed is clearly inspired by biology, but adapted to engineering self-organising systems.

In biological systems, used as inspiration for self-organising mechanisms, two main entities can be observed: (1) the *organisms* that collaborate in the biological process (e.g. ants, fish, bees, cells, virus, etc.) and (2) the *environment*, a physical space where the organisms are located. The environment provides *resources* the organisms can use (e.g. food, shelter, raw material) and *events* that can be observed by the agents and can produce changes in the system (e.g. toxic clouds, stormy thunder, or a fire). Organisms can communicate with each other, sense from the environment and act over the environment. Moreover, organisms are autonomous and proactive and they have partial knowledge of the world. The environment is dynamic and acts over the resources and over the organisms (e.g. it can kill organisms, destroy resources, change the topology of the space where the organisms live, change the food location, remove food, add new food, etc.). The communication between the organisms can be direct (e.g. dolphins sending ultra-sounds through the water, or beavers emitting sounds to alert about a predator presence, etc.) or indirect using the environment to de-

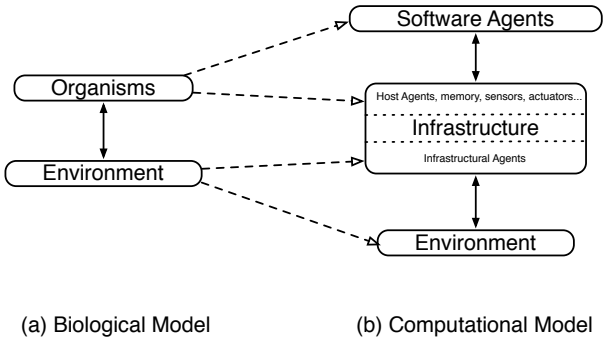


Figure 2.2: Relevant entities of the biological and computational models.

posit information that other agents can sense (e.g. pheromones in ant colonies, morphogens in the specialisation of cells, etc).

The biological model may be summarised in two layers: organisms and environment, see Figure 2.2 (a). To create a computational model inspired by the biological model, the computational model adds a new layer in addition to the two existing layers (i.e. Software Agents that corresponds to organisms layer in the biological model and environment), Figure 2.2(b). This new layer, called the *infrastructure* layer, is necessary because in an engineered system, the software agent must be hosted in a device with computational power that provides the agents with the ability to interact with the environment (i.e. sensing the environment through sensors or acting in the environment through actuators), and to communicate with other agents. Moreover, in our computational model the environment can not act over the information, thus the infrastructure contains also part of the environment implementation.

The entities proposed in the computational model are: (a) the *agents* which are pro-active software entities, (b) the *infrastructure*, which contains *hosts* with computational power, sensors and actuators; and (c) the *environment*, the space where the infrastructure is located. *Events* are phenomena of interest that appear in the environment that can be sensed by the agents exploting sensors in host's devices. Each agent needs a host to be executed, to communicate with other agents, to sense events or to act in the environment. Thus, the infrastructure provides the agents with all the necessary tools to simulate organisms' behavior and a place where information can be stored and possibly read by other agents. In most of the biological processes, the environment plays a key role, due to its ability to act over the entities present in the system (e.g. spreading and removing chemical signals in the environment). To tackle this ability, each host in the infrastructure has a software embedded in it, called *Infrastructural Agent (IA)*. Both IA's and agent's behaviours must be designed to follow self-organising patterns. IAs play an important role when agents can move freely over hosts. For instance, IAs may be responsible for managing information de-

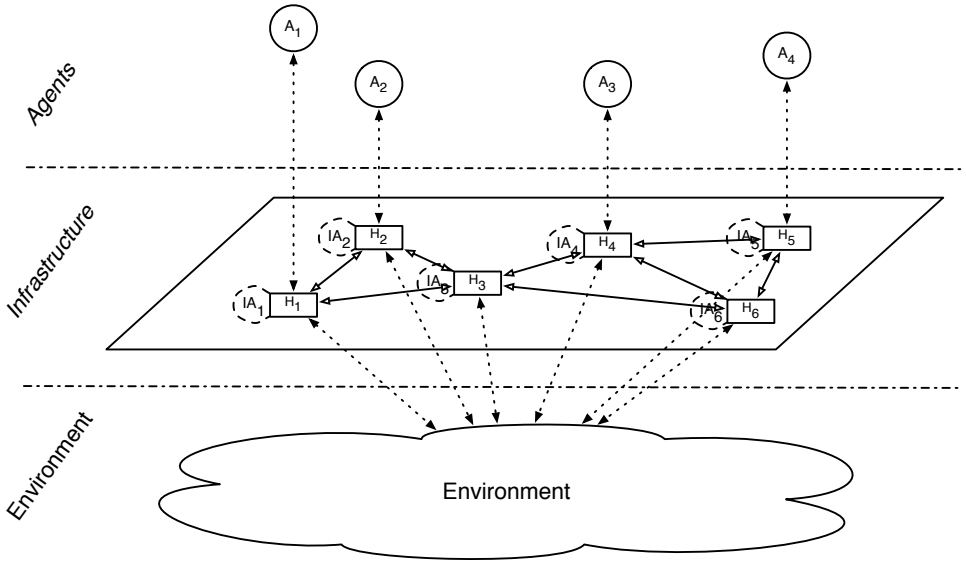


Figure 2.3: Model

posited in hosts by the agents or spread information over other hosts. In other cases, IA is a software embedded into a middleware providing built-in features (e.g. evaporation of digital pheromones).

Figure 2.3 shows the different layers of the computational model and their corresponding interactions. The top layer represents software agents in the system. Agents use the infrastructure layer to host themselves, communicate with each other, sense and act with the environment, and to deposit information that other agents can read. There are two variants in the model: when agents can move freely over the hosts (e.g. mobile agents) or when they are coupled to the host (e.g. swarm of robots). The separation between the agents layer and the infrastructure enables to cover a large variety of scenarios.

Table 2.1 summarises the different kinds of systems that can be modeled with this approach. On the one hand, software agents may be mobile or coupled with hosts. On the other hand the infrastructure may be fixed (i.e. stationary hosts) or mobile. Mobile hosts may be controlled by the agents (e.g. a robot) or not (e.g. PDA's movements under the control of its owner). This last scenario is typical of pervasive systems where several mobile devices, such as, PDAs, laptops or mobile phones are located in a common physical space (e.g a shopping mall, a museum, etc.), forming what is usually referred to as an opportunistic infrastructure, where the nodes move according to the movements of the user carrying them, and the agents freely jump from one node to another. An example of this architecture is the Hovering Information Project presented in Chapter 4. Sensor networks are a good example of systems where agents are mobile and hosts

Mobile Agents	Mobile Hosts	Known Application
no	no	Sensor Networks
no	yes - controlled	Swarm Robotics
yes	no	Sensor Networks
yes	yes - uncontrolled	Pervasive Scenarios

Table 2.1: Application Examples - different kinds of agents and hosts and corresponding known systems.

are not. Additionally, sensor networks are also representative of systems where not only hosts but also agents are static, as reported in [Vinyals et al., 2011].

To summarise, the entities used in the computational model are:

- **Agents**: autonomous and pro-active software entities running in a host.
- **Infrastructure**: the infrastructure is composed by a set of connected Hosts and Infrastructural Agents. A **Host** is an entity with computational power, communication capabilities, sensors and actuators. Hosts provide services to the agents. An **Infrastructural Agent** is an autonomous and pro-active entity, acting over the system at the infrastructure level. Infrastructural agents may be in charge of implementing those environmental behaviors present in nature, such as diffusion, evaporation, aggregation, etc.
- **Environment**: The Environment is the space where the Infrastructure is located. An **Event** is a phenomenon of interest that appears in the Environment and that may be sensed by the Agents using the sensors provided by the Hosts.

2.2 Pattern Scheme

In software engineering, a design pattern describes a reusable solution for a commonly recurring problem. Software design patterns were proposed by [Gamma et al., 1995], [Buschmann et al., 1996] and [Lind, 2003] for the development of object-oriented software. The goal of using a software design pattern scheme to describe self-organising mechanisms is to provide a systematic way to apply these mechanisms in engineering self-organising system. The description of a pattern and its relation with other patterns makes easy to decide when and how to apply a mechanism and how to adapt or compose the mechanism to deal with new problems.

Many bio-inspired self-organising mechanisms have been proposed in literature. These mechanisms allow to achieve a good performance when coping with the openness, unpredictability, and dynamism inherent to today's decentralised and distributed application domains. However, the knowledge and experience on how, when, and where to use them is spread across the corresponding

Name	The pattern's name.
Aliases	Alternative names used for the same pattern.
Problem	Which problem is solved by this pattern and situations where the pattern may be applied.
Solution	The way the pattern can solve the problems.
Typical Case	Domain where the mechanism is usually applied.
Inspiration	Biological process that inspires the design pattern.
Forces	Prerequisites for using the pattern and aspects of the problem that led the implementation including parameters (trade-offs).
Entities	Entities that participate in the pattern and their responsibilities.
Dynamics	How do the entities of the pattern collaborate to achieve the goal. Typical scenario describing the run-time behavior of the pattern.
Environment	Environment requirements to apply the pattern. Infrastructure needed.
Implementation/ Simulation	Hints of how the pattern could be implemented. Including parameters that must be tuned.
Known Uses	Examples of application where the pattern has been applied successfully.
Consequences	Effect on the overall system design.
Related Patterns	Reference to other patterns that solve similar problems, can be combined with this pattern or present conflicts with this pattern.

Table 2.2: Description fields

literature. This issue motivated us to focus on novel researches targeted at proposing a scheme similar to those presented in [De Wolf and Holvoet, 2007] and [Gardelli et al., 2007] to describe design patterns in self-organising systems.

To describe patterns, we use the scheme shown in Table 2.2, which is based on the scheme proposed in [Gardelli et al., 2007] and extended with some ideas from [De Wolf and Holvoet, 2007]. In addition to the Gardelli scheme we add *Inspiration*, the biological process where a mechanism is inspired, *Solution*, how the mechanism solves the problem, and *Typical case*, the domain where the mechanism has been usually applied. Adding the biological process that inspires the design pattern makes easy to understand the motivation and properties of the pattern.

2.3 Basic Patterns

Basic Patterns are used to compose more complex patterns on the middle layer (Section 2.4) and on the top layer pattern (Section 2.5). These patterns describe basic mechanisms that have been used isolated in the literature making important contributions to self-organising systems.

2.3.1 Spreading Pattern

The Spreading Pattern is a basic pattern for information diffusion/dissemination. The Spreading Pattern progressively sends information over the system using direct communication among agents. The spreading of information allows the agents to increment the global knowledge of the system by using only local interactions.

Aliases: spreading is also known as diffusion, dissemination, flooding, broadcast, epidemic spreading and propagation.

Problem: *agents' reasoning suffers from the lack of knowledge about the global system.*

Solution: a copy of the information (received or held by an agent) is sent to neighbours and propagated over the network from one node to another. Information spreads progressively over the system and reduces the lack of knowledge of the agents while keeping the constraint of the local interaction.

Typical Case: an agent wants to communicate an information to all the agents in the system.

Inspiration: spreading appears in important biological processes, such as, Quorum Sensing, Morphogenesis, Chemotaxis, and patterns of animal coats (e.g. stripes). For instance, when bacteria (e.g. salmonella [Bassler, 2002]) want to attack an organism, the bacteria wait until the density of bacteria is enough to defeat the defenses of the host organism. Each bacterium has only local sense of the system. However, bacteria need to have global knowledge in order to coordinate the attack. To increase the global knowledge of each bacterium, each one spreads a chemical substance called autoinducer. The concentration of autoinducer in the environment (i.e. host body) is an estimation of the density of bacteria. When the autoinducer goes beyond a threshold, the bacteria execute a coordinated attack. This example shows how sometimes to achieve a coordinated task, the individuals (in this case the bacteria) need information that come beyond the local knowledge of the individuals.

Forces: if spreading occurs with high frequency, the information spreads over the network quickly but the number of messages increases. A quick spread is desired when the environment is changing continuously and the agents must know the new values and adapt themselves. It may happen that the information is only interesting for agents close to the source. In that case, information spread only up to a determined number of hops, reducing the number of messages. Another way to reduce the number of messages is to determine the number of neighbouring nodes that receive the information. It has been demonstrated that

it is not necessary to send the information to all the neighbouring nodes in order to ensure that every node has received the information [Birman et al., 1999].

Entities-Dynamics-Environment: The entities involved in the spreading process are the hosts, agents, and infrastructural agents. The spreading process is initiated by an agent that first spreads the information. When information arrives to the neighbouring hosts, infrastructural agents re-send the information. The process continues even when all the hosts in the system have the information. Following our model, the spreading process is initiated by agents and the information is re-sent by infrastructural agents. However, the Spreading Pattern can be implemented in a system where agents are not mobile. In these cases, agents are embedded in the hosts and could be the agents responsible for re-sending the received information.

The dynamics is usually extended to avoid infinite loops and waste duplicated deliveries (e.g. when one agent receives the same information it has sent before, the agent does not resend that information).

Implementation: the most common technique used to spread information is Broadcast. Broadcast is a transmission technique where the information is sent from one node to multiple receiving nodes, avoiding to send the same information node by node. The broadcast implementation avoids the MAC identification phase. As a consequence, the messages necessary to establish communication are avoided, but the sender does not receive a confirmation from the receiving nodes (i.e. a sender can not guarantee that the receiving nodes have received the information). When the Broadcast is used iteratively (also called flooding) to spread information over a network, the Broadcast Stormy Problem [Tseng et al., 2002] appears. Thus, a straightforward broadcast by flooding will result in serious redundancy, contention, and collisions. Large scale networks where the radius of nodes overlap, emphasise the Broadcast Stormy Problem. It makes impossible to apply Broadcast in many cases. To solve the Broadcast Storm Problem, an optimised broadcast can be implemented. The optimised broadcast may be implemented following different schemes: probabilistic, counter-based, distance-based, location-based, and cluster-based schemes. The best performance is achieved with location-based which avoids redundant rebroadcasts without compromising reachability. Location-based schemes assume that all nodes known their position and a MAC identification phase. Even when the MAC identification phase increments the number of messages used for establishing communication, optimised Broadcast is recommended [Tseng et al., 2002].

Known uses: the Spreading mechanism has been applied to several applications: from Swarm motion coordination, to coordination in games, to problem optimisation, etc. For more information about known applications see related patterns and their known uses.

Consequences: when the Spreading Pattern is applied, the agents in the system receive information beyond their local context. There is an increment in the network load. This increment becomes extreme when the environment is very dynamic and agents must keep updated information as soon as possible.

Related Patterns: the Spreading Pattern is used by the Morphogene-

sis Pattern (Section 2.5.1), the Quorum Sensing Pattern (Section 2.5.2), the Chemotaxis Pattern (Section 2.5.3), the Gossip Pattern (Section 2.4.3), and the Gradient Pattern (Section 2.4.1).

2.3.2 Replication Pattern

The idea of replication is to give software agents the ability to replicate themselves. The replicas are exactly the same and execute the same functionality or offer the same services. The name replication pattern has been used before to define the action of copying information [Babaoglu et al., 2006, Gardelli et al., 2007]. For information replication, see Spreading Pattern (Section 2.3.1). We propose to restrict the use of the Replication Pattern to the copy of software agents. Thus, we differentiate between the spreading mechanism used to spread information over the environment or the replication mechanism used to increase the number of agents in the system according to system requirements. An example to clarify this difference is the following. Let A be an agent that is giving services in a MANET network. Due to a high number of requests, A can not answer the request dealing with system requirements. In that case, A can decide to replicate itself and use in this way the resources of another node in the MANET. Notice that this example is completely different from one agent inside a MANET informing all the nodes about a specific information, and then, the agent sends a broadcast message to all the nodes in the MANET.

Aliases: duplication, redundancy.

Problem: *the number of software agents necessary to achieve a given task is not known or must change along the time to satisfy the system requirements. The high dynamicity in the environment can kill agents and they need to be created again.*

Solution: Agent replication permits to create new agents in the system in a completely autonomous and decentralised manner. Also, the replication permits to create new agents in specific locations.

Typical cases: The replication pattern is used typically in networks and decentralised applications to improve the accessibility time and coverage in a scalable way, highly dynamic environments and ad-hoc infrastructures. Replication deals with accessibility problems and provides reliability and fault-tolerance.

Inspiration: Replication exists in nature as a mechanism to ensure survivability, robustness, and security. An example of survivability are viruses. Viruses use the cell structure to replicate themselves and to ensure their survivability. Human cells are an example of robustness, owning a local copy of the DNA for recovering from minor mutations. Other examples are epidemic spreading [Bailey, 1975] and the proliferation process in the immune system [Janeway et al., 2001].

Forces: Increasing the number of replicas in a system, the system becomes more robust and the accessibility time decreases. However, the memory required and the number of messages sent increase. Thus, the number of replicas in the system must be enough to satisfy the system requirements (accessibility, survivability, memory usage, etc.) using the minimum number of agents.

Entities-Dynamics-Environment: Entities involved in the replication process are agents and the hosts in the environment. The agents decide when to replicate and a replica consists in a copy of itself that moves to in one host in the environment. The copies are not modified during the replication process. Replication can be executed periodically or can be decided by each agent following local policies. Thus, local policies determine when and where replicating.

Implementation: Different kinds of replication algorithms exist in the literature. For cases of copies, a simple broadcast sends a copy to all the neighbours (Chapter 4). For serving requests with high performance, services are replicated under request and create replica close to the destination [Jamjoom et al., 1999], reducing latency. Basically, the idea is the same, that is, to create a copy of an agent among the network increasing the number of agents.

Known uses: Some applications where the replication has been used are:

- Hovering Information [Fernandez-Marquez et al., 2011], where mobile agents use the replication to offer a direct access to the information for all the nodes inside an area.
- In grid computing [Bell et al., 2003], where the local copy of services are replicated over the network to reduce the network latency.
- Replication of services in sensor networks to reduce the latency and the bandwidth [Jamjoom et al., 1999].

Consequences: When agents have the ability to replicate themselves, in most cases the system acquires the self-healing property, becoming more robust in front of hardware or software failures. On the other hand, replication makes possible to adapt the number of agents necessary to deal with the system's requirements.

Related Patterns: The replication pattern is not related to other patterns. It is a basic pattern that can be perfectly applied isolated or combined with other patterns.

2.3.3 Aggregation Pattern

The Aggregation Pattern is a basic pattern for information fusion. The dissemination of information in large scale MAS may produce network and memory overload, thus, the necessity of synthesizing information. This excess of information is deposited by the agents or taken from the environment using hosts' sensors. The Aggregation Pattern reduces the amount of information in the system and produces meaningful information. Aggregation Pattern was proposed in [Gardelli et al., 2007].

Alias: information fusion.

Problem: *excess of information produced by the agents may produce network and memory overloads. Information must be distributively processed to reduce the amount of information and to assess meaningful information.*

Solution: aggregation consists in locally applying an aggregation operator to process the information and to synthesize macro information. This operator can take many forms such as filtering, merging, aggregating, or transforming.

Inspiration: in the nature, the aggregation (sum) of ants' pheromones allows colony to find the shortest path to the food and to discard longer paths (e.g. two pheromone scents together create an attractive field bigger than a single pheromone scent). In nature, aggregation is a process performed by the environment. Even when there are no agents present in the system, the environment keeps doing the aggregation process.

Forces: aggregation applies on all the information available locally or only on part of that information. The parameter involved is the amount of information that is aggregated; it relates to the memory usage in the system. This pattern is not repetitive (i.e. there is no frequency involved). The pattern applies only once, even though it can be repeatedly invoked from within another pattern.

Entities-Dynamics-Environment: aggregation is executed either by agents or by infrastructural agents. In both cases, the agents aggregate the information that they access locally. The information comes from the environment or from other agents. Information that comes from the environment is typically read by sensors (e.g. temperature, or humidity). According to the model presented in Section 2.1, aggregation is executed by an agent that receives information from the host where the agent is residing. Such host is either a sensor reading information from the environment or a communication device receiving information from neighbouring hosts. The aggregation may be applied by any agent that receives information independently of the underlying infrastructure The aggregation process is not repetitive and finishes when one agent executes the aggregation function.

Implementation: available information takes the form of a stream of events. Aggregation of information can take various forms: from a simple operator (e.g. max, min, or avg) like in ACO, to more complex operators (e.g. Kohonen Self-organising Map to aggregate sensor data in clusters [Lee and Chung, 2005]). Aggregation operators are classified into four different groups [Chen and Kotz, 2002]:

(1) *Filter*: operator that selects a subset of the received events (e.g. a sensor taking 10 measures per second, but the application processes only 1 per second)

(2) *Transformer*: operators changing the type of the information received (e.g. GPS coordinates transform into countries where the positions are located)

(3) *Merger*: operators that unify all information received and output the information received as a single piece of information (e.g. the input is the position of many sensors and the output is the corresponding tuple of positions);

(4) *Aggregator*: this operator applies a specific operation (e.g. max, min, or avg) to one or more incoming inputs; input and output types may be different.

Known uses: aggregation has been used in ACO algorithms [Dorigo and Di Caro, 1999], where aggregation is used to create higher concentrations when two or more pheromones are close to each other. In [Parunak et al., 2002] the aggregation is also used in digital pheromones for autonomous coordination of swarming UAVs. Moreover, aggregation has been used in information fusion, which studies how to aggregate individual belief bases into a collective one [Grégoire and Konieczny, 2006], or for truth-tracking in MAS [Pigozzi and Hartmann, 2007].

Consequences: aggregation increases the efficiency in networks by reducing the number of messages and thus, increasing battery life. Moreover, aggregation provides a mechanism to extract macro-information in large-scale systems, such as extracting meaningful information from data reads obtained from many sensors. Then, the amount of memory used by the system is reduced.

Related Patterns: The aggregation pattern is used by the Digital Pheromone Pattern (Section 2.4.2) and the Gossip Pattern (section 2.4.3). Its combination with the Evaporation Pattern and the Spreading Pattern has been fruitful in a wide range of different applications.

2.3.4 Evaporation Pattern

Evaporation is a basic pattern to reduce the relevance of information along time. Thus, recent information becomes more relevant than information processed time ago. Evaporation deals with dynamic environments where the information used by agents can become outdated. In real world scenarios, the information usually changes along time and its detection or prediction is usually costly or even impossible. Thus, when agents have to modify their behaviours taking into account information from the environment, information gathered recently must be more relevant than information gathered a long time ago.

Evaporation was proposed as a design pattern for self-organising multi-agent systems in [Gardelli et al., 2007]. Prior to its introduction as a pattern in [Gardelli et al., 2007], evaporation has been used in different applications, usually related to Ant Colony Optimisation [Dorigo, 1992].

Aliases: Penalisation, degradation, decay, depletion.

Problem: *Outdated information can not be detected or its detection involves a cost that needs to be avoided.*

Solution: Evaporation is a mechanism to periodically reduce the relevance of information along time. Thus, recent information becomes more relevant than older information.

Typical cases: The idea of evaporation is related to decision-making in dynamic environments. Typical cases involve agents that are driving their behaviour by taking into account information from dynamic environments. The information taken may become outdated and may misguide the agent's decision making. Evaporation permits to take into account information acquired in the past, but its relevance in the decision making process is lower than information received recently.

Inspiration: Evaporation is present in the nature. For instance, in ant colonies [Deneubourg et al., 1983], when ants deposit pheromones in the environment, these pheromones attract other ants and drive their movements from the nest to the food and vice-versa. Evaporation acts over the pheromones reducing their concentration along the time until they disappear. This mechanism allows the ants to find the shortest path to the food, even when environment changes occur (such as, new food locations or obstacles in the path). Ants can find the new shortest paths by forgetting the old paths. As time goes by, evaporation decreases the intensity of the pheromones in the environment. Thus, pheromones added recently are more relevant for the ants decision making. That is, pheromones deposited recently are more attractive than pheromones deposited a long time ago.

Forces: Evaporation is controlled by the evaporation factor parameter (i.e. how much the information is evaporated) and the evaporation frequency parameter (i.e. frequency of evaporation execution). Their influence is as follows:

- A higher evaporation factor leads to a faster evaporation of the information relevance. However, the same effect could be also achieved by increasing the evaporation frequency.
- The evaporation factor and evaporation frequency must be set accordingly to the dynamics of the environment. Then, if the evaporation is too fast, we may lose information. However, if the evaporation is too slow the information may become outdated and misguide the agents' behaviour. Regarding memory usage, a higher evaporation factor releases memory, but also reduces the information available in the system.
- Exploration versus exploitation: When the evaporation is applied to collaborative search or optimisation algorithms, changing the evaporation factor the system varies the balance between exploration and exploitation. A high evaporation rate reduces the agents' knowledge about the environment, increasing the exploration, and producing fast adaptation to environment changes due to the higher exploration. However, a higher evaporation factor decreases the performance when no environment changes occur due to an excess of exploration that is not required.

Entities-Dynamics-Environment: Evaporation can be applied to any information present in the system. Periodically the relevance of this information decays over time. We distinguish two cases. In the first case, there is only one entity, an agent that encapsulates the information and decays the relevance itself. In the second case, the information is deposited by one agent in one host and infrastructural agents interact with the host to decay the information's relevance. There are no specific infrastructural requirements for using the evaporation mechanism.

Implementation: The Evaporation Pattern is executed by agents that need to update the relevance of its internal information, or by infrastructural agents that change the relevance of the information deposited in the environment. In

both cases the evaporation is directly applied to quantifiable parameters or, if the parameter is not quantifiable, to a quantifiable attribute that indicates its relevance. Then, evaporation is applied to the relevance attribute, i.e, the relevance of the unquantifiable parameter depends on its relevance attribute. The evaporation is applied periodically.

As it was explained before, when the dynamics of the environment is unknown (i.e. frequency of changes), it is difficult to decide the best evaporation factor or term. In Chapter 3 we propose a dynamic evaporation rate for dynamic problem optimisation using a multi-swarm particle optimisation approach. Each particle in the swarm adjusts its evaporation factor at time t depending on the local knowledge of the environment. The dynamic evaporation rate helps to tune the evaporation parameter when no knowledge about the environment dynamicity is known a priori.

Known uses: Evaporation has been used mainly in Dynamic Optimisation Problems. Examples of algorithms that use evaporation are Ant Colony Optimisation (ACO) [Dorigo and Di Caro, 1999], QSOE (Quantum Swarm Optimisation Evaporation) presented in Chapter 3, and [Weyns et al., 2007] where evaporation is performed using a parameter called freshness associated to the information.

Consequences: Evaporation enables adaptation to environment changes. However, the use of evaporation in static scenarios may lead to a decrease of performance, due to the loss of information associated with this mechanism. In static environments, all the information taken is useful and it has no sense its evaporation because it never becomes outdated. The Evaporation Pattern adds to the system the ability to self-adapt to environment changes and increases the tolerance to noise, as shown in Chapter 3.

Related Patterns: The Evaporation Pattern is used by higher level patterns, such as, the Digital Pheromone Pattern (Section 2.4.2) or the Gradient Pattern (Section 2.4.1).

2.3.5 Repulsion Pattern

The Repulsion Pattern is a basic pattern for motion coordination in large scale MAS. The Repulsion Pattern enables the agents to get a uniform distribution in a specific area or to avoid collisions between them. Agents can adapt their position when the desired area changes or when some nodes disappear.

Problem: *agents' movements have to be coordinated in a decentralised manner to achieve a uniform distribution and to avoid collisions between them.*

Solution: The Repulsion pattern creates a repulsion vector that guides agents to move from a high concentration of agents to regions with lower concentration. Thus, after few iterations agents achieve a more uniform distribution in the environment.

Typical Case: To create a uniform distribution inside a required area, even when the area is moving or is changing the shape through time, i.e. pattern formations.

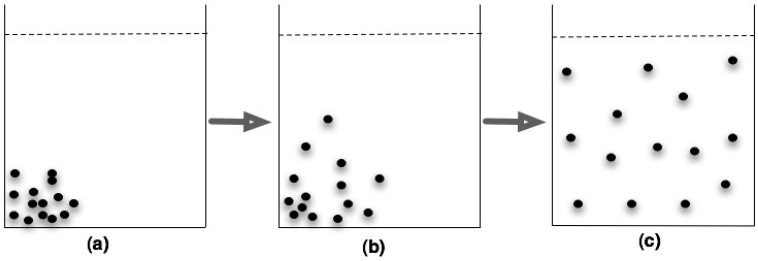


Figure 2.4: Diffusion in science

Inspiration: The repulsion mechanism appears in a wide range of biological self-organising processes, such as, the diffusion process in physical science, the flocking of birds or schools of fishes. The diffusion process, for example, describes the spread of particles through random motion from regions of higher concentration to regions of lower concentration. For instance, Figure 2.4 represents the different steps of the diffusion process. First, a concentration of ink is deposited in the glass of water, step (a). We observe the initial state where the particles are all around in one corner of the glass. The corner with the particles, therefore, contains a higher concentration of ink's particles. Second, the particles begin to move in the diffusion process, from regions of higher concentration to regions of lower concentration, step (b). The concentration of the particles in this step is higher the closer the particles are to the corner. Finally, we observe how the diffusion process has randomly moved around all the particles inside the water, producing a uniform random distribution of the particles. At this point the different ink's concentrations disappear. Inside a container, the particles reach a uniform distribution after the diffusion process. However, in an open space, the diffusion process spreads the particles until the concentration is so low that it is considered negligible. As Figure 2.4 shows, the diffusion process finishes when the system reaches an even concentration. That is, when the concentration gradient becomes zero. After that, particles reach a uniform distribution. The Repulsion Pattern focuses on getting a uniform distribution, while the Gradient Pattern (Section 2.4.1) is focused on the dissemination of the information through the environment. Even when both of them are inspired by the same biological process, the applications and implementations are completely different. In [Cheng et al., 2005] the repulsion mechanism was presented as inspired by gas theory. The gas theory also produces a diffusion where the time to reach a uniform concentration is shorter than that of the diffusion process.

Forces: The main parameters involved in the repulsion pattern are the repulsion frequency (i.e. how frequent the repulsion is applied) and the repulsion radius (i.e. how strong the repulsion is). The increment of the repulsion frequency involves a faster spreading of the agents and faster adaptation when the formation (or area) desired changes. However, it increases the number of messages, because the repulsion pattern requires information about the position

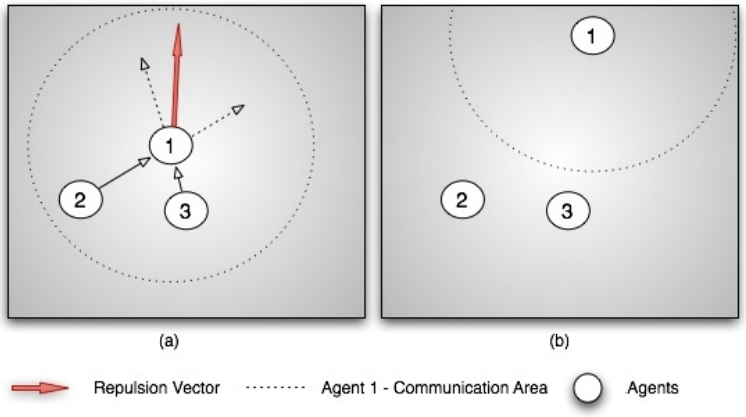


Figure 2.5: Repulsion

of neighbours. The repulsion radius should be limited to the communication range of agents, because it makes no sense to move to one location where the concentration of agents is unknown. Thus, the movement of one agent in each repulsion step must be limited to its communication range.

Entities-Dynamic-Environment: The repulsion can be applied to systems whose agents are fixed in hosts (e.g. robotic swarms) or to software agents that freely move in a network composed by hosts. In both cases the dynamics between them is the same. When repulsion is applied, the agent that executes the repulsion sends a position request to all the neighbouring agents. After the agent receives neighbours' replies, it calculates the desired position and moves to that position. When the environment is not continuous, as in the mobile agents case, the agent moves to the host closest to the desired position. In this case the position request must be sent also to the hosts. To apply the Repulsion Pattern, agents should know their positions.

Implementation: To reach a uniform distribution, the repulsion pattern uses an equation that calculates a repulsion vector between the particles that is inversely proportional to the distance between them. The repulsion equation is implemented as follows: Let R be the repulsive radius; d_i the distance between a given node and neighbouring node i; $\vec{p}$ the position of the given node and $\vec{p_i}$ the position of the neighbouring node i. Then, the position p and the movement vector $\vec{m}$ are given by:

$$p_{t+1} = p_t + \vec{m} \tag{2.1}$$

$$\vec{m} = \sum_i \frac{\vec{p} - \vec{p_i}}{d_i}(R - d_i) \tag{2.2}$$

Figure 4.2 shows how agent 1 is repelled by agents 2 and 3 when it applies

the repulsion mechanism. In Figure 4.2(a) the agent 1 executes the equation 2.2 to create the repulsion vector. In Figure 4.2(b) the agent 1 moves by following the repulsion vector. Notice that Repulsion Pattern can be only implemented when agents directly communicate.

Known uses: The repulsion pattern has not been proposed before. However, several applications have used the repulsion mechanism, such as swarm robotics for pattern formation [Cheng et al., 2005], where the system achieves shape formation by simultaneously allowing agents to disperse within the defined 2D shape. In Particle Swarm Optimisation (PSO) repulsion coordinated the position of explorer particles in a multi-swarm approach [Fernandez-Marquez and Arcos, 2009b]. The Hovering Information Project (Chapter 4) uses repulsion to coordinate the position of pieces of information to ensure the accessibility to this information using minimum memory.

Consequences: Repulsion does not involve replication, i.e. during the repulsion process no new agents are created. Repulsion is a continuous process that produces a uniform distribution of the agents in the system. Even when the agents are uniformly distributed in the environment, the repulsion mechanism continues working, producing a self-adaptation process when the number of agents change (i.e. self-repairing formation in swarms of robots) or environment changes occur.

Related Patterns: The repulsion mechanism is used by the Flocking Pattern (Section 2.5.4).

2.4 Composed Patterns

From the composition of some low level patterns, several mechanisms are analyzed in this section. Composed patterns can be used isolated or extended by other high level patterns. Low level patterns can be easily combined in many ways to deal with new problems. However, in this section we present those that have been widely used in the literature.

2.4.1 Gradient Pattern

The Gradient Pattern is an extension of the Spreading Pattern where the information is propagated in such a way that it provides an additional information about the sender's distance: either a distance attribute is added to the information; or the value of the information is modified such that it reflects its concentration: higher values (i.e. higher concentration) meaning the sender is closer, such as in ants' pheromone. Additionally, the Gradient Pattern uses the Aggregation Pattern to merge different gradients created by different agents. Here again, different cases apply: either only the information with the shortest distance to the sender is kept, or the concentration of the information increases.

