Self-healing Model for Distributed Environments based on Artificial Life Techniques

Arles Ernesto Rodríguez Portela
Systems Engineer, M.Sc. Eng. Systems and Computing, Ph.D.(c)
Code: 300742

Universidad Nacional de Colombia
Engineering Faculty
Computer Systems and Industrial Engineering Department
Bogotá, D.C., Colombia
2018
Self-healing Model for Distributed Environments based on Artificial Life Techniques

ARLES ERNESTO RODRÍGUEZ PORTELA
Systems Engineer, M.Sc. Eng. Systems and Computing, Ph.D.(c)
Code: 300742

Dissertation Presented to Obtain the Title of Ph.D. in Computer and Systems Engineering

Advisor
Jonatan Gómez Perdomo, Ph.D.

Coadvisor
Ada Diaconescu, Ph.D.

Research line
Intelligent Systems

Research group
on Artificial Life ALIFE

Universidad Nacional de Colombia
Engineering Faculty
Computer Systems and Industrial Engineering Department
Bogotá, D.C., Colombia
2018
Title in English

Self-healing Model for Distributed Environments based on Artificial Life Techniques

Título en español

Modelo de auto-recuperación para Sistemas Distribuidos basado en Técnicas de Vida Artificial

Abstract: Nowadays, distributed systems are a fundamental part of cloud-based systems, such as Google and cyber-physical systems like smart cities and electric grids. Achieving robustness and providing failure detection and recovery in distributed systems is a difficult problem because they are subject to local conditions and can fail unexpectedly. The main goal of this research is to define algorithms to achieve robustness and self-healing for solving the problem of failure detection and recovery in distributed systems. This research integrates different approaches inspired from nature: it improves robustness for distributed data-collection tasks performed by failure-prone mobile agents employing techniques inspired from animal foraging and swarm intelligence. Results show how agents are able to collect and replicate data from the entire target space despite agent failures. Then, the performance and robustness of the pheromone-based algorithm and random exploration are studied for data collection in complex networks, with different topologies (Lattice, Small-world, Community and Scale-free). Experimental results show that network topologies impact data collection and synchronisation and that the proposed pheromone-based approach can improve performance and success rates across most networks. Next, a replication based self-healing mechanism is proposed. The proposed replication approach uses communication time-outs to determine agent failure, and learns time-outs automatically to minimise false positives. Finally, a model to self-heal the structure of a complex network from node failures is proposed. This model differs from existing approaches in the creation of replicas from existing failing nodes and its links instead of rewiring the network to recover its functionality. Experimental results show that it is possible to recover failures in nodes if nodes know the topology. However, in some cases the topology is unknown or changes dynamically. To solve this problem, the data-collection strategies studied previously are applied to synchronise the network topology. Results show the benefits of this approach with respect to a reference multicast-based solution. By using mobile agents, a good part of the network is maintained with lesser overloads in terms of number of messages compared with multicast. Additionally, the strategy to replicate failing mobile agents is extended to deal with failures in nodes, making possible for agents to synchronise the topology data and to enable nodes holding this information to recover failed agents and neighbouring nodes at the same time. The obtained results provide key information that may help to design distributed systems covering applications like sensor networks, swarm robotics, server clusters, clouds and Internet of Things (IoT).
**Resumen:** Actualmente, los sistemas distribuidos son una parte fundamental de sistemas basados en la nube, tales como Google y sistemas ciber físicos como ciudades inteligentes y redes eléctricas. Obtener robustez y proveer detección y recuperación de fallas en sistemas distribuidos es un problema difícil porque dichos sistemas están sujetos a condiciones locales y pueden fallar inesperadamente. El objetivo principal de esta investigación es definir algunos algoritmos para lograr robustez y auto-recuperación para resolver el problema de detección y recuperación de fallas en sistemas distribuidos. Esta investigación integra diferentes enfoques inspirados en la naturaleza: mejora la robustez en una tarea de recopilación de datos distribuidos usando agentes móviles propensos a fallas basados en la búsqueda de alimento de animales y especialmente en la inteligencia de enjambres. Los resultados muestran cómo los enjambres recogen y replican información de todo el espacio incluso cuando ocurren fallas. A continuación, se estudia el rendimiento y la robustez del modelo basado en feromonas y la exploración aleatoria para recopilar datos en redes complejas con diferentes topologías (Lattice, Small-world, Community y Scale-free). Los resultados experimentales muestran cómo las topologías de red impactan en la recopilación y sincronización de datos y cómo el enfoque de enjambres propuesto puede mejorar el rendimiento y las tasas de éxito en la mayoría de las redes. A continuación, se define una técnica de auto-curación que crea réplicas de agentes anómalos que les permite completar una tarea de sincronización de datos incluso para altas tasas de fallas. El enfoque de repilación propuesto aprende y estima los tiempos límite del movimiento de agentes minimizando los falsos positivos. Finalmente, se propone un modelo para auto-curar la estructura de una red compleja a partir de fallas en los nodos. Una diferencia con otros trabajos revisados, es que el modelo crea réplicas de los nodos que falleen y sus enlaces en lugar de reconectar la red para recuperar la funcionalidad. Los resultados experimentales muestran que es posible recuperar fallas en los nodos si los nodos conocen la topología. Sin embargo, hay casos en que la topología es desconocida o cambia dinámicamente. Para resolver este problema, las estrategias estudiadas para recopilar datos se aplican para sincronizar la topología de la red y compararlas con respecto a una solución de multidifusión de referencia. Mediante el uso de agentes móviles, una buena parte de la red se mantiene con menos sobrecargas en términos de cantidad de mensajes en comparación con multidifusión. Además, la estrategia para replicar los agentes móviles que falleen se amplía para tratar las fallas en los nodos haciendo posible que los agentes sincronicen los datos de topología y logrando que los nodos puedan recuperar los agentes móviles y otros nodos al mismo tiempo. Los resultados obtenidos proveen información clave que puede ayudar a diseñar sistemas distribuidos cubriendo aplicaciones como redes de sensores, robótica de enjambres, clústeres de servidores, nubes e Internet de las cosas.

**Keywords:** Self-healing, distributed systems, multi-agent systems, animal motion, local interactions, complex networks, self-organisation

**Palabras clave:** Auto-reparación, Sistemas Distribuidos, Sistemas Multi-agente, Movimiento de animales, interacciones locales, redes complejas, auto-organización
Dedication

This work is dedicated to my mom María, my brother Johan, my sister Mónica, my love Yiyiz and my niece Sofía, as well as the people of the Alife research group from Universidad Nacional de Colombia.
Acknowledgements

To Dr. Jonatan Gómez, for his advice, support, encouragement and for his incredible ideas.

To Dr. Ada Diaconescu, for allowing my internship in Telecom ParisTech, for her advice, dedication and continuous help and support.