Aliases: The Gradient Pattern is a particular kind of computational fields.

Problem: *Analogously to the Spreading Pattern (Section 2.3.1), the Gradient Pattern tackles the lack of global knowledge to estimate the consequences of*

the actions performed by agents beyond their communication range. Moreover, Gradient Pattern deals with the problem of knowing the information source location/distance.

Solution: Information spreads from the location it was initially deposited and aggregates when it meets other information. During spreading, additional information about the sender's distance and direction is provided: through a distance value (incremented or decremented); or by modifying the information to represent its concentration (concentration is lower when information is further away from the sender). When one agent receives the gradient information, it also knows the direction and the distance (i.e. hops' number or concentration value) where the information comes from. During the aggregation process, an aggregator operator keeps only the pair $(information, distance)$ with the highest (or lowest) *distance* value, or it modifies the concentration.

Typical Case: Coordination of agents' behaviour depending on the information of other agents (e.g. position, direction), including agents beyond their communication range.

Inspiration: Gradients appear in a lot of biological processes. The most known are the quorum sensing, the morphogenesis and the chemotaxis processes. In all of these processes, gradients permit to communicate biological entities (e.g. cells, bacteria, etc.) through keeping the constraint of local interaction. For instance, in the bacteria quorum sensing, the bacteria hosted by one organism wait to achieve a sufficient number of bacteria to attack together. The way these bacteria communicate each other is using a substance called autoinducer that creates a gradient. More details about these biological processes can be found in the Quorum Sensing Pattern (Section 2.5.2), Morphogenesis Pattern (Section 2.5.1) and Chemotaxis Pattern (Section 2.5.3).

Forces: High frequencies of gradient update involve a network overload. However, the adaptation to environmental changes is faster. Lower updating frequencies reduce the network overload, but can produce wrong values when environment changes occur. There is also a trade-off between the diffusion radius (number of hops) and the load in the network. A higher diffusion radius brings information further away from its source, providing a guidance also to distant agents. However, it involves an increment in the load and may overwhelm the network [Beal, 2009].

Entities-Dynamic-Environment: The entities that act in the Gradient Pattern are agents, hosts and infrastructural agents. Analogously to the Spreading Pattern, when a gradient is created the agent spreads the information to its neighbours (i,e. hosts controlled by agents or infrastructural agents). We distinguish two: (1) the neighbours forwarding the received information modifying the distance attribute by incrementing or decrementing its value, (2) the neighbours forwarding the aggregation of multiple gradients that are locally present.

Implementation: agents start the process by sending information (with a counter added equal to zero) to all neighbours (i.e. hosts controlled by agents or infrastructural agents). When a host receives information, the infrastructural agent increments the counter embedded into the information by one and

it forwards the information again to all the neighbours.

When a host receives the same information repeated, only one of the information copies received, stay stored in the host and is forwarded (usually the information with the minimum counter). Thus, the value of the counter increases as one moves away from the source agent (reflecting the distance to the source). Taking into account that the information about the gradient stay in the network, stored in the hosts, the number of hops can provide different information.

Notice that this pattern can be applied without hosts and infrastructural agents, just with agents (e.g swarms of robots). Moreover, this basic implementation is not self-healing. Thus, if a gradient source disappears the gradients are not automatically updated. To solve this problem, the Gradient Pattern can be combined with the Evaporation Pattern. This combination is known as active gradients. In actives gradients, evaporation decreases the values of gradients along the time. Then, changes in the networkv(either topology or agents' movements) are prevented. However, the number of messages in the network increases. Other self-healing variants are proposed in [Beal et al., 2008, Viroli et al., 2011].

Known uses: Gradient Pattern has been used in problems, such as, coordination of swarm of robots, coordination of agents in video games, or routing in sensor networks. The gradient is used in higher level patterns, such as, Morphogenesis Pattern, Chemotaxis Pattern or Quorum Sensing Pattern. Thus, more details about known uses and their referenced papers are provided in these sections.

Consequences: The Gradient Pattern adds extra information (counter) to the information. This extra information can be used to limit the number of hops the information reaches.

Related Patterns: The Gradient Pattern is an extension of the Spreading Pattern and at the same time is used from higher level patterns, such as the Morphogenesis Pattern (Section 2.5.1), the Chemotaxis Pattern (Section 2.5.3) and the Quorum Sensing Pattern (Section 2.5.2).

2.4.2 Digital Pheromone Pattern

The Digital Pheromone Pattern is a swarm coordination mechanism based on indirect communication. In this pattern, agents deposit digital pheromones in hosts. A digital pheromone is a mark, that is spread over the environment using the infrastructure. Then, other agents beyond the communication range can receive the information generated by digital pheromones. Digital pheromones are stored in the hosts and stay alive even when agents that deposited digital pheromones disappear. Digital pheromones can be either identic likewise the Ant Colony Optimisation Algorithm [Dorigo and Di Caro, 1999], or specialised for a specific task, likewise swarming vehicle control [Sauter et al., 2005].

Problem: *Coordination of agents in large scale environments using indirect communication.*

Aliases: The Digital Pheromone Pattern is also known as Stigmergy or Pheromone.

Solution: Digital Pheromone provides a way to coordinate agents' behaviours using indirect communication in high dynamic environments. It also provides simplicity, scalability, and robustness [Sauter et al., 2005].

Typical Case: Swarm robotics where decisions about movements are an important function (e.g. collaborative search).

Inspiration: The Digital Pheromone Pattern takes inspiration from ant colonies. Ant colonies are able to find food sources using local interactions and indirect communication based on pheromones. The colonies present interesting characteristics desired in computing systems. On the one hand, colonies are able to adapt to environment changes (such as, new obstacles, new food sources, food sources that become empty, etc.). On the other hand, colonies are failure tolerant, in the sense that, even when a high percentage of ants die, colonies are able to re-organise them-selves and ensure their survival.

Forces: The implementation of the Digital Pheromone Pattern involves the implementation of the Aggregation, Spreading, and Evaporation pattern. Thus, the main forces to consider are the following:

Regarding the evaporation, how much evaporation is applied at each iteration and how frequent the evaporation iteration is applied. See Evaporation Pattern in (Section 2.3.4) for more details.

The Gradient Pattern is composed by the Aggregation and Spreading Patterns. Thus, the higher frequently pheromones are spread the higher bandwidth is used. Also, spreading longer pheromones more information is received by the agents and also more memory and bandwidth is consumed.

Tuning the parameters depends on the application and may be different for each digital pheromone. For example, the parameters of a pheromone representing a plane's position are different from those that represent an enemy base. Thus, pheromones representing the plane positions would evaporate faster than the pheromones representing enemy bases.

Entities-Dynamic-Environment: Agents are the only entities that can deposit pheromones. Pheromones are deposited in hosts and infrastructural agents apply spreading, aggregation, and evaporation mechanisms. Thus, pheromones are spread though the network, aggregated in each host when two or more pheromones' information arrive, and evaporated along the time until disappear. Before a pheromone disappears due to the evaporation mechanism, it is available in the hosts and can be looked up by agents directly connected.

Implementation: Digital Pheromones are usually implemented by using multiplicative static evaporation (i.e. the same evaporation factor is applied periodically over the pheromone's information). Independently of the patterns used to implement the Digital Pheromone Pattern, pheromones can be deposited in hosts, (i.e. following the proposed model), simulated by software [Sauter et al., 2005], or implemented using RFID sensors [Mamei and Zambonelli, 2007].

Known uses: Digital Pheromones have been used mainly in autonomous coordination of swarming UAVs [Parunak et al., 2002, Sauter et al., 2005]. More-

over, applications of digital pheromones can be found in the Ant Foraging Pattern description (Section 2.5.5).

Consequences: As reported in [Sauter et al., 2005], the implementation of Digital Pheromones for swarm coordination provides to the system: (1) Simplicity: compared with the logic necessary in a centralised approach, (2) Scalability: the Digital Pheromones work in a totally decentralised manner, i.e. applicable in large scale MAS, and (3) Robustness: due to decentralisation and the continuous self-organising process the digital pheromones provide, the agents may fail and the system continues its execution without problems.

Related Patterns: The Digital Pheromone Pattern is composed by the Evaporation Pattern (Section 2.3.4) and the Gradient Pattern (Section 2.4.1), which is composed by the Aggregation Pattern (Section 2.3.3) and the Spreading Pattern (Section 2.3.1). Moreover, the Digital Pheromone pattern is exploited by the top-layer Foraging Pattern (Section 2.5.5).

2.4.3 Gossip Pattern

The goal of the Gossip Pattern is to obtain an agreement about the value of some parameters in the system in a decentralised way. All the agents in the system collaborate to progressively reach this agreement: all of them contribute with their knowledge by aggregating their knowledge with the others' knowledge and by spreading this aggregated knowledge. Gossip was proposed as an Amorphous computing primitive mechanism by Abelson et al. [Abelson et al., 2000b]. Gossip is also known as epidemic communication.

Problem: *Agents need to reach an agreement with only local perception and in a decentralised way, in large scale MAS.*

Solution: The gossip mechanism combines the aggregation mechanism (Section 2.3.3) with the spreading mechanism (Section 2.3.1) to progressively reach an agreement taking into account the information of all the agents in the system. Information is spread to neighbours, where it is aggregated with local information. Aggregated information is spread further to progressively reach an agreement.

Inspiration: gossip is inspired from the human social behaviour linked to spreading rumors. People add their own information to the information received from other people, increasing their knowledge and spreading this knowledge further. When the process is repeated several times, people start to share the same knowledge and finally they reach an agreement.

Forces: the Gossip Pattern is composed by the Spreading and Aggregation Patterns. Thus, it presents the same tradeoffs. As in spreading, the main problem of gossip is the network overload that is produced by the continuous broadcast performed by the agents. To reduce the network overload, optimised broadcast can be applied (e.g. not all the neighbours receive the information). The number of neighbours that receive the information is the tradeoff of this pattern. The more neighbours receiving the information, the more robust the system becomes in case of failures, but a higher network overload is produced.

Entities-Dynamics-Environment: the entities involved in the gossip mechanism are agents, infrastructural agents, and hosts. Gossip is a combined pattern. Thus, the dynamics between the entities is the same as aggregation and spreading. Analogously to spreading, only an agent can initiate the process. When one agent initiates a gossip process, it sends the information (e.g. parameters and values) to a subset of its neighbours. Agents that receive the information, aggregate the information received with its own information and resend the aggregated information to its own neighbours. The same behavior is produced by infrastructural agents when no agent is hosted in one host and the host receives an information. The process finishes when the information received is the same than previously sent, i.e. when an agreement is reached.

Implementation: One interesting example of implementation appears in [Haas et al., 2006], where a probabilistic gossip is proposed. It was demonstrated that executing the gossip (broadcast) with a probability between 0.6 and 0.8 is enough to ensure that almost every node gets the message in almost every execution. This optimisation decrements the number of messages by 35%.

Known uses: Kempe et al. [Kempe et al., 2003] analysed simple gossip-based protocols for the computation of sums, averages, random samples, quantiles, and other aggregation functions, and demonstrate that this protocol converges to the agreement faster than the uniform gossip. Norman et al. [Salazar et al., 2010] propose an aggregation based on an Evolutionary Algorithm. They present a mechanism for coordination in large convention spaces (finding a common vocabulary in their case). The Evolutionary Algorithm approach keeps the diversity throughout the agreement process (not 100% of agents get the same agreement), this feature guarantees that when the scenario changes the system can quickly achieve a new agreement. It was demonstrated that this approach is resilient to unreliable communications and guarantees the robust emergence of conventions.

Consequences: the main advantage of gossip is the robustness: even in the presence of failures, the pattern is able to assess the agreement.

Related Patterns: the Gossip Pattern is composed by the Spreading Pattern (Section 2.3.1) and the Aggregation Pattern (Section 2.3.3).

2.5 Top Layer Patterns

Top layer patterns are composed or extended patterns from the middle layer or bottom layer patterns. In this section composed mechanisms are presented as self-organising design pattern.

2.5.1 Morphogenesis Pattern

The goal of the Morphogenesis Pattern is to achieve (or trigger) different behaviours in agents depending on their position in the system. The Morphogenesis Pattern exploits gradients: positional information is assessed through one or multiple gradient sources generated by other users. Morphogenesis was proposed as a

self-organising mechanism in [Mamei et al., 2006, Sudeikat and Renz, 2008]. We differentiate between the morphogenesis that refers to positional information and chemotaxis that refers to directional information as proposed in [Nagpal, 2004]. The morphogenesis process in biology has been considered as one inspiration source for gradient fields.

Problem: *In large-scale decentralised systems, agents decide on their roles or plan their activities based on their spatial position.*

Solution: In the Morphogenesis Pattern, specific agents spread morphogenic gradients over the system. Then, agents calculate their position in the system by computing their relative distance to the morphogenetic gradient sources. Analogously to the Gradient Pattern, the Morphogenesis Pattern increases the global knowledge of the agents because they can sense gradients that are beyond their communication ranges.

Typical Case: The Morphogenesis Pattern is typically used in large scale MAS where agents use their positions to decide their roles or to plan their activities.

Inspiration: In the biological morphogenetic process some cells create and modify molecules (through aggregation) which diffuse (through spreading), creating gradients of molecules. The spatial organisation of such gradients is the morphogenesis gradient, which is used by the cells to differentiate the role that they play inside of the body, e.g. in order to produce cell differentiations. As a result of the morphogenesis process, multi-celular organisms can develop, from a single zygote, a complex system of tissues and cell types.

Forces: The forces presented in this pattern are the same of the Gradient Pattern (Section 2.4.1).

Entities-Dynamic-Environment: The entities involved in the morphogenesis process are agents, hosts, and infrastructural agents. At the beginning some of the agents spread one or more morphogenic gradients, this process is implemented using the Gradient Pattern. Other agents sense the morphogenetic gradient in order to calculate their relative positions. Depending on their relative positions, the agents adopt different roles and coordinate their activities in order to achieve collaborative goals.

Implementation: An interesting implementation of the morphogenesis gradient to estimate the position is found in [Beal, 2009]. In that paper, a self-healing gradient algorithm with a tunable trade-off between precision and communication cost is proposed. In [Mamei et al., 2004] the motion coordination of a swarm of robots is implemented by using both Morphogenesis and Chemotaxis Patterns (Section 2.5.3).

Known uses: The Morphogenesis Pattern was used in [Bojinov et al., 2000] to implement control techniques for modular self-reconfigurable robots (metamorphic robots). In [Nagpal, 2002] the morphogenesis is used to create a robust process for shape formation on a sheet of identically programmed agents. In that paper, it was demonstrated that the gradients are extremely robust against random death, since there is no fixed hierarchy or centralised control, the system can not be easily disrupted.

Consequences: The Morphogenesis Pattern provides to the agents a mechanism to coordinate their activities based on their relative positions. Like the other mechanisms previously presented, robustness and scalability are properties presented by this pattern.

Related Patterns: The Morphogenesis Pattern exploits the Gradient Pattern (Section 2.4.1). Notice that the Morphogenesis Pattern can be combined with the Digital Pheromone Pattern. In this case, the role and behaviour of the agents depend on the distances to the pheromones.

2.5.2 Quorum Sensing Pattern

Quorum sensing is a decision-making process for coordinating behaviour and for taking collective decisions in a decentralised way. The goal of the Quorum Sensing Pattern is to provide an estimation of the number of agents (or of the density of the agents) in the system using only local interactions. The number of agents in the system is crucial in these applications, where a minimum number of agents is needed to collaborate on specific tasks.

Problem: *Collective decisions in large-scale decentralised systems requiring a number of agents or estimation of the density of agents in a system using only local interactions.*

Solution: The Quorum Sensing Pattern allows to take collective decisions through an estimation by individual agents of the agents' density (assessing the number of other agents they interact with) and by determination of a threshold number of agents necessary to take the decision.

Typical Case: in large scale MAS, the agents' density can be a crucial variable for the decision making, due to the existence of tasks that require a minimum number of agents.

Inspiration: the Quorum Sensing Pattern is inspired by the Quorum Sensing process (QS). QS is a type of inter-cellular signal used by bacteria to monitor cell density for a variety of purposes. An interesting example of this is bio-luminescent bacteria (Vibrio Fischeri) that can be found in some species of squids. This bacteria self-organise their behaviour to produce light only when the density of bacteria is sufficiently high [Miller and Bassler, 2001]. These bacteria constantly produce and secrete certain signaling molecules called auto-inducers. In presence of a high number of these bacteria, the level of autoinducers increases exponentially (i.e. the higher autoinducer level a bacterium detects, the more autoinducer produces and secretes).

Another interesting example is given by the colonies of ants (Leptothorax albipennis) [Sahin and Franks, 2002], when the colony must find a new nest site. A small portion of the ants looks for new potential nest sites and assess their quality. When they return to the old nest, they wait for a certain period of time before recruiting other ants (i.e. higher assessments produce lower waiting periods). Recruited ants visit the potential nest site and make their own assessment about its quality. Recruited ants return to the old nest and repeat the recruitment process. Because of the waiting periods, the number of ants present in the best nest will tend to increase. Eventually, the ants in this nest will sense

that the rate at which they encounter other ants in that nest has exceeded a particular threshold, indicating that the quorum number has been reached.

Forces: the Quorum Sensing Pattern uses gradients, thus it presents the same parameters that the Gradient Pattern (Section 2.4.1). Additionally, it is necessary to determine the threshold that triggers the collaborative behaviour. Quorum sensing provides an estimation of the density of agents in the system (i.e. beyond the local knowledge of the agents). However, this pattern is not a solution to calculate the number of agents necessary to carry out a collaborative task (i.e. to identify the threshold value).

Entities-Dynamic-Environment: The entities involved in the Quorum Sensing Pattern are the same that in the Gradient Pattern (Section 2.4.1). That is, agents, hosts, and infrastructural agents. The concentration is estimated by the aggregation of the gradients.

Implementation: There is not a specific implementation for the Quorum Sensing Pattern. However, biological systems presented above give us some ideas about how to implement the pattern. Two different ways to implement the Quorum Sensing Pattern may be proposed: (1) to use the Gradient Pattern to simulate auto-inducers where gradient concentrations provide agents with an estimation of the agents' density (i.e. like in bioluminiscent bacteria); or (2) likewise ant systems, agents' density can be estimated through the frequency to which agents are in a communication range. The use of gradients provides better estimations than the use of frequencies. However, it is more expensive computationally and it requires more network communications.

Known uses: Britton et al. [Britton and Sack, 2004] present an interesting example of the use of the Quorum Sensing Pattern: quorum sensing is used to increase the power saving in Wireless Sensor Networks. The quorum sensing permits to create clusters based on the structure of the observed parameters of interest, and then only one node for each cluster sends the information on behalf of the quorum. Another known example is the coordination of Autonomous Swarm Robots [Sahin and Franks, 2002].

Consequences: Each agent can estimate the density of nodes or other agents in the system using only local information received from neighbours, even when the system is really large and there agents do not have an identifier (i.e. are anonymous).

Related Patterns: The Quorum Sensing Pattern depending on its implementation uses the Gradient Pattern (Section 2.4.1).

2.5.3 Chemotaxis Pattern

The Chemotaxis Pattern provides a mechanism to perform motion coordination in large scale systems. The Chemotaxis mechanism was proposed by [Nagpal, 2004] in 2004. Analogously to the Morphogenesis Pattern, the Chemotaxis Pattern exploits the Gradient Pattern (Section 2.4.1): agents identify the gradient direction in order to decide the direction of their next movements.

Problem: *decentralised motion coordination in large scale systems aiming at detecting sources or boundaries of events.*

Solution: Agents locally sense the gradient information, they then follow the gradient in a specified direction: they follow higher gradient values, lower gradient values, or equipotential lines of gradients.

Typical Case: Chemotaxis has been mainly used as a gradient based approach to Locate diffuse event contours and diffuse event sources.

Inspiration: In biology, chemotaxis is the phenomenon in which single or multicellular organisms direct their movements according to certain chemicals present in their environment. Some examples in nature where chemotaxis appears are: leukocyte cells moving towards a region of a bacterial inflammation or bacteria migrating towards higher concentrations of nutrients [Wolpert et al., 2007].

It is important to note that in biology, chemotaxis is also a basic mechanism of morphogenesis. It guides cells during development so that they will be placed in the final right position. In this paper, we refer to terms chemotaxis and morphogenesis in a slightly different manner: chemotaxis for motion coordination following gradients, and morphogenesis for triggering specific behaviour based on relative positions determined through a gradient.

Forces: The Chemotaxis Pattern presents the same forces as the involved in the Gradient Pattern (Section 2.4.1).

Entities-Dynamic-Environment: The concentration gradient guides the agent's movement in three different ways, as shown on Figure 2.6: (1) Attractive Movement, when agents change their positions following higher gradient values; (2) Repulsive movement, when agents follow lower gradient values, producing an increment in the distance between the agent and the gradient source; and (3) Equipotential movement, when agents follow a specific gradient value or gradient values between a minimum and a maximum threshold.

Implementation: Chemotaxis can be implemented in two different ways. On the one hand, the system can use gradients existing in the environment to coordinate the agent's positions or directions. For example, Ruairi et al. [Ruairí and Keane, 2007] use attractive and equipotential movements to detect the contour of diffuse events using a multi-agent approach over a sensor network infrastructure. In this study, agents move over the sensor network following the gradient created by the diffuse event sources (existing in the environment), and when the agent reaches a gradient threshold, it follows equipotential movements to detect the contour of diffuse events.

On the other hand gradient fields could be generated by the agents. As an example, Mamei et al. [Mamei and Zambonelli, 2005] uses a gradient-based approach to coordinate the position of bots in the Quake 3 Arena video game. In this video game there is one agent controlled by the user (the prey) and a set of agents called bots (predators) trying to trap the agent controlled by the user. Each agent spreads a gradient in the system, thus the bots are repulsive between each other and attractive towards the agent driven by the user. Using gradients, the emerging behaviour of the bots is to attack the target from different directions.

Known uses: [Mamei et al., 2004] use Chemotaxis to coordinate the posi-

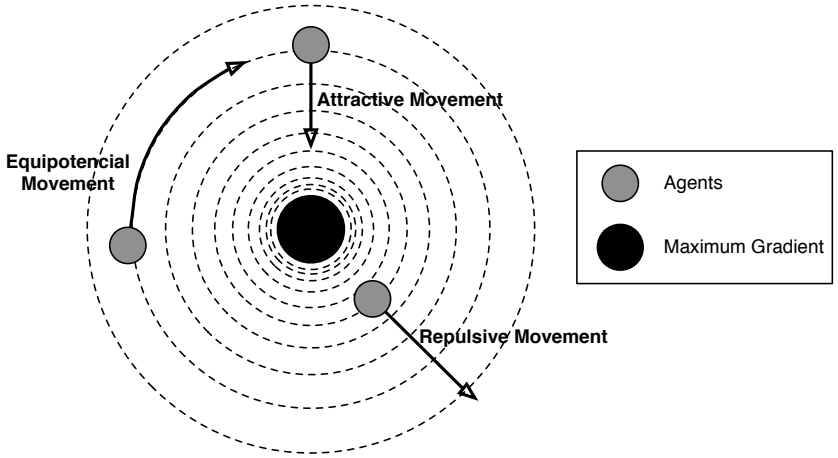


Figure 2.6: Chemotaxis Pattern

tions in swarms of simple mobile robots. In that paper, chemotaxis is implemented as follows: robots change the direction randomly when they detect that they are not following the gradient in their preferred direction. Notice that in that paper the mechanism used is called morphogenesis, despite of the actual use of the Chemotaxis Pattern. We adopted the name Morphogenesis to refer to positional information and Chemotaxis to directional information, following [Nagpal, 2004].

Consequences: The implementation of Chemotaxis Pattern analogously to the Morphogene Pattern produces an increment of the network load due to the spreading of gradients over the network. Using the Chemotaxis Pattern, agents can coordinate its movements even when they do not know about their position or the neighbouring positions.

Related Patterns: The Chemotaxis Pattern extends the Gradient Pattern (Section 2.4.1).

2.5.4 Flocking Pattern

Flocking is a kind of self-organising behaviour of a herd of animals of similar size and body orientation, often moving in masse or migrating in the same direction and with a common group objective. The Flocking Pattern is able to control dynamic pattern formations and move agents over an environment while keeping the formation and interconnections between them. Different disciplines have been interested in the emergent behaviour of flocking, swarming, schooling and herding. Several examples can be found in [Olfati-saber, 2006]. The forces that drive the flocking behaviour were proposed in 1986 by Craig W. Reynolds [Reynolds, 1987]. They are known as Reynolds rules: (1) Flocking Centering, (2) Obstacle Avoidance and (3) Velocity Matching. The Flocking Centering rule

captures the intuition that individuals try to keep close to nearby flockmates because they always try to move toward the flocking center. Obstacle avoidance rule pursues collision avoidance with nearby flockmates. Velocity matching is related to the ability to move the flocking with all the individuals at the same speed.

Problem: *Dynamic swarming pattern formation.*

Solution: The flocking Pattern provides a set of rules for moving the agents over the environment while keeping the formation and interconnections between them.

Typical case: Motion coordination of large scale MAS, mainly, 3-D simulations.

Inspiration: This pattern is inspired by the behaviour of a group of birds when they are foraging or flying and in the schooling of fish when they are avoiding a predator attack or foraging. For example, in a school of fishes when they are under a predator attack the movement of the first fish that senses the predator presence produces a fast movement alerting the other fishes by the waves of pressure sent though the water. The schooling of fish can change its formation in order to avoid a predator attack, recovering the initial formation after the attack. Analogously, the birds fly in flocking and they split the flocking in presence of one obstacle. When birds overtake the obstacle, they join together.

Forces: Parameters such as, avoidance distance, maximum velocity and maximum acceleration must be tuned to achieve the desired visualisation. However, regarding the typical case, as far as we know these parameters are independent.

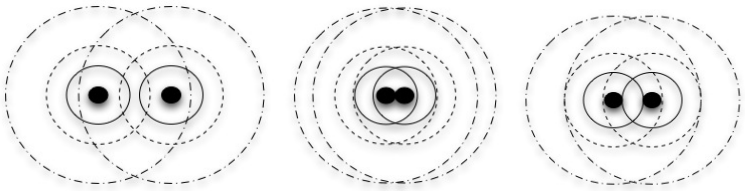
Entities-Dynamic-Environment: The entities participating in the Flocking Pattern are only agents using direct communication. Basically, agents sense the position of their neighbours and keep constant a desired distance. When the distance is changed due to external perturbations, each agent responds in a decentralised way to control the distance and to recover the original formation.

Implementation: Details about the algorithm and theory can be found in [Olfati-saber, 2006]. Here we present some basic concepts about the algorithm and the implementation. Analogously to the free-flocking algorithm presented in [Olfati-saber, 2006], each agent's motion is control by the equation 2.3

$$\vec{u}_i = \oint_i^g + \oint_i^d + \oint_i^\gamma \tag{2.3}$$

Where $\oint_i^g$ is a gradient based term that represents the flock centering and obstacle avoidance (rules 1 & 2).

Figure 2.7 represents how two agents that are communicated each other change their behaviour following the first term. In (a) agents are attracted, because they are allocated in attractive zones. In (b) they repel each other because they are too much close. Finally, in (c) they are in the neutral zone where the term becomes zero. When all the agents in the flocking are allocated in the neutral area, they form a stress-free structure. $\oint_i^d$ is a velocity consensus/alignament term that represents the velocity matching rule (rule 3). Finally,



(a) Attraction (b) Repulsion (c) Equilibrium

Figure 2.7: Metric distance Model - Movements

f_i^γ is the navigational feedback term that drive the group to the objective.

Known uses: The first application of the Flocking Pattern was modeling animal behaviour for films. Specifically, it was used to generate crowds which move more realistically. Flocking has also been used to control the behaviour of Unmanned Air Vehicles (UAVs) [Crowther and Riviere, 2002], Autonomous mobile robots [Hay, 2002, Jadbabaie et al., 2003], Micro or Miniature Aerial Vehicles (MAV) [Nardi et al., 2006] and Mobile Sensor Networks [La and Sheng, 2009a, La and Sheng, 2009b].

Consequences: Flocking tries to generalise the behaviour of flocking, independently of individuals (birds, penguins, fish...). Its behaviour does not depends on the methods used for the generation of agents' trajectories. The Flocking Pattern provides robustness and self-healing properties in front of agents' failures and communication problems.

Related Patterns: The Flocking Pattern extends from the Repulsion Pattern (Section 2.3.5). In fact, repulsion can be seen as a simplification of the Flocking Pattern. Both of them are gradient based, but not in the sense of the Gradient Pattern, because the Gradient Pattern uses diffusion to send the information beyond its communication range. However, it is an open possibility to use the Flocking Pattern with Gradients, when it may exist more than one role on agents in the flocking and it is desired to keep them in a formation even when they are not directly communicated. This case can fit a squadron of UAV's where some of them must be allocated in the perimeter of the formation because they have special devices. In this case, UAV's with special devices can used the Gradient Pattern to communicated with the other UAV's with spacial devices using the flock as a network infrastructure to spread the gradient.

2.5.5 Foraging Pattern

Foraging is the activity where a set of ants collaborate to find food. The Foraging Pattern is a decentralised collaborative search pattern. Mainly, the Foraging Pattern has been used in optimisation and swarm robotics.

Alias: Ant Foraging, Ant Colony Optimisation, Ant System

Problem: *Decentralised search and optimisation problems.*

Solution: The Foraging Pattern provides rules to explore the environment

in a decentralised manner and exploit resources.

Typical Case: Foraging Pattern is mainly applied to optimisation problems.

Inspiration: The Foraging Pattern is inspired by the Ant Colony Foraging behaviour. In ant colonies, ants self-organise their behaviour to find food close to the nest. The ant colonies use stigmergy communication, i.e. ants modify the environment through depositing a chemical substance called pheromones. This pheromone drives the behaviour of other ants in the colony. The pheromone concentrations are used to recluit other ants. Following the highest pheromone concentration, ants find the shortest path from the nest to the food, and they can adapt this path when obstacles appear or when the food is depleted.

Forces: Each ant has a probability to follow gradients or not. When one ant is not following the gradients, the ant walks randomly over the environment looking for new sources. When the probability of exploration is high, ants can adapt fastly to environmental changes. Whereas, a low exploration provides a fast convergence, i.e. most of the ants follow the path to the source. Due to the lack of exploration, when the source is depleted the ants spend too much time to find new sources. The Ant Foragin Pattern present also the same forces as the Digital Pheromone Pattern (Section 2.4.2).

Entities-Dynamic-Environment: The entities involved in the Foraging Pattern are the same as the Digital Pheromone Pattern (Section 2.4.2), and interact in the same way. When one agent senses the presence of a digital pheromone decides to follow the gradient or to move randomly.

Implementation: according to some random probability, agents either follow scouts (are recruited to exploit food), or perform some random search. In the case of ants, scouts deposit pheromones in their environment, that are later sensed by other ants to find food sources.

Known uses: The Foraging Pattern has been mainly applied as Ant Colony Optimisation (ACO) [Dorigo, 1992] in applications such as, scheduling [Martens et al., 2007, Blum, 2005], vehicle routing problems [Toth and Vigo, 2002, Secomandi, 2000, Bachem et al., 1996], or assignment problems [Lourenço and Serra, 1998].

Consequences: The system achives high quality performance in NP-Hard search problems.

Related Patterns: Foraging Pattern is composed by the Digital Pheromone Pattern (Section 2.4.2). Thus, the Foraging Pattern uses Evaporation (Section 2.3.4), Spreading (Section 2.3.1), and Aggregation (Section 2.3.3) patterns.

2.6 Summary

Table 2.3 summarises the problems that each pattern presented in this book solves.

Pattern's Name	Problem
Spreading	Agents' reasoning suffers from the lack of knowledge about the global system.
Replication	The number of software agents necessary to achieve a task is not known or must change along time to satisfy the system requirements. The high dynamicity in the environment can kill agents and they need to be created again.
Aggregation	Excess of information produced by the agents may produce network and memory overloads. Information must be distributively processed to reduce the amount of information and to assess meaningful information.
Evaporation	Outdated information can not be detected or its detection involves a cost that needs to be avoided.
Repulsion	Agents' movements have to be coordinated in a decentralised manner to achieve a uniform distribution and to avoid collisions between them.
Gradients	Agents suffer from lack of global knowledge that prevent them to know the consequences of their action beyond their communication range. Moreover, The agent behaviour not only depends on the information of other agents, also on the position or direction of other agents.
Digital Pheromone	Coordination of agents in large scale environments using indirect communication.
Gossip	Agents need to reach an agreement with only local perception and in a decentralised way.
Morphogenesis	In large-scale decentralised systems, agents decide on their roles or plan their activities based on their spatial positions.
Quorum Sensing	Collective decisions in large-scale decentralised systems requiring a number of agents or estimation of the density of agents in a system using only local interactions.
Chemotaxis	Decentralised motion coordination in large scale systems aiming at detecting sources or boundaries of events.
Flocking	Dynamic swarming pattern formation.
Foraging	Decentralised search and optimisation problems.

Table 2.3: Patterns Table

Chapter 3

Dynamic Optimisation

In this chapter the Evaporation Pattern, proposed in section 2.3.4, is incorporated to the Particle Swarm Optimisation (PSO) algorithm to deal with dynamic optimisation problems.

The goal of dynamic optimisation is to minimise or maximise a target function that changes along time. Many real-time decision problems, such as dynamic scheduling problems and dynamic vehicle routing problems, can be formulated as a dynamic optimisation problem.

When the size of the problems is very high, finding a solution with traditional optimisation algorithms requires days of execution, i.e. traditional algorithms are not feasible for dynamic optimisation. Meta-heuristics algorithms are able to find feasible solutions within a reasonable computation cost, demonstrating its applicability to solve dynamic optimisation problems.

A successful family of meta-heutistic algorithms are population-based algorithms. Most distinguished algorithms inside this subset are: Evolutionary Algorithms (EA), Ant Colony Optimisation (ACO), and Particle Swarm Optimisation (PSO). In all of them, the desired behaviour emerges from the interactions between individuals. The coordination of individuals' movements over the search space is considered a self-organising process.

To solve real-world optimisation problems, several uncertainty issues have to be considered. As it was described in [Jin and Branke, 2005], uncertainty issues can arise from four different origins: the evaluation of the fitness function may be subject to noise; the design variables may be subject to perturbations; the fitness function may be only approximated; or the optimum of the problem may change over time. In this chapter we are interested in addressing optimisation problems when i) the problem may change over time (dynamic environments) and ii) the evaluation of the fitness function is subject to noise.

Particle Swarm Optimisation is a population-based algorithm, where a set of particles initially spread over the search space, iteratively collaborate each other to improve the candidate solution. Particle Swarm Optimisation (PSO) has been proved as an efficient mechanism for static functions. Since 2001 when Parsopoulos and Vrahatis [Parsopoulos and Vrahatis, 2001] showed that PSO

could track slowly moving optima without any changes at all, several authors have proposed different variants to improve the performance of PSO in dynamic environments.

Specifically, in order to improve the original PSO to deal with dynamic environments, two main problems have to be addressed [Blackwell, 2007]: the *outdated memory problem* (due to the environment dynamism) and the *diversity loss problem* (due to particles' convergence). The outdated memory problem is related to the storage of the best position found. When a change in the environment occurs, the best solution found may become obsolete and may misguide the particle's search. The diversity loss problem appears when the swarm has converged and an environment change occurs. If the peak where the swarm converged disappears or changes its location, the low velocity of the particles inhibits the tracking and the particles may oscillate around a false attractor.

As we will describe in the next section, several variants of the PSO algorithm have been proposed to deal with the diversity problem. In the existing approaches, the outdated memory problem is easily solved by re-evaluating periodically the best positions found. This solution has been proved sufficient in environments when the fitness function is not subject to noise.

Regarding to the evaluation of fitness functions that are subject to noise, the main challenge appears when particles try to detect changes in the environment. The noise associated to the evaluation of the fitness function makes this detection difficult. [Parsopoulos and Vrahatis, 2001] presented some experimental results showing that PSO is somehow noise tolerant. However, some detected changes may be false and some real changes may not be detected.

The goal of this research is to demonstrate that the evaporation mechanism, proposed as a basic design pattern for self-organising systems, is a contribution to improve the performance of PSO algorithms in dynamic and noisy environments. Specifically, we propose an extension of the *Multi Quantum Swarm Optimisation* (mQSO) algorithm [Blackwell and Branke, 2006] that incorporates an evaporation mechanism to improve the convergence of PSO in dynamic multimodal environments when the evaluation of the fitness function is subject to noise. Moreover, the evaporation mechanism proposed avoids the continuous detection of changes in the environment.

This chapter is organised as follows. Next section briefly reviews the research related to our problem. Section 3.2 describes how the evaporation mechanism has been used and how it is incorporated into the existing mQSO algorithm. Section 3.3 reports experiments demonstrating the contribution of the evaporation mechanism. Finally, in Section 3.4 we draw some conclusions and set paths to future research.

3.1 Background

This section presets the state of the art of PSO in dynamic and noisy optimisation to understand the contribution of applying the evaporation mechanism to existing PSO approaches.

PSO was proposed aiming to produce a collaborative intelligence behavior by borrowing the analogy of social interaction, rather than purely individual cognitive abilities. The movements of each particle are based on the combination of both a cognitive and a social model. The cognitive model drives each particle to its best position so far. The social model drives each particle to the best position found by particles belonging to its neighbourhood.

3.1.1 Particle Swarm Optimisation

In PSO, each particle has a position $\vec{p}_i$ and a velocity $\vec{v}_i$. Initially, the set of particles is randomly distributed in the search space with a random initial velocity. The position and velocity of each particle are modified iteratively. The movement of the particles is determined by combining some aspect of its experience, its best position found $\vec{b}_i$, with social information, the best position of its neighbours $\vec{g}$. At each algorithm iteration, all particles evaluate the objective function at its current location (fitness value) $f_u(\vec{p}_i)$.

Since 1995 when James Kennedy and Eberhart proposed the PSO algorithm, some extensions and optimisations of their parameters have been realised [Poli et al., 2007]. One of them, the constriction coefficients, has been well accepted by the community. The constriction coefficients control the convergence of the particles and allow an elegant and well-explained method for preventing explosion, ensuring convergence and eliminating the arbitrary V_{max} parameter. Using contriction coefficients, the movement of each particle follows the next two equations:

$$\vec{p}_i = \vec{p}_i + \vec{v}_i \tag{3.1}$$

$$\vec{v}_i = \chi(\vec{v}_i + \vec{U}(0, \phi_1)(\vec{b}_i - \vec{p}_i) + \vec{U}(0, \phi_2)(\vec{g} - \vec{p}_i)) \tag{3.2}$$

where:

- $\vec{p}_i$ is the current position of the particle i;
- $\vec{v}_i$ is the velocity of the particle i;
- χ is the constant multiplier that ensures the convergence;
- $\vec{b}_i$ is the best position found by the particle i;
- $\vec{g}$ is the global best solution found by the particles; and
- $\vec{U}(0, \phi_i)$ represents a vector of random numbers uniformly distributed in $[0, \phi_i]$.

The velocity of particles is composed of two components: the cognitive part $\vec{U}(0, \phi_1)(\vec{b}_i - \vec{p}_i)$ that drives the particles to its best position found, and the social part $\vec{U}(0, \phi_2)(\vec{g} - \vec{p}_i)$ that drives the particles to the best position found by the swarm.

3.1.2 PSO in Dynamic Environments

Optimisation in dynamic environments is a challenging problem for PSO. Different extensions of PSO have been proposed to improve its adaptiveness in dynamic environments. Extensions (see [Poli et al., 2007] for an overview) propose solutions such as resetting the position of particles frequently or using a multi-swarm model.

Diversity loss has been addressed either by introducing randomisation, repulsion, dynamic networks, or multi-populations [Blackwell, 2007]. An important contribution was to keep the diversity along the algorithm execution, instead of resetting the position of particles whenever a change in the environment occurs. That is, introducing the notion that not all the particles tend to reach the optimal position. some particles are continuously exploring the search space while other are converging to the peaks. These explorer particles have been implemented in different ways.

In [Fernandez-Marquez and Arcos, 2009b] we proposed an exploration mechanism based on heterogeneous swarms that combine attractive and repulsive particles. Repulsive particles keep a formation that allows a continuous exploration of the search space, whereas attractive (PSO) particles collaborate to improve the solution. The mechanism maintains the diversity property allowing the swarm to self-detect the changes of the environment.

Charged Particle Swarm Optimisation [Blackwell and Bentley, 2002] (CPSO) use repulsion between a subset of swarm particles, to avoid the convergence of the whole swarm. Quantum Swarm Optimisation [Blackwell, 2007] (QSO) uses the notion of a cloud of particles, that are randomly positioned around the swarm attractor. Both methods have been tested in a multi-swarm context and QSO has shown a higher performance.

The Collaborative Evolutionary-Swarm Optimisation (CESO) [Lung and Dumitrescu, 2007] is a hybrid approach that also uses two different sets of particles to preserve the diversity. Specifically, the diversity is maintained using crowding techniques. The performance of CESO is higher than QSO but the mechanism to detect changes in the environment is the same.

Recently, [Du and Li, 2008] have proposed MEPSO (Multi-strategy Ensemble Particle Swarm Optimisation). MEPSO outperforms other approaches but for unimodal environments where changes have high severity. To detect changes, MEPSO uses re-evaluation. Moreover, when a change is detected a re-randomisation is performed.

The outdated memory problem has been tackled by setting best positions as their current positions or by re-evaluating best positions to detect the changes in the environment (increasing the computation cost) and then resetting the memory of the particles. Most of these existing approaches assume that either the changes are known in advance by the algorithm or that they can be easily detected. These hypotheses are not feasible in many real problems due to the presence of noise and its unpredictable nature.

Multi Quantum Swarm Optimisation (mQSO)

Multi Quantum Swarm Optimisation (mQSO) is an algorithm proposed for dealing with multi-modal dynamic problems. mQSO divides the swarm in a number of subswarms with the goal of exploiting different promising peaks in parallel. The multiswarm approach, increases the diversity and decreases the probability to finalise the search in a local optimum.

Moreover, each swarm consisted of two different kind of particles: i) *PSO particles* that try to reach a better position by following the standard PSO algorithm and ii) *quantum particles* that orbit around the subswarm attractor within a radius r_{cloud} to keep the diversity along the algorithm execution. Quantum particles address the diversity loss problem. The position of quantum particles is calculated with the following equation:

$$\vec{p}_i \in B_n(r_{cloud}) \tag{3.3}$$

where B_n denotes the d-dimensional ball of the swarm n centered on the swarm attractor $\vec{g}_n$ with radius r_{cloud}.

The idea of mQSO is that each swarm reaches one peak and tracks it along the algorithm execution. To ensure that two swarms are not exploiting the same peak, an exclusion mechanism is proposed as a form of swarm interaction. The exclusion mechanism uses a simple competition rule among swarms that are close (at most at distance r_{excl}) to each other. The winner is the swarm with the best fitness value at its swarm attractor. The loser swarm is expelled and reinitialised in the search space.

When there are more peaks than swarms, not all peaks can be tracked by swarms. Because of the changes in the environment, any local maximum may become a global maximum. Thus, whenever this new optimum is not tracked by any swarm, the performance of the system decreases. To prevent this, an anti-convergence operator is applied whenever all swarms have converged, i.e. when for all swarms the max distance found betweem 2 particles is less than r_{conv}. At that moment, anti-convergence expels the worst swarm from its peak by reinitializing the particles of the swarm. As a result, there is at least one swarm watching out for new peaks.

mQSO was proved as a good mechanism in multi-peak dynamic environments. The dynamicity is expressed by small changes applied to the peak locations, heighs, and widths.

3.1.3 PSO in Noisy functions

Noisy fitness functions have been addressed by different researches and are a key issue in real-world problems. Different authors have demonstrated that noisy fitness functions are not a handicap for PSO effectiveness in static environments. In 2001, Parsopoulos and Vrahatis studied the behaviour of PSO when a Gaussian distributed random noise is added to the fitness function. They demonstrated that PSO remained effective in the presence of noise.

In 2005, [Pugh et al., 2005] proposed the Noise-resistant variant, where each particle takes multiple evaluations of the same candidate solution to assess a fitness value. It was demonstrated that noise-resistant PSO has a better performance than the original PSO. The disadvantage of this approach is that in order to improve the confidence of a fitness value, multiple evaluations have to be performed.

In [Bartz-Beielstein et al., 2007] the stagnation effect is analyzed for additive and multiplicative noise sources. Bartz-Beielstein et al propose the use of a statistical sequential selection procedure, together with PSO, to improve the accuracy of the function estimation and to reduce the number of evaluations of samples. Moreover, the authors show that the tuning of PSO parameters is not enough to eliminate the influence of noise.

Another proposal for reducing the number of re-evaluations is Partitioned Hierarchical PSO [Janson and Middendorf, 2006]. PH-PSO organises the neighbourhood of the swarm in a dynamic tree hierarchy. This organisation allows the reduction of the number of sample evaluations and can be used as a mechanism to detect the changes in noisy and dynamic environments. In PH-PSO, the mechanism used to detect the changes is based on the observation that changes occur within the swarm hierarchy.

3.2 Evaporation Mechanism

As we have described in the previous section, there is some evidence that PSO is able to deal with noisy environments but many of the current PSO extensions for dynamic multi-modal problems have not been tested in noisy environments. The main drawback of the current proposals is that they continuously check for changes in the environment. This strategy of continuously checking, cannot be directly applied when the result of the evaluation of a specific position is subject to noise. Noise provides different values in each evaluation of a given point. These differences maybe misinterpreted as environment changes causing the continuously resetting of the memory of particles.

A possible solution to deal with noisy environments is to incorporate a filter that tries to minimise the problem of receiving different fitness values each time a specific position is requested. In addition of the issue of determining the best threshold for the filter in unknown environments, we will show in the experiments section that the use of noise filters is not the best solution.

An alternative approach to deal with noise is to consider that the confidence on a given fitness value degrades with the time. Our proposal is to extend the mQSO algorithm with an evaporation mechanism (Section 2.3.4) that will avoid to continuously check up on changes in the environment and, as a second beneficial consequence, will improve the performance of mQSO in noisy fitness evaluation problems.

3.2.1 Evaporation Mechanism

To solve the outdated memory problem when changes in the environment are not known in advance or they cannot be detected due to the noisy fitness function, we claim that a mechanism for continuously forgetting is better than a periodical resetting approach and even better than the use of an ad-hoc threshold to filter the noise in the fitness function. Moreover, a continuous mechanism avoids the assumption that changes can be predicted or detected in some way.

We propose to use the evaporation mechanism presented as design pattern for self-organising systems (Section 2.3.4) to reduce the fitness value along time of the best position found by each particle. This mechanism will penalise optima that were visited a long time ago. Thus, evaporation provides an automatic dissipation mechanism over the information taking into account the acquisition time. The idea of evaporation is not new, as it was presented in the Evaporation Pattern description (Section 2.3.4). ACO systems use evaporation in pheromone trails as a mechanism to achieve a signal degradation [Garnier et al., 2007] and to self-adapt to environment changes. Moreover, in [Fernandez-Marquez and Arcos, 2008], some preliminary results were shown using an evaporation factor in unimodal dynamic environments.

Different approaches can be used as evaporation factors. The main approaches are the use of either a subtractive or a multiplicative factor. A subtractive factor will decrease, at each particle iteration, the fitness value in a constant factor ν following the equation:

$$s_{ni} = s_{ni} - \nu \tag{3.4}$$

where given a particle i belonging to the swarm n, the fitness value of the best position found $\vec{p}_{ni}$ is stored at s_{ni} (best solution).

Analogously, a multiplicative factor will decrease the fitness value by multiplying it with a constant α following the next equation:

$$s_{ni} = s_{ni} \times \alpha \tag{3.5}$$

where α is an evaporation factor such that $\alpha \in (0, 1)$ and $\times$ is the multiplier operator.

In the experiments section we show that with the subtractive factor mQSO achieves lower errors. Nevertheless, multiplicative factors can be used achieving also good performance results. Then, we incorporated a subtractive factor ν. Specifically, at each algorithm step, each particle evaluates the fitness of its current position and updates s_{ni} either, to the current fitness when it is higher than s_{ni}, or applying the evaporation equation. That is:

```
if (f_u(p⃗_ni) > s_ni) then
    s_ni = f_u(p⃗_ni);
    b⃗_ni = p⃗_ni;
else
    s_ni = s_ni − ν
end
```
(3.6)

where f_u is the fitness function and $\vec{p}_{ni}$ is the current position of the particle i of swarm n.

3.2.2 mQSOE

In this section we introduce the new algorithm called multi Quantum Swarm Optimisation Evaporation (mQSOE). In this new algorithm the exisitng mQSO has been extended with the evaporation mechanism. The main advantages of this extension (mQSOE) are twofold: i) it avoids the false change detection produced by the noisy fitness function; and ii) because it has not to check for environment changes, it saves evaluations of the fitness function. Thus, the savings can be used to improve the algorithm performance.

The modification of the mQSO algorithm is simple: the test for environmental changes is eliminated and the evaporation equation (3.4) is added when updating the particle and swarm attractors. Figure 3.1 presents the mQSO algorithm extended with the evaporation mechanism. At the beginning, the particles of all swarms are randomly initialised. Given a particle i of a swarm n, $\vec{v}_{ni}$ is the velocity of the particle, $\vec{p}_{ni}$ is the position of the particle, $\vec{b}_{ni}$ is the position of the best fitness found by the particle, and s_{ni} is the fitness value of the best position $\vec{b}_{ni}$. Once the particles are initialised, the best positions $\vec{g}_n$ and fitness values s_n of the swarms are calculated.

In the main loop, the first step is to apply the anti-convergence operator when all swarms have converged. A swarm has converged when the maximum distance found between its particles is lower than the r_{conv} parameter. Anti-convergence operator resets the worst swarm. As we presented before, anti-convergence operator ensures that at least one swarm is ever exploring the search space to detect new peaks that may become important.

Next to the anti-convergence operator, the exclusion operator is applied. The exclusion operator detects when two swarms have converged to a close position and then resets the worst of them. Two swarms are too close when the distance between their attractors is lower than r_{excl}.

After applying all operators, the particles are moved according to the equations defined for their type: PSO particles are updated following the standard PSO equations (3.2) and (3.1) while QSO particles are randomised according (3.3). Finally, after updating the particle's position, the best position found by the particle $\vec{b}_{ni}$ and the best fitness found s_{ni} is also updated. The evaporation mechanism is applied to this step according to (3.6).

```
//Initialisation
foreach particle ni do
    Randomly initialise v_ni, p_ni;
    b_ni = p_ni;
    s_ni = f_u(p_ni);
end
foreach swarm n do
    g_n = argmax{f_u(p_ni)}
    s_n = max{f_u(p_ni)}
end
repeat
    //Anti-convergence
    if all swarms have converged then
        //randomise worst swarm
        RandomiseSwarm(worst_swarm)
    end
    //Exclusion
    foreach pair of swams n, m do
        if swarm attractor g_n is within r_excl of g_m then
            if (s_n ≤ s_m) then
                RandomiseSwarm(n)
            else
                RandomiseSwarm(m)
            end
        end
    end
    foreach swarm n do
        foreach particle i of swarm n do
            //Update particle
            Apply equations (3.2) (3.3) depending on particle type.
            if (f_u(p_ni) > s_ni) then
                s_ni = f_u(p_ni)
                b_ni = p_ni
            else
                //Apply evaporation factor
                s_ni = s_ni − ν
            end
            //Update attractor
            g_n = b_ni such as i has max{s_ni}
            s_n = max{s_ni}
        end
    end
until number of function evaluations;
```

Figure 3.1: mQSOE algorithm (mQSO extended with the evaporation mechanism)

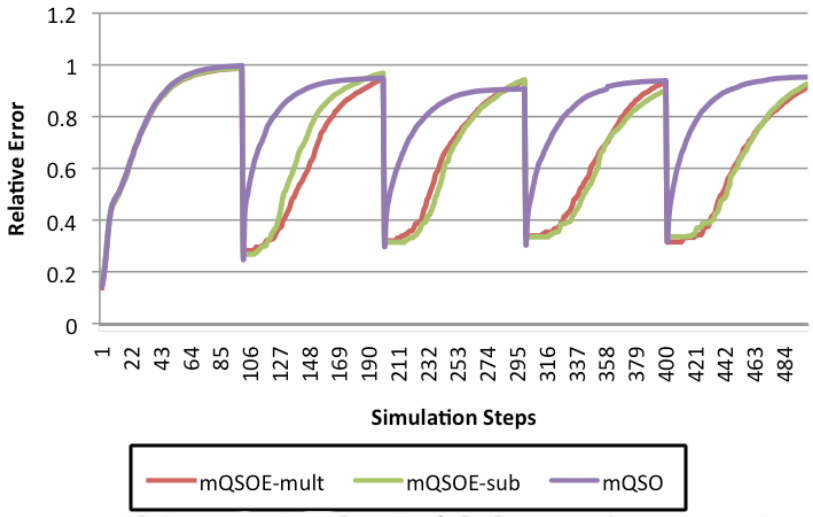


Figure 3.2: Slow Adaptation Behavior

3.2.3 Dynamic Evaporation

As it is shown later on in the experimental section, mQSOE (mQSO + Evaporation) achieves a better performance than the original algorithm mQSO when the fitness function is subject to noise. However, when the fitness function is not subject to noise, mQSO achieves a slightly better performance than mQSOE. To solve this issue, we propose a new evaporation mechanism able to improve mQSOE in both scenarios (in noise free scenarios and in noisy scenarios).

Studying the behavior of mQSOE, we observed that a high evaporation factor produces a fast adaptation (fast response after environment changes occur), but particles can not reach a good solution in the optimisation process (bad convergence), because particles may not use their cognitive part. That is, particles forget the best position found very fast and are not capable of reaching good solutions. Contrarily, a low evaporation factor achieves a fast and effective convergence, but adaptation is very low. Figure 3.2 exposes the slow adaptation observed in mQSOE-mult and mQSOE-sub compared with mQSO using environment change detection. We observe that, even when both approaches achieve the same performance, the delay introduced by evaporation produces negative consequences when the frequency of changes increases.

To solve this problem, we propose a dynamic evaporation factor using only the particle's local information, i.e. its velocity and the difference between the fitness of its best position found and the fitness of its current position (i.e a gradient-based policy). The particle's velocity presents the following characteristics: (1) the velocity module is very high before converging, and (2) the velocity module decreases when particles are close to the convergence point.

On the other hand, when particles have converged and an environment change occurs, the gradient becomes high. Thus, the intuition is that the evaporation factor must be high when the velocity module is slow (indication of convergence) and the gradient is high. Moreover, the evaporation factor must be low when the particle's velocity is high, contributing to the swarm convergence. Specifically, dynamic evaporation is implemented using the following equations:

$$s_{ni} = s_{ni} * \alpha_{dyn} \tag{3.7}$$

$$\alpha_{dyn} = 1 - \frac{(\frac{s_{ni} - fitness(\vec{p}_{ni})}{s_{max}} + \frac{|\vec{V}_{ni}|}{|\vec{V}_{max}|})}{2} \tag{3.8}$$

where α_{dyn} is the dynamic evaporation factor, s_{ni} the fitness value of the best position found, s_{max} the maximum fitness reachable, $|\vec{V}_{ni}|$ is the velocity module of the particle, $|\vec{V}_{max}|$ the maximum velocity module of the particles. Both maxima are used to normalise the evaporation factor. Notice that when a new best position is found, the evaporation is not applied until next iteration, (see algorithm 3.3).

Analogously to mQSOE, mQSODE (dynamic evaporation) extends the original mQSO adding the dynamic evaporation mechanism. The modification of the mQSO algorithm is simple: the test for changes is eliminated and the dynamic evaporation equation (3.4) is added when updating the particle's memory.

Figure 3.3 summarises the mQSO algorithm extended with dynamic evaporation equations (i.e. evaporation mechanism). The evaporation mechanism is applied according to (3.7) and (3.8). The detailed description of the different algorithm steps was presented in the previous section.

3.3 Experiments

The goal of the experiments is to analyze the effect of the evaporation mechanism by comparing with the standard mQSO algorithm. Previously to compare both algorithms, we performed experiments to determine the best value for the subtraction factor ν; we compared the performance of using a subtractive or a multiplicative factor; and we analyzed the independence of the evaporation factor with respect to the peak heights and the peak shifts (severity).

To evaluate the performance of the evaporation mechanism and to compare mQSOE with mQSO, we have used the Moving Peaks Benchmark (MPB) [Branke,]. MPB is a benchmark developed to compare dynamic optimisation algorithms by modeling problems less complex than the real world but more complex than a simple simulation. MPB allows the design of search spaces that change over time (in the height, width and location of peaks).

3.3.1 Experimental framework

The settings for MPB parameters correspond to MPB's scenario 2 with 10 peaks. Specifically, we used a 5-dimensional search space with dimension ranges from 0

```
// Initialisation
foreach particle ni do
    Randomly initialise v_ni, p_ni;
    b_ni = p_ni;
    s_ni = f_u(p_ni);
end
foreach swarm n do
    g_n = argmax{f_u(p_ni)}
    s_n = max{f_u(p_ni)}
end
repeat
    // Anti-convergence
    if all swarm have converged then
        // Remember to randomise worst swarm
        randomiseSwarm(worst_swarm);
    end
    // Exclusion
    foreach pair of swams n, m do
        if swarm attractor g_n is within r_excl of g_m then
            if (s_n ≤ s_m) then
                randomiseSwarm(n);
            else
                randomiseSwarm(m);
            end
        end
    end
    foreach swarm n do
        foreach particle i of swarm n do
            // Update particle
            Apply equations (3.2) (3.3) depending on particle type.
            if (f_u(p_ni) > s_ni) then
                s_ni = f_u(p_ni)
                b_ni = p_ni
            else
                // Apply evaporation factor
                Equations (3.7), (3.8)
            end
            // Update attractor
            g_n = b_ni such as i has max{s_ni}
            s_n = max{s_ni}
        end
    end
until number of function evaluations;
```

Figure 3.3: mQSODE algorithm (mQSO extended with dynamic evaporation equations)

Params	values	Params	values
movrand	random	num. of peak	10
num. of dimensions	5	minheight	30
maxheight	70	stdheight	50
minwidth	0.1	maxwidth	12
stdwidth	0.0	mincoordinate	0
maxcoordinate	100	vlength	1
height_severity	7.0	width_severity	1.0
use_basis_function	true	correl. lambda	true
eval per change	5000	peak_function	cone

Table 3.1: Standard settings for MPB

to 100. The peak function used is a cone and the maximum distance that one peak moves is 1. Table 3.1 summarises all the MPB settings.

A run consisted of 100 peak changes in a random direction (correlation coefficient = 0). All peaks changed every 5000 steps (function evaluations) their height (height $\in$ [30, 70]), width (width $\in$ [1, 12]), and position (vLenght = 1). Results are based on averages over 50 runs with uncorrelated peak changes.

The performance of the algorithms has been assessed using the *offline error* measure. Offline error is the average over, at every point in time, the error of the best solution found since the last change of the environment. This measure is always greater or equal to zero and would be zero in perfect tracking.

To aggregate noise to the fitness function, we modified MPB such as the fitness function incorporates a noise factor γ in the following way:

$$fitness(\vec{p}) = MPBfitness(\vec{p}) + \frac{2 * \theta - 1}{2} * \gamma \tag{3.9}$$

where θ generates a uniform random number between [0..1] and γ varies between 0 and 30 depending on the experiment.

The parameters of mQSO and mQSOE are the same, except the evaporation factor ν that only exists in mQSOE. We used 100 particles grouped in 10 different swarms. Each swarm has 10 particles, 5 PSO and 5 QSO. The parameters r_{excl} and r_{conv} were set to 31.5 following the authors' recommendations [Blackwell and Branke, 2006]. The standard PSO parameters have been settled following [Blackwell, 2007], $\chi = 0.729843788$, $\phi_1 = 2.05$, and $\phi_2 = 2.05$.

3.3.2 Determining the evaporation factor

The first experiments conducted aimed at determining the best evaporation factor ν and to analyze the dependency of this factor with respect to the noise level γ. We performed experiments by varying γ from 0 to 30 and varying ν from 0 to 30.

The best results were achieved for low values of ν. Thus, we focused our analysis for values of ν ranging from 0 to 4. Figure 3.4 shows the offline er-

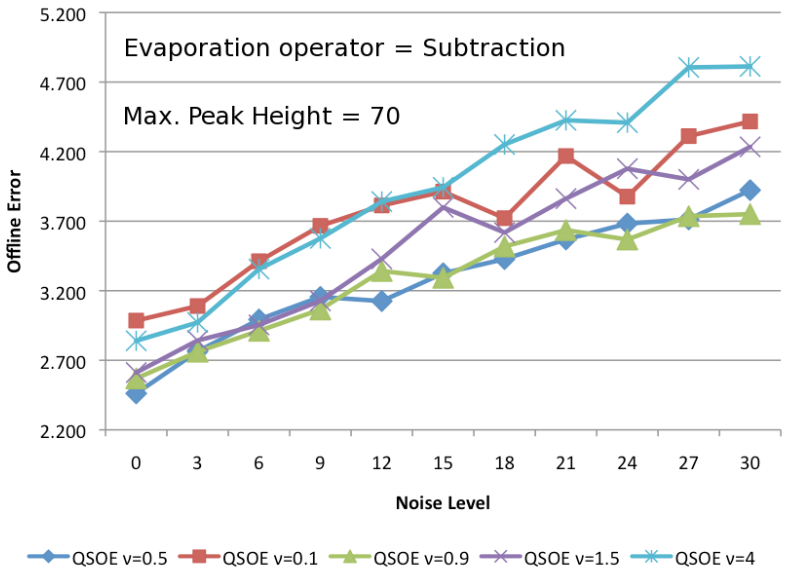


Figure 3.4: Varying Evaporation Factor

ror when varying the noise level and the evaporation factor. The vertical axis represents the offline error, and the horizontal axis the noise level. First, as it was predictable, the error increases when the noise increases. Second, the best performance is achieved with low evaporation values (lower than 1), but with a minimum evaporation (values like 0.1 behave also bad).

Our decision was to choose $\nu = 0.5$. This decision was motivated by two main reasons: it presents a good performance (best performances are achieved for $\nu \in [0.4, 0.9]$) and it presents the smoothest behavior. Moreover, the experiments show that all values of ν follow the same behavior when the noise level increases. Thus, the best evaporation value is not changing with the noise level, i.e. the evaporation factor is independent of the noise level.

3.3.3 Subtraction versus multiplication as evaporation operator

The goal of these experiments was to compare the performance of a subtraction operator with the performance of a multiplier operator. Specifically, we compared the best result achieved in the previous experiments ($\nu = 0.5$) with mQSOE where the evaporation mechanism was substituted by the equation (3.5). We performed experiments by varying α from 0 to 1.

Figure 3.5 compares the error of the subtractive operator with the error of best multiplicative factors. To achieve errors similar to the subtractive operator,

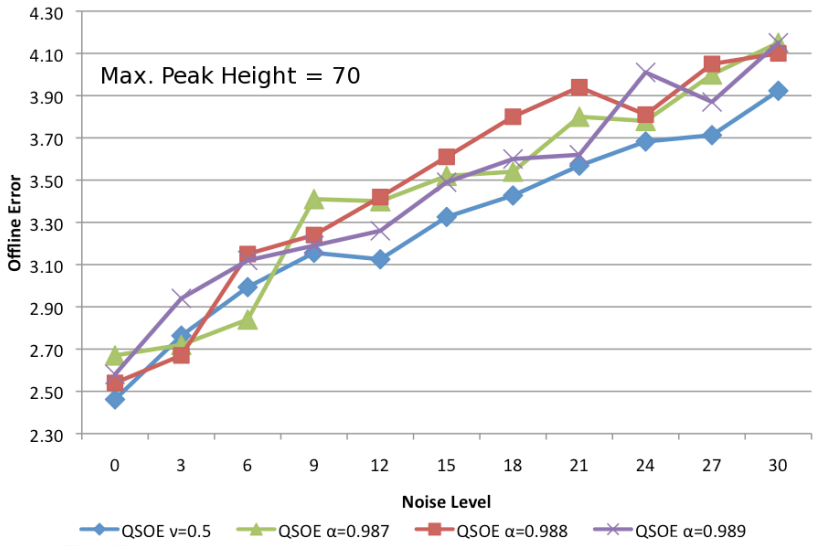


Figure 3.5: Multiplicative versus subtractive evaporation

the values of α must be close to 1. The best performance of a multiplicative factor is achieved with $\alpha = 0.989$. Specifically, the results are similar to the best subtractive operator but the subtractive operator has a smother behavior and achieves lower errors. Nevertheless, as a conclusion we have to say that both operators can be used as evaporation mechanisms because the performance difference is not significative.

3.3.4 Independence of peak heights and peak shifts

The goal of these experiments is to analyze the dependency of the evaporation factor with respect to the heights of the peaks and the distance of the peak movements (severity). All the previous experiments were conducted with peak heights from 30 to 70 and with a severity of 1.

We analyzed the performance of the evaporation factor with lower peak heights. Specifically we repeated the experiments with $height \in [10..30]$. Figure 3.6 shows the results obtained. Notice that the offline error is lower because the worst error is now also lower. Also, the offline error difference among the different factors decreases. Nevertheless, $\nu = 0.5$ is still the best value indicating that the evaporation factor is not dependent of the peak heights.

To compare the performance of mQSOE with reported results of mQSO, we set the severity parameter to 1. Nevertheless, we conducted experiments varying the severity parameter to analyze the performance of mQSOE. Since we have previously analyzed the performance of the evaporation factor with different

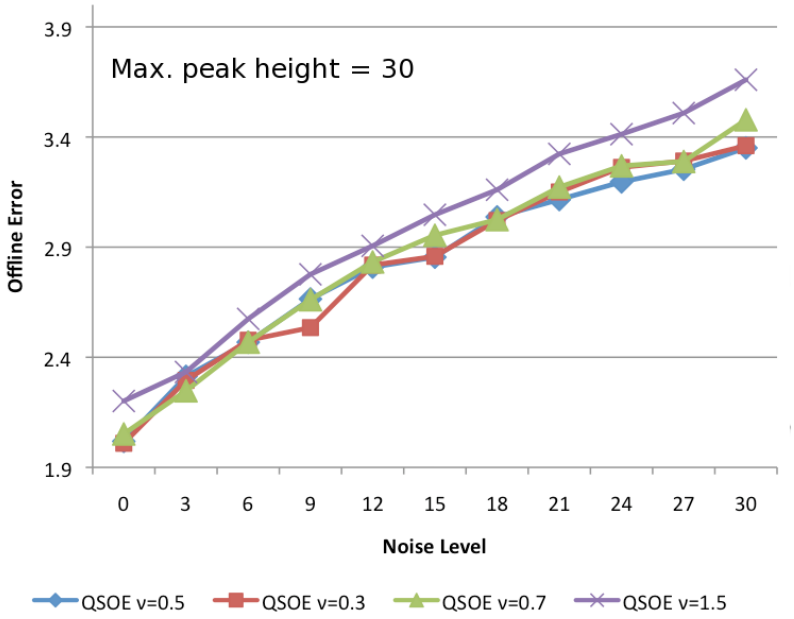


Figure 3.6: Evaporation performance when decreasing the peak height

noise levels, in these experiments the noise level was set to zero. Figure 3.7 shows that the error increases when the severity increases but it affects to all the evaporation settings with the same proportion. Thus, $\nu = 0.5$ presents also a competitive behavior and the experiments corroborate the independence of the evaporation factor with respect of the severity.

3.3.5 mQSO versus mQSOE

Finally, we are ready to compare the offline error of mQSO versus our mQSOE. The goal of the experiment is to study the behaviour of both algorithms regarding the increment of noise in the evaluation of the fitness function. As in the previous experiments, γ varied from 0 (no noise) to 30 (approx. 23% of noise).

Table 3.2 shows that when the noise level increases, the performance of both algorithms decreases (as expected). Moreover, in presence of noise mQSOE demonstrates better performance than the original mQSO. However, in the absence of noise mQSO reaches a better performance than using evaporation. The reason is that environment changes are instantly detected by mQSO because it is constantly re-evaluating swarm attractors. However, the evaporation mechanism is a little bit slower but more robust in presence of noise.