To my dissertation Committee members for taking some minutes of their valuable time to read this work and for their valuable suggestions.
Contents

List of Tables IV

List of Figures V

Introduction VIII

1. Background and Related Work 1
   1.1 Distributed Systems ................................................. 1
      1.1.1 Simulation of Distributed Systems ......................... 1
      1.1.2 Distributed Data Collection and Replication Tasks ....... 3
         1.1.2.1 Terrain Exploration and Coverage ....................... 3
         1.1.2.2 Data Collection and Synchronisation in Sensor networks, 
            Data Centres, Server Farms and Clouds .................. 4
   1.2 Autonomic Computing and Self-healing ........................ 5
      1.2.1 Autonomic System Architecture ............................. 6
      1.2.2 Self-healing .................................................. 6
      1.2.3 Network Self-healing and Self-organisation ............... 7

2. Failure Resistant Mobile Distributed Systems: Self-Organised Terrain Exploration 10
   2.1 Towards Failure-Resistant Mobile Distributed Systems Inspired by Swarm Intelligence and Trophallaxis ................................. 11
      2.1.1 The Problem: World exploration ............................. 13
         2.1.1.1 Termites Definition .................................. 14
         2.1.1.2 Failure Definition .................................... 15
CONTENTS

2.1.2 Solution Approaches .................................................. 16
   2.1.2.1 Sequential Exploration with One Termite .................. 16
   2.1.2.2 Random Exploration with Communication .................. 17
   2.1.2.3 Lévy Walk Exploration With Communication ............... 18
   2.1.2.4 Pheromone-based Exploration ............................... 18

2.1.3 Results Analysis .................................................... 23

2.2 Foraging-Inspired Self-Organisation for Terrain Exploration with Failure-Prone Agents .......................... 25
   2.2.1 Basic Approaches .................................................. 25
   2.2.2 Experiments and Results ........................................ 27
   2.2.3 Hybrid Approaches and Evaporation .......................... 30
      2.2.3.1 Lévy Walkers and Carriers ................................ 30
      2.2.3.2 Lévy Walkers and Carriers with Lévy Walks .......... 31
      2.2.3.3 Carriers and Pheromone Evaporation .................... 31
      2.2.3.4 Carriers with Lévy Walks and Pheromone Evaporation .. 32
   2.2.4 Experiments and Results with Hybrid Approaches and Evaporation ........................................ 32
   2.2.5 Discussion .......................................................... 35

2.3 Summary ................................................................. 35

3. Self-organised Data Collection in Complex Networks .......................... 37
   3.1 Introduction ........................................................... 38
   3.2 The Problem of Data-Collection in Complex Networks .............. 39
      3.2.1 Small-world Networks ........................................ 40
      3.2.2 Scale-free Networks ......................................... 40
      3.2.3 Community Networks ......................................... 41
      3.2.4 Forest Hub & Spoke ........................................ 41
      3.2.5 Line, Circle and Lattice .................................... 42
   3.3 Agent Motion Algorithms and Failures ................................ 42
   3.4 Experiments and Results ............................................ 45
      3.4.1 Experimental settings ....................................... 45
      3.4.2 Results and discussion ..................................... 46
   3.5 Summary ............................................................... 55
4. Replication-Based Self-healing of Mobile Agents Exploring Complex Networks

4.1 Introduction ..................................................... 56
4.2 Data Replication Problem ..................................... 58
4.3 Agent Replication Approach .................................. 60
4.4 Experiments and Results ...................................... 61
   4.4.1 Experiments without agent movement delays .......... 62
   4.4.2 Experiments with agent movement delay ............... 63
   4.4.3 Results in terms of round number ..................... 63
   4.4.4 Results in terms of memory consumption .......... 65
4.5 Summary ......................................................... 66

5. Self-healing Networks via Self-organising Mobile Agents 67

5.1 Introduction ..................................................... 68
5.2 Network Self-healing Problem ................................. 69
5.3 Proposed Approach ............................................ 72
5.4 Data Collection Techniques .................................. 73
5.5 Agent replication algorithm .................................. 74
   5.5.1 Emergence of loops in agents replication ............ 76
   5.5.2 Modification of algorithm to produce experiments with no loops .. 76
5.6 Experiments and Results ..................................... 77
   5.6.1 Analysis of Mobile Agents Replication ................ 78
   5.6.2 Defining an initial round for failures .................. 81
5.7 Summary ......................................................... 82

Conclusions and Future Work 84

Bibliography 89
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Summary of Experiments for Sequential Exploration ( \text{Population} = 1, width = 50, height = 50 ). A ( p_f = 4 \times 10^{-4} ) is added because ( 1/(\text{width} \times \text{height}) = 4 \times 10^{-4} ).</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Summary of average of information collected and number of agents with complete information ( \text{pop} = 10, 30, 50, width = 50, height = 50, \text{maxiter} = 3000 ).</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>Summary of averages of the number of rounds required for the best agents to collect all information ( \text{pop} = 10, 30, 50, width = 50, height = 50, \text{maxiter} = 3000 ).</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>Information Collected and Success Rates</td>
<td>28</td>
</tr>
<tr>
<td>2.5</td>
<td>Success rates for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C-Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-Lw and Evap)</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>ANOVA and Wilcoxon Test for Carriers vs. Random by Topology</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>False Positives with Replication and without Delay</td>
<td>62</td>
</tr>
<tr>
<td>4.2</td>
<td>Differences in Rounds of Replication with Delay and No Replication with a ( p_f = 0 )</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>Example of chain protocol and followedAgents vector by node</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Example of chain protocol and followedAgents by node</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Architecture of an Autonomic System based on [41] .......................... 7
1.2 Self-healing Process ................................................................. 8

2.1 World Exploration Using Stigmergy with 10 Agents: white points are seekers and blue points are carriers; unexplored terrain is initially light blue; pheromone traces are red and fade into blue as concentrations decrease. ... 16
2.2 Sequential Exploration Algorithm .................................................. 17
2.3 Communication and Status Determination of Termite r. ....................... 21
2.4 Pheromone exploration with 10 termites, white points are SEEKERS and blue points are CARRIERS. .......................................................... 21
2.5 SEEKERS (red line) vs. CARRIERS (blue line), $p_f = 0$ a) $Pop = 10$, b) $Pop = 30$, c) $Pop = 50$. ............................................................... 22
2.6 Random, Levy Walk and Pheromone Exploration .................................. 24
2.7 Overview of Basic Solution Approaches ............................................. 26
2.8 Box-plot of average information collected by agents for random motions (random), Lévy walks (levywalk), Seekers and Carriers (sandc) and Seekers and Carriers with Lévy walks (sandclw). ........................................ 27
2.9 Box-Plot of round number for random motions (random), Lévy walks (levy-walk), Seekers and Carriers (sandc) and Seekers and Carriers with Lévy walks (sandclw). ........................................ 29
2.10 Box-Plot of messages sent for random motions (random), Lévy walks (levy-walk), Seekers and Carriers (sandc) and Seekers and Carriers with Lévy Walks (sandclw). ........................................ 30
2.11 Box-plot of information collected for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C. w Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-LW and Evap) .............................. 33
2.12 Box-Plot of round number for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C. w Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-LW and Evap) ........................................ 34