The bad performance of mQSO with noisy fitness functions is produced be-

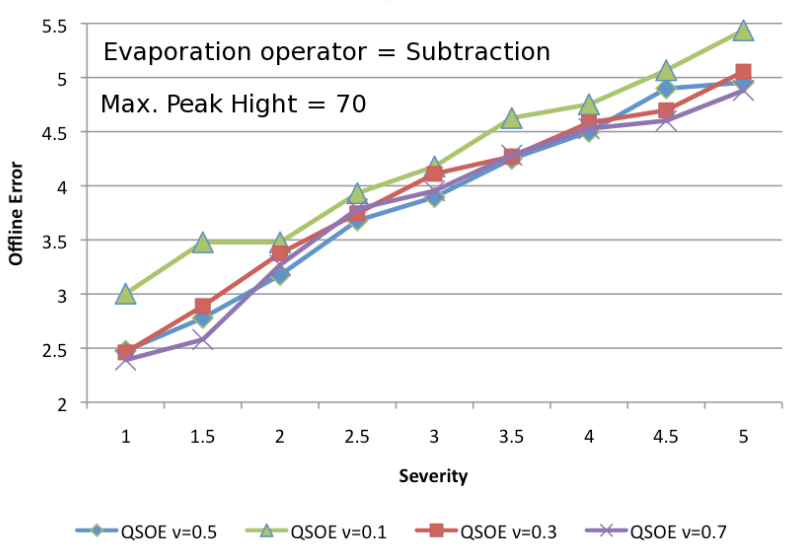


Figure 3.7: mQSOE performance when changing severity

cause the algorithm confuses noise with changes in the environment and, consequently, it is continuously re-initialisating the particle's memory. That is, at each algorithm iteration the best position found ($\vec{b}_{ni}$) is changed to the current position degrading the search mechanism.

3.3.6 Filtering noise in mQSO

In order to avoid the continuously resetting of mQSO, we experimented with the incorporation of a noise filter into the original mQSO. Specifically, we used a threshold value β to determine whether a change observed in the environment is effectively considered a real change. Then, a change in the environment is only considered when the difference between the stored fitness value and the current evaluation is higher than this threshold. Thus, we add the condition (3.10) in the function that checks for a change.

$$\begin{array}{l} \textbf{if } \beta < |s_{ni} - f_u(\vec{p}_{ni})| \textbf{ then} \\ \quad | \quad changeProduced = true \\ \textbf{else} \\ \quad | \quad changeProduced = false \\ \textbf{end} \end{array} \tag{3.10}$$

where β is the ad-hoc threshold that controls the tolerated noise.

The goal of this experiment is to analyze the performance of mQSO modified with (3.10). Two different values for the threshold were analyzed: $\beta = 15$ and

γ	mQSO	mQSOE
0	2.206 ± 0.529	2.462 ± 0.620
3	3.397 ± 0.727	2.763 ± 0.600
6	3.581 ± 0.640	2.993 ± 0.562
9	4.107 ± 0.526	3.156 ± 0.554
12	4.468 ± 0.475	3.125 ± 0.472
15	5.094 ± 0.765	3.326 ± 0.661
18	5.397 ± 0.652	3.427 ± 0.534
21	6.090 ± 0.670	3.568 ± 0.514
24	6.603 ± 0.822	3.683 ± 0.668
27	6.985 ± 0.618	3.713 ± 0.564
30	7.609 ± 1.015	3.923 ± 0.419

Table 3.2: Average of offline error (± Std. Dev.) achieved by mQSO and mQSOE introducing different noise levels

$\beta = 30$. The first value is a threshold that corresponds to the middle noise level. The second one is the highest noise level.

Figure 3.8 shows how the use of a noise threshold ($\beta > 0$) improves the performance of the original mQSO in presence of noise. However, noise filters produce a lower performance when the fitness function is not subjected to noise. This decrement is caused because the algorithm confuses small environment changes with noise.

It can be observed that mQSO with $\beta = 15$ presents the best performance when $\gamma = 15$. Nevertheless, this behavior is not robust and decays for other values of γ. This result is not surprising because when β and γ share the same value, the noise threshold is correlated with the error in the fitness evaluation. That is, when β and γ are 15, we ensure that the noise is never confused with a change in the environment. Nevertheless, as Figure 3.8 shows, for γ values different than β the offline error grows. The reason is because when γ is lower than β, noise is confused with environment changes and produces a memory resetting when it is not necessary. On the other hand, when γ is higher than β, it increases the probability of not detecting the changes in the environment, preventing the peak tracking.

For $\beta = 30$, environment changes are treated as noise. Thus, most of the changes in the environment are not detected and, consequently, the memory of the particles is not re-initialised. As a consequence, swarms are not able to track the peaks.

Comparing the performance of mQSO with noise filters and mQSOE, mQSOE presents a more robust behavior. mQSOE is not achieving the best performance at all the different noise levels: without noise the standard mQSO presents the lowest error; and mQSO extended with a noise filter presents the best performance when the noise level in the fitness function is the same that the noise threshold at mQSO. However, the noise threshold cannot be fixed because

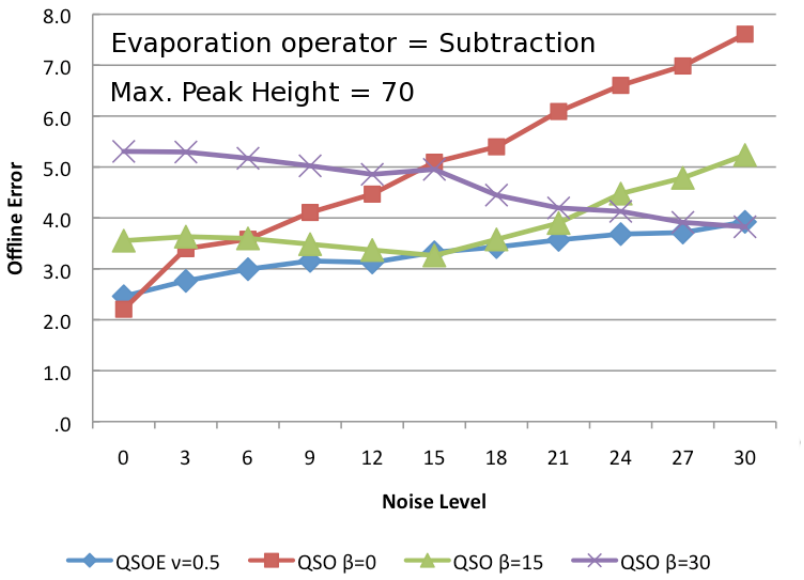


Figure 3.8: Comparing the offline error when introducing different noise thresholds

in real problems the noise level is never static. Even setting the threshold in mQSO, with the noise level value that the fitness function has (best case, but not feasible in real world), mQSOE achieve similar performance.

Summarizing, mQSOE is demonstrated as a good solution for dealing with dynamic environments where the evaluation of the fitness function is subject to noise. Moreover, the evaporation mechanism does not require an explicit checking of changes.

3.3.7 Dynamic Evaporation

In this final experiment the goal was to demonstrate the higher performance of the proposed mQSODE when the fitness function is not subjected to noise. As it was presented, the main problem of using evaporation is the decrease of performance in this situation. Moreover we show how dynamic evaporation presents the same performance as mQSOE algorithm in presence of noise.

Performance when the fitness function is noise-free

Figure 3.9 shows the comparison between multiplicative and substractive evaporation (mQSOE), the original mQSO, and the dynamic evaporation (mQSODE). The results are calculated with the small step change type. Horizontal axis represents the simulation steps and vertical axis represents the relative error averaged

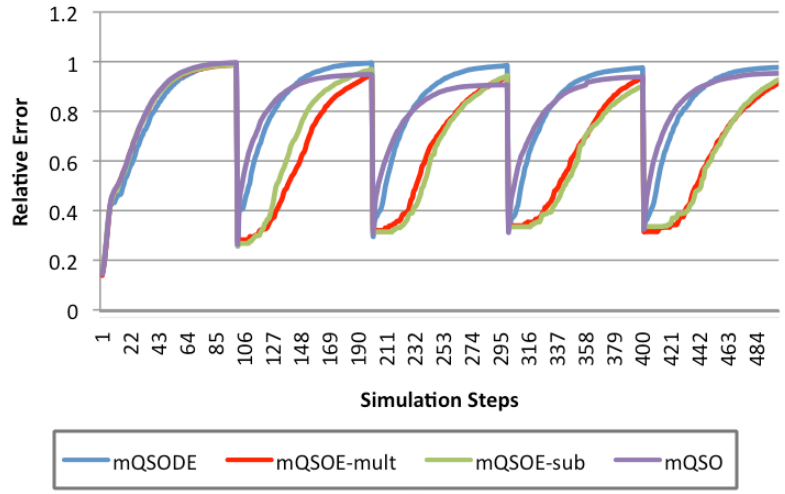


Figure 3.9: T1 - Noise free Environment

Type	Description	Type	Description
T1	Small Step	T2	Large Step
T3	Random	T4	Chaotic
T5	Recurrent	T6	Recurrent with noise

Table 3.3: Types of Changes

over 60 runs. Constant-based evaporation methods (mQSOE-mult and mQSOE-sub) present a delay after the environment changes, due to the slow evaporation process, as we presented in the previous section. The dynamic evaporation mechanism is able to act after a change with similar speed than mQSO (that is detecting environment changes). The faster reaction of our approach when environment changes occur allows a better convergence. Notice that, in Figure 3.9, if the system increases the frequency of changes, mQSOE-sub and mQSOE-mult cannot reach convergence due to the slow reaction when an environment change occurs. However, mQSODE and mQSO may converge and present a better performance when the frequency of changes increases.

Figure 3.10 summarises average maxima (the average of worst convergences) for each algorithm when different change types are applied (types of changes are described in Table 3.3). mQSODE presents a better performance than the constant-based evaporation mechanisms and the standard mQSO.

But mQSODE is not only presenting the best performance in the worst cases, it is also the best according to average means, i.e. the average of all convergences realised after changes, for the majority of change types. Figure 3.11 presents

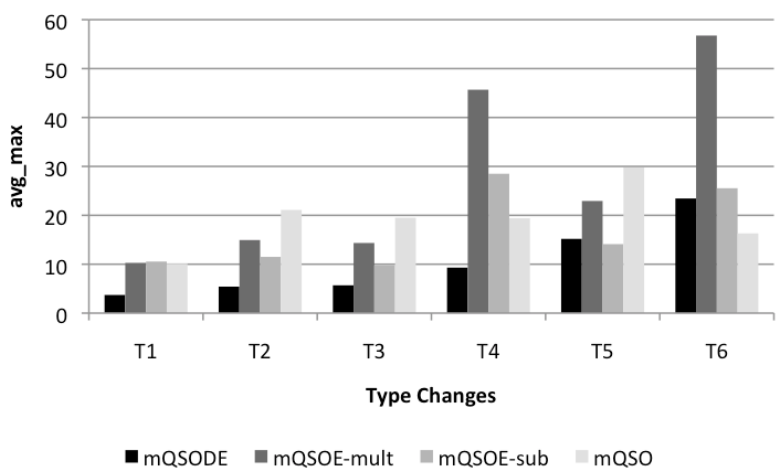


Figure 3.10: Average Maxima - Noise-free environment

the results of mQSODE for average_means. The performance of mQSODE is achieved because its faster reaction to environment changes by adjusting the evaporation factor depending of the local state of the particle.

Figure 3.12 represents the average of the best cases. Regarding this measure, the original mQSO presents a better performance than our approach, i.e., even though mQSODE presents a better performance than mQSO in average, mQSO is able to reach some best results. However, the behaviour of mQSO is not as robust as the mQSODE. Figure 3.13 exposes how the mQSO algorithm presents a higher standard deviation than mQSODE.

In noise free environments the dynamic evaporation mechanism improves the existing algorithms (mQSOE-sub and mQSOE-mult). Moreover, the problem reported for those algorithms, i.e. the decreasing of performance when the fitness function is not subjected to noise, has been solved for the new approach. mQSODE presents similar results than the standard mQSO, that does not use the evaporation, for two reasons: a close reaction time when an environment change occurs, and the fact that mQSO uses fitness function evaluations that mQSODE can use in the optimisation process.

Performance in presence of noise

Figure 3.14 shows how mQSODE solves the problem of slow adaptation after a change and moreover, achieves a similar performance than the mQSOE-mult and mQSOE-sub algorithms.

mQSODE presented a higher performance than the other (see Figure 3.17). Moreover, mQSODE achieves a clear higher performance than the others in the average, worst, and lower standard deviation. We consider this result a

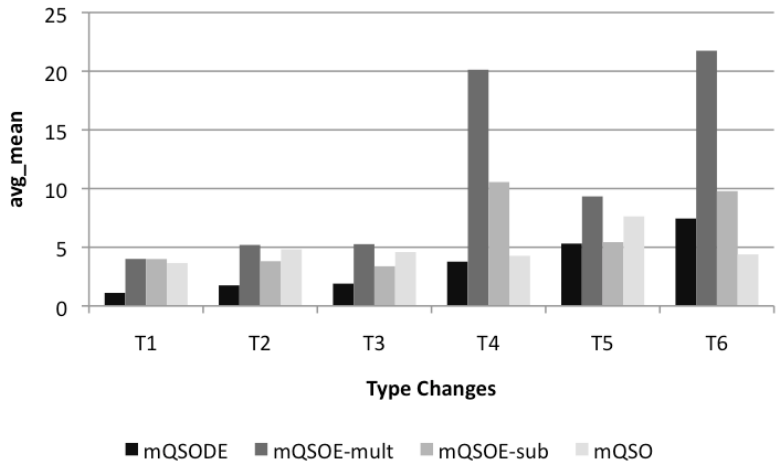


Figure 3.11: Average Means - Noise-free environment

big success, to achieve a low standard deviation, because it highlights a robust behavior in the optimisation process. Our proposal presents slightly lower results than mQSOE-sub and mQSOE-mult regarding the average best. Nevertheless, this difference is not statistically significant.

3.4 Conclusions

In this chapter we show how we have applied an evaporation mechanism to deal with optimisation problems when the problems change over time (dynamic environments) and the evaluation of the fitness function is subject to noise. We have incorporated the evaporation mechanism as an extension of mQSO, a well known algorithm for dynamic multimodal problems. Nevertheless, the solution proposed in this research is general and can be incorporated to other existing algorithms that detect the changes in the environment by re-evaluating swarm attractors.

We have shown that the evaporation mechanism applied to Particle Swarm Optimisation is robust to different simulation conditions such as the peak heights, the severity, and the level of noise. Furthermore, we reported experiments that introduce the possibility of using either subtractive or multiplicative evaporation (mQSOE) factors and also a dynamic evaporation (mQSODE).

Experiments have shown that the performance of the proposed mQSOE extension outperforms the standard mQSO when the fitness evaluation is noisy. Moreover, mQSODE (dynamic evaporation) achieves a better performance than mQSOE (fixed evaporation) and better than mQSO when the fitness function is not subject to noise. Also, we have shown that the detection of changes

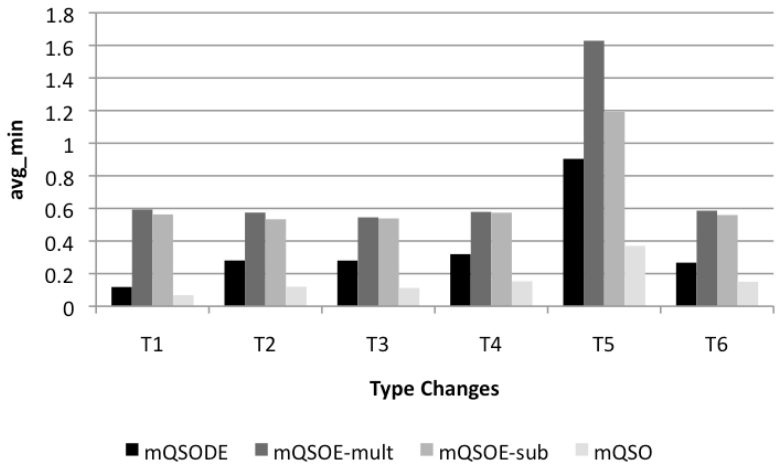


Figure 3.12: Average Minima - Noise-free environment

when noise is present is not a trivial issue. Specifically, we have shown how the introduction of noise filters is not able to outperform mQSOE nor mQSODE.

Moreover, the evaporation mechanism has an additional benefit: the effort avoided in re-evaluating swarm attractors can be used to achieve a better performance by spending them on the swarm's convergence.

One of the research questions proposed in this book was: can the basic mechanisms extracted by decomposition of complex mechanisms be applied isolated and make contributions to other communities?

In this work we demonstrate that a basic mechanism as the evaporation can make important contribution and can be used isolated. Even when the evaporation was extracted from the pheromone pattern which is composed by aggregation, spreading and evaporation. It has been demonstrated that the evaporation can work without the presence of aggregation or spreading.

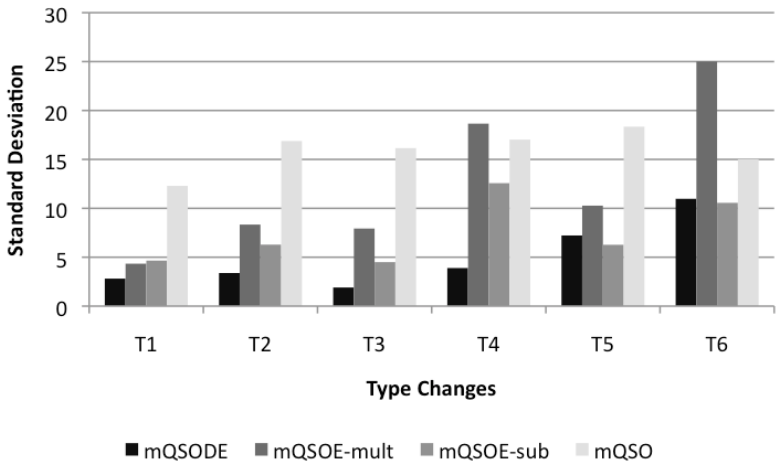


Figure 3.13: Standard Deviation - Noise-free environment

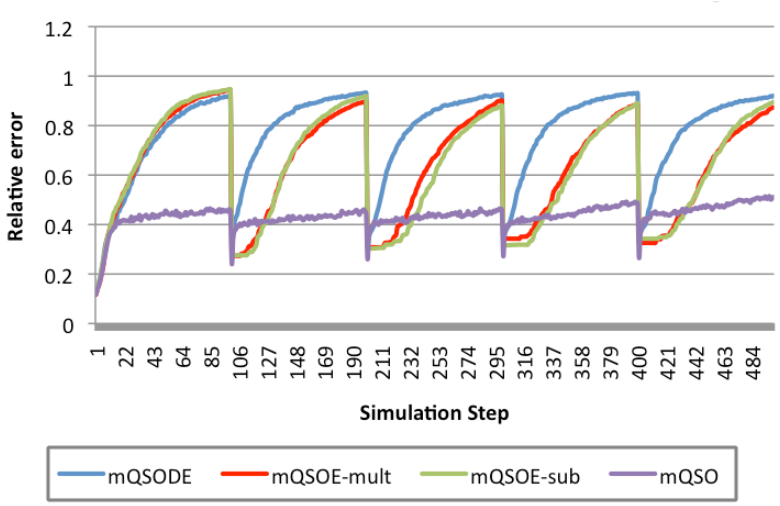


Figure 3.14: T1 - Noisy Environment

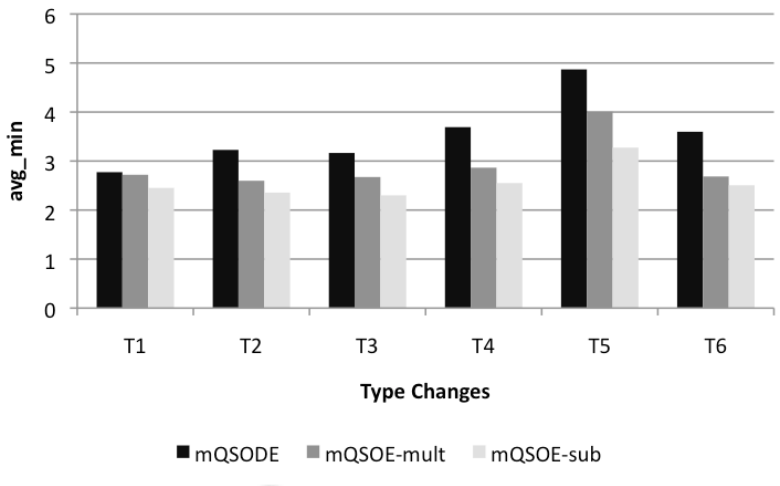


Figure 3.15: Average Minima - Noisy Environment

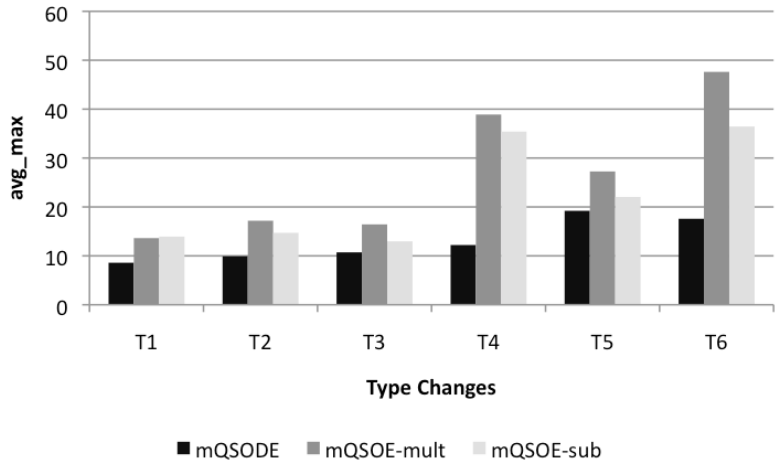


Figure 3.16: Average Maxima - Noisy Environment

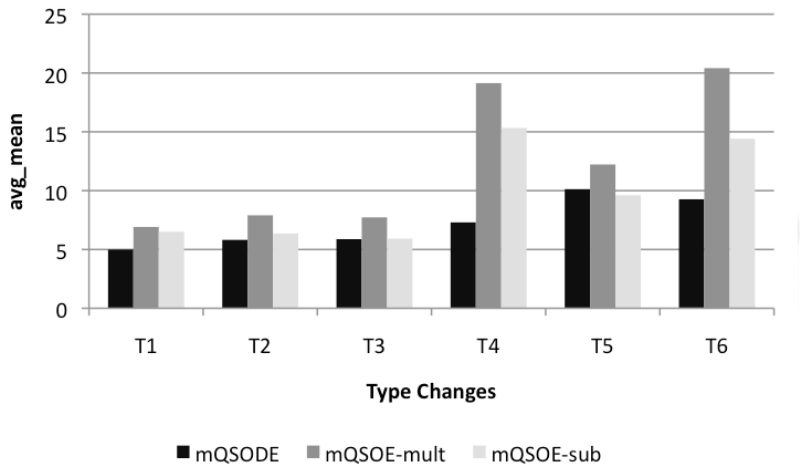


Figure 3.17: Average Means - Noisy Environment

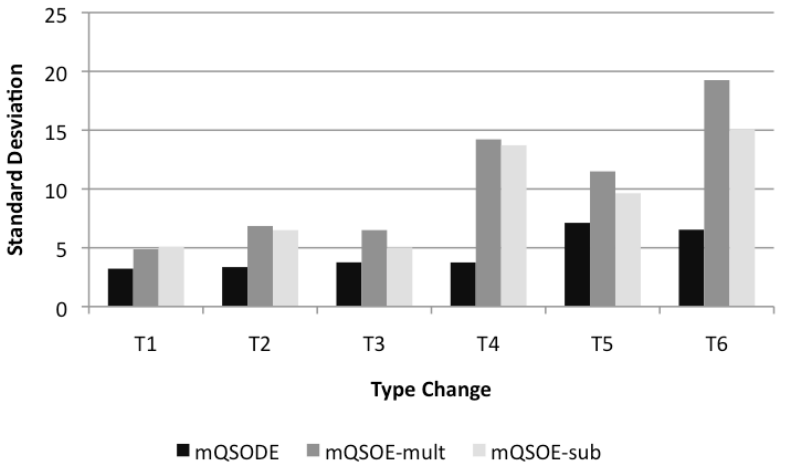


Figure 3.18: Standard Deviation - Noisy Environment

Chapter 4

Hovering Information in Spatial Computing

In this chapter we define and analyse a collection of algorithms based on the Replication Pattern (Section 2.3.2) and the Repulsion Pattern (Section 2.3.5), for persistent storage of information at specific geographical zones exploiting the resources of mobile devices located in these areas. The proposed algorithms are implemented and tested inside the Hovering Information Project [1].

Hovering Information is a mobile computing paradigm where pieces of self-organising information are responsible to find their own storage on top of a dynamic set of mobile devices. In Hovering Information a piece of information is deposited in the environment. Its goal is to serve the information to all other devices in a fixed geographical area. Thus, a piece of information deposited in the environment must ensure its survivability by replicating it-self to other devices. Moreover, the information and its replica must self-organise their positions to cover the fixed geographical area using the minimum number of replicas. The Replication Pattern provides robustness and fault-tolerance to the system, while repulsion provides a decentralised motion coordination where information reaches a uniform distribution inside a determined area. The proposed system provides a persistent storage (and access) of information at specific locations on top of a volatile (mobile and uncontrolled) storage media. By persistent we mean that the data must be stored and accessible at a fixed geographical area for the duration required by the application. Contrarily to other approaches of data dissemination, our approach uses a viral programming model. Data performs an active role in the storage process. It acts as a virus or a mobile agent finding its own storage and relocating when necessary. We consider geographical areas of any shape and size. Simulation results show that our algorithms are scalable and converge quickly, even though none of them outperform the others in all performance metrics considered.

[1] http://www.dcs.bbk.ac.uk/∼dimarzo/projects/HoverInfo.html

4.1 Introduction

Spatial computing builds on the Amorphous Computing concept [Abelson et al., 2007] by considering physical geographical zones as computing elements (e.g. geographical zones able to process tasks, collaborate with each other in order to produce some specific result), while making abstraction of underlying computational devices (i.e. sensors, mobile phones or robots).

Amorphous Computing considers computational particles dispersed irregularly in an environment, communicating locally with each other. They form what is referred to as an amorphous computer. Particles are programmed identically but may store different values. Particles are all similar and generally stationary. Mobile particles considered so far are either swarms of robots or self-assembling robots. Primitives for programming amorphous computing take inspiration from self-organising natural systems and corresponding applications show a high level of robustness to particle errors.

Our goal is to provide a spatial memory service for mobile users applications, where both stationary and mobile devices provide memory storage. A spatial memory is a set of (active) geographical zones (of any shape, possibly overlapping) each acting as a memory cell able to store any kind of information. This memory would constitute a base service for the Spatial Computing paradigm. For instance, a Spatial Search and Rescue service could coordinate an emergency service to rescue survivors of a natural disaster by exploiting data about survivors' position and data about rescue team availability, both stored in such a memory. Additionally, data can change while it is replicating (e.g. to create gradient fields) or different pieces of data can self-aggregate to create new data.

The challenge is to provide persistent storage (and retrieval) of information at specific locations on top of a volatile (mobile and uncontrolled) storage media. By persistent we mean that the data must be stored and accessible at a fixed geographical area for the duration required by the application (from a few minutes, to several hours or several days).

Our work so far concentrated on persistent *storage* algorithms using the concept of Hovering Information, thus providing a way of implementing the idea of Spatial Memory in an infrastructure-free and self-organising way. Hovering Information is a mobile computing paradigm where pieces of self-organising information are responsible to find their own storage on top of a dynamic set of mobile devices. Once deployed, the hovering information service is a location-based service disseminating geo-localised information generated by and aimed at mobile users. It supports a wide range of pervasive applications, from urban security to stigmergy-based systems. A piece of Hovering Information is a geo-localised information residing in a highly dynamic environment such as a mobile ad hoc network. This information is attached to a geographical area, called "an anchor area". A piece of hovering information is responsible to keep itself alive, available and accessible to other devices within its anchor area. Hovering information uses the Replication Pattern (Section 2.3.2) and the Repulsion Pattern (Section 2.3.5). The properly use of those patterns in combination with a set of

policies allows the system to replicate strategically a minimum number of hovering information pieces necessary to satisfy the above requirements. It does not rely on any central server. The appealing characteristics of the hovering information concept is the absence of a centralised entity and the active participation of the information in the storage and retrieval process.

Therefore, this chapter presents a theoretical investigation of new variants of persistent self-organising storage algorithms. We focus on one single piece of information replicated into a predefined area and consider only the storage aspect of the spatial memory problem mentioned above. We evaluate algorithms' performance by measuring metrics such as number of messages exchanged among mobile nodes, memory consumption, and accessibility rates.

4.2 Background

Current proposals for location-based services consider data as a passive entity moved around by the infrastructure intimately linked with the users (publishers or subscribers). Data is either stored on a fixed infrastructure and delivered to users when they reach a certain location [Eugster et al., 2005], or for infrastructureless scenarios, where the publication space moves along with the publisher and a subscriber space must overlap the publisher space to access the information [Eugster et al., 2009]. Other works for data dissemination over MANETs exploit mobile devices but do not address persistence of the information or do so for specific scenarios only (e.g. intravehicular networks) [Leontiadis et al., 2009]. Data itself is passive and not self-organising. Tasks of propagating or routing information are left to the infrastructure (mobile or fixed nodes).

In our work, we consider data as an active self-organising entity acting as a mobile agent, which decides on its own where to go next and how to be stored. Data has a life of its own; it is dissociated from the (human) users who produce and exploit it, and from the mobile devices who act as storage media. In [Konstantas and Villalba, 2006] the concept of hovering information was introduced. Hovering Information was initially applied to provide persistent storage algorithms for circular areas and investigated performances for single (identical) pieces of information [Villalba Castro et al., 2008]. After that, Hovering Information was also applied to multiple (different) pieces of hovering information [Castro et al., 2008]. Initial replication algorithms and their corresponding simulations took into account wireless characteristics of mobile devices and concentrated on circular areas only. Specific issues such as scalability or speed of convergence were not investigated. Simulations were performed with OMNET++[2], a simulation tool for wireless devices. Although good for gaining performance results, these simulations did not provide a satisfactory visualisation of the propagation of the pieces of information. We decided to revise initial algorithms and to undergo additional thorough simulations using Repast[3], an

[2] http://www.omnetpp.org/
[3] http://repast.sourceforge.net/

agent-based simulation tool providing visual simulations still allowing us to collect performance results.

Hovering information and spatial memory are related to different concepts such as memory, middleware, or dissemination of data. Mainly, the works related with Hovering Information service are: Location-based publish/subscribe [Eugster et al., 2009], where the information is available to the subscriber when the publisher and subscriber communication range are overlapped. Thus, the publication space moves along with the publisher. Dissemination Services [Leontiadis and Mascolo, 2007, Scellato et al., 2008, Datta et al., 2004], present an interesting starting point for replication algorithms but they do not offer a solution to ensure the persistence of information. Virtual Nodes [Dolev et al., 2005], where the GeoQuorum approach proposes the implementation of an atomic shared memory in ad hoc networks. Persistent Node [Beal, 2003], where, analogously to Hovering Information, data is the active entity and storage medium is rather passive towards the data. However, the Persistent Node approach assumes a no mobile storage medium and Viral Programming [Butera, 2007].

The novelty of the hovering information service (and later of the spatial memory) resides in the combination of the above described techniques, in particular the combination of virtual memory, persistent node (active and moving data) and viral programming (mobile code). This combination allows to deal with new problems not addressed before. One example of case study is the presence of ice on the road. One car detects ice and deposits a piece of information with an anchor area center in the ice. The piece of information will be alive and warning other cars before they reach the ice, whereas there is at least one car to store and serve the information. Notice that the information is an autonomous entity, its goal is to stay alive in the anchor area, replicate the number of times necessary to cover the whole anchor area and self-organise their positions to serve the warning information to all cars in the anchor area, while using the minimum number of replicas. A detailed studio about the difference between the hovering information service and related concepts commented above can be found in [Fernandez-Marquez et al., 2011],

We consider the concept of hovering information as a service: pieces of hovering information come with *policies* specifying the replication mode (speed and availability level) and the time to live (garbage collection). For instance, a piece of hovering information spreading an emergency message should best follow the broadcast algorithm (even if it is more expensive) because of safety issues, since all users must be quickly informed. A piece of hovering information carrying an advertisement message could be cheaper to spread and does not need to be accessible to everybody in the area. Additionally, pieces of hovering information may adapt to the environment. For instance, if using a specific algorithm their target of spreading cannot be achieved because the number of nodes suddenly drops, they can switch to another algorithm in order to maintain the same level of service.

The notion of policies allows to switch from one algorithm to another: pieces

of hovering information use the algorithm that is most appropriate to the type of information (e.g. emergency information vs advertisement), to the replication mode (faster vs slower) by changing the repulsion rate, to accessibility rate threshold (100% of the users in the anchor area must have access to the information vs it is acceptable if only 80% of the users have access to the information).

4.3 Hovering Information Concept

4.3.1 Hovering Information

A piece of Hovering Information h is a geo-localised information attached to a geographical area, called the *anchor area*. The main goal of h is to self-replicate among neighbouring mobile nodes in order to maintain itself in the specified anchor area and make itself accessible to mobile nodes in that area.

A piece of hovering information h is defined as a tuple:

$$h = (id, A, n, data, policies, size);$$

where id is the hovering information identifier, A is the anchor area (see below), n is the mobile node where h is currently hosted, $data$ is the actual data carried by h, $policies$ are the spreading policies of h, and $size$ is its size.

In this chapter, we do not investigate the active usage of policies for enhancing adaptation. The policy investigated is the dissemination algorithm applied by the pieces when they spread in their environment. For the sake of completeness, we give the full definition of h (inc. policies and size). As we said above, policies can be used to dynamically change the algorithm at run time.

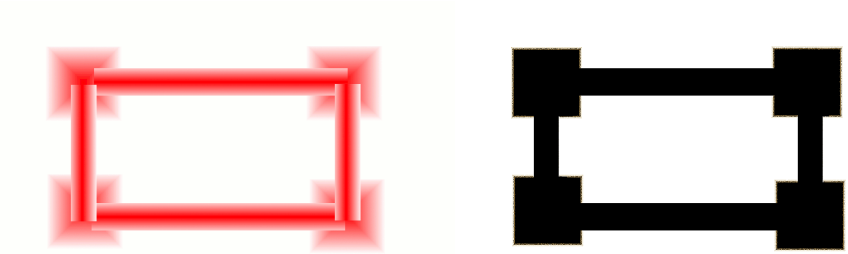
4.3.2 Anchor Areas

Hovering information spread into indoor or outdoor spaces such as motorways, pedestrian roads, shopping centers or leisure areas. The shape of the area can vary from a simple circle centered on a focal point to more elaborated shapes of any type (regular, irregular, convex or not, etc.).

Anchor areas presented in this chapter do not have any restriction in the shape, as in previous works [Di Marzo Serugendo et al., 2007] (where anchor areas were circular). We call these areas amorphous areas. The goal of a piece of hovering information is to replicate itself in the area in order to be accessible to nodes in that area (i.e. a piece of information needs to know the anchor area).