2.13 Box-Plot of messages sent for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C. w Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-LW and Evap) ........................................ 34

3.1 Small-world Networks: \( n = 100, k = 4, a) \beta = 0.3, b) \beta = 0.5, c) \beta = 0.9 \) ........................................ 40

3.2 Scale-free Networks: \( sn = 4, steps = 97, a) \eta = 1, b) \eta = 2, c) \eta = 4 \) ........................................ 41

3.3 Community Networks: a) Community Central, and b) Community Circle Network: \( n_{clusters} = 4, \beta = 0.5, degree = 4 \) and \( m = 100 \) ........................................ 42

3.4 Forest Hub & Spoke ........................................ 42

3.5 Line, Circle and Lattice ........................................ 43

3.6 Box-plot of Round Number for Some Selected Complex Networks with \( p_f = 0 \) ........................................ 48

3.7 Betweenness Centrality (std. dev.) Correlation to the Round Number (log) ........................................ 49

3.8 Global Information Collected for Scale-free \( sn = 4, \eta = 1, steps = 97 \) and \( p_f = 0 \) ........................................ 50

3.9 Success Rates and Messages Sent for \( p_f = 0.001 \) ........................................ 51

3.10 Global Information Collected for Scale-free \( sn = 4, \eta = 1 \) and \( steps = 97 \) and Circle, with \( p_f = 0.001 \) ........................................ 52

3.11 Success Rates and Messages Sent for \( p_f = 0.003 \) ........................................ 52

3.12 Global Information Collected for Community Network \( \beta = 0.3 \) and Forest Hub-and-Spoke, with \( p_f = 0.003 \) ........................................ 53

3.13 Success Rates For Complex Networks \( p_f = 0.005 \) ........................................ 53

3.14 Community Network Beta=0.5. a) Box-plot of round number with \( p_f = 0.001, b), c) and d) Success rates for \( p_f = 0.003,0.005 \) and \( 0.008 \) respectively ........................................ 54

4.1 Complex Network Topologies Selected for Experiments ........................................ 59

4.2 \( p_f \) vs. success rates for Community network \( n = 100, \beta = 0.5, degree = 4 \) with no replication ........................................ 62

4.3 Simulation sorted by date vs number of agents (max, median, min) in a Small-World network \( n = 100, \beta = 0.5, degree = 4 \) ........................................ 63

4.4 Simulations Sorted by Date vs. Number of Agents (max, median, min) in a Small-world Network \( n = 100, \beta = 0.5, degree = 4 \) with delay ........................................ 64
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5 a) Simulation sorted by date vs. false positives b) Number of rounds necessary to successfully complete a simulation versus $pf$ - Small-World network $\beta = 0.5$, degree $= 4$ with replication</td>
<td>65</td>
</tr>
<tr>
<td>4.6 Simulation Sorted by Date vs. Memory Consumption</td>
<td>65</td>
</tr>
<tr>
<td>5.1 Complex Networks Selected for Experiments</td>
<td>70</td>
</tr>
<tr>
<td>5.2 Round Number vs. Network Similarity</td>
<td>71</td>
</tr>
<tr>
<td>5.3 Summary of Results for the Selected Networks</td>
<td>79</td>
</tr>
<tr>
<td>5.4 Box-plots of round vs number of agents generated for the selected Small-world network with an initial population of 10 agents a) no loop b) with loop</td>
<td>80</td>
</tr>
<tr>
<td>5.5 Box-plot of nodes sorted by degree versus number of replicas generated for a) Forest Hub &amp; Spoke and b) Scale-free with 10 agents</td>
<td>81</td>
</tr>
<tr>
<td>5.6 Consolidated similarities of experiments without failures for 100, 250 and 1000 rounds</td>
<td>83</td>
</tr>
</tbody>
</table>
Introduction

Nowadays, most computing applications rely on distributed systems, from cloud-based systems, such as Google, Amazon, Facebook and Apple, to cyber-physical systems, such as smart cities, building, electric grids and cars. For example, Facebook looks to the end user like a single application. However, in the background, many independent components communicate and cooperate via a network to offer the services of messages and inbox, friends, newsfeeds, likes and reactions, comments, etc [47]. The ability to hide different processes and resources interacting is known as transparency [93]. This feature is hard to obtain because each component acts locally, there is no shared memory, each component only knows local past states of others while performing computations, and individual components can fail [66].

Information and Communication Technology (ICT) organisations invest important resources in the prevention and recovery of distributed systems from failures [54]. Corporate access to services requires availability rates of 99.999% and downtime costs are in the order of thousands of dollars per minute [43]. Maintenance operations require to be as fast and reliable as possible and the main investments are focused on training system administrators with the ability to deal with and manage different specific technologies [43]. Maintenance is a difficult task in large distributed environments and human mistakes may happen. System administration expertise is expensive, there may be a shortage of experts because of the extensiveness of distributed systems requiring maintenance, and even experts may be unable to react fast enough to deal with failures within minutes [43]. In 2017, a disruption in the Amazon s3 service occurred due to a command entered incorrectly that removed a large set of servers causing failures in other important subsystems [2]. Hence, fast automated solutions are needed to enable systems to self-heal.

To help system administrators in their management task and inspired from nature, the concept of self-healing was proposed in 2001 [41]. This concept arises from the autonomic nervous system and its capacity to self-regulate vital functions like heart rates and body temperature. Applying this idea to computing systems, the aim is to have a system able to manage itself. Specifically, self-healing is an adaptation process that provides a system with the ability to detect components that are failing and then replace, eliminate or repair
them without disruptions in the system operation [53]. Additionally, a system has self-
healing abilities if it can monitor and recover itself by deciding and performing recovery
actions according to an initial specification and without external help [68].

Heartbeats are the most used technique to monitor the “well-being” of distributed
systems and detect failures. Using heartbeats, an active process in a component regularly
sends a message to other processes to announce that it is still up and running [93]. However,
in large distributed systems, determining which processes send heartbeats to others is
a problem because the number of monitoring messages increases with the network size
and scalability problems may appear [84]. To solve this problem different approaches have
created groups making a process being monitored by a small number of other processes
[37]. Groups can be created randomly [37, 40] or by defining specific heuristics [84].

In terms of the recovery actions there are some approaches that include isolation of
components to avoid cascading errors [21], handling emotions to increase social abilities
between the system elements producing intelligent decision-making [104], and integrat-
ing management information to maintain a significant a priori knowledge and incorporate
new knowledge about failures and possible healing actions [90]. Additionally, other works
concentrate on self-repair the network structure of large scale distributed systems by cre-
ating new links between remaining nodes when nodes fail to avoid network splitting [29].
Other approaches are focused on distributed consensus in case of failures to provide data
replication based on a leader election process [59].

Self-healing approaches in distributed systems can be extended to other applications
like space exploration. A spacecraft in [85] is proposed as a distributed collection of pico-
satellites. Authors mentioned that future missions will treat communications in these pico-
satellites as a network structure instead of using point-to-point communication between
the space vehicle and a ground station. In these environments, self-healing is required to
maintain the communications infrastructure even when some pico-satellites fail due to the
hostility of the outer space.

In [69, 7, 70] a self-healing mechanism for repairing distributed programs of mobile
agents exploring a bi-dimensional environment was proposed. This mechanism allowed
agents to detect and repair failures without a central control while they carried-out food
searching activities using stigmergy. To achieve that, a set of diagnosis and healing actions
were included in the behaviour of the agents. These actions allowed each agent to learn
and provide feedback about possible failures and changes that must be included in other
agents based on its current status. With time, failures are repaired and self-healing is
obtained as an emergent property.

From these previous works some of the outstanding issues considered in this thesis
include:

- Detect, locate and diagnose failures in the shortest possible time [41].
INTRODUCTION

• Determine and implement a reconfiguration that ensures the continuity of the system.

• Provide a distributed simulation that allows to define different distributed environments explored by mobile agents in order to explore and evaluate different self-healing mechanisms, rather than only heartbeats.

• For reasons of scalability and robustness in distributed systems, self-healing mechanisms should be decentralised without dependence on a leader election (at least in case of having complementary information).

• Introduce learning functions to enhance self-healing capabilities about corrective actions.

• Explore different algorithms inspired from nature, evaluate their suitability and robustness in data collection and replication scenarios. This mechanism must be fully decentralised without dependence on a leader election (at least in case of having complementary information).

• Perform experiments in distributed systems with different network topologies. In experiments are done in a community network with 3 clusters and in only the lattice topology is considered in experiments.

• Heal nodes and rebuild the network structure: with the growing virtualisation of servers in a cloud, some self-healing approaches can be proposed to repair instances of services or virtual machines.

Problem Definition

Taking into account some previous researches, this dissertation addresses the problem of providing a distributed system with the ability to detect and recover from failures by itself. Inspired from nature and from previous works, three main sub-problems identified are: 1) determining distributed system behaviours to simulate; 2) defining different types of failures in distributed systems; and, 3) developing self-healing processes based on artificial life techniques.

The first sub-problem that arises is how to model distributed behaviours using a simulated multi-agent environment. Having an accurate simulation environment replicating the relevant behavioural aspects of targeted systems is essential for accurately evaluating various self-healing approaches applicable to those systems. A multi-agent system is chosen according to its ability to define autonomous components and local behaviours (e.g. failures, local communications, agents movement, etc.). Additionally, animal foraging techniques can be modelled via agents and can be the base for defining a data collection and synchronisation task in distributed environments. This task is important because it provides a base to maintain information in case of failures in real distributed applications.
like terrain exploration, network coverage and data replication in server farms or data centres.

The second sub-problem is to define failures and evaluate robustness in the selected distributed environments. Three types of failures are considered here. Firstly, agents may fail while performing their tasks (e.g. failing software, or robot agents running out of battery). Secondly, in networked environments, agents may be lost in communication (e.g. unreliable transmission in mobile ad-hoc networks). Thirdly, network nodes may fail, potentially disrupting communication between neighbouring nodes and/or causing the failure of software agents executing on those nodes.

The third sub-problem is to deal with the defined failures. This part implies to build a monitoring, detection and self-healing model inspired by local interactions and self-organisation. Initially, assumptions and constraints must be defined for the self-healing model. Next, it is required to evaluate the suitability of the proposed approach in terms of recovery, scalability and overheads of the self-healing process. This work aims to recover mobile agents from failures providing a more robust data collection task and maintaining a network structure with insights in applications like clouds or server farms.

Goal

The purpose of this work is to develop a self-healing model for distributed environments based on artificial life techniques. This dissertation concentrates on the following objectives:

**To model distributed system behaviours using a simulated multi-agent environment:** A multi-agent simulator is developed for exploring decentralised self-healing functions and evaluating robustness in distributed systems. Within this simulator, experiments with failure-prone agents are modelled [75]. They cooperate to achieve collective data-management tasks in different types of distributed environments, such as data collection from uncharted terrains [71] and data synchronisation across complex networks [73]. Agents explore their environment, collect and update local data, and exchange data with agents that they encounter, until the collective task is completed.

**To define different types of failures of distributed systems in the multi-agent environment:** This work models different types of failures based on the notion of failure probability. The first part of this work models failures in agents while they perform a data collection task in bi-dimensional environments and complex networks. The second part of this work presents failures in links that produce agent failure while agents move between nodes. The third part models failures in nodes producing the destruction of a network topology that makes impossible for a system to complete distributed computations and also produces cascade failures in mobile agents.
To build a failure monitoring and detection process based on message propagation and on the formation of autonomous groups between agents: First, experiments intend to determine which exploration strategies are more robust in case of agent failures, faster in terms of simulation rounds, and lighter in terms of resource overheads. Robustness is the starting point to propose a self-healing function based on local agent replication of mobile agents. In short, each distributed node keeps track of agents departing from neighbouring nodes. Upon arrival at a new node, agents send a confirmation message back to their departing node, which consequently stops tracking them. Additionally, nodes are provided with a mechanism to detect failures in other nodes based on internal information of local perceptions and external information provided via multicast and by mobile agents.

To design a planning model based on an autonomous selection of possible healing actions: On the one hand, nodes are able to recover mobile agents that fail during communication by approximating a time-out based on local messages. Furthermore, a model to recover a complex network structure is proposed. This model consists in evaluating the difference between external information (from mobile agents and via multicast) and the local perception of each node about its neighbours. New replicas of nodes and their links are recreated obtaining a network similar to the original. On the other hand, the replication algorithm proposed in case of communication failures is extended to create replicas of mobile agents that fail when nodes crash. In consequence, the proposed approach is designed to maintain some mobile agents synchronising the topology information with the actual network structure at the same time. Nodes require mobile agents to synchronise the network topology information and agents require nodes to be recreated in case of failures.