Figure 4.1 represents an anchor area with 4 halls connected by 4 corridors. Nodes can move freely in whatever direction (also outside the corridors), i.e. the hall and corridors are not delimited by walls. The information must fill the anchor area, remain located inside it and must not spread outside. In particular, if a node is located in the center of Figure 4.1 (thus not in a corridor), it may have access to a replica but should not store one.

This is a typical area for a shopping center and for a piece of information that must be relevant to people located in the corridors and meeting points, lifts or stairs (4 corners), but not when they are inside specific shops.



(a) Gradient (b) Binary

Figure 4.1: Amorphous Areas

Binary Matrix and Gradient Matrix

The Amorphous Areas (i.e. anchor areas without a regular shape) are implemented using a matrix. A matrix is a discretisation of the real space where the nodes reside (i.e. the environment). A piece of information uses the matrix to know the position of the host inside amorphous area. Given a host position, the Binary matrix (Figure 4.1(b)) denotes only whether or not a position is inside the amorphous area. Moreover, the gradient matrix (Figure 4.1(a)) adds gradients to denote the relevance inside the anchor area, and thus, providing a way to coordinates the movements of information to more relevant positions inside the amorphous area. Thus, the Binary Matrix is defined by $(matrix \subset \mathbb{E}\, x\, \{0,1\})$ where 0 represents positions outside of the area and 1 positions inside the area. The gradient Matrix is defined by $(matrix \subset \mathbb{E}\, x\, \mathbb{N})$. Where $\mathbb{E}$ is the whole environment considered, i.e. the set of all geographic coordinates where mobile nodes can move. Analogously to the binary Matrix, 0 represent positions outside of the area, however, those positions inside the anchor are defined using a gradient, where higher values represent more relevant positions inside the area and lower values are located close to the border.

The matrix has the size of the space of the area we consider. For instance, let us consider a museum area of 300m x 300m. The grid size is 300 x 300: one point in the matrix refers to one meter in the real word. For the binary approach, each position in the matrix contains the information bit 0 or 1. A piece of information, which wants to know if it is in the anchor area, checks its position and looks in the matrix to know if its position is inside the area (1) or outside (0). For the gradient area, we use a matrix too, but now the information in each point is a byte. The 0 value refers to the area outside the anchor area. The value increases when the replica is inside the anchor area, 255[4] being the inner part of the anchor area (darker shade on the pictures). Thus, the center of the corridor has a value of 255 and also the center of the halls.

The actual set of coordinates that are in an amorphous area is given by:

$$A(matrix) = \{b \in \mathbb{E} \mid ((b, val) \in matrix \ \wedge \ val \neq 0)\}.$$

[4]this could be less: 4, 8, 16, etc.)

We consider that the overhead for storing the matrix is neglictible compared to the actual data (few bits for each square meter).

4.3.3 Assumptions

Pieces of hovering information have the following information (at any point in time t):

- Knowledge of the anchor area (either a pair $A = (a, r)$, anchor location and radius; or a binary or gradient matrix);
- Position of the node where it is currently in;
- Position of neighbouring nodes and which of them has the information already.

We make the assumption that the information itself is more expensive to spread around than getting information about position and data id stored by neighbouring nodes. Due to the dynamic nature of the nodes, a hovering information service provides a best effort service accommodating imprecise positions or unexpected movement of nodes.

4.4 Hovering Information Algorithms

In this section different replication algorithms are proposed based on the Replication Pattern (Section 2.3.2) and Repulsion Pattern (Section 2.3.5). To the existing replication algorithms in the Hovering Information project, we contribute with the repulsion mechanisms. The repulsion achieves a better distribution of the pieces of information, reducing the number of replicas necessary to ensure the accessibility in the area. This section describes the existing replication mechanisms in Hovering Information (Replication with Broadcast and Replication with Attractor Point) and their extension using the replication mechanism.

4.4.1 Replication with Broadcast

At every simulation step (every second of simulation time) each piece of information executes the Broadcast replication algorithm. In a real implementation, pieces of hovering information would apply the algorithm at a regular interval (e.g. every second), but not in a synchronous way as it is for the simulations. In the broadcast algorithm, the replication is triggered when the information is inside the area (Algorithm 1). Broadcast replicates to all nodes in the communication range that do not hold a replica. Although this process can be viewed as a kind of multicast, we prefer to call it broadcast because the effect is to put a replica in each node. Broadcast spreads data in the neighbourhood (to those nodes that are not yet storing the information). There is no selection of a subset of nodes based on their geographical position as in geocast.

```
pos ← NodePosition();
if (IsInAnchorArea(pos)) then
    Broadcast();
end
```

Algorithm 1: Broadcast Replication Algorithm

Notice that when the replication is implemented using broadcast, it can be misunderstood with the spreading pattern (Section 2.3.1). However, we defined it as replication because the replica of information is not only sending an information, it involves an information service that is created in the destination node. Replication involves a copy of the agent (i.e. piece of information) and the replica offers the same services than the original agent. More details about the differences between Replication and Spreading Patterns are presented in Section 2.3.2.

4.4.2 Replication with Attractor Point

The Attractor Point algorithm avoids broadcasting to all neighbouring nodes. At each simulation step, a piece of information replicates only to the Kr nodes in communication range that present highest level of gradients. Then, the Replication with Attractor Point needs a gradient matrix to be implemented and can not be applied with a binary matrix. The algorithm is triggered when the information is inside the area (Algorithm 2).

```
pos ← NodePosition();
neigbourNodes ← NodeNeigbours();
if (IsInAnchorArea(pos)) then
    selectNodes ← SelectKrHighestGradient(neigbourNodes);
    Multicast(selectNodes);
end
```

Algorithm 2: Attractor Point Replication Algorithm

4.4.3 Cleaning

A piece of hovering information located outside of the amorphous area removes itself and frees the memory of the node in which it was stored. The cleaning algorithm ensures that the hovering information pieces stay in the anchor area and do not spread all over the environment. Additionally, we consider that at most one replica of the same piece of hovering information is stored in a given node at a certain point in time.

4.4.4 Repulsion

The repulsion mechanism provides motion coordination to spread the pieces of information over the anchor area achieving a uniform distribution even the the

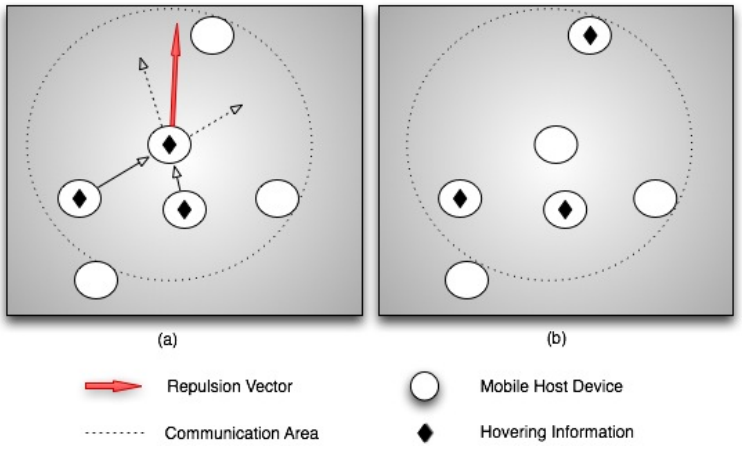


Figure 4.2: Repulsion

hosts are moving. The uniform distribution of the pieces of information produces good accessibility levels while keeping a minimum number of replicas (in order to use less memory). If two or more replicas are close to each other, one of them will move away (removing itself from its current location and replicating further away). Additionally, repulsion enables to easily fill amorphous shapes: the information spreads along the shape until it fills it. The repulsion mechanism, inspired by Flocking behavior, has been used in self-repairing formation for swarms of mobile agents [Cheng et al., 2005] and as an exploration mechanism in multi-swarm optimisation algorithms [Fernandez-Marquez and Arcos, 2009b]. The main difference between our system and the self-repairing formation for swarms of mobile agents is that pieces of hovering information do not have any control over the movement of the mobile nodes. Then, hosts produce a non uniform discretisation of the environment where the pieces of information can be located. Moreover, the hosts' movements involve that the available hosts' position to store a piece of information is changing continuously. Notice that the repulsion algorithm does not change the number of replicas in the systems.

Figure 4.2(a) shows how a replica creates a repulsion vector inversely proportional to the distance between itself and neighbouring replicas, and how this replica subsequently moves to another node following the repulsion vector, as shown in Figure 4.2(b). Contrarily to the previously described Broadcast and Attractor Point, the replica, which applies the repulsion mechanism, removes itself from the current node.

Let h be a piece of information, r a replica of h, and $n(r)$ the mobile node where r is currently located. Using the repulsion mechanism, the desired position for r at time $t+1$, $\vec{P}_d(r)_{t+1}$, is calculated as follows:

$$\vec{P}_d(r)_{t+1} = \vec{P}(r)_t + \vec{R}(r)_t, \tag{4.1}$$

```
pos ← NodePosition();
neigbourNodes ← NodeNeigbours();
if (IsInAnchorArea(pos)) then
    CalculateDesirePositionbyRepulsion();
    equations (4.1), (4.2)
    MoveToNodeClosestDesirePosition();
    if (¬ExistPieceOfInformation(neigbourNodes)) then
        Broadcast();
    end
end
```

Algorithm 3: Broadcast with Repulsion - Binary matrix

where $\vec{P}(r)_t$ is the position of r at time t and $\vec{R}(r)_t$ is the repulsion vector at time t.

$$\vec{R}(r)_t = \sum_{i \in R(r,t)} \frac{\vec{P}(n(r))_t - \vec{P}(n(i))_t}{dist(n(r), n(i))} \times (r_{comm}(n(r)) - dist(n(r), n(i))) \quad (4.2)$$

where:

- $R(r,t)$ is the set of replicas of h in communication range of $n(r)$ at time t;
- $dist(n(r), n(i))$ is the Euclidean distance between the node $n(r)$ and the node $n(i)$;
- $\vec{P}(n(j))_t$ is the position of node $n(j)$ where the replica j is stored at time t;
- $\times$ is the multiplier operator; and
- r_{comm} is the communication range.

Once the desired position $\vec{P_d}(r)_{t+1}$ is known, the replica r must choose which node in its communication range is the closest to the desired position. If the closest node is itself, then repulsion is not applied. Otherwise r replicates to the new node and deletes itself from $n(r)$. Basically, these equations produce a repulsion vector that moves the replica of information to the less dense area inside its communication range.

4.4.5 Broadcast Repulsion

Broadcast with Repulsion (see Algorithm 3) uses broadcast as a replication algorithm and repulsion to spread away the replicas. A piece of information replicates to all the neighbours when it is inside the anchor area and there are no other replicas in the communication range. Moreover, a piece of information executes

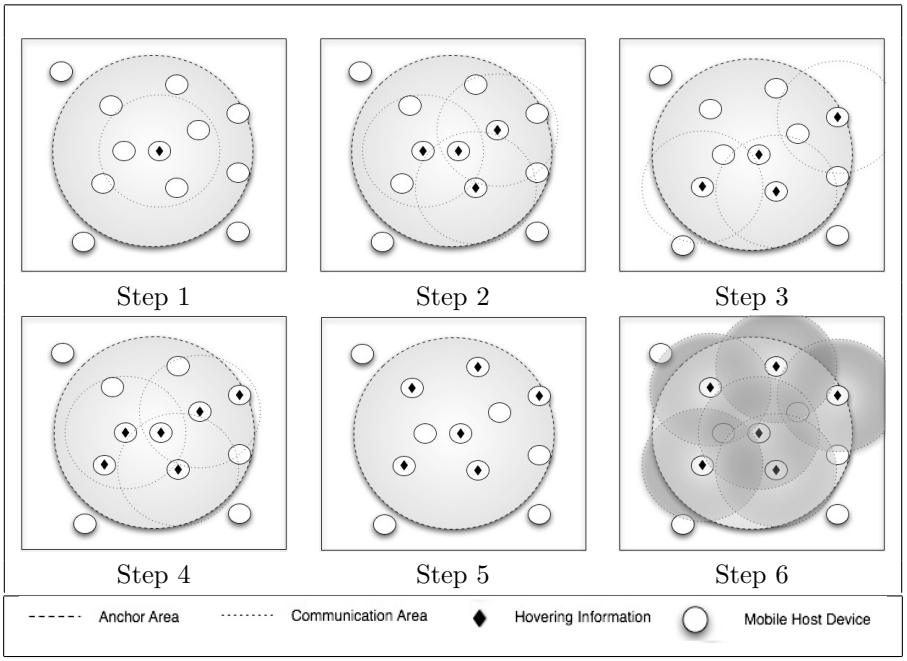


Figure 4.3: Broadcast Repulsion Steps

repulsion when it is inside and there is one o more replicas in the communication range. Thus, the replication with broadcast creates the replicas necessary to cover the shape and the repulsion is the responsible of distributing the pieces of information uniformly over the area. Broadcast Repulsion is applied to the Binary Matrix, since the broadcast replicates to all the neighbours and does not use the gradient information.

The main steps of the algorithm are shown in Figure 4.3. At the beginning, Step 1, the first piece of information is created and the anchor area is defined. This piece of information executes the broadcast, since there is no other piece of information in its communication range. Step 2 shows how the nodes that are in communication range get a copy of the piece of information. At Step 3, the pieces of information spread away due to the repulsion algorithm. After the repulsion step, two of the three replicas have no other replica in their communication range, then they execute the Broadcast algorithm and replicate to all the nodes in their communication range, Step 4. At Step 5 the pieces of information spread further due to the repulsion and finally fill the anchor area. Step 6 shows how, without getting stored in every node, the initial piece of information becomes available to all nodes in the anchor area (i.e. it “fills” or “covers” the whole anchor area).

```
pos ← NodePosition();
neigbourNodes ← NodeNeigbours();
if (IsInAnchorArea(pos)) then
    CalculateDesirePositionbyRepulsion();
    equations (4.1), (4.2)
    MoveToNodeClosestDesirePosition();
    if (¬ExistPieceOfInformation(neigbourNodes)) then
        selectNodes ← SelectKrHighestGradient(neigbourNodes);
        Multicast(selectNodes);
    end
end
```

Algorithm 4: Attractor Point with Repulsion - Gradient Matrix

4.4.6 Attractor Point Repulsion

Attractor Point with Repulsion (see Algorithm 4), analogously to the broadcast repulsion, uses replication, and repulsion to spread away the replicas. However, In Attractor Point with Repulsion, instead of replication to all nodes in the communication range, a piece of information replicates to the Kr neighbouring nodes with higher level of gradients when it is inside the anchor area and no other replicas are present in the communication range. The use of gradient provides a better location of the new replicas created. Analogously to the Broadcast Repulsion algorithm, a piece of information executes repulsion when it is in the anchor area and no more replicas are present in the communication range. Then, the algorithm behaviour is similar to the Broadcast Repulsion, but, Attractor Point repulsion reduces the number of replicas created in each replication process. The Attractor Point Repulsion needs the gradient matrix to choose the Kr nodes with highest gradient.

4.5 Simulation Results

We conducted diverse simulations involving the algorithms described above for different scenarios. For each algorithm, the performances are measured and compared against a collection of metrics. We used the Recursive Porous Agent Simulation Toolkit (REPAST)[5], a Java environment for agent simulations. In addition to performance measures, it provides an event scheduler to simulate concurrency and a two-dimensional agent environment that we used to visualise the nodes and the spreading of the pieces of hovering information in the environment.

Although the WLAN 802.11 standard provides communication ranges up to 70m for indoors and 250m for outdoors, most authors (e.g. [Roth, 2003]) consider that this is too high for realistic situations, and propose smaller values.

[5] http://repast.sourceforge.net/

Blackboard	1200m x 700m
Mobility Model	Random Way Point
Nodes speed	1m/s to 2m/s
Communication Range	40m or 80m
Replication Time (T_R)	1s
Repulsion Time	10 s
Cleaning Time (T_C)	1s

Table 4.1: Scenario's Settings

We defined two scenarios: The *indoor scenario* where the communication range is 40 meters and the *outdoor scenario* where the communication range is 80 meters. The anchor area used is presented in Figure 4.1 and the mobility model used for the nodes is Random Way Point. This mobility pattern, in contrast to other patterns like Random Direction or Random Walk, creates the highest density of nodes at the center of the environment. By locating the anchor area at the center, a similar concentration of nodes across our different experiments is ensured, i.e. we can infer meaningful conclusions. The goal of the simulations is to compare Attractor Point (AP), Broadcast (BB), Attractor Point Repulsion (APR) and Broadcast Repulsion (BBR) in the different scenarios proposed where the goal is to ensure high accessibility levels with a minimum number of messages and a minimum number of nodes actually storing the information.

Table 4.1 summarises the parameter settings for the scenarios simulated. For each experiment, we executed 50 runs, each run spending 1000 simulation seconds. The results presented are the average over these 50 runs. For the simulations, we used a synchronous model, i.e. we measured the metrics by analysing mobile devices activity (e.g. how many times they received, replicated, discarded data) during a specific period of time.

4.5.1 Metrics

To measure the performance of each algorithm in the different scenarios proposed, we defined a collection of metrics focused on: the survivability of pieces of information, the accessibility of information in the anchor area, the number of messages sent, and the memory used (i.e. number of replicas). To define the metric we use the following notation: $HoverInfo_t = (\mathcal{N}_t, \mathcal{H}_t)$ is a Hovering Information System at time t. $\mathcal{N}_t$ is the set of nodes present in the system at time t. $\mathcal{H}_t$ is the set of pieces of hovering information and any replica present in the system at time t. Since we consider only one piece of information in this work, $\mathcal{H}_t$ stands for h and all its replicas (same identity and data but stored at different nodes). Note that h itself is just a specific replica.

Survivability

A piece of hovering information is alive at some time t if there is at least one node hosting a replica r of this information. The survivability of h at time t is given by the boolean value:

$$sv(t) = \begin{cases} 1 & \text{if } \exists r \in \mathcal{H}_t, n(r) \in \mathcal{N}_t \\ 0 & \text{otherwise.} \end{cases}$$

The survivability along a period of time is defined as the ratio between the amount of time during which the hovering information has been alive and the overall duration of the observation. Specifically, the survivability of h between time t_c (creation time of h, always 0 in this work) and time t is given by:

$$SV(t) = \frac{1}{t - t_c} \sum_{\tau=t_c}^{t} sv(\tau).$$

Accessibility

A piece of hovering information h is accessible by a node n at some time t if the node is able to get this information, i.e. if it exists a node m being in communication range of the interested node n containing a replica of h. Let $n \in \mathcal{N}_t$ be a mobile node, the accessibility of h for n at time t is given by the boolean value:

$$ac(n,t) = \begin{cases} 1 & \text{if} : \exists r \in \mathcal{H}_t, n(r) : in\ A_C(n) \\ 0 & \text{otherwise.} \end{cases}$$

where, $A_C(n)$ is the area in communication range of node n.

The accessibility of h at time t is given by:

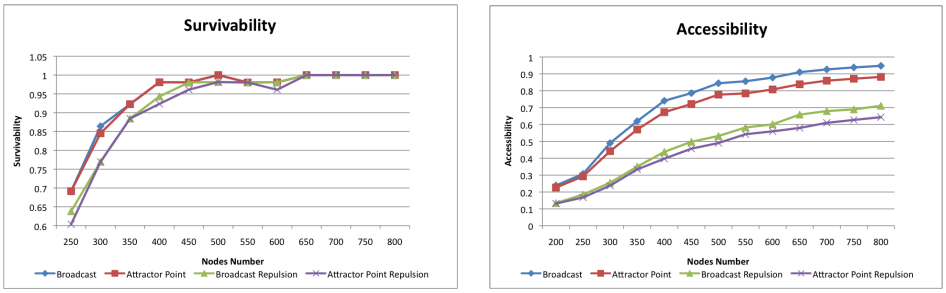
$$AC(t) = \frac{S(\bigcup_{r \in \mathcal{H}_t} A_C(n(r)) \cap A)}{S(A)}$$

where $S(X)$ denotes the surface area of X, $A_C(n(r))$ is the area of communication of the node n storing replica r, and A is the anchor area.

Messages

A message is sent each time a replica self-replicates to another node. Let $msg(n,t)$ be the number of messages sent by node n between 0 and t, the number of messages sent at time t is given by:

$$MSG(t) = \sum_{n \in \mathcal{N}_t} msg(n,t)$$



(a) (b)

Figure 4.4: Survivability and Accessibility - Indoor scenario

Memory

The memory that the system uses at time t is the total number of replicas in the system at time t: $|\mathcal{H}_t|$. Then, the Rate of Memory used at time t is:

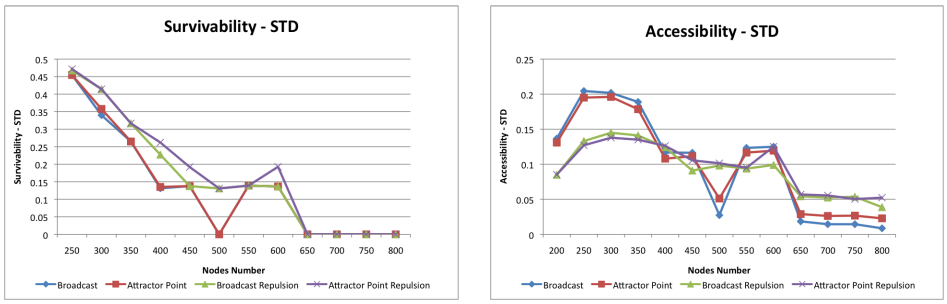
$$MEM(t) = \frac{1}{t - t_c} \sum_{\tau=t_c}^{t} |\mathcal{H}_\tau|$$

4.5.2 Indoor Scenario

In indoor scenarios the communication range of the nodes is set to 40m. The anchor area considered is the one shown in Figure 4.1 (4 corridors and 4 corners), localised in a bi-dimensional space of 1200m by 700m, and the minimum number of nodes is above 200.

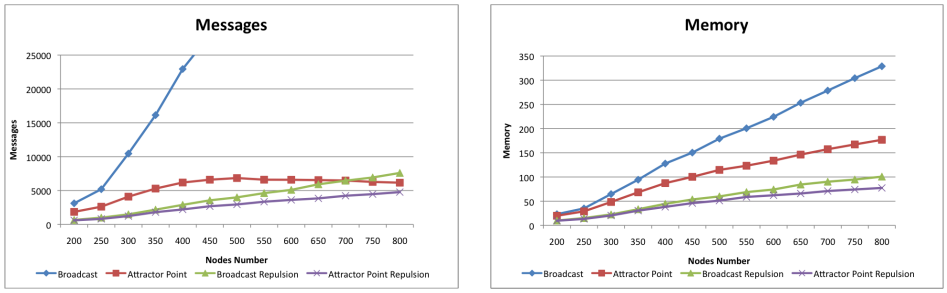
Figure 4.4 shows survivability and accessibility rates of the four algorithms considered in this section. Broadcast and Attractor Point outperform their respective variants with repulsion since more nodes store the data. However, this outperform in the accessibility produces an increment in the number of replicas, as shown in figure 4.6(b).

Figure 4.5(a) shows the standard deviation (STD) related to survivability. We may observe how STD is going down when the number of nodes increases. That is, the probability of failure decreases when the number of nodes increase. When the number of nodes is higher than 600, the STD tends to 0, i.e. at least 600 nodes are necessary to ensure the survivability of the hovering information along the simulation. Figure 4.5(b) shows the STD related to accessibility. The best STD is achieved when the number of nodes in the system is enough to ensure the survivability of the hovering information during the whole simulation. Notice that in both above mentioned figures an “accidental” increase of the STD may be observed at 550 and 600 nodes, because one of the 50 runs failed to reach survivability. This effect is also visible in Figure 4.4(a), a small drop in survivability for 550 and 600 nodes.



(a) (b)

Figure 4.5: Survivability and Accessibility STD - Indoor scenario



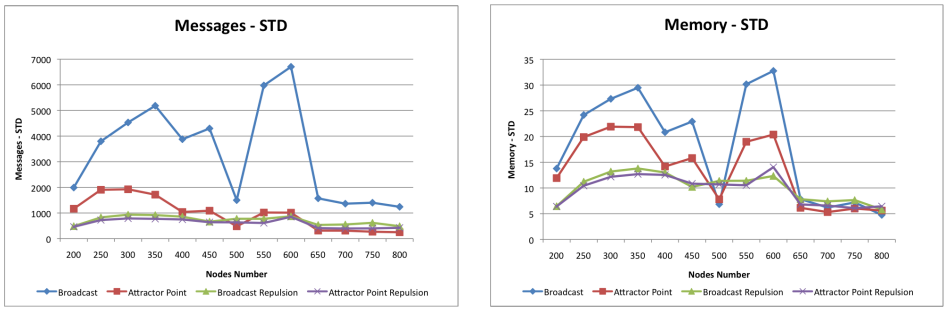
(a) (b)

Figure 4.6: Messages and Memory - Indoor scenario

Figure 4.6(a) shows that Broadcast employs much more messages and storage memory than the other algorithms. It is clearly outperformed by the rest.

Figure 4.7(a) shows the STD of messages achieved by the different algorithms in the amorphous area. Except Broadcast that presents a high STD, the rest of the algorithms present an STD lower than 10%. Figure 4.7(b) shows the STD related to memory. All the algorithms reach a STD for memory lower than 8% when the system has enough number of nodes to sustain the data. Here again, we observe that the same runs that affected the STD in Figures 4.5(a) and (b) affect also Figures 4.7(a) and (b) for 550 and 600 nodes. As it occurs in all the STD figures presented in this work, a low number of nodes involves an unstable survivability, producing an increment in the STD.

The Attractor Point with Repulsion outperforms its variant without repulsion for both the number of messages and the memory storage. This result is motivated by the fact that the repulsion variant replicates only when there is no other replica in the communication range.

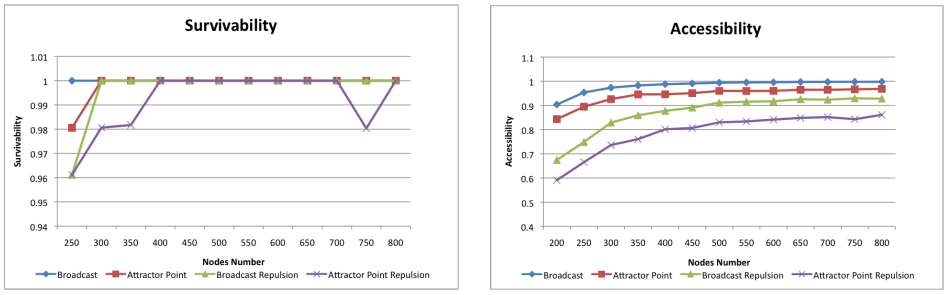


(a) (b)

Figure 4.7: Messages and Memory STD - Indoor scenario

4.5.3 Outdoor Scenario

In the outdoor scenario, the communication range is set to 80 meters. Figure 4.8 shows survivability and accessibility rates for the outdoor scenario. Because of the increased communication range, both rates are better for all algorithms. Broadcast and Attractor Point still outperforming (even though slightly) the repulsion variants for accessibility rates.



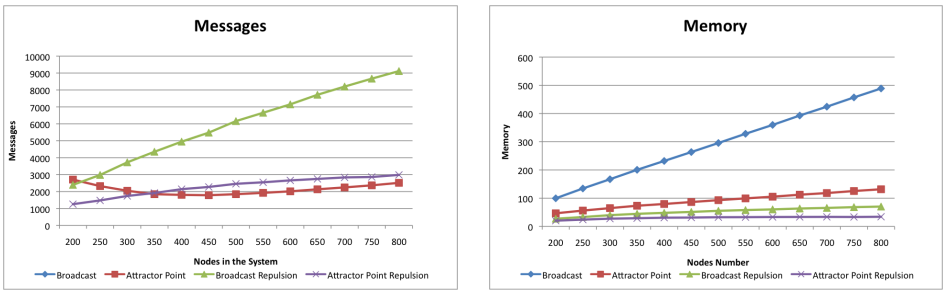
(a) (b)

Figure 4.8: Survivability and Accessibility - Outdoor scenario

Analogously to the indoor scenario, Broadcast is clearly outperformed by the other three algorithms for both the number of messages sent around and the memory used. To facilitate the understanding, the messages for Broadcast are not included in Figure 4.9. Attractor Point (both variants) presents similar levels of messages whereas the repulsion variant is using less memory.

4.5.4 Analysis of algorithms

This section investigates scalability issues, the impact of the repulsion rate on the results, cases where a newly created piece of information does not succeed



(a) (b)

Figure 4.9: Messages and Memory - Outdoor scenario

to spread and dies almost immediately, the minimum number of nodes below which the system cannot work, and a recovering scenario in case of massive failures of nodes. More extended study about the failures can be found in [Fernandez-Marquez et al., 2010].

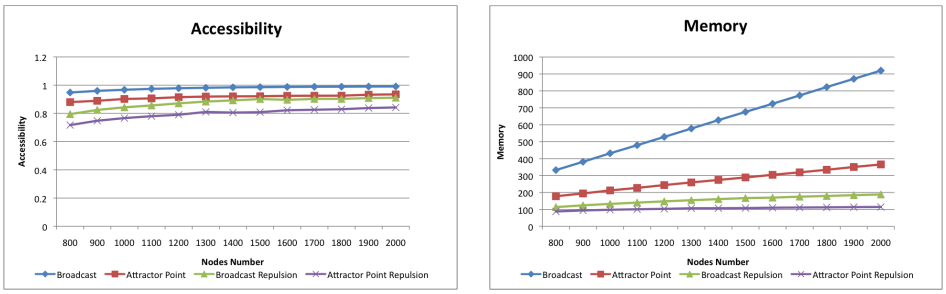
Scalability

In this section we study the behaviour of the algorithms for very large numbers of nodes (up to 2000). This experiment involves the anchor area of Figure 4.1 and a communication range of 40m. We varied the number of nodes from 800 to 2000. Analogously to the other experiments, the Attractor Point and Attractor Point with Repulsion are using the gradient matrix (Figure 4.1(a)), while the Broadcast and Broadcast with Repulsion are using the binary matrix (Figure 4.1(b)).

The four algorithms achieved very good accessibility rates due to the very large number of nodes involved, as shown in Figure 4.10(a). Broadcast and Attractor point present better accessibility than the variants with repulsion, since they use also more memory, as shown in Figure 4.10(b). The main point is that the repulsion variants are scalable regarding memory consumption. The memory levels remain constant despite the higher number of nodes.

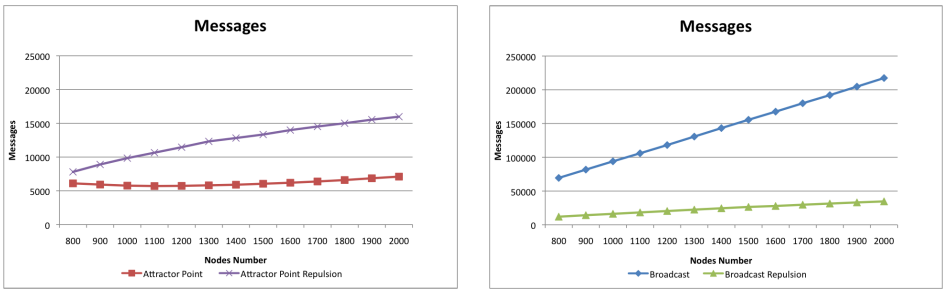
The main drawback of applying repulsion is the high number of messages sent around when comparing with the variants without repulsion. Attractor Point (without repulsion) is scalable in terms of messages: the number of messages remains constant even though the number of nodes increases (Figure 4.11(a)). Broadcast (both variants) is not scalable at all in terms of messages (Figure 4.11(b)).

To conclude, for large numbers of nodes, Attractor Point scales better in terms of messages than the Attractor Point with Repulsion, but employs more memory (up to four times more). The repulsion variant scales in terms of memory but uses far more messages. It is also worth noting that the Attractor Point doesn't use any mechanism to spread the information over the area. The area gets filled as a result of the movements of nodes. Nodes help the algorithm in



(a) (b)

Figure 4.10: Accessibility and Memory - Scalability Indoor scenario



(a) (b)

Figure 4.11: Messages - Scalability in Indoor scenario

spreading the information. The repulsion version ensures that the information spreads over the area, even when the nodes are stationary or less mobile. Broadcast (without repulsion) is not scalable and its variant with repulsion must be preferred.

Repulsion Interval

An important parameter of the repulsion mechanism is the repulsion interval, i.e. the time between two repulsion executions. When the repulsion interval is shorter, the number of messages and the accessibility rate increase. When the repulsion interval is long, the number of messages decreases, but also the accessibility rate.

In this experiment we computed the accessibility rate, the number of messages, and the memory usage along 1000 simulation steps, over the average of 50 runs for Broadcast with Repulsion and Attractor Point with Repulsion. For this experiment we set the number of nodes to 500 and the communication range to 40m. We considered the same anchor area as above.

Figure 4.12 shows that accessibility rates are better with a shorter repulsion

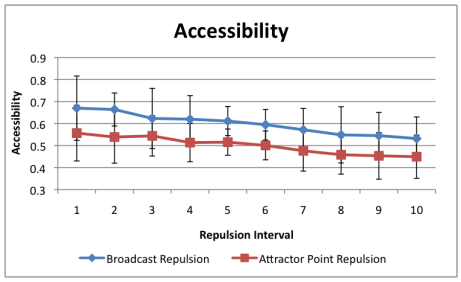
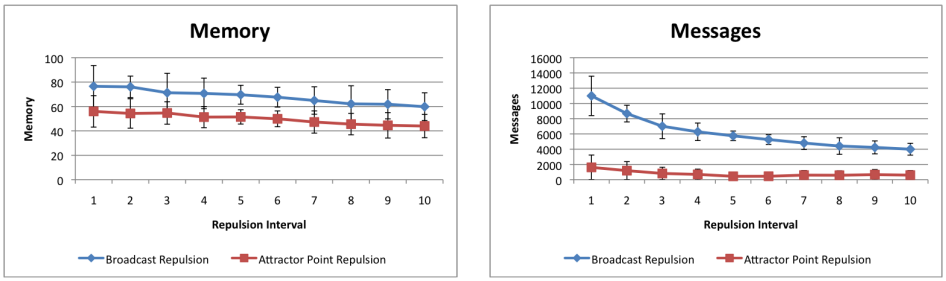


Figure 4.12: Accessibility - Varying the repulsion interval



(a) (b)

Figure 4.13: Memory and Messages - Varying the repulsion interval

interval (every 1s or 2s), because repulsion spreads faster the pieces of information over the area. Repulsion adapts quickly to topology changes by moving around the replicas. When we increment the repulsion interval, repulsion applies less frequently (every 4s or less), it adapts less quickly to topology changes, and accessibility rates decrease. For memory consumption, a shorter repulsion interval causes more nodes to store replicas, while a longer repulsion interval decreases the memory used. A shorter repulsion rate increases the number of time repulsion applies, i.e improves the exploration of the space and is able to find better places wherein to replicate increasing accessibility. The drawback is a higher memory consumption, as shown in Figure 4.13.