Main Contributions

The main contribution of this work is the development of several algorithms to achieve robustness and self-healing of distributed systems based on artificial life techniques. These techniques are proposed for applications such as data collection and synchronisation in distributed environments. Also, the data-collection application is employed to enable agents to collect and share information about a network's topology, which is in turn essential for self-repairing the network structure. A model to recover mobile agents while collecting data is also provided. The main contributions of this dissertations are summarised as follows.

Failure Resistance in Mobile Distributed Systems via Swarm Intelligence and Trophallaxis

This work addresses the problem of collecting and synchronising data in a bidimensional environment via a swarm of failing agents [72, 71]. Particularly, this work:
• Proposes to model agent failure via a predefined (configurable) probability, which makes agents crash while they explore, get local data and collect information.

• Presents an agent communication mechanism allowing neighbouring agents (i.e. situated on the same node) to exchange their local information enabling agents to acquire global data faster via cooperation. These local exchanges speed-up global data collection.

• Studies different decentralised techniques inspired from animal motions like Lévy walks and stigmergy-based approaches.

• Analyses exploration approaches in terms of speed of data collection, failure resistance, messages exchanges and scalability.

Completion speed is an essential characteristic to achieve better success rates. Mechanisms that favour exploration are more resistant to failures than those that focus on increasing communication among agents. Additionally, increasing agents density also speeds up the data collection process.

**Self-organised Data Collection in Complex Networks**

This work studies robustness and performance of different multi-agent algorithms for data collection and synchronisation in complex networks [73]. Specifically, this work:

• Models different types of network topologies, both traditional (line, circle, lattice, Hub & Spoke, Forest Hub & Spoke) and complex (Small-world, Community and Scale-free) with several parameters.

• Adapts the stigmergy model proposed for bi-dimensional environments to enable mobile agents to explore and collect data from nodes across various targeted networks.

• Simulates different failure probabilities in mobile agents.

• Profiles the suitability of the proposed exploration approaches with respect to various network topologies and to various failure rates.

Experimental results show that a pheromone-based exploration technique may result in a faster data collection for most experiments in different topologies and, in consequence, in a more robust exploration. Additionally, by observing some network parameters, a possible relation was found between the network topology and the data-collection performance given by the speed of data collection in rounds and the standard deviation of the between-ness centrality. A higher deviation indicates a higher data collection time. Experimental results also revealed that different exploration algorithms (e.g. random) performed better in certain network topologies (e.g. hub and spoke or scale-free with high degree of the hubs).
Replication-Based Self-healing of Mobile Agents Exploring Complex Networks

This work introduces a first self-healing technique. This technique consists in the replication of mobile agents that explore complex networks and that may fail during communication between network nodes [74]. Failed agents are recreated by nodes based on local interactions. More specifically, this work:

- Defines a failure probability when agents move among nodes.
- Models different communication delays between network nodes, hence requiring adaptation of time-out estimates to avoid false positives that would lead to agent over-replication.
- Proposes a model that maintains references of agents that leave a node and move to another one. Local time-outs are estimated by agent notifications.
- Proposes a replication model that ensures task completion even with relatively high failure rates. If a node does not receive a notification from an agent in this expected delay, the node creates a new replica of this mobile agent and injects its local data aggregate into it.
- Analyses experimental results in terms of success in the data synchronisation task, speed of data collection with and without agent replication, memory consumption and false positives.

Experimental results show that the proposed agent replication approach avoids over-replication and under-replication by learning to approximate time-outs, which enables the agent population to maintain (more or less) its initial size, despite agent failures.

Self-healing Networks via Self-organising Mobile Agents

This work introduces a model to self-heal a complex network structure from node failures. The model replicates failed nodes and recreates their connections to neighbouring nodes, hence preserving network topology. In particular, this work:

- Models node failures via a failure probability; in turn, node failures also cause the loss of the agents visiting those nodes.
- Establishes assumptions regarding the frequency of failures and the speed of the healing process.
- Proposes an algorithm that recreates nodes and their links when they fail.
- Defines two types of agents: static agents that manage the network self-repair process and mobile agents that collect and distribute information about network topology.
• Compares the performance of multicast and mobile agents when the topology is unknown or changes dynamically.

• Extends the replication model of mobile agents proposed in [74], making possible to recover agents when nodes fail.

• Performs experiments that maintains the network structure and mobile agents at the same time.

Experiments show how mobile agents are able to collect and synchronise data about the topology of a network. The network structure is recovered with less overheads in terms of number of messages when compared to a reference multicast-based approach. Experimental results also present a relation in the degree of each node and the number of replicas of mobile agents produced.

**Dissertation Outline**

The remainder of this work is organised as follows:

**Chapter 1** provides the background on the concepts and related work used to propose the self-healing model and some differences with some existing approaches in the literature.

**Chapter 2** introduces the bi-dimensional environment and the problem of robustness in a data collection task with failure-prone agents. Additionally, several approaches based on animal foraging are studied and their failure resistance is evaluated in terms of success rates, messages overheads and round number to task completion.

**Chapter 3** introduces mobile agents performing data collection in selected complex networks with different failure probabilities. Movement algorithms are profiled in terms of their robustness, speed in rounds, and type of network, global information collected, success rates versus failure probability and overheads in terms of number of messages.

**Chapter 4** presents the replication algorithm of mobile agents if they fail while moving between nodes.

**Chapter 5** details the network self-healing approach and extends the replication model of mobile agents.

Finally some conclusions and future work are outlined.
Background and Related Work

Some concepts are key to propose a self-healing model for distributed environments based on artificial life techniques. First, the distributed system definition addressed in this dissertation is presented with some related work to simulate it. Then, the self-healing property that arises as an adaptation process in autonomic computing is defined. After that, some ways to achieve self-healing in distributed systems are introduced.

1.1 Distributed Systems

A distributed system is a collection of autonomous components (e.g. nodes, processes, agents, robots) that coordinate their activities and share resources via a network to offer services to end users. Transparency is required because user interaction with a distributed system is offered in a uniform way, thus, it appears to users as a single system [93].

A distributed system can be seen as a “natural” object based on the collective work of connected components through communication and cooperation. Components have partial knowledge of the environment, act locally and may fail [66]. In this fashion, a distributed system should also be easy to expand, making possible to connect or disconnect several components and resources. A distributed system can be scalable in terms of size, since more users and resources can be added in an easy way; in terms of geographical distribution, regarding resources and network topologies; and administratively scalable in terms of management [93].

1.1.1 Simulation of Distributed Systems

Distributed system simulation is a way to test and validate different distributed algorithms, applications and/or configurations independently of specific computing and network resources. In some cases, real distributed systems are not suitable for testing and can be
time consuming, thus, experiments may not be reproducible due to resource dynamics and customization. Software and network devices have many possible standards and implementations that are very difficult to configure and control when understanding, testing or developing distributed algorithms [97]. Consequently, simulation offers a balance between complex theoretical models that can be hardly implemented or unrealistic and the time and resources expended to use and configure a real experimental platform [19].

Scalability and accuracy are the main concerns in distributed systems simulation [19]. A simulation is scalable if it is capable of implementing some validated simulation models, defines different network topologies, simulates resource failures and, if complex behaviours are simulated fastly and with precision. Accuracy refers to get realistic results and can be measured by predicting the time for completing communication of links, estimate bandwidth sharing or the bandwidth behavior of many flows competing on a random topology [97, 28]. A balance between the speed of the simulation and its accuracy is required [97].

Different models of distributed systems simulation include:

- **CPU delays emulation**: In this case, CPU usage is calculated based on an analytical model, which means determining a computing cost and dividing it by a computing speed [88].

- **Network simulation**: It estimates the information about bandwidth (in bytes) and latency time to send data (seconds). The most accurate way to study protocols over distributed systems are Packet Level Simulators [97]. These simulators calculate bandwidth, delay and routing [16, 65] and implement all protocol layers, but are slow when simulating large scale distributed systems. Packet level simulators use discrete-event simulation by which a flow over a network path can be represented as a sequence of events regarding packet arrivals and departures between nodes and routers [97]. The slow simulation times of packet level simulators are even larger than real times, because simulation time increases with the number of packets to simulate [28].

- **Storage Models**: It simulates storage devices based on software configurations such as file-systems and hardware components like buses, controllers or types (e.g. RAID, SSD, etc.) [4].

This work is focused on designing a model based on agents with the aspects of scalability and accuracy. The simulators found in the literature revision do not model the environment and do not model behaviours using the notion of agent. Additionally, these simulations are oriented to specific tasks. This dissertation studies different distributed tasks like communication coverage by mobile robots, robot based terrain exploration, sensor networks decentralised algorithms or data synchronisation in computing clusters or clouds.

Multi-agent Systems (MAS) are used to model complex systems because they are based on an algorithmic description of individuals that simulate a determined expected
behaviour [77]. As a difference with the previously presented models of distributed systems simulation, a MAS uses a bottom-up approach opposed to the top-down approach of formal models. With MAS it is possible to model agent behaviours using discrete time via computation in rounds [66]. Simulating a system using the multi-agent paradigm aims at identifying global behaviours or tendencies from individual behaviours (bottom-up). In the specific case of this work, to model individual behaviours helps to identify the main factors that can increase the robustness and produce self-healing of a simulated distributed system.

For this reason, simulation is designed from scratch to model each component of a distributed system as an agent. In this way, a distributed system is defined as a collection of processes $\prod = \{p_1, p_2, ..., p_j\}$, where $p_i$ is able to communicate with $p_j$ and $p_i, p_j \in \prod$ are nodes connected via a communication channel [84]. Communication is based on sending and receiving messages and can be defined in each node as a buffer. This buffer contains messages that have been sent to the process but not yet received [83]. Communication channels are bidirectional (unless unidirectional is defined) and the ability to send messages between processes or nodes determines the network topology [83].

There are two communication primitives [83, 27]:

- Send($m$, $q$): It sends a message $m \in M$ where $M$ is a fixed universe, to a process $q$.
- Receive($m$, $q$): It cleans the buffer of the process $q$ and processes the message $m$.

### 1.1.2 Distributed Data Collection and Replication Tasks

Main tasks of distributed systems include data-collection and replication. By these tasks, multiples copies of data are created to provide high integrity, availability and performance [5]. The main scenarios that require data-collection and replication include mobile distributed systems, sensor networks, clouds and Internet of Things (IoT) environments.

#### 1.1.2.1 Terrain Exploration and Coverage

Terrain exploration and coverage problem can be defined as a robot or a set of robots visiting every location in a continuous unknown terrain and avoiding obstacles to achieve a determined task [86]. A common task in multi-robot environments is to acquire as most information as possible in the shortest amount of time with the main assumption that robots have zero knowledge about the terrain [63].

Communication coverage intends to form a network over a determined space by defining robots with locomotion and sensing capabilities [103]. In terms of locomotion, algorithms based on random movements in a terrain are simulated and studied [11]. The main principles in communication coverage include moving agents to unexplored areas and avoiding
other agents to maximize the space explored \cite{46}. Additionally, as a main assumption, agents only have local information obtained from its local sensors while exploring.

Animal foraging is studied as a possible way to achieve communication coverage. On the one hand, fruit flies or spider monkeys present certain patterns when looking for food or exploring a determined environment. \cite{12} shows that these patterns can be modelled via Lévy walks. Lévy walks follow a super-diffusive approach that explores and covers a determined space in a faster and effective way compared to random walks or repulsive forces. In \cite{67,80}, Lévy walks are used to model human motion in a determined space to form opportunistic mobile social networks.

Social insects and stigmergy are studied in mobile networks because they provide a decentralised way of achieving coordination using the environment \cite{49}. By stigmergy, a pheromone trace left in the environment guides exploration and exploitation of new paths. Pheromone in terms of robotics can be defined in two main approaches: simulated if the pheromone is defined as a logical grid and the environment is not altered \cite{3,96,61}; or situated, that is, robots left a trace that modifies the environment and use sensors to detect it. In \cite{62} chemical sensors are used to sense alcohol trails and in \cite{32} light sensors for pheromones are implemented to guide agents through light projections.

In \cite{86}, a robustness analysis of a simulated pheromone approach is performed. However analysis depends of the number of agents to estimate if the terrain is marked and there are not experimental results about failures versus amount of terrain explored of success in the terrain exploration task. Additionally, in terms of exploration with unreliable agents, agents sense the world in a defective way in \cite{56} and can obtain information from themselves (direct information) or from others (indirect information). As a result, they achieve coherence in the information as a main result.

Another approach in robust exploration in autonomous robots includes using previous knowledge of environment to guide exploration and create a map of the explored terrain \cite{63}. \cite{63} provides a robustness mechanism to drive the robot back to its starting location with the map if it is consistent or without a map via rewinding the trajectory taken by the robot.

1.1.2.2 Data Collection and Synchronisation in Sensor networks, Data Centres, Server Farms and Clouds

Data centres, server farms and clouds are other kind of distributed systems. They are formed by a huge amount of computing resources connected via a network \cite{93}. Data replication is an essential task in these environments because it provides robustness in case of failures in resources. Some areas for application are secure communications \cite{34}, log machine replication in databases \cite{5,59} and information processing in sensor networks (computing averages of local observations) \cite{60}. 
Consensus is a well known mechanism of replication in distributed systems and provides an agreement over replicas stored locally in each component [5]. Paxos [44] and Raph [59] are consensus approaches based on a leader election process. The leader replicates and decides information to copy in other components. In this work, replication is produced by a self-organised process instead of a leader election. Self-organisation relies on collective behaviours that emerge from local interactions, thus, stigmergy is one of the approaches studied in this work to replicate information via exploration of the network [25]. Another assumption in this work is that information in components is complementary. This aspect differs and is complementary to traditional consensus approaches, where conflicts in information are solved by the leader.