The main issue with a shorter repulsion interval is that the increment in the number of messages is not proportional to the accessibility rate. In Figure 4.13(b), we may observe that for short repulsion intervals, the number of messages increases. This increment is not linear like the accessibility or the memory values. The gain in accessibility is weak compared to the large number of additional messages needed to reach that level. Additionally, the repulsion interval depends on the dynamics of the network. When the network is very dynamic, i.e. the topology is continuously changing, a short repulsion interval allows to keep up with the change. For less dynamic networks, a short repul-

sion interval causes replicas to move around continuously without improving the performances.

Convergence

In this section our goal is to analyze the velocity of convergence when covering the whole area for each algorithm. We measure first the convergence at the initialisation and second we investigate the self-healing property of the system when, due to an extreme case of nodes failure, a large number of nodes are disconnected, all at once, and the system has to converge again to fill the area. Simulations run over the anchor area given by Figure 4.1. We set the number of nodes to 500 and we measured the accessibility at different simulation steps. The executions run over 1000 simulations steps and at the 500th simulation step, nodes in a portion of the area are manually disconnected, forcing the system to converge again. We performed two sets of simulations: one for the indoor and another one for the outdoor scenario. We consider that the system has converged in covering the whole area when the accessibility rate is higher than 0.8. That is, when 80% or more of the nodes in the anchor area have access to the information.

Indoor Scenario. Figure 4.14 shows that Broadcast provides the best convergence speed. This algorithm reaches the accessibility of 0.8 at the 150th simulation step. This very fast convergence is due to the fact that the information replicates to all the nodes and thus the area gets filled very quickly. The cost is the high level of messages and memory usage (Figure 4.11). Attractor Point reaches the accessibility of 0.8 at the 280th simulation step (Figure 4.14). Broadcast with Repulsion and Attractor Point with Repulsion need more than 500 simulation steps to reach an accessibility of 0.8.

At the 500th simulation step, nodes in a portion of the area are disconnected and the accessibility suddenly drops. The speed of convergence after the failure is similar for the four algorithms. As we may see in Figure 4.15(b), the four algorithms follow the same slope. That is, the time that the MANET takes to fix the network after the failure, i.e. the time required to repopulate the area. Whatever the speed of convergence of the algorithm, after a failure in the network, the speed of convergence is limited by the speed of the nodes in repopulating the area, i.e. the speed of convergence decreases due to the lack of nodes to fill the shape.

Outdoor Scenario. With a communication range of 80 meters, the convergence speed increases. Figure 4.16(a) shows that the Broadcast algorithm reaches 80% of accessibility at the 8th simulation step, Attractor Point at the 65th, Broadcast with Repulsion at the 85th and Attractor Point with Repulsion at 150th. The more memory an algorithm consumes, the quicker it converges. When the failure occurs (Figure 4.17(b)) we may observe that the convergence speed is similar (same slopes).

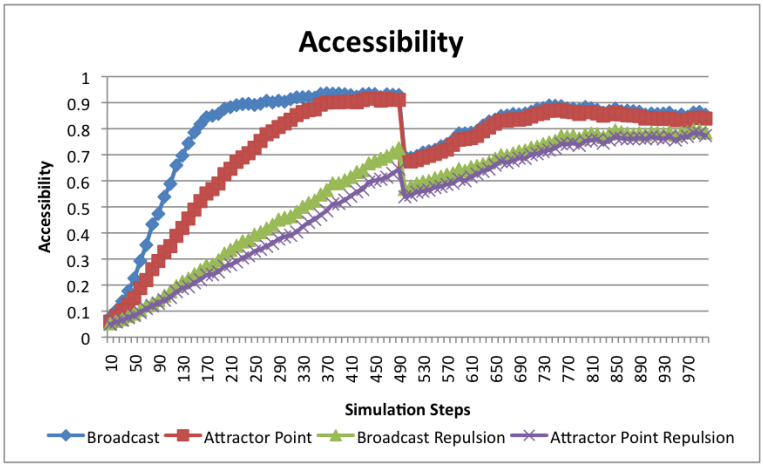
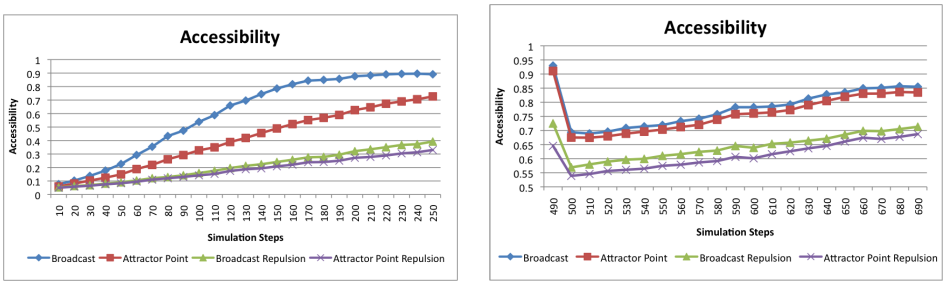


Figure 4.14: Accessibility - steps 0 to 1000 - Indoor Scenario



(a) Steps 0 - 250 (b) Steps 490 - 700

Figure 4.15: Accessibility with zoom - Indoor Scenario

Faults - Initialisation phase

One of the key issues we encountered was the initialisation of the information, i.e. the period of time between the creation of a piece of information and the moment the information covers the whole area. In many cases, due to a lack of nodes in the neighbourhood or to random movements, the piece of information *cannot* replicate and dies before the anchor area is fully covered. We observed that once the initialisation phase succeeds, i.e. once a large part of the anchor area is covered with replicas, the system gains in robustness. Indeed, the probability that *all* the replicas leave the anchor area without replicating to neighbouring nodes is lower than during the initialisation process (when only one or few replicas are available). During the initialisation phase, the system is very fragile and sensitive to random movements.

In this experiment we executed each algorithm 5000 times and each run spent

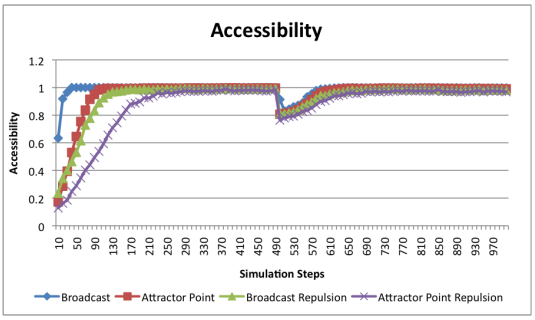
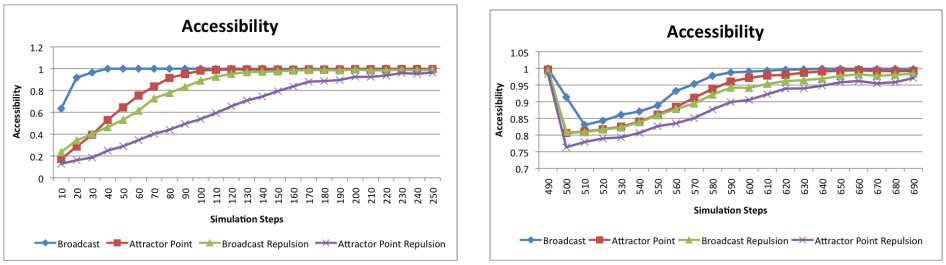


Figure 4.16: Accessibility - steps 0 to 1000 - Outdoor Scenario



(a) Steps 0 - 250 (b) Steps 490 - 700

Figure 4.17: Accessibility with zoom - Outdoor Scenario

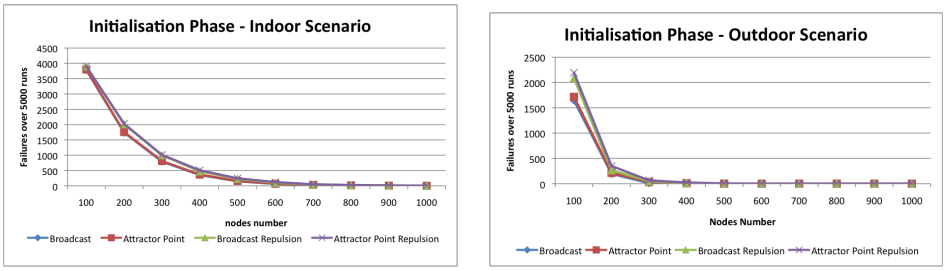
500 simulation seconds, i.e. enough time to ensure that the information was initialised successfully. Over the 5000 runs, we counted the number of times the information dies before the end of the run, corresponding to system failures. We studied the 4 algorithms: Broadcast (BB), Attractor Point (AP), Broadcast with Repulsion (BR), Attractor Point with Repulsion (APR). The number of failures is reported in Table 4.2 (Indoor and Outdoor scenarios) and Figures 4.18(a), 4.18(b).

The indoor case is clearly more sensitive to initial conditions than the outdoor case because of the shorter communication range. The likelihood that random conditions prevent a piece of information to replicate to neighbouring nodes is higher in indoor scenarios. Outdoor, the repulsion variants fail to initialise more frequently than their counterpart without repulsion. Indoors, failures are comparable among the four algorithms. Finally, the more the nodes in the environment, the less the risk of failure during the initialisation phase.

Broadcast and Attractor Point ensure a good survivability rate. They also present a better convergence speed. The price is a higher level of memory storage. Broadcast with Repulsion and Attractor Point with Repulsion present a very low use of memory. For this reason, the best option is to use algorithms like Broadcast or Attractor Point during the initialisation of the system. Later,

	Indoor Scenario				Outdoor Scenario			
Nodes Number	BB	AP	BR	APR	BB	AP	BR	APR
100	3780	3799	3897	3900	1646	1716	2088	2194
200	1738	1757	2009	2028	195	213	267	350
300	791	811	998	1012	12	34	48	65
400	354	368	472	509	0	7	9	24
500	147	160	226	244	0	1	0	5
600	64	69	102	123	0	0	1	1
700	27	32	37	50	0	0	0	2
800	14	16	21	27	0	0	0	1
900	5	6	11	13	0	0	0	2
1000	2	2	6	7	0	0	0	3

Table 4.2: Init Test in Indoor and Outdoor Scenarios (amorphous()



(a) Indoor (b) Outdoor

Figure 4.18: Faults - Initialisation Phase

the information could switch to Broadcast with Repulsion or Attractor Point Repulsion in order to keep the use of memory low.

Faults - Critical mass of nodes

We focus here on analyzing the minimum number of nodes that the system needs in order to keep the information alive (survivable) during the whole simulation time. In this experiment we started with a large number of nodes, 600, in order to ensure the correct initialisation of the system (and to avoid initialisation failures discussed above). At every 15000 simulation seconds, one node is removed at random. As time goes by, the probability of failure increases (i.e. the probability of the information failing to survive). Once all replicas have disappeared from the system, we count the remaining number of nodes in the system. We run each algorithm 50 times and we present the average and deviation (Table 4.3). This information provides the minimum number of nodes below which the system is not viable. We observe that the repulsion variants need more nodes than their variants without repulsion. Experiments with another shapes have demonstrated

	Indoor Scenario		Outdoor Scenario	
Algorithm	Nodes Number	Std. Dev.	Nodes Number	Std. Dev.
BB	167.63	7.86	93.83	5.12
AP	169.02	7.86	98.92	5.02
BR	185.83	10.35	111.98	6.61
APR	186.20	10.63	116.18	6.79

Table 4.3: Nodes limit in Indoor and Outdoor Scenarios

that small anchor areas emphasise this difference.

4.6 Conclusions

This chapter proposes a viral programming approach to perform persistent storage of data at specific spatial locations on top of mobile computing devices. The main goal is to make the data accessible to a maximum number of users inside a required area, without flooding all nodes with the data, and without sending a prohibitive number of messages among the nodes (i.e. using the minimum number of replicas and messages sent).

We investigated four algorithms, Broadcast, Attractor Point, and their respective variants with Repulsion (to avoid replicating at nearby locations). All the algorithms proposed implement the replication pattern and two of them combine the replication with the repulsion to get a better distribution of the pieces of information in the area. Results show that Broadcast converges very quickly, but needs high levels of memory and employs more messages than the Attractor Point. The variants with Repulsion consume less memory and their memory consumption is scalable, but the number of messages among nodes is very high. Attractor Point (without repulsion) does not outperforms the other algorithms for all metrics, but reaches high levels of accessibility even for low number of nodes in the area, converges rather quickly in covering the whole area, employs less messages, and slightly more memory than its variant with Repulsion.

For anchor areas both indoor and outdoor, Attractor Point (both versions) is clearly the best options. Specifically, for higher availability rates (above 95%) Attractor Point should be preferred whereas for acceptable lower availability rates (80%) and less memory storage, Attractor Point with Repulsion is the best option.

In the Appendix, we provide some simulation images using different anchor areas for all the algorithms analysed in this chapter. Pictures show how the distribution of pieces of information improve when repulsion is used, and how the repulsion variants use less memory (number of nodes storing pieces of information) than the variants without repulsion. Moreover, we provide some simulation images for these algorithms when the nodes fail in a sub-area. The images show the faster refill after the failure.

We may see how the convergence speed varies for these algorithms. The

Attractor Point with Repulsion takes more time to spread replicas over the whole area (280 seconds of simulation time), while Broadcast quickly fills the shape (at the 18th simulation second already). The difference in the number of nodes storing a replica (darker dots) is also clearly visible.

Notice that the Random Way Point model, used for the simulations, is such that the nodes are most of the time in the middle of the environment. So, when an anchor area is on the border of the environment, the system will not work well because of a lack of nodes. Alternatively, if the mobility model does not follow the Random Way Point model (e.g. Random Walk), the density of the nodes may not be guaranteed in the anchor area, and results will not be the same as those reported here.

Finally, the Repulsion and Replication Patterns implemented in the different algorithms reached a good performance in terms of robustness and adaptability in front of environmental changes.

Chapter 5

Detecting Diffuse Event Sources in Noisy WSN Environments

This chapter proposes and evaluates the use of the chemotaxis Pattern applied in sensor networks to localise dynamically changing diffuse events. Localizing dynamically changing diffuse event sources in real environments is still an open problem in Wireless Sensor Networks (WSN). The dynamism of the environment, the energy limitations of the sensors, and the noise associated to the sensors' measurements pose a challenge that a realistic solution has to deal with.

We propose a decentralised mobile agent approach to detect diffuse event sources in dynamic and noisy environments, using a Wireless Sensor Network infrastructure. In the search process, the agents' motion coordination is based on the Chemotaxis Pattern (Section 2.5.3). The chemotaxis pattern exploits the gradient created in the environment by the diffuse events, following a distributed and decentralised algorithm based on local interactions and local knowledge of the environment. Reported experiments show that our approach efficiently adapts in tracking the event sources as they appear, is scalable, and robust to noise and failures.

5.1 Introduction

The localisation of diffuse event sources and plumes is a problem that appears in a wide range of real applications such as toxic gas detection, detection of underwater leaks, or detection of acoustic and heat sources. Diffuse events are huge phenomena that can spread in a 2D or 3D space without a regular shape. A diffuse event consists of one *source* and its *plume*. The source is the focus of the event whereas the plume is the area or space the diffuse event covers. Plume sizes and shapes are constantly changing due to the environment dynamism that

acts over them (the wind, obstacles, ...).

In some scenarios, the source is fixed and does not vary with time, while the plume varies constantly. A recent example is provided by the eruption of the Eyjafjallajökull volcano in Iceland. The source is well known and somehow fixed, while the changing ash plume is the main point of concern. In other scenarios, the sources themselves vary (in location and number) over time and it is imperative to detect all of them as quickly as possible. For instance, in 2002 the Prestige tanker was damaged and began losing its cargo during a storm. The Prestige was carrying approximately 81.000 tons of oil. The oil spread over the sea near the Spanish and Portuguese coasts. Due to the wind and sea currents and the way the tanker sank, the oil split into several disjoint spots. The different spots of oil moved over the sea and continued splitting into new spots, rendering the recuperation of the oil and the cleaning process difficult. Ultimately, this accident lead to a huge ecological disaster, the oil spills stretching on more than 1000 km. The detection and tracking of the spots was a difficult task that could have been addressed with the use of sensor networks. Another real example of dynamically changing diffuse event sources are the bush fires in Australia in 2009. Because of the wind, embers were blown ahead of the fire front, new spot fires then started where the embers landed. In this particular example, the presence of smoke complicated the localisation of the main fire focuses. Infrared vision sensors, as used in the project Spread[1] have been used to localise hot temperature spots and to predict fire movements, thus demonstrating the usefulness of sensors in tracking fires. In scenarios where sources are dynamically changing, localizing as soon as possible all diffuse event sources is crucial (e.g. to avoid the spreading of toxic gas and possible large disasters). We consider that the sensor network and the localisation of diffuse event sources may play a key role in these kind of scenarios.

So far, approaches exploiting WSN, have essentially concentrated on detecting plumes using centralised algorithms [Ruairí and Keane, 2007], on detecting a single source (global optimum) in static and noise-free environments [Blatt and III, 2006, Ermis and Saligrama, 2006], or detecting multiple sources with sensors well distributed in the environment and following a centralised strategy [Weimer et al., 2009]. More generally, regarding the detection of static diffuse event sources in non-noisy environments, Ruair et al. [Ruairí and Keane, 2007] demonstrated that existing algorithms for target tracking do not scale well when they are applied to the localisation of diffuse events. These algorithms require that each sensor reports the data to the sink when it reads a sensor value higher than a threshold. Since diffuse events can cover large areas, a large number of sensors would try to report the data to the sink, producing a network overload.

To the best of our knowledge, the problem of detecting dynamically changing diffuse event sources in noisy WSN environments has not been addressed before. Our work focuses on the detection of diffuse event sources in *dynamic* and *noisy* environments. The main task is to detect not only the main event source (i.e.

[1] http://www.algosystems.gr/spread/index.html

location of the global optimum given for instance by the highest temperature or the highest density of oil) but also any residual event sources that may become new principal events (i.e. local optima becoming global optima). Thus, the goal is to detect *all* event sources *dynamically appearing* over time in the system. Additionally, any realistic solution to the problem has to deal with the imprecision related to sensors' measurements and the noise introduced by the environmental changes (e.g. weather conditions or ocean currents).

To track diffuse event sources, we consider sensor networks covering large areas created by a vast number of connected devices spread randomly in the environment. Despite the improvement in the technology, which has made possible the development of ultra-small fully autonomous and communicating sensors, one of the most important requirements in a WSN remains the design of energy-efficient algorithms able to extend the network lifetime [Vinyals et al., 2011]. A quick detection of dynamically changing diffuse event sources in large sensing areas requires decentralised self-organising approaches able to adapt to the dynamicity of the environment, robust to noise, and that scale without being greedy on energy consumption. This work proposes a decentralised multi-agent approach, based on the Chemotaxis Pattern, exploiting local interactions among sensors. Analogously with the chemotaxis process in bacteria (i.e swimming towards the highest concentration of food molecules), the mobile agents coordinate their movements towards the highest concentration of gradients. Thus, mobile agents move by hopping among the sensors until they reach a diffuse event *source*.

The chapter is organised as follows. First, we discuss related work. Then, we briefly explain the lower power listening mode assumed in this work for the sensors. Next, we describe our model and approach. Then, we report on simulations and discuss the performance of our approach in terms of messages sent, number of sensors' measurements, and resilience to noise and failures. We also performed a study on the impact of the parameters used.

5.2 Related Work

Localisation of diffuse event sources differs from target tracking [Yang et al., 2006] and environment monitoring [Corkill et al., 2007]. These related problems are concerned either with the prediction of object movements or with the creation of a model to monitor the changes in a specific area. We assume that diffuse events are phenomena whose behavior is unpredictable because of two main reasons: the environment dynamism and the high latency that WSN require to track objects. Moreover, the appearance of diffuse events cannot be predicted by any model.

The problem of localizing diffuse event plumes in a WSN has been addressed by Ruair et al. [Ruairí and Keane, 2007] who propose a MAS approach to map the contours of large diffuse events. Agents are distributed over a WSN playing different roles: an agent playing the *leader* role and operating on one sensor, and multiple agents playing the *member* role and operating on sensors adjacent to the location of the leader agent. Agents change their role by following a

gradient-based strategy, with the aim of covering an event's contour (plume). The proposed mechanism can be adapted to deal with multiple sources, but it has not been demonstrated to be enough for dynamic and noisy environments.

Blatt et al. [Blatt and III, 2006] and Ermis et al. [Ermis and Saligrama, 2006] proposed different algorithms to detect and localise sources that emit acoustic waves. They consider static and noisy-free environments, and their goal is to assess the global optimum value avoiding the local optima of the acoustic signals.

When the cost of the sensors is expensive, sensors are allocated strategically and a centralised solution produces really good results [Weimer et al., 2009]. When the data sampling periods are much larger than the communication time, a centralised approach for detection and localisation is feasible. Indeed, the time required to coordinate the nodes is smaller than the sampling time. This solution however does not scale to a large number of non expensive sensors spread randomly over the space, since we cannot assume that all nodes are sampling at each period.

Finally, as the main studies in dynamic multi-modal optimisation have demonstrated [Blackwell, 2007, Lung and Dumitrescu, 2007], in highly dynamic environments detecting only the global optimum is not sufficient because the diversity of the exploration is a required feature. A current trend in dynamic multi-modal optimisation is to localise most of the best local optima to guarantee a fast adaptation to environmental changes [Fernandez-Marquez and Arcos, 2009a], thus, when a local optimum becomes to global optimum is quickly found.

5.3 Sleep/Wake Modes

The required life time of sensors for environment monitoring can reach several years. In order to achieve this requirement, a sensor must be in sleep mode most of the time. A sensor consumes energy while it takes measurements, is computing or while it is communicating (sending or listening for data). Communication is the most energy consuming activity of the sensor [Croce et al., 2008]. The energy used in the communication device, even in idle listening is three orders of magnitude higher than when the node in the sleep mode.

Different proposals to deal with energy efficiency at the MAC layer in sensor networks communication have been presented. Two main approaches can be identified [Na et al., 2008]. On the one hand, the synchronised listening (SL) approach causes sensors to turn on and off their radio at regular intervals; sensors must be synchronised to communicate with each other. The synchronisation has an extra cost and sensors cannot send data when they need to, they have to wait for the wake up events to do so. On the other hand, the low power listening (LPL) approach allows sensors to send information when they want. The only requirement is, for the sender, to send a large preamble data in order to synchronise with other sensors in communication range. Potential receiving sensors wake up asynchronously to detect and synchronise with any detected preamble.

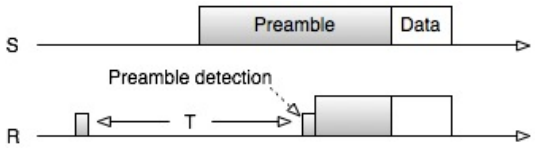


Figure 5.1: Low Power Listening (taken from [Na et al., 2008])

We consider that in emergency scenario like forest fires, or a gas leaks, a sensor should not wait until the next wake up period, but the sensor must be able to send the information in a short period of time. Therefore in this work we assume the LPL approach. The Low Power Listening (LPL) approach reduces the idle listening time, by incorporating a duty cycle in the physical layer. This approach is motivated by the idea that most of the time sensors do not need to communicate, because interesting events rarely occur. Basically, LPL increments the size of the data sent by the transmitter and reduces the cost from the receiver. Figure 5.1 shows how the receiver wakes up asynchronously and checks whether there is a preamble or not. If the preamble is detected, the receiver continues listening until it receives the data, otherwise it turns off the radio until the next cycle T. LPL can be applied to those devices where switching the radio on/off takes little time. Recently, further improvements have been realised in both approaches (SL, LPL) [Na et al., 2008].

5.4 Our Approach

The aim of our approach is to localise the diffuse event sources as soon as possible, minimizing sensors' measurements and communication. Diffuse events appear and disappear over time. Basically, the idea is to find those sensors closest to diffuse events. One of the contributions of this algorithm is that the search of the diffuse event sources is executed in a decentralised way, by collaboration. This proposal produces better scalability when diffuse events spread over a huge number of sensors. Once we find the sources, the number of sensors that report the information about the diffuse event sources localisation is very low compared with the traditional tracking algorithms used in sensor networks, where every sensor that samples a value higher than a fixed threshold sends the information to the sink.

We assume a WSN where the sensors are spread randomly over a 2-dimensional space. All sensors are identical and reactive. Over the WSN there is a middleware that permits a set of agents to move from one sensor to another and have access to the sensor data and sensor communication devices. All agents run the same algorithm and agents have only access to local information. Communication between agents is only allowed when they reside in adjacent sensors, that is, a hop-by-hop communication protocol is not assumed. Sensors only communicate with other sensors when an agent hosted in some sensor requires

information.

We propose a distributed and decentralised approach based on a mobile MAS where agents freely move over the sensor network to localise the sources of diffuse events that are randomly appearing and disappearing along the time. Moreover, agents are responsible for monitoring the localised events once the source is reached. They are responsible for requiring measures from the sensors.

Our approach pursues a number of active agents lower than the number of sensors, as we show later on. As a consequence, a low number of environment measurements are performed. Because we cannot control the number of active diffuse events, we include a mechanism to control the number of mobile agents that live in the WSN. This mechanism controls the number of agents in the WSN in a decentralised way and without additional communication cost.

To deal with energy constraints, we use a GPS-free algorithm where our main goals are to reduce the number of sensors' measurements and the bandwidth used. The GPS-free approach reduces WSN cost [Savvides et al., 2001] and works either in indoor or underwater environments with high energy constraints.

Our approach performs two different explorations: (1) a *global exploration* thanks to the random generation of new agents on the WSN; and (2) a *local exploration* that drives agents to the sources. Global exploration is required to continuously monitor new diffuse events as they appear. We consider that the system *converges* when, for each active event, there is an agent located at the sensor nearest its source (i.e. *all* event sources are monitored).

To ease the discussion, we use the notion of mobile agents. However, to further reduce computation and communication costs, the actual movement of the agents can be replaced by moving a token (instead of a whole agent). In that case, each sensor hosts a stationary agent and the movement would consist in sending a token among the sensors until the token reaches the diffuse event source. The mobile agent approach has the advantage of providing a simpler design of the system's behaviour. In addition, we are also considering to extend the functionalities of mobile agents, such as monitoring both the plume and the source applying flocking techniques. In these cases, a token-based approach would not appropriately capture that swarm behaviour.

5.4.1 Sensors

Sensors are responsible for creating agents. Sensors provide an infrastructure to host agents allowing the agents to access their data and communication devices. Sensors are most of the time in the sleep state, that is, with the wireless communication turned off and using low energy. Sensors do not know their position, i.e. no global position system is assumed. Sensors are identical and they run the same software. Transmission collisions are handled by lower MAC layer protocols and are not considered in this work. Sensors follow the Low Power Listening (LPL) mode described in Section 5.3 and no multi-hop protocol is assumed. Sensors are reactive to agents request. No proactive behavior is assumed from the sensor side. Every T_w ticks, a sensor creates an agent with probability P_a. It

```
if (timeElapsed(T_w)) then
    if (Random() < P_a ) then
        CreateAgent()
    end
end
if (sensorReadRequestEvent()) then
    sendSensorData()
end
```

Algorithm 5: The Sensor Algorithm

is important to note that the creation of an agent does not change the communication state, if the sensor is in the sleep state, it will stay so until it switches to the awake state because of a communication request (i.e. data received from a nearby sensor or sent on request of the agent). The P_a parameter controls the number of agents that are created across the whole environment. A high P_a value implies a high global exploration and also a higher cost, i.e. an increment on the sensors' measurements and on the number of messages sent. Moreover, sensors send data measurements when they receive *data requests*. These are sent by an agent to a neighbour sensor when it performs local exploration. The sensor algorithm is sketched in Algorithm 5. For simplicity purposes, we do not show the change of communication state (sleep to awake to sleep again). The sensor is always in the sleep mode, except when it sends or receives data.

5.4.2 Mobile Agents

Mobile Agents are responsible to actively track diffuse event sources and monitor them once they have reached the source. Mobile Agents use the WSN as an infrastructure that enables them to move over the space, to obtain sensor data, and to communicate with other sensors or agents using the sensors' communication devices. The agent procedure has to deal with uncertain data (mistaken measurements) and with a weak infrastructure that can fail at any time (sensors can break down, sensor data may contain noise, and communications can fail).

The goal is to design a robust agent algorithm that allows agents to monitor diffuse events with a high performance. The agents decide when a sensor must read a sensor data or when a sensor must communicate its sensor data to a neighbour sensor. Sensors are managed by the agents, i.e. they are not proactive.

To deal with the requirements, low number of sensor reads and low number of communication messages, the number of active agents must be considerably lower than the number of sensors. We consider the following policies: (1) when an agent is created, it first checks whether another agent exists in another sensor within its communication range, the agent with a higher creation timestamp finishes its execution; and (2) when two different agents reach the same sensor, only one of them continues its execution (i.e. two agents cannot coexist at the same sensor).

```
if (agentsInNeighbourhood()) then
 | exit()
end
while (true) do
   sensorData = readSensor()
   if (sensorData <= 0) then
    | exit()
   end
   neighbours = selectAdjNodes (n_s)
   requestReads (neighbours)
   bestSensor = selectBestSensor (neighbours)
   if (bestSensor.data > sensorData) then
      moveToSensor(bestSensor)
      if (existAgentInSensor () ) then
       | exit()
      end
   end
end
```

Algorithm 6: The Agent Algorithm

The intuition is that when agents are created, they try to reach the closest diffuse event source by following the shortest path according to a gradient-based strategy. Specifically, each agent uses the sensor data of the neighbouring sensors to guide its movements and finally find the source. Following Algorithm 6, when an agent is created, it first checks if there is another agent placed in one of the adjacent sensors. If that is the case, the most recent agent finishes its execution. Otherwise, it reads the sensor data and checks if a given event plume is detected. If nothing is detected (the measured value is too low), it finishes its execution. When an event is detected, the execution continues by choosing n_s adjacent sensors and sending a sensor data request to the selected n_s sensors. When all the answers are received, the agent selects the best sensor. That is, the sensor providing the highest sensor data read (e.g. highest gas concentration or highest temperature). If the data of the best neighbour sensor is higher than the data the agent has measured on its host sensor, the agent migrates to the selected sensor. After migrating, if another agent is already hosted at that sensor, the migrating agent finishes its execution. Otherwise, the main loop starts again (reading the sensor data of the host sensor).

When an agent reaches the source of a diffuse event (i.e. when it does not move between consecutive reads), it continuously monitors the event until an environment change occurs (i.e. it sends the information to the sink). An event source may disappear or change its location. When it disappears, the data obtained from the sensor becomes zero and the agent finishes its execution. When an event source changes its position (i.e. the event moves slightly), the requests to the neighbour sensors will guide the agent to the new source location.

Parameters	Values	Parameters	Values
movrand	random	num. of peak	1-3
num. of dimensions	2	minheight	30
maxheight	100	stdheight	50
minwidth	0.1	maxwidth	5.0
stdwidth	0.0	mincoordinate	0
maxcoordinate	100	peak_function	cone

Table 5.1: Standard settings for MPB

5.5 Experiments

The goal of this section is to demonstrate the performance of our approach in simulated scenarios and to perform a study of the impact of the parameters of our proposal. Specifically, we analyze the performance of our approach when the number, i.e. density, of the sensors changes; when local and global exploration vary; or when the system is subject to different noise levels. Moreover, we measure the exploration cost and we study the robustness of our approach in front of network failures.

The simulation has been implemented using REPAST [Samuelson and Macal, 2006] to model sensors and agents, and the Moving Peaks Benchmark (MPB) [Branke,] to model environment changes (diffuse events). MPB is a benchmark created to compare dynamic function optimisation algorithms, providing a fitness function changing along the time. The function is composed by different peaks (cones) that change in width, height and position. These peaks are used as diffuse events in our simulation. Analogously to Chapter 3, MPB is modified to aggregate noise to the sensor reads, thus, the fitness function incorporates a noise factor γ in the following way:

$$SensorValue(\vec{p}) = MPBValue(\vec{p}) + (2 * \theta - 1) * \gamma \tag{5.1}$$

where θ generates a uniform random number between [0..1] and γ, the noise factor, varies between 0 and 10 depending on the experiment.

A simulation is a run of $T_S = 2 \times 10^5$ ticks, where an environment change occurs at each $t_c = 200$ ticks. That is, a simulation holds 1000 environment changes. In each environment change diffuse events change the position, intensity, and size. The results reported are the averages of these 1000 changes. Simulations take place in a rectangular space of $10^3 \times 10^3$ square meters where 1000 sensors are distributed randomly. The number of diffuse events vary from 1 to 3 with a radius of the plume ranging from 30 to 5000 meters and the frequency of an agent creation event is $T_w = 20$ ticks. From the results reported later, the probability of creating an agent $P_a = 0.5\%$ and the number of nearby sensors receiving a data request from an agent $n_s = 3$, (table 5.1 summarises the configuration of MPB).

Figure 5.2 shows an example of a simulated scenario, where the sensors are

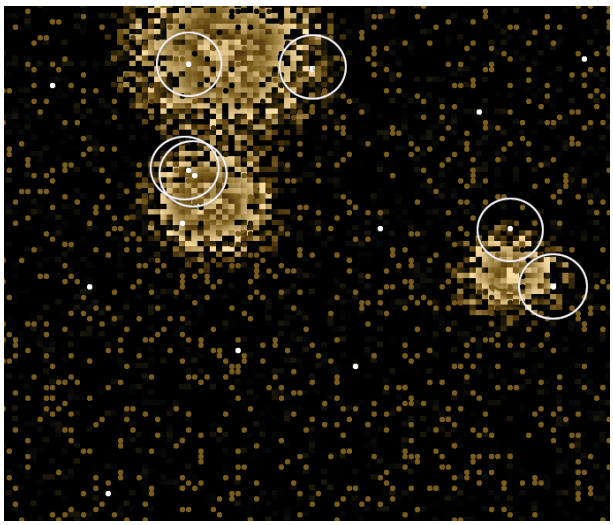


Figure 5.2: Snapshot of a noisy scenario

spread over the space and 3 diffuse events are active. Gray blurred regions represent the diffuse events perceived with noise, i.e. event plumes do not form a continuous space. Small filled points represent the sensors. Gray filled points represent sensors not hosting agents. White filled points represent sensors with a hosted agent. Circles represent communication range of sensors hosting an agent that has detected an event; n_s sensors within the circle will receive the data requests.