Simulation in this work concentrates on modelling the network structure instead of simulating bandwidth and latency time. The literature revision showed that topological properties impact network performance [18, 98, 44, 58], thus, complex networks, are defined to model some networks in experiments. A complex network is a large collection of interconnected nodes with non-trivial topological properties. Complex networks include some features like relative small distances between nodes [98, 101] or power-law degree distributions [10].

Scale-free networks are complex networks that follow a power-law distribution in terms of degree. A power-law distribution implies that some nodes have a disproportionate number of links compared to the average degree by node. Some examples of real scale-free networks include the WWW, email or protein interaction networks. These networks are highly resistant to accidental failures, but rather vulnerable to targeted node attacks [10].

Small-worlds are a complex network that presents relatively short paths between any pair of network nodes, even in very large networks. As an example, the Internet has evolved in a Small-World way because IP packets cannot use more than a threshold of physical links, thus, typical distances between two points are relatively small [94].

1.2 Autonomic Computing and Self-healing

Autonomic computing was proposed by Paul Horn from IBM for the National Academy of Engineers at Harvard University in 2001. Horn presented a relationship between the autonomic nervous system and computing systems with self-managing [53]. By abstracting the concept from biology to computer science, the following questions were proposed: what these systems should look like? How these systems should work? And what obstacles must be faced when designing and understanding the behaviour of these systems?

According to IBM, autonomic computing is mainly an ability of an ITC (Information Technology and Communications) infrastructure to adapt based on certain policies and objectives. This adaptation is produced by a system without human intervention to self-reconfigure, self-heal, self-optimise, and self-protect [26].
1.2.1 Autonomic System Architecture

An autonomic system can be seen as a set of autonomic elements. Each element is responsible for managing a particular component of a system, its own state and its interactions with an environment [100]. The environment consists in signals and messages from other elements and the external world.

The internal behavior of an autonomic element and the set of relationships with other elements are based on [41]:

- Goals that a designer has embedded in it,
- Goals incorporated by another system and,
- Subcontracts with other elements, with its tacit or explicit consent.

Several elements may require assistance from others to reach their goals, and this assistance is scalable at some levels from a specific role to cover all the system. Individual components are fault-tolerant in a lower level; while, at higher levels, fixed behaviors, connections and relationships provide more flexibility and dynamism to the autonomic system. In high level terms, each element solves troubles while performing its work [41]. Figure 1.1 shows how autonomic elements establish relationships between them to achieve a global goal. This relationships can be of association or engagement.

1.2.2 Self-healing

Self-healing is based on the ability of a system to detect components that are failing while it is performing a task [53]. A system with this property is capable of detecting failures in components and then, replacing, eliminating or repairing itself without disrupting its operation [53].

Self-healing means that the system operates at less than 100%. If a system is capable of restoring all its operational capability, then the system is fault tolerant [68]. Figure 1.2 presents self-healing as an adaptation process of a system that starts a set of healing actions when it reaches a threshold given in terms of deviation of a expected behaviour. As some researches remark [76, 68, 69, 83], a monitoring process starts to determine possible critical states and, a set of recovery actions is carried out to recover the system from this critical state.

Monitoring a system refers to the detection of relevant events by polling the system state or by reports from the system [68]. A monitoring process involves an information interchange among elements of the system, related to information dissemination processes. The way in which nodes communicate determines the kind of monitoring.

Heartbeats are the most used technique to monitor distributed systems. By heartbeats, an active process regularly sends a message to all the other members in a group to
announce that it is still up and running. Whenever such a heartbeat message is missing, another application will automatically activate the highest-ranked process that is currently inactive \[93\]. Some monitoring techniques are based on replication, like performing background audits to determine whether a service is functioning correctly or not. Scalability of monitoring is a critical issue in large scale distributed systems because of the number of monitoring messages and the network size.

To deal with this problem, some strategies are based on self-organisation and consider the suitability of monitoring relations as a key point to provide scalability \[84, 40\]. The number of monitoring messages is reduced if each process is monitored by a small number of processes. Scalability can be evaluated by measuring the average number of monitoring messages versus the network size \[83\] or in terms of application requirements like quick failure detection by some non-faulty process and accuracy of failure detection \[37\].

1.2.3 Network Self-healing and Self-organisation

Recovering servers from failures is a critical task in clouds and data centres to offer availability and reliability \[43\]. In the literature review, it is shown that dealing with failures can produce isolated components and network disintegration. To tackle failures, several healing strategies have been developed. Some works in the literature revision \[30, 6, 64\]
provide a healing strategy based on the creation of new links between remaining nodes, to avoid network splitting.

In [30], distances are maintained as short as possible compared with the original topology. In [6], a multicast routing algorithm is proposed to provide connection to orphan nodes in case of failures that create partitions in a network. In [64], self-healing in Small-world networks is provided via activation of backup links. Complementary to the previous approaches, in this dissertation a node self-healing model based on self-organisation is studied instead of rewiring a network.

Nowadays, server technologies are passing from physical to virtual architectures because they offer portability and efficiency. Maintaining server functionality is crucial to provide and process information [102]. Failures can be produced by denial of service attacks (DoS) or computer viruses causing server hang. In [102], a self-repair model is presented. The model resets the failing node based on the detection of DoS attacks, or unusual activity. However, human mistakes can occur or physical hardware can fail [2] and this approach is not able to monitor this type of failures. [81] claims that not all implementations are affected by the same vulnerability and proposed an approach that provides different heterogeneous implementations based on biological diversity and using docker. Docker [1] is a platform to run applications isolated in a container including its dependencies and libraries as an alternative to the creation of virtual machines.

Self-organisation takes inspiration from nature (insect swarms for example) to provide systems that achieve certain global goals based on local interactions [17]. As [43] remarks, the main challenge to obtain self-organisation consists in identifying the rules or processes that produce the desired global behaviour. Here, multi-agent systems define agents capable of independent actions and with the ability of interacting with others. In order to achieve
their tasks, they are required to cooperate, coordinate and negotiate which each other, being a base to model distributed tasks and self-healing actions [8].