In the simulations we use the number of data sensor reads and the number of messages sent as an estimation of the cost to reach convergence, i.e. when *all* diffuse event sources of a given scenario have been detected. These values are measured for each environment change: from the tick a new scenario arises until convergence is reached (all events sources detected). We consider a failure of the system if the system cannot reach convergence before a new change in the environment (200 ticks), i.e. at least one of the sources has not been detected. Once the system has reached convergence, agents continue exploring and monitoring events. At that moment agents are ready to send the sensor data to the sink. The cost of sending the information to the sink depends on the routing algorithm used and it is not addressed in this work. Thus, the monitoring reads and routing messages are not counted here, because they depend on the routing algorithm and on external parameters such as the desired monitoring frequency. Our counting of reads and messages stops when agents reach event sources. We performed an additional experiment to measure the number of sensor data reads and messages when no diffuse events are present, i.e. the cost of the global exploration.

Sensor Number	Reads	Msgs	Failures	Adj. avg.
500	311.30	518.82	35.5%	9.2
1000	397.38	651.64	15.2%	18.76
2000	681.67	1115.48	5.8%	37.09
4000	1222.37	1998.11	4.6%	74.94
8000	2339.69	3816.25	3.2%	149.7175

Table 5.2: Varying Sensor Number without Noise

5.5.1 Varying the number of sensors in WSN

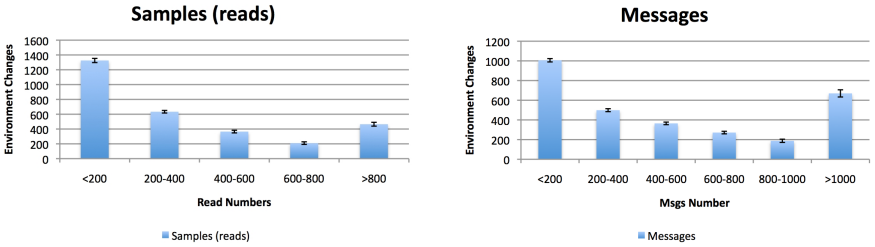
This first experiment had two goals: (1) to demonstrate that the complexity of our approach grows linearly with the WSN size (i.e. our approach is scalable) and (2) to demonstrate the adaptability or our approach to different WSN densities. The different densities used in this simulation have been established following [Intanagonwiwat et al., 2002]. In this experiment the number of sensors varies from 500 to 8000 and noise is not applied to sensor data reads.

The first observation is that, when the density of sensors increases, the number of failures decreases, i.e. agents are able to find better paths to navigate toward event sources (see Table 5.2). Notice that the number of failures reaches a 35% only when the number of sensors is low (500). This percentage of failures can be reduced by incrementing the P_a probability or by reducing the T_w interval, as we will present in the next experiment. The number of consumed resources varies according to the size and location of the diffuse events. Fast convergences are reached with only 15 sensor reads whereas hard scenarios require more than 1000 reads. Notice that difficult scenarios are those where the diffuse events have overlapping areas or where at least one of the diffuse events is covered by a low number of sensors (small diffuse event). Notice that we consider convergence only when all the event sources of a scenario are located.

The results achieved in this first simulation show that our approach is able to find all the diffuse event sources with a probability of 85% when the number of reads is $\sim 40\%$ the number of sensors, and the number of messages is $\sim 60\%$ the the number of sensors (line 2 of Table 5.2). The number of messages and reads grows linearly with the number of sensors, while the number of failures decreases (good scalability).

5.5.2 Quality of Convergence

In the experiments we analyzed the average of the number of reads and the average of the number of messages the approach needs to reach convergence. Figure 5.3(a) shows how, for most of the scenarios, our approach is able to reach convergence in less than 200 reads. The black line on the top of the bars shows the standard deviation over 5 runs where each run has 3000 environment changes. More precisely, 1300 convergences of a total of 3000 are assessed with less than 200 reads, while 450 scenarios require more than 800 reads or do not



(a) Reads histogram

(b) Msgs histogram

Figure 5.3: Performance Results

converge at all. Figure 5.3(b) shows that similar results are obtained for the number of messages: 30% of the convergences are reached with less than 200 messages.

5.5.3 Varying the Noise Factor

The goal of these experiments was to evaluate the performance of our proposal in the presence of different noise levels. Specifically, the noise factor γ varied from 0 to 10. Notice that when the environment is subject to noise, the plume does not follow a monotonous decrease when moving away from the source. Table 5.3 shows how, when the noise factor increases, the performance of the system decreases (in terms of reads). However, when the noise level is equal to or lower than 4, the percentage of failures decreases. The explanation is that noise introduces a stochastic behaviour that increases the exploration in the search. This increment in the exploration increases the number of reads and messages, but produces a better convergence (less percentage of failures). Notice also that our algorithm is robust to noise. Indeed, even when the noise factor is $\pm 10\%$ the algorithm is able to reach convergence, that is to detect the optimum sensor for all the diffuse events in 75% of the scenarios.

5.5.4 Varying Local Exploration

In this experiment we studied the performance of the algorithm when we vary local exploration in a noise-free environment. Local exploration is controlled by the number of sensors that an agent uses to decide its next location (n_s). In Table 5.4, we may observe that even when we increase to 10 the number of requested sensors, the number of failures is not significantly decreasing. The reason behind this result is that increasing local exploration is not enough to detect all the diffuse event sources. Specifically, global exploration is the main factor of failures. As expected, the number of messages and sensor reads increases when local exploration is higher. From the results of this experiment (see Table 5.4), we set the parameter $n_s = 3$.

γ	Reads	Msgs	Failures
0%	397.38	651.64	15.2%
±2%	547.70	907.64	13.4%
±4%	698.31	1160.37	15.1%
±6%	776.21	1291.31	18.6%
±10%	878.73	1461.43	25.1%

Table 5.3: Varying the noise factor γ

n_s	Reads	Msgs	Failures
1	446.41	625.68	15.4%
2	425.96	663.72	16.7%
3	397.38	651.64	15.2%
4	435.79	740.80	12.6%
5	517.21	903.14	14.1%
6	525.74	934.45	14.4%
10	752.18	1391.42	13.1%

Table 5.4: Varying the n_s parameter

P_a	T_w	Reads	Msgs	Failures
0.2	20	322.30	538.29	37.7%
0.5	20	397.38	651.64	15.2%
1	20	484.01	783.83	4.5%
5	20	654.21	1024.92	5.0%

Table 5.5: Varying Agent Creation Probability, P_a

5.5.5 Varying Global Exploration

In the previous experiment we observed that, even increasing local exploration, the number of failures is not reduced. Thus, the goal of this experiment is to reduce system failures by increasing global exploration and to measure the cost associated to this strategy. Global exploration is controlled by the frequency (T_w) of the sensors to create agents and by the probability (P_a) to actually do so. Both parameters can increase or decrease the number of agents that are exploring the space at the same time. We performed an study assessing the contribution of these parameters to the global exploration ratio, the relation between the global exploration ratio and system failures, and the cost of the exploration when reducing system failures.

Table 5.5 shows how, when the exploration rate increases due to an increased probability P_a of creating an agent, the number of failures decreases. However, the price is an increment of the number of reads and messages. Similar results are found when the frequency T_w is increased (see Table 5.6). In both experiments we are increasing the number of agents that explore the WSN. As a conclusion of the results, P_a and T_w can be used to customize our approach depending of the search priority. This trade-off between the quality of the results and the cost can be used to control the priority of the search process. Emergency situations will tend to increase the exploration cost. Notice that even when we reduce the percentage of failures to 0.3%, the number of reads and messages present good results. Indeed, the algorithm is able to find the sensor closest to the event with 654 reads in an environment with 1000 sensors.

Experimental results have demonstrated that, even in the presence of a high

P_a	T_w	Reads	Msgs	Failures
0.5	5	576.26	920.21	1.8%
0.5	10	488.36	790.16	4.4%
0.5	20	397.38	651.64	15.2%
0.5	50	307.75	513.11	38.5%

Table 5.6: Varying Frequency, T_w

Noise	Reads	Msgs
0%	49.28	0±0
±2%	100.22	108.74
±5%	99.46	107.87
±10%	101.78	111.05

Table 5.7: The Exploration Cost

noise level, the number of failures is reduced by incrementing the global exploration. For instance, increasing P_a to 2% and the noise level to 10% the number of reads is 1190 and the number of messages is 1947 whereas the number of failures is 162 (16.2%), i.e. same number of failures achieved without noise. Thus, the global exploration level can reduce the number of failures produced by the lack of sensors in the WSN or by the presence of noise.

5.5.6 The Exploration Cost

The goal of these experiments was to measure the exploration cost when no diffuse events are present in the system (most frequent case). Specifically, we tested our approach when different noise levels are applied. Notice that noise is acting as false plumes that temporarily drive agents through the WSN. Table 5.7 shows how, when the noise level increases from 0% to ±2%, the exploration cost increases by 50%. Thus, we may conclude that noise increments the exploration cost. However, this increment remains constant even when we increment the noise to ±5%, or even to ±10%. Thereby, the performance of our approach does not depend on the noise level.

5.5.7 Tolerance to WSN failures

Finally, we analyzed the robustness of our approach when sensors fail. To that purpose, a probability of failure was added to each sensor. Sensor failures are simulated as follows: just before T_w a percentage of sensors are declared broken down (state is off). Then, those sensors cannot be used until the next T_w interval, where the sensors may continue to be broken or have become fixed. In Table 5.8 we may observe that the increment in the sensor failures involves a decrease of system convergences. However, when exploration is increased (e.g. increasing

Failure Prob.	Reads	Msgs	Failures
0%	449.91	740.33	13.7%
5%	455.76	751.13	15.1%
10%	409.97	675.38	16.9%
20%	422.26	697.46	19.6%
40%	463.55	772.60	30.7%

Table 5.8: Failure Tolerance

the probability of agent creation from 0.5 to 2.0) the system is able to decrease the failures to 47 (with an average of reads of 705 and messages of 1145). Thus, we may conclude that our approach reaches the convergence even with a high probability of sensor failures.

5.6 Conclusions

In this chapter, we have proposed a new approach, based on the Chemotaxis Pattern using Mobile Multi-Agent technology, to detect diffuse event sources in dynamic and noisy environments working on a wireless sensor network infrastructure. To our knowledge, this problem has not been addressed before. Our approach proposes a distributed and decentralised algorithm based on local interactions and local knowledge of the environment. Different strategies have been designed to guarantee a low number of agents maintaining the performance of the system.

We studied the performance of our proposal on different scenarios: changing the density of the sensors; varying local and global exploration ratios; applying noise to the data that sensors gather; and subjecting sensors to failures. Experimental results have shown that the presence of noise, sensor failures, and the lack of sensors diminish the performance of our approach. However, it has been detailed how this degradation can be alleviated by increasing the exploration level. Increasing the exploration level involves a reasonable rise on the cost to reach the convergence. Importantly, in our approach the cost of global exploration does not depend on the noise level. Because our approach is not introducing any assumption on the sensor positions, we plan to explore its capabilities in scenarios like underwater applications or 3-Dimension spaces.

Chapter 6

Conclusions and Future Work

Aimed of contributing to the engineering self-organising systems, this book presents a catalog of bio-inspired self-organising mechanisms used in large scale Multi-Agent Systems (MAS). The mechanisms presented in this book are spread over the literature and applied in ad-hoc way. Even when those mechanisms have achieved relevant contributions in different fields, when and how to apply these mechanisms is still an open issue. We classify and identify each mechanism using a design pattern structure. Thus, the mechanisms can be applied (composed or adapted) easily to solve existing self-organising problems in large scale MAS.

Complex mechanisms have been decomposed, identifying their internal basic mechanisms. These basic mechanisms have been presented also as design patterns, providing a complete catalog of mechanisms organised in three different layers, Figure 6.1. In the bottom layer, basic mechanisms that can be easily combined to create composed mechanisms or can be used isolated. In the middle layer, existing composed patterns, with contributions well known in the literature. The top layer presents high level mechanisms that exploit the basic and composed mechanisms proposed in the bottom and middle layer in different ways. In Figure 6.1, Patterns' names distinguished with bigger size are those we have implemented and exploited to demonstrate their contribution in different fields.

To describe the interactions and dynamics between the entities involved in each pattern, we proposed a general computational model that can be applied to model a wide range of existing multi-agent applications. Some of the proposed mechanisms described in this book have been applied to three different domains:

1. In dynamic optimisation where the existing Particle Swarm Optimisation algorithm has been extended with the Evaporation Pattern improving its performance in dynamic and noisy optimisation.

2. In Spatial Computing, where a new infrastructureless storage system is

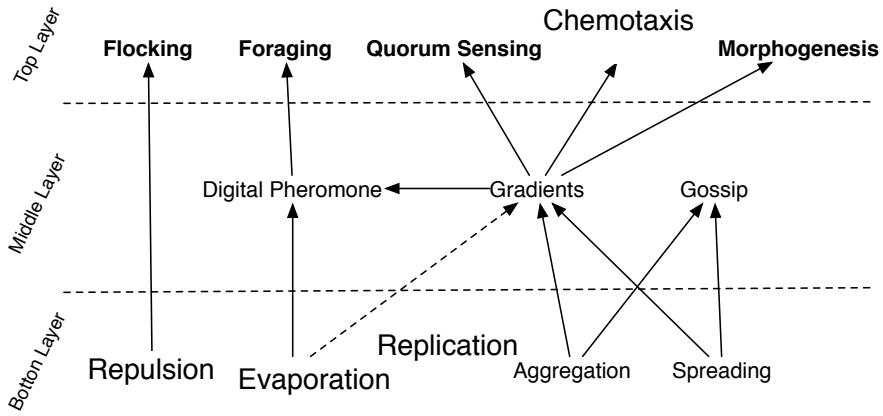


Figure 6.1: Design Patterns

proposed based on Mobile Multi-Agent Systems using a high dynamic mobile network. In this system the agents decide when to replicate and collaborate between them to ensure the information coverage in a specific area. In this domain, the Replication Pattern and the Repulsion Pattern are applied together. This composition of basic patterns shows how the basic patterns are easily used to compose new patterns or adapt existing patterns.

3. In sensor networks, the location and tracking of diffuse event sources problem is proposed as a new interesting problem with important application in real word domains. The Chemotaxis Pattern, used to tackle this new problem has demonstrated to achieve a good performance and presents good tolerance in front of sensor network failures and noisy and dynamic environments.

This book is a step forward to engineering self-organising systems. The mechanisms proposed in this book achieve a desired emergent behaviour from the local interactions. However, even when these mechanisms can be applied systematically to solve existing problems, it is necessary to have a good experience to adapt the different parameters of each mechanism. The adaptation of parameters to optimise the performance of each mechanism is still an open issue. Moreover, even when these mechanisms lead engineers to implement self-organising systems. The engineering of self-organising systems is not yet widely exploited and nowadays is a promising open field. Thus, we could summarise the future work as follows:

- To describe a design process for engineering self-organising systems using the proposed patterns. Basically the main steps in the design process

would be: (1) to identify the problems and requirements of the system; (2) to select the patterns from the proposed catalog that solve these problems; (3) to compose the selected patterns as composed mechanisms; and (4) to add the policies for the proper development of the application.

- The study of self-adaptation of each pattern parameters and policies, in order to optimise the performance of each pattern automatically when they are applied in different domains.
- To tackle new existing problems with the mechanisms proposed in this book.

6.1 Publications related to this research

The work presented in this book has been published in several international conferences and journals. Specifically, the relation of these publications with the book chapters in the following:

- Chapter 2: Bio-inspired design patterns.
  - Description and Composition of Bio-Inspired Design Patterns: the Gradient Case. In Proceedings of the 3rd Workshop on Bio-Inspired and Self-* Algorithms for Distributed Systems (BADS 2011). (To appear).
  - Description and Composition of Bio-Inspired Design Patterns: the Gossip Case. In Proceedings of the 8th IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe 2011). (To appear).
- Chapter 3: Dynamic Optimisation.
  - Adapting particle swarm optimisation in dynamic and noisy environments. In Proceedings of IEEE Congress on Evolutionary Computation (CEC 2010), pages 765-772.
  - An evaporation mechanism for dynamic and noisy multimodal optimisation. In Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO 2009), pages 17-24.
  - Evaporation as a self-adaption mechanism for PSO. In Brueckner, S. Robertson, P., and Bellur, U. editors, In Proceedings of the 2th International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), pages 465,466.
- Chapter 4: Hovering Information in Spatial Computing.
  - Infrastructureless spatial storage algorithms. ACM Transactions on Autonomous and Adaptive Systems (TAAS). (To Appear).

  - Infrastructureless storage in dynamic environments. In Proceedings of the 25th Symposium On Applied Computing (SAC 2010). ACM, New York, NY, USA 1334-1338.

- Chapter 5: Detecting Dynamically Changing Diffuse Event Sources in Noisy WSN Environments.

  - Decentralised Approach for Detecting Dynamically Changing Diffuse Event Sources in Noisy WSN Environments. In Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010), pages 257-258.
  - Decentralised Approach for Detecting Dynamically Changing Diffuse Event Sources in Noisy WSN Environments (extended version). In Proceedings of the 8th European Workshop on Multi-Agent Systems (EUMAS 2010).

Bibliography

[Hay, 2002] (2002). *Self-Organized Flocking with Agent Failure: Off-Line Optimization and Demonstration with Real Robots.*

[Abelson et al., 2000a] Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas F. Knight, J., Nagpal, R., Rauch, E., Sussman, G. J., and Weiss, R. (2000a). Amorphous computing. *Commun. ACM*, 43(5):74–82.

[Abelson et al., 2000b] Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas F. Knight, J., Nagpal, R., Rauch, E., Sussman, G. J., and Weiss, R. (2000b). Amorphous computing. *Commun. ACM*, 43(5):74–82.

[Abelson et al., 2007] Abelson, H., Beal, J., and Sussman, G. j. (2007). Amorphous computing. Technical Report MIT-CSAIL-TR-2007-030, Computer Science and Artificial Intelligence Laboratory, MIT.

[Babaoglu et al., 2006] Babaoglu, O., Canright, G., Deutsch, A., Caro, G. A. D., Ducatelle, F., Gambardella, L. M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. (2006). Design patterns from biology for distributed computing. *ACM Trans. on Autonomous and Adaptive Sys*, 1:26–66.

[Bachem et al., 1996] Bachem, A., Hochstättler, W., and Malich, M. (1996). The simulated trading heuristic for solving vehicle routing problems. *Discrete Appl. Math.*, 65:47–72.

[Bailey, 1975] Bailey, N. T. (1975). The mathematical theory of infectious diseases and its applications / norman t.j. bailey.

[Bartz-Beielstein et al., 2007] Bartz-Beielstein, T., Blum, D., and Branke, J. (2007). *Metaheuristics. Progress in Complex Systems Optimization*, chapter Particle Swarm Optimization and Sequential Sampling in Noisy Environments. Springer.

[Bassler, 2002] Bassler, B. L. (2002). Small talk. cell-to-cell communication in bacteria. *Cell*, 109(4):421–424.

[Beal, 2003] Beal, J. (2003). Persistent nodes for reliable memory in geographically local networks. Technical Report AI Memo 2003-011, Artificial Intelligence Laboratory, MIT.

[Beal, 2009] Beal, J. (2009). Flexible self-healing gradients. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1197–1201, New York, NY, USA. ACM.

[Beal et al., 2008] Beal, J., Bachrach, J., Vickery, D., and Tobenkin, M. (2008). Fast self-healing gradients. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1969–1975, New York, NY, USA. ACM.

[Bell et al., 2003] Bell, W., Cameron, D., Carvajal-Schiaffino, R., Millar, A., Stockinger, K., and Zini, F. (2003). Evaluation of an economy-based finle replication strategy for a data grid. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 661 – 668.

[Birman et al., 1999] Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y. (1999). Bimodal multicast. *ACM Transactions on Computer Systems*, 17:41–88.

[Blackwell, 2007] Blackwell, T. (2007). Particle swarm optimization in dynamic environments. In Yang, S., Ong, Y.-S., and Jin, Y., editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, pages 29–49. Springer.

[Blackwell and Bentley, 2002] Blackwell, T. and Bentley, P. (2002). Dynamic search with charged swarms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pages 19–26.

[Blackwell and Branke, 2006] Blackwell, T. and Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472.

[Blatt and III, 2006] Blatt, D. and III, A. O. H. (2006). Energy-based sensor network source localization via projection onto convex sets. *IEEE Transactions on Signal Processing*, 54(9):3614–3619.

[Blum, 2005] Blum, C. (2005). Beam-aco: hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591.

[Bojinov et al., 2000] Bojinov, H., Casal, A., and Hogg, T. (2000). Multiagent control of self-reconfigurable robots. In *In Proceedings of International Conference on Multiagent Systems*, pages 143–150.

[Branke,] Branke, J. The moving peaks benchmark website. www.aifb.uni-karlsruhe.de/ jbr/movpeaks/.

[Britton and Sack, 2004] Britton, M. and Sack, L. (2004). The secoas project: development of a self-organising wireless sensor network for environmental monitoring. In *2nd International Workshop on Sensor and Actor Network Protocols and Applications*, Boston.

[Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-oriented software architecture: A system of patterns*. John Wiley & Sons.

[Butera, 2007] Butera, W. (2007). Text display and graphics control on a paintable computer. In *SASO '07: Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 45–54, Washington, DC, USA. IEEE Computer Society.

[Castro et al., 2008] Castro, A. A. V., Di Marzo Serugendo, G., and Konstantas, D. (2008). Hovering information - self-organising information that finds its own storage. In *Proceedings of the 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 193–200, Washington, DC, USA. IEEE Computer Society.

[Chen and Kotz, 2002] Chen, G. and Kotz, D. (2002). Solar: A pervasive-computing infrastructure for context-aware mobile applications. Technical report, Department of Computer Science, Dartmouth College Hanover, NH, USA 03755.

[Cheng et al., 2005] Cheng, J., Cheng, W., and Nagpal, R. (2005). Robust and self-repairing formation control for swarms of mobile agents. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 59–64, Menlo Park, California. AAAI Press.

[Corkill et al., 2007] Corkill, D. D., Holzhauer, D., and Koziarz, W. (2007). Turn Off Your Radios! Environmental Monitoring Using Power-Constrained Sensor Agents. In *First International Workshop on Agent Technology for Sensor Networks (ATSN-07)*, pages 31–38, Honolulu, Hawaii.

[Croce et al., 2008] Croce, S., Marcelloni, F., and Vecchio, M. (2008). Reducing power consumption in wireless sensor networks using a novel approach to data aggregation. *The Computer Journal*, 51:227–239.

[Crowther and Riviere, 2002] Crowther, B. and Riviere, X. (2002). Flocking of autonomous unmanned air vehicles. In *Proceedings of the 17th Bristol UAV Conference*.

[Datta et al., 2004] Datta, A., Quarteroni, S., and Aberer, K. (2004). Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in mobile ad-hoc networks. In *IC-SNW'04, International Conference on Semantics of a Networked World*, LNCS, pages 126–143.

[De Wolf and Holvoet, 2007] De Wolf, T. and Holvoet, T. (2007). Design patterns for decentralised coordination in self-organising emergent systems. In *Proceedings of the 4th international conference on Engineering self-organising systems*, ESOA'06, pages 28–49, Berlin, Heidelberg. Springer-Verlag.

[Deneubourg et al., 1983] Deneubourg, J., Pasteels, J., and Verhaeghe, J. (1983). Probabilistic behaviour in ants: A strategy of errors? *Journal of Theoretical Biology*, 105(2):259 – 271.

[Di Marzo Serugendo et al., 2007] Di Marzo Serugendo, G., Villalba Castro, A., and Konstantas, D. (2007). Dependable requirements for hovering information. In *Supplemental Volume - The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 36–39.

[Dolev et al., 2005] Dolev, S., Gilbert, S., Lynch, N. A., Shvartsman, A. A., and Welch, J. L. (2005). Geoquorums: implementing atomic memory in mobile ad hoc networks. *Distrib. Comput.*, 18:125–155.

[Dorigo, 1992] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy.

[Dorigo and Di Caro, 1999] Dorigo, M. and Di Caro, G. (1999). *The ant colony optimization meta-heuristic*, pages 11–32. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

[Du and Li, 2008] Du, W. and Li, B. (2008). Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Inf. Sci.*, 178:3096–3109.

[Ermis and Saligrama, 2006] Ermis, E. and Saligrama, V. (2006). Detection and localization in sensor networks using distributed FDR. In *Proceedings of the Conference on Information Sciences and Systems (CISS-06)*, Princeton, New Jersey, USA.

[Eugster et al., 2009] Eugster, P., Garbinato, G., Holzer, A., and Luo, J. (2009). Effective location-based publish/subscribe in manets. In *IEEE International Conference on Pervasive Computing and Communications (PerCom'09)*.

[Eugster et al., 2005] Eugster, P. T., Garbinato, B., and Holzer, A. (2005). Location-based publish/subscribe. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 279–282, Washington, DC, USA. IEEE Computer Society.

[Fernandez-Marquez and Arcos, 2008] Fernandez-Marquez, J. L. and Arcos, J.-L. (2008). Evaporation as a self-adaptation mechanism for pso. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 465–466, Washington, DC, USA. IEEE Computer Society.

[Fernandez-Marquez and Arcos, 2009a] Fernandez-Marquez, J. L. and Arcos, J. L. (2009a). An evaporation mechanism for dynamic and noisy multimodal optimization. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 17–24, New York, NY, USA. ACM.

[Fernandez-Marquez and Arcos, 2009b] Fernandez-Marquez, J. L. and Arcos, J. L. (2009b). Keeping diversity when exploring dynamic environments. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1192–1196, New York, NY, USA. ACM.

[Fernandez-Marquez et al., 2010] Fernandez-Marquez, J. L., Arcos, J. L., and Di Marzo Serugendo, G. (2010). Infrastructureless storage in dynamic environments. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1334–1338, New York, NY, USA. ACM.

[Fernandez-Marquez et al., 2011] Fernandez-Marquez, J. L., Di Marzo Serugendo, G., and Arcos, J. L. (2011). Infrastructureless spatial storage algorithms. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass.

[Gardelli et al., 2007] Gardelli, L., Viroli, M., and Omicini, A. (2007). Design patterns for self-organizing multiagent systems. In Wolf, T. D., Saffre, F., and Anthony, R., editors, *2nd International Workshop on Engineering Emergence in Decentralised Autonomic System (EEDAS) 2007*, pages 62–71, ICAC 2007, Jacksonville, Florida, USA. CMS Press, University of Greenwich, London, UK.

[Garnier et al., 2007] Garnier, S., Gautrais, J., and Theraulaz, G. (2007). The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31.

[Grégoire and Konieczny, 2006] Grégoire, E. and Konieczny, S. (2006). Logic-based approaches to information fusion. *Information Fusion*, 7:4–18.

[Haas et al., 2006] Haas, Z. J., Halpern, J. Y., and Li, L. (2006). Gossip-based ad hoc routing. *IEEE/ACM Transaction Networking.*, 14(3):479–491.

[Intanagonwiwat et al., 2002] Intanagonwiwat, C., Estrin, D., Govindan, R., and Heidemann, J. (2002). Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 457–458, Washington, DC, USA. IEEE Computer Society.

[Jadbabaie et al., 2003] Jadbabaie, A., Lin, J., and Morse, A. S. (2003). Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001.

[Jamjoom et al., 1999] Jamjoom, H., Jamin, S., and Shin, K. (1999). Self-organizing network services. In *University Michigan*.

[Janeway et al., 2001] Janeway, C., Travers, P., Walport, M., and Shlomchik, M. (2001). *Immunobiology - The immune system in health and disease*. Garland Publishing.

[Janson and Middendorf, 2006] Janson, S. and Middendorf, M. (2006). A hierarchical particle swarm optimizer for noisy and dynamic environments. *Genetic Programming and Evolvable Machines*, 7(4):329–354.

[Jin and Branke, 2005] Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments - a survey. *IEEE Transactions on Evolutionary Computation*, pages 303–316.

[Kempe et al., 2003] Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 482–, Washington, DC, USA. IEEE Computer Society.

[Konstantas and Villalba, 2006] Konstantas, D. and Villalba, A. (2006). Towards hovering information. In *Proceedings of the First European Conference on Smart Sensing and Context (EuroSSC 2006)*, pages 161–166.

[La and Sheng, 2009a] La, H. M. and Sheng, W. (2009a). Flocking control of a mobile sensor network to track and observe a moving target. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3586–3591, Piscataway, NJ, USA. IEEE Press.

[La and Sheng, 2009b] La, H. M. and Sheng, W. (2009b). Moving targets tracking and observing in a distributed mobile sensor network. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 3319–3324, Piscataway, NJ, USA. IEEE Press.

[Lee and Chung, 2005] Lee, S. and Chung, T. (2005). Data aggregation for wireless sensor networks using self-organizing map. In Kim, T., editor, *Artificial Intelligence and Simulation*, volume 3397 of *Lecture Notes in Computer Science*, pages 508–517. Springer Berlin / Heidelberg.

[Leontiadis et al., 2009] Leontiadis, I., Costa, P., and Mascolo, C. (2009). Persistent content-based information dissemination in hybrid vehicular networks. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–10, Washington, DC, USA. IEEE Computer Society.

[Leontiadis and Mascolo, 2007] Leontiadis, I. and Mascolo, C. (2007). Opportunistic spatio-temporal dissemination system for vehicular networks. In *MobiOpp '07: Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking*, pages 39–46, New York, NY, USA. ACM Press.

[Lind, 2003] Lind, J. (2003). Patterns in agent-oriented software engineering. In *Proceedings of the 3rd international conference on Agent-oriented software engineering III*, AOSE'02, pages 47–58, Berlin, Heidelberg. Springer-Verlag.

[Lourenço and Serra, 1998] Lourenço, H. R. and Serra, D. (1998). Adaptive approach heuristics for the generalized assignment problem. Technical report,

Economic Working Papers Series No.304, Universitat Pompeu Fabra, Dept. of Economics and Management.

[Lung and Dumitrescu, 2007] Lung, R. I. and Dumitrescu, D. (2007). A collaborative model for tracking optima in dynamic environments. *IEEE Congress on Evolutionary Computation (CEC)*, pages 564–567.

[Mamei et al., 2006] Mamei, M., Menezes, R., Tolksdorf, R., and Zambonelli, F. (2006). Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52:443–460.

[Mamei et al., 2004] Mamei, M., Vasirani, M., and Zambonelli, F. (2004). Experiments of morphogenesis in swarms of simple mobile robots. *Journal of Applied Artificial Intelligence*, 18:903–919.

[Mamei and Zambonelli, 2005] Mamei, M. and Zambonelli, F. (2005). Motion coordination in the quake 3 arena environment: A field-based approach. In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 264–278. Springer Berlin / Heidelberg.

[Mamei and Zambonelli, 2007] Mamei, M. and Zambonelli, F. (2007). Pervasive pheromone-based interaction with rfid tags. *ACM Transactions on Autonomous and Adaptive Systems*, 2.

[Martens et al., 2007] Martens, D., De Backer, M., Vanthienen, J., Snoeck, M., and Baesens, B. (2007). Classification with ant colony optimization. *IEEE Transactions on Evolutionary Computation*, 11:651–665.

[Miller and Bassler, 2001] Miller, M. B. and Bassler, B. L. (2001). Quorum sensing in bacteria. *Annual Review of Microbiology*, 55(1):165–199.

[Na et al., 2008] Na, J., Lim, S., and Kim, C.-K. (2008). Dual wake-up low power listening for duty cycled wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2008:46:1–46:11.

[Nagpal, 2002] Nagpal, R. (2002). Programmable self-assembly using biologically-inspired multiagent control. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, pages 418–425, New York, NY, USA. ACM.

[Nagpal, 2004] Nagpal, R. (2004). A catalog of biologically-inspired primitives for engineering self-organization. In *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering. Volume 2977 of Lecture Notes in Computer Science*, pages 53–62. Springer.

[Nardi et al., 2006] Nardi, R. D., Holl, O., Woods, J., and Clark, A. (2006). Swarmav: A swarm of miniature aerial vehicles. In *Proceedings of the 21st Bristol International UAV Systems Conference.*

[Olfati-saber, 2006] Olfati-saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51:401–420.

[Parsopoulos and Vrahatis, 2001] Parsopoulos, K. E. and Vrahatis, M. N. (2001). Particle swarm optimizer in noisy and continuously changing environments. *Artificial Intelligent and soft computing*, pages 289–294.

[Parunak et al., 2002] Parunak, H., Purcell, M., and Connell, R. (2002). Digital pheromones for autonomous coordination of swarming uavs. In *Proceedings of First AIAA Unmanned Aerospace Vehivales, Systems, Technologies, and Operations Conference (AIAA)*. American Institute of Aeronautics and Astronautics.

[Pigozzi and Hartmann, 2007] Pigozzi, G. and Hartmann, S. (2007). Aggregation in multi-agent systems and the problem of truth-tracking. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 07), 1418 May 2007, Honolulu, Hawaii, USA*, pages 674 – 676.

[Poli et al., 2007] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. an overview. *Swarm Intelligence*, 1:33–57.

[Pugh et al., 2005] Pugh, J., Martinoli, A., and Zhang, Y. (2005). Particle swarm optimization for unsupervised robotic learning. *proceedings of IEEE swarm intelligence symposium(SIS)*, pages 92–99.

[Reynolds, 1987] Reynolds, C. W. (1987). 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. ACM.

[Roth, 2003] Roth, J. (2003). The critical mass problem of mobile ad-hoc networks. In *Proceedings of IADIS International Conference e-Society*, pages 243–250. IADIS Press.

[Ruairí and Keane, 2007] Ruairí, R. M. and Keane, M. T. (2007). An energy-efficient, multi-agent sensor network for detecting diffuse events. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical Intelligence*, pages 1390–1395. Morgan Kaufmann Publishers Inc.

[Sahin and Franks, 2002] Sahin, E. and Franks, N. R. (2002). Measurement of Space: From Ants to Robots. In *International Workshop Biologically-Inspired Robotics*, pages 241–247, Bristol, UK.

[Salazar et al., 2010] Salazar, N., Rodriguez-Aguilar, J. A., and Arcos, J. L. (2010). Robust coordination in large convention spaces. *AI Communications*, 23(4):357–372.

[Samuelson and Macal, 2006] Samuelson, D. and Macal, C. (2006). Agent-based simulation comes of age. *OR/MS Today*, (4):34–38.

[Sauter et al., 2005] Sauter, J. A., Matthews, R., Van Dyke Parunak, H., and Brueckner, S. A. (2005). Performance of digital pheromones for swarming vehicle control. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 903–910. ACM.

[Savvides et al., 2001] Savvides, A., Han, C.-C., and Strivastava, M. B. (2001). Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 166–179, New York, NY, USA. ACM.

[Scellato et al., 2008] Scellato, S., Mascolo, C., Musolesi, M., and Latora, V. (2008). Epcast: Controlled Dissemination in Human-Based Wireless Networks Using Epidemic Spreading Models. In *Bio-Inspired Computing and Communication: First Workshop on Bio-Inspired Design of Networks, BIOWIRE 2007 Cambridge, UK, April 2-5, 2007 Revised Selected Papers*, pages 295–306, Berlin, Heidelberg. Springer-Verlag.

[Secomandi, 2000] Secomandi, N. (2000). Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research*, 27(11-12):1201–1225.

[Sudeikat and Renz, 2008] Sudeikat, J. and Renz, W. (2008). Engineering environment-mediated multi-agent systems. chapter Toward Systemic MAS Development: Enforcing Decentralized Self-organization by Composition and Refinement of Archetype Dynamics, pages 39–57. Springer-Verlag, Berlin, Heidelberg.

[Toth and Vigo, 2002] Toth, P. and Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512.

[Tseng et al., 2002] Tseng, Y.-C., Ni, S.-Y., Chen, Y.-S., and Sheu, J.-P. (2002). The broadcast storm problem in a mobile ad-hoc network. *Wireless Networks*, 8(2/3):153–167.

[Villalba Castro et al., 2008] Villalba Castro, A., Di Marzo Serugendo, G., and Konstantas, D. (2008). Hovering information - self-organising information that finds its own storage. In *Proceedings of International IEEE Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC'08)*, pages 193–200.

[Vinyals et al., 2011] Vinyals, M., Rodriguez-Aguilar, J. A., and Cerquides, J. (2011). A survey on sensor networks from a multiagent perspective. *The Computer Journal*, 54:455–470.

[Viroli et al., 2011] Viroli, M., Casadei, M., Montagna, S., and Zambonelli, F. (2011). Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems*. To Appear.

[Weimer et al., 2009] Weimer, J., Sinopoli, B., and Krogh, B. (2009). Multiple source detection and localization in advection-diffusion processes using wireless sensor networks. In *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, RTSS '09, pages 333–342, Washington, DC, USA. IEEE Computer Society.

[Weyns et al., 2007] Weyns, D., Holvoet, T., and Helleboogh, A. (2007). Anticipatory vehicle routing using delegate multi-agent systems. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 87 –93.

[Wolpert et al., 2007] Wolpert, L., Jessell, T., Lawrence, P., Meyerowitz, E., Robertson, E., and Smith, J. (2007). *Principles of Development*. Oxford University Press, Oxford, 3rd edition.

[Yang et al., 2006] Yang, L., Feng, C., Rozenblit, J. W., and Qiao, H. (2006). Adaptive tracking in distributed wireless sensor networks. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pages 103–111, Washington, DC, USA. IEEE Computer Society.

Appendix A

Hovering Information

This appendix provides additional information about experiments we performed regarding Hovering Information (Chapter 4). Specifically, some simulation images using different anchor areas. Figure A.1 shows how the different algorithms proposed can cover a simple circle anchor area. Small gray nodes represent nodes that have not access to the information, circular white filled nodes represent nodes with information access, and circular gray filled nodes are storing a piece of information. In the figure we may observe how the Attractor Point Repulsion algorithm uses less nodes storing the information than the others. Moreover, the repulsion variants achieve a better distribution of the piece of information inside the shape. Similar results are achieved with the different anchor areas.

Figure A.2 shows three steps of the Attractor Point Repulsion Algorithm. In Step 1 one piece of information is deposited in one node. In step 2, the replication and repulsion located new replicas following the shape. In this way, in few iterations more than half of the anchor area is covered. In step 3, the anchor area is completely covered. It may be observed how the number of replicas used to cover the anchor area is really low compared with the number of nodes located inside the anchor area. Moreover, it may be observed a uniform distribution of pieces of information over the anchor area. Figure A.3, is a continuation of the previous simulation, where nodes inside a sub-area are eliminated from the environment. Steps 4, 5 and 6 show how the Attractor Point Repulsion can refill the anchor area even in presence of node failures. Similar results were achieved by the other algorithms proposed. Figures A.4 and A.5 show the same experiment than figures A.2 and A.3 but instead of using Attractor Point Repulsion, the algorithm used is broadcast. Comparing the simulation steps of different algorithms, we may observe that the Broadcast algorithm achieved a faster convergence than the Attractor Point Repulsion algorithm. However, the number of replicas used by the Broadcast algorithm is higher than replicas used by the Attractor Point Repulsion. Regarding the recovering time after node failures, both algorithms present similar performance, because both algorithms need to wait until the network is recovered from the failure.

Figures A.7 and A.8 show three different steps of the simulation using the

Broadcast algorithm and the Broadcast Repulsion algorithms respectively. The anchor area used in both simulations is showed in Figure A.6. Broadcast simulation compared with Broadcast Repulsion simulation used more pieces of information to cover the shape. However, analogously to previous simulation, without repulsion the algorithm can cover the area using less number of iterations.

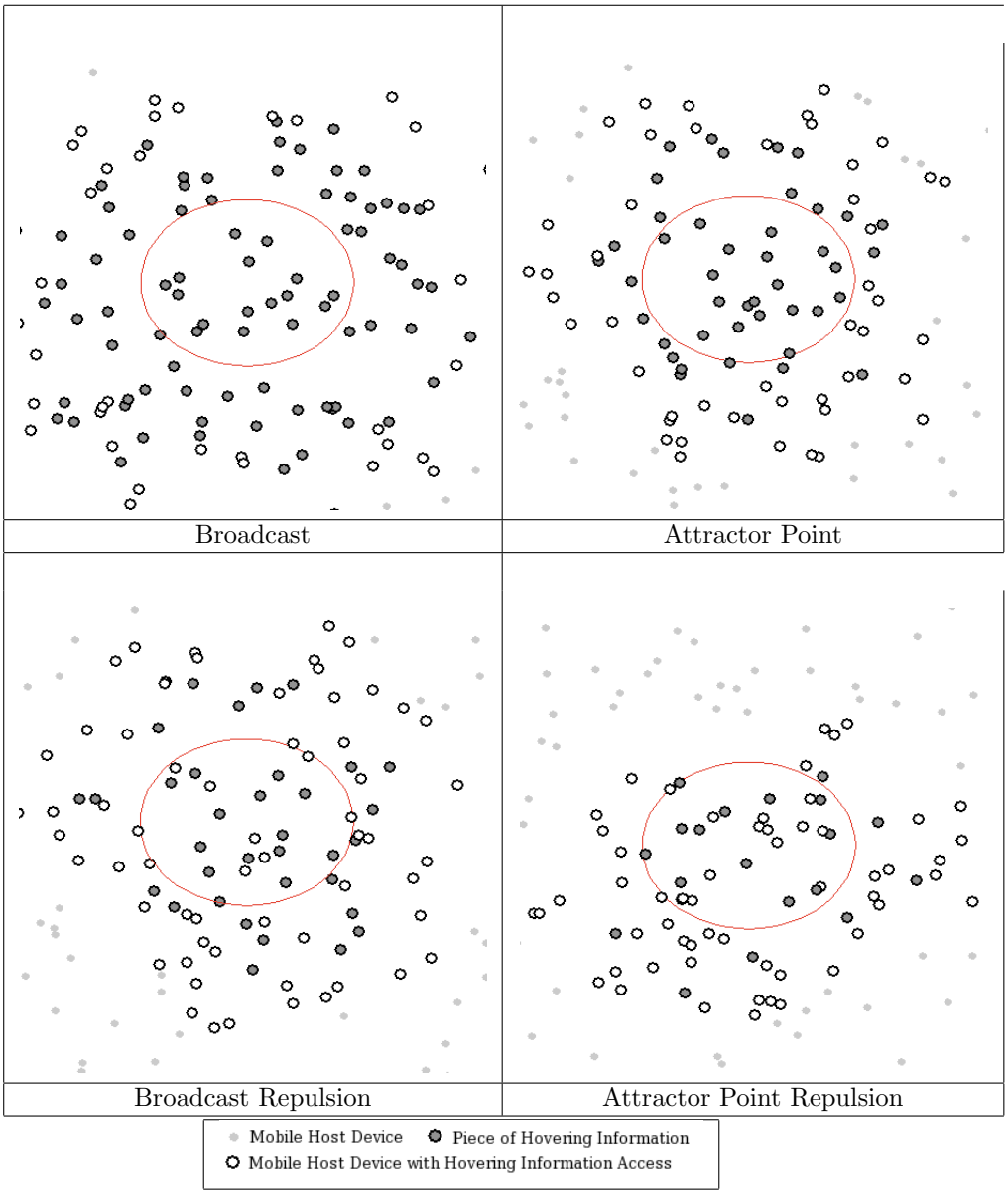
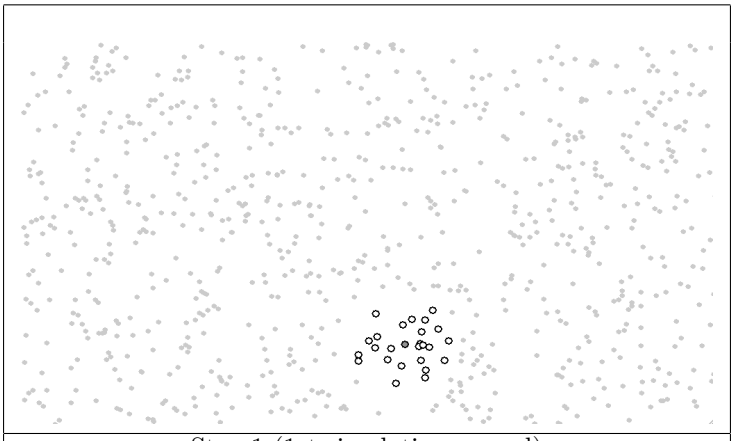
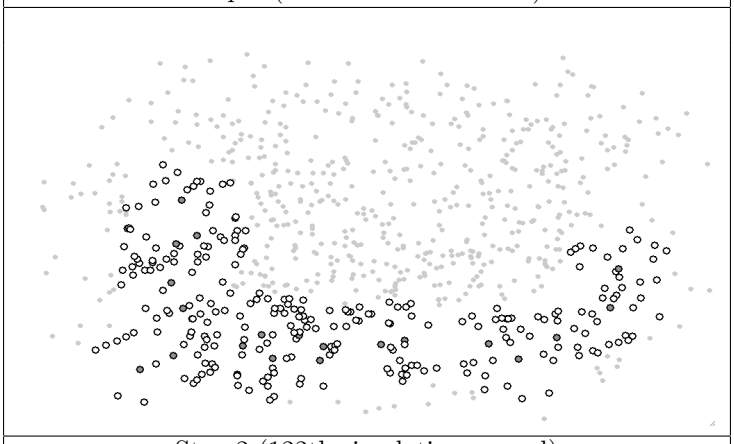


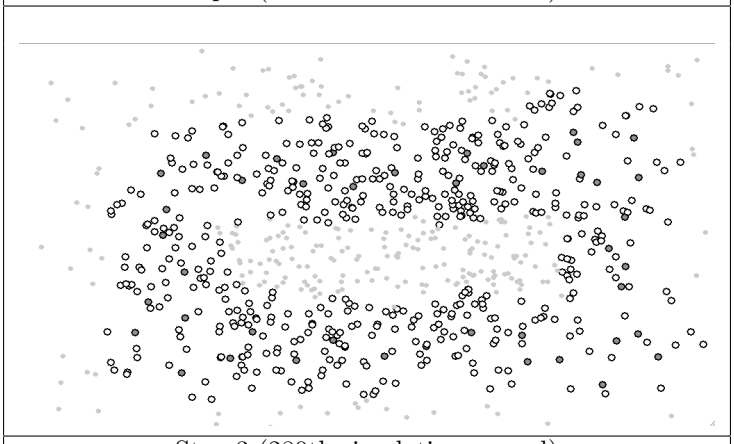
Figure A.1: Broadcast, Attractor Point, Broadcast Repulsion and Attractor Point Repulsion



Step 1 (1st simulation second)



Step 2 (122th simulation second)



Step 3 (280th simulation second)

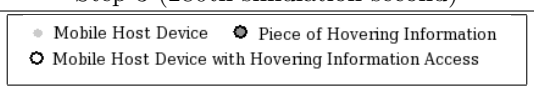
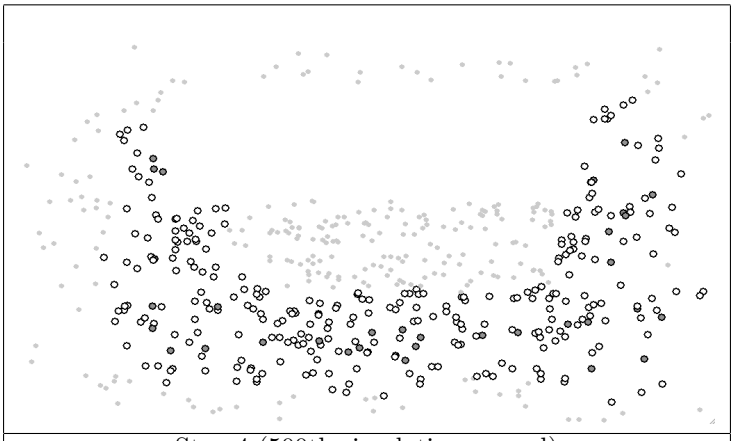
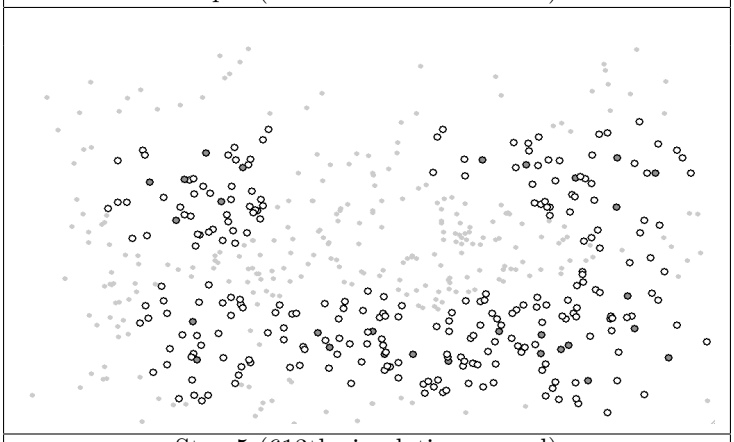


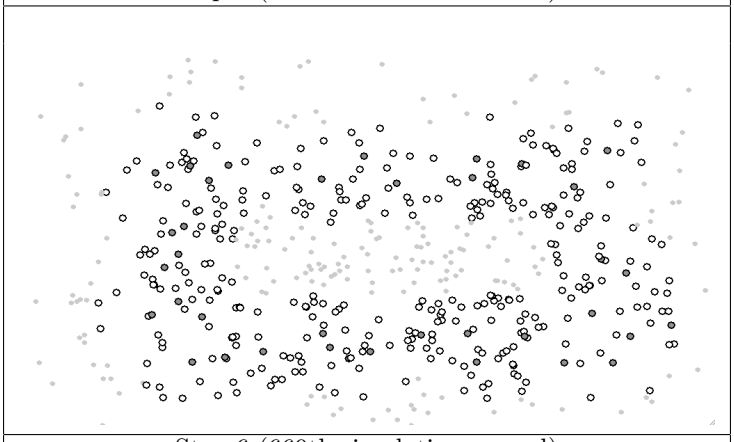
Figure A.2: Attractor Point Repulsion - Convergence - Steps



Step 4 (500th simulation second)



Step 5 (613th simulation second)



Step 6 (669th simulation second)

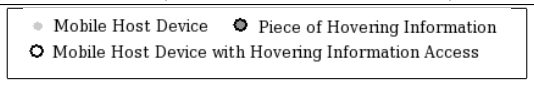
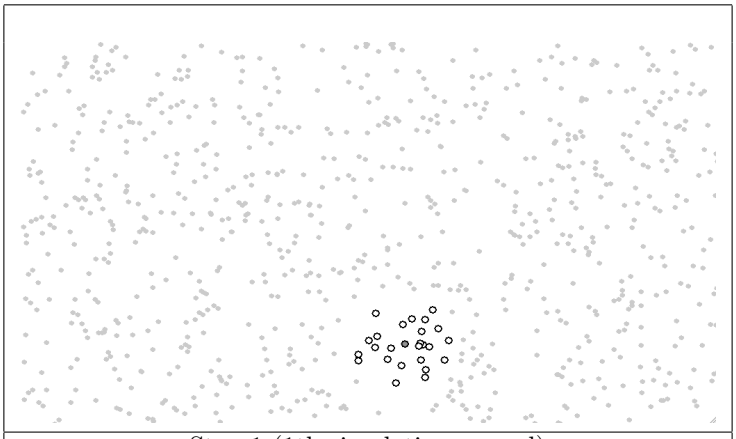
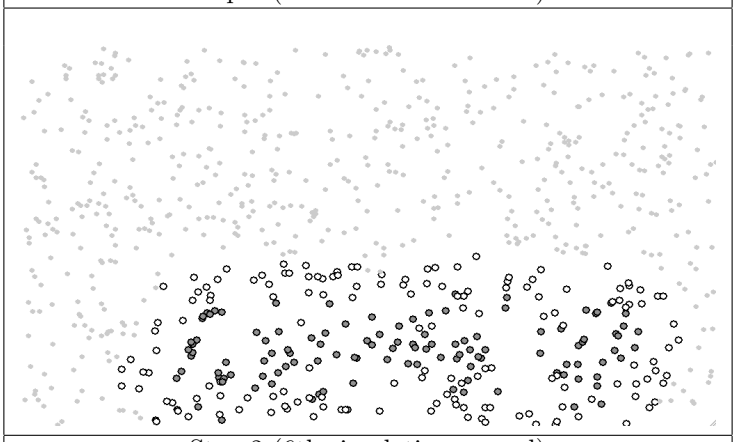


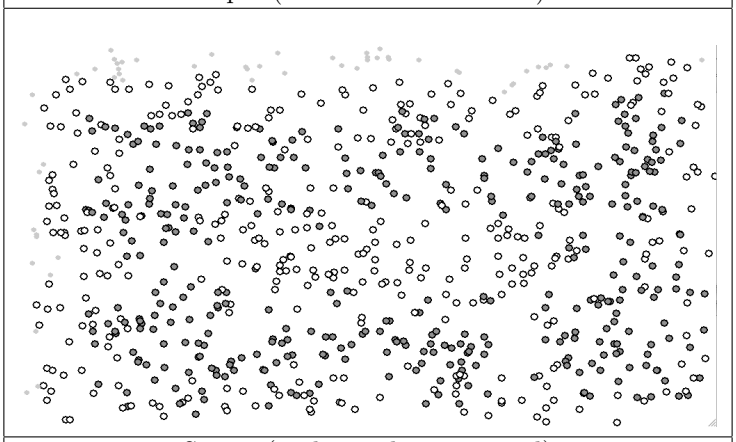
Figure A.3: Attractor Point Repulsion - nodes in area fail - Steps



Step 1 (1th simulation second)



Step 2 (6th simulation second)



Step 3 (18th simulation second)

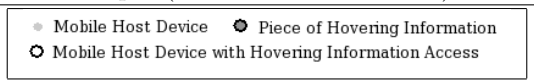
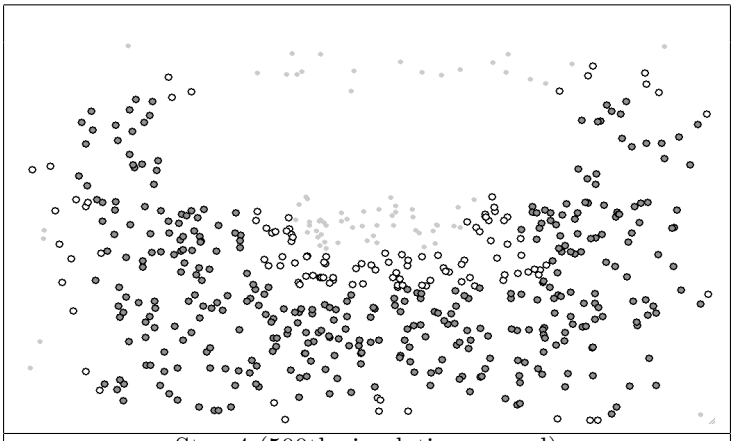
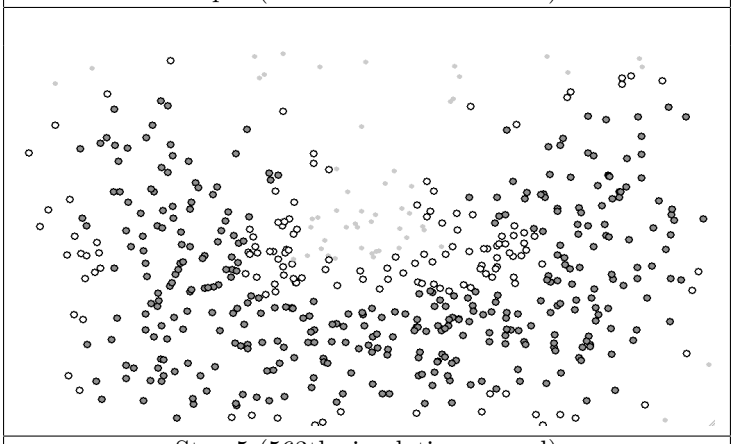


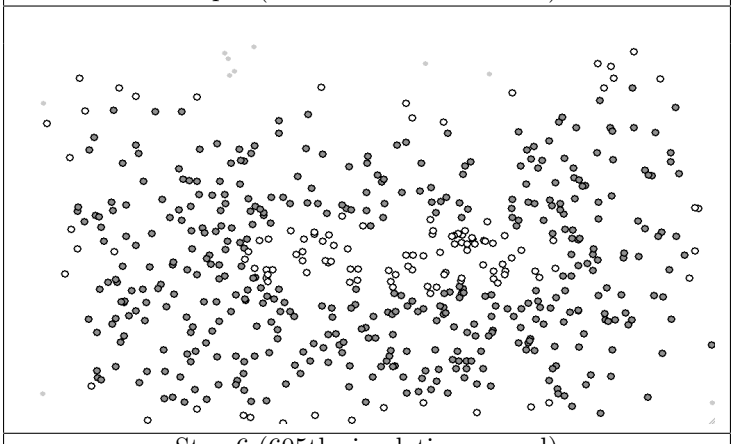
Figure A.4: Broadcast - convergence - Steps



Step 4 (500th simulation second)



Step 5 (563th simulation second)



Step 6 (605th simulation second)

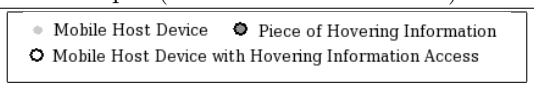


Figure A.5: Broadcast - Nodes in Area Fail - Steps

Figure A.6: Amorphous Shape 5

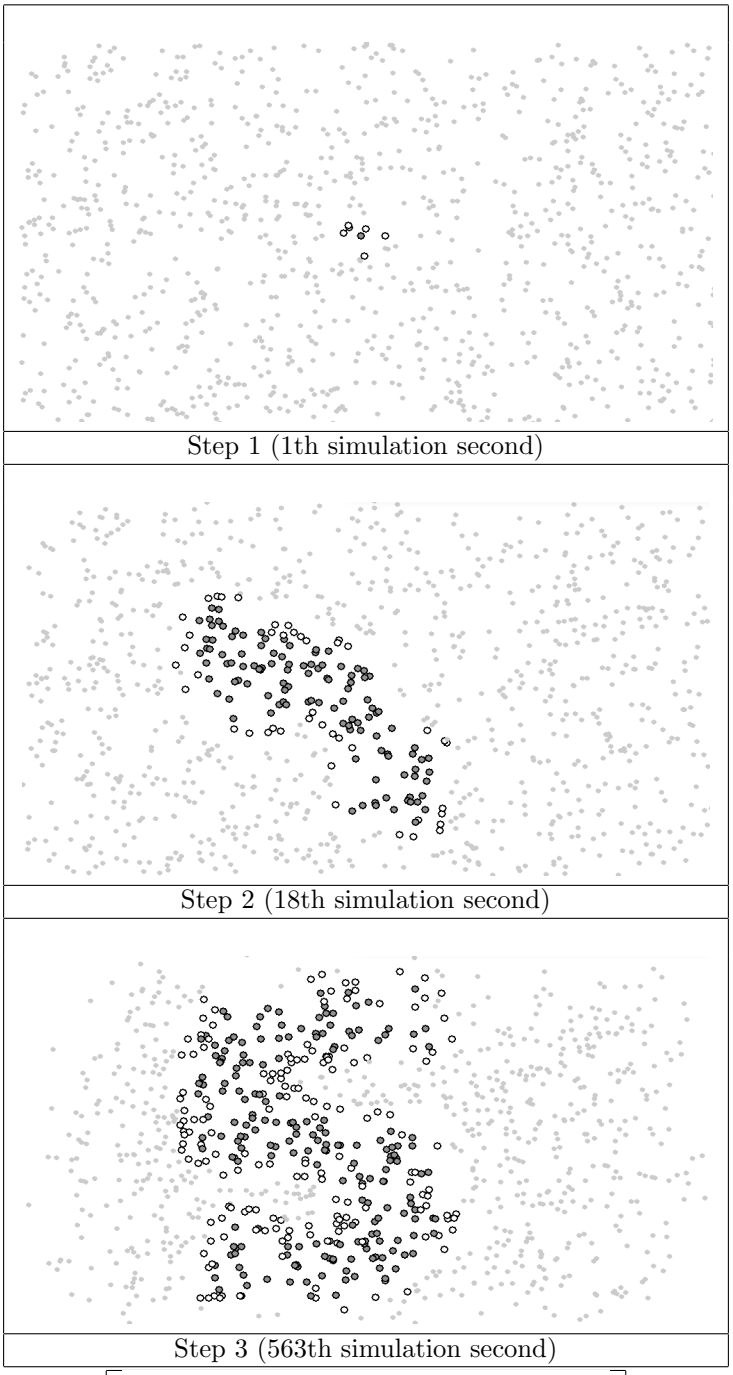
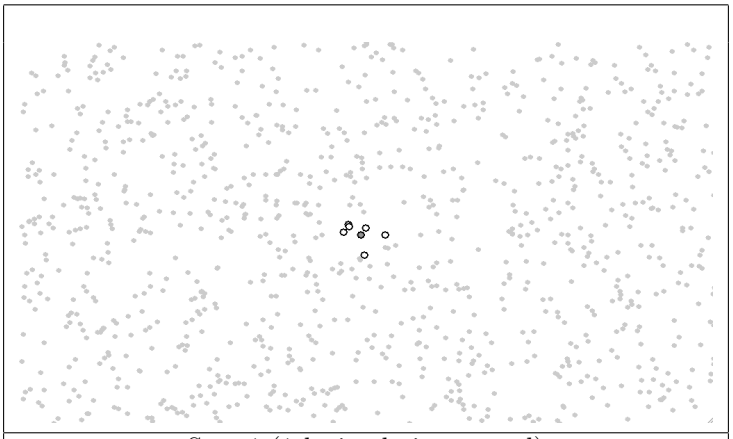
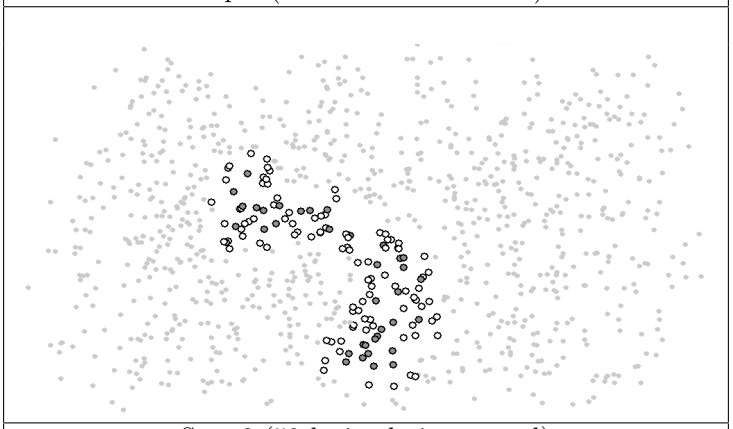


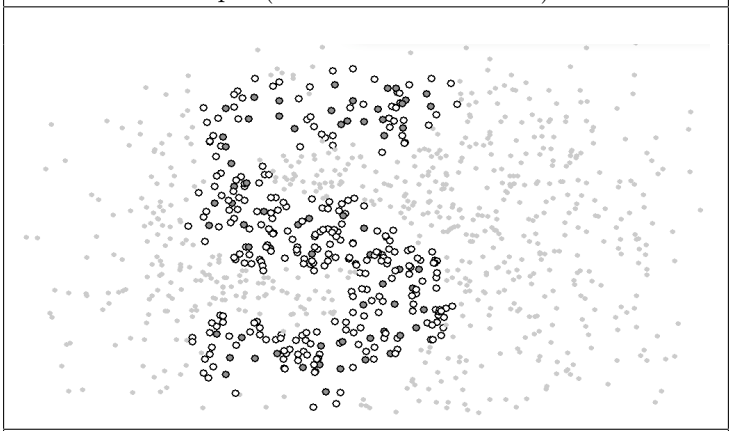
Figure A.7: Broadcast Simulation Steps in scenario '5'



Step 1 (1th simulation second)



Step 2 (50th simulation second)



Step 3 (589th simulation second)

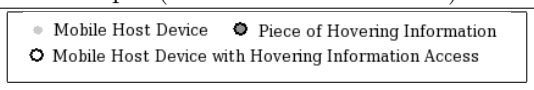


Figure A.8: Broadcast Repulsion Simulation Steps in scenario '5'

Monografies de l'Institut d'Investigació en Intel·ligència Artificial

Num. 1. J. Puyol, *MILORD II: A Language for Knowledge-Based Systems.*

Num. 2. J. Levy, *The Calculus of Refinements, a Formal Specification Model Based on Inclusions.*

Num. 3. Ll.Vila, *On Temporal Representation and Reasoning in Knowledge-Based Systems.*

Num. 4. M. Domingo, *An Expert System Architecture for Identification in Biology.*

Num. 5. E. Armengol, *A Framework for Integrating Learning and Problem Solving.*

Num. 6. J. Ll. Arcos, *The Noos Representation Language.*

Num. 7. J. Larrosa, *Algorithms and Heuristics for Total and Partial Constraint Satisfaction.*

Num. 8. P. Noriega, *Agent Mediated Auctions: The Fishmarket Metaphor.*

Num. 9. F. Manya, *Proof Procedures for Multiple-Valued Propositional Logics.*

Num. 10. W. M. Schorlemmer, *On Specifying and Reasoning with Special Relations.*

Num. 11. M. López-Sánchez, *Approaches to Map Generation by means of Col-laborative Autonomous Robots.*

Num. 12. D. Robertson, *Pragmatics in the Synthesis of Logic Programs.*

Num. 13. P. Faratin, *Automated Service Negotiation between Autonomous Computational Agents.*

Num. 14. J. A. Rodriguez-Aguilar, *On the Design and Construction of Agent-mediated Electronic Institutions.*

Num. 15. T. Alsinet, *Logic Programming with Fuzzy Unification and Imprecise Constants: Possibilistic Semantics and Automated Deduction.*

Num. 16. A. Zapico, *On Axiomatic Foundations for Qualitative Decision Theory A Possibilistic Approach.*

Num. 17. A. Valls, *ClusDM: A Multiple Criteria Decision Method for Heterogeneous Data Sets.*

Num. 18. D. Busquets, *A Multiagent Approach to Qualitative Navigation in Robotics.*

Num. 19. M. Esteva, *Electronic Institutions: from Specification to Development.*

Num. 20. J. Sabater, *Trust and Reputation for Agent Societies.*

Num. 21. J. Cerquides, *Improving Algorithms for Learning Bayesian Network Classifiers.*

Num. 22. M. Villaret, *On Some Variants of Second-Order Unification.*

Num. 23. M. Gómez, *Open, Reusable and Configurable Multi-Agent Systems: A Knowledge Modelling Approach.*

Num. 24. S. Ramchurn, *Multi-Agent Negotiation Using Trust and Persuasion.*

Num. 25. S. Ontañon, *Ensemble Case-Based Learning for Multi-Agent Systems.*

Num. 26. M. Sánchez, *Contributions to Search and Inference Algorithms for CSP and Weighted CSP.*

Num. 27. C. Noguera, *Algebraic Study of Axiomatic Extensions of Triangular Norm Based Fuzzy Logics.*

Num. 28. E. Marchioni, *Functional Definability Issues in Logics Based on Triangular Norms.*

Num. 29. M. Grachten, *Expressivity-Aware Tempo Transformations of Music Performances Using Case Based Reasoning.*

Num. 30. I. Brito, *Distributed Constraint Satisfaction.*

Num. 31. E. Altamirano, *On Non-clausal Horn-like Satisfiability Problems.*

Num. 32. A. Giovannucci, *Computationally Manageable Combinatorial Auctions for Supply Chain Automation.*

Num. 33. R. Ros, *Action Selection in Cooperative Robot Soccer using Case-Based Reasoning.*

Num. 34. A. García-Cerdaña, *On some Implication-free Fragments of Substructural and Fuzzy Logics.*

Num. 35. A. García-Camino, *Normative Regulation of Open Multi-agent Systems.*

Num. 36. A. Ramisa Ayats, *Localization and Object Recognition for Mobile Robots.*

Num. 37. C.G. Baccigalupo, *Poolcasting: an intelligent technique to customise music programmes for their audience.*

Num. 38. J. Planes, *Design and Implementation of Exact MAX-SAT Solvers.*

Num. 39. A. Bogdanovych, *Virtual Institutions.*

Num. 40. J. Nin, *Contributions to Record Linkage for Disclosure Risk Assessment.*

Num. 41. J. Argelich Romá, *Max-SAT Formalisms with Hard and Soft Constraints.*

Num. 42. A. Casali, *On Intentional and Social Agents with Graded Attitudes.*

Num. 43. A. Perreau de Pinnick Bas, *Decentralised Enforcement in Multiagent Networks.*

Num. 44. I. Pinyol Catadau, *Milking the Reputation Cow: Argumentation, Reasoning and Cognitive Agents.*

Num. 45. S. Joseph, *Coherence-based Computational Agency.*

Num. 46. M. Atencia, *Semantic Alignment in the Context of Agent Interaction.*

Num. 47. M. Vinyals, *Exploiting the Structure of Distributed Constraint Optimization Problems to Assess and Bound Coordination Actions in MAS.*

Num. 48. D. Villatoro, *Social Norms for Self-policing Multi-agent Systems and Virtual Societies.*

Num. 49. J. L. Fernandez-Marquez, *Bio-inspired Mechanisms for Self-organising Systems.*