Emergence can be seen as formation of order from disorder [51]. Self-organisation is a prerequisite to call a system emergent because order is formed without external intervention [52]. [51] remarks that a large population of agents that interacts without a central control and displays properties not existing in the individual level presents emergence.
DESIGNING DISTRIBUTED ALGORITHMS FOR MOBILE AD-HOC SENSOR SYSTEMS IS DIFFICULT, BECAUSE OF THEIR ASYNCHRONOUS COMMUNICATION, MOBILITY, ABSENCE OF SHARED MEMORY AND HIGH RISK OF FAILURES. TO DEAL WITH THESE CHALLENGES, SOME TECHNIQUES LIKE REPLICATION AND CONSENSUS ARE PROPOSED IN THE LITERATURE. HOWEVER, TECHNIQUES LIKE CONSENSUS DEPEND ON A LEADER ELECTION, AND THIS LEADER CAN FAIL. THIS CHAPTER PRESENTS SOME ADVANCES INSPIRED FROM NATURE FOR THE DESIGN OF A DECENTRALIZED, SCALABLE WAY FOR GETTING AND SYNCHRONIZING INFORMATION AMONG COMPONENTS IN A POINT-TO-POINT WAY [71, 72].

To achieve this, the problem of getting and synchronizing information is addressed by defining a distributed system as a swarm of agents (termites) that look for information. Termites are designed with the task of exploring a simulated environment, sensing some desired data distributed throughout the space, and sharing their local knowledge about the environment with other nestmates only if they are neighbours. However, when failure rates increase, it is less probable that a termite completes the entire task by itself before all termites fail.

In order to allow at least one termite to gather the complete information from the environment, several solution approaches are proposed, e.g. sequential exploration with one agent as reference, random movements with local information exchanges, Lévy walks and a pheromone-based exploration algorithm inspired by Ant Colony System. This algorithm allows a termite to explore data in a world and enables it to search other nestmates with more information by using a trace and by defining a search status given the amount of local information that the termite has. Results show how swarms manage to collect and replicate information from the entire space even when failures occur. By local interactions, almost all the termites get complete information from a defined world before failing, without a central control and with simple local rules.
2.1 Towards Failure-Resistant Mobile Distributed Systems
Inspired by Swarm Intelligence and Trophallaxis

A distributed system consists in a collection of components (e.g. processes, agents, robots and nodes), connected via a network, that coordinate their activities and share system resources, offering different services to users and appearing to them as a single system [93]. Coordination and cooperation between processes are necessary. Thus, from the design point of view, communication is a key point of distributed systems because each process has partial knowledge of the environment, acts in a local way and can fail [66].

Distributed systems must be scalable, that is, being adaptable to changes in terms of size (to provide an easy way to add or delete resources), geographical localization of components and easy management independently of its size [93]. Scalability involves the design and implementation of decentralised algorithms dealing with components that can fail, have no complete information, take decisions in a local way and use point-to-point communication because broadcast is not possible. Lack of scalability usually implies the performance loss of a system while it grows up [93].

On the one hand, autonomic computing addresses complexity with the idea of a computer system that adapts to changes without human intervention [43]. Inspired by nature, the idea is that systems elements manage themselves, while also provide their services [41]. The internal behavior of an autonomic element and the set of relationships with other elements are based on goals that a designer has embedded in it and on goals incorporated by other systems through subcontracts with other elements, with its tacit or explicit consent [41]. A set of self-* properties is required to achieve the self-management goals: adapting to the addition or deletion of components (self-configuration), detecting and recovering from failures without disruption in the system operation (self-healing), finding improvements in the efficiency of a system (self-optimization), and anticipating and preventing threats (self-protection) [43, 41].

On the other hand, replication is proposed as an essential component of failure tolerance in distributed systems. A well known mechanism of replication in distributed systems is consensus. By consensus, an agreement is expected on the replicas value given the local information in each process [5]. Different areas of application include state and log machine replication in databases [5, 59], motion planning, alignment problems [55] and information processing in sensor networks (computing averages of local observations) [60].

Some existing consensus algorithms like Paxos [44] and Raph [59] are based on a leader election process. This leader receives information and replicates it to other processes. However, several natural systems do not require leader direction and self-organisation relies on collective behaviors that emerge from local interactions, as happens in insect colonies where collaboration emerges from stigmergy [25].

Mobile ad-hoc sensor systems are employed increasingly for distributed tasks in unreliable conditions, such as terrain exploration and measuring. Self-organising solutions
can help to ensure reliability, availability and scalability, while using unreliable components (or agents) with limited resources. It enables agents to act independently, and to exchange and combine their partial solutions into a (more) complete result, which can be transmitted to users before all agents fail.

An important challenge in mobile sensor systems is to develop algorithms that allow components (e.g. processes, agents, robots or nodes) to collect and aggregate information by using range-limited communication and focusing on mobility [103]. To tackle this challenge, some constraints must be considered: each component has partial knowledge of the environment, acts locally and may fail. Coordination and cooperation between components are essential for reaching global objectives while the way in which components communicate and move are key for the system success [66]. Scalability is also required as systems must support increasing component numbers, which must take local decisions and communicate in a peer-to-peer fashion as broadcast is not available [93]. Application areas of interest include: communication coverage by mobile robots; robot-based terrain exploration, such as altitude charting; decentralised computations in sensor networks, such as average functions for measurements; or, automatic record replication for data synchronisation and self-healing in distributed environments, such as computer clusters or clouds. Each of these application domains features different resource constraints, failure risks, and performance targets.

Communication coverage with mobile robots has been studied using a combination of sensing and locomotion to form a network over a targeted space [103]. Some approaches were proposed based on random motions and initial robot locations within a terrain [11 103]. Robot foraging and communication coverage share some of the same principles: make agents move towards unexplored areas and avoid other agents in order to maximize the explored space; distribute sensor nodes throughout the terrain; and define an exploration strategy from local perceptions [46]. Some exploration approaches are inspired from foraging in animals, like fruit flies or spider monkeys where patterns are super-diffusive and appear to obey a search for resources that matches a Lévy walk pattern [12 15 14]. Lévy walks were also applied to model opportunistic mobile networks assuming that humans move following this model [67 80]. Additionally, stigmergy was adopted for inter-robot communication through the environment, requiring only that a robot passes close enough to a location where “communication” was placed to be affected by it [3 9]. There are two main ways to define pheromones in robots. The first one is simulated, in which the environment is not modified, and it is mapped instead as a logical grid with different pheromone concentrations (robots share references of places, or use a map defined as a shared memory) [3 96 61]. The second one is situated, in which the environment is modified and robots use specialized sensors to detect it – e.g. chemical sensors for alcohol-based trails [87], infrared sensors [62], or light sensors for pheromones implemented as light projections onto the ground [32].

Exploration with unreliable agents was also proposed in [56], by using a trust measure between agents that sense the world in a defective way. The approach uses a motion
strategy that is based either on agent reliability, or on a random motion in case agents do not have sufficient information. Similar to this work, [56] also established a difference between direct information (collected by each agent) and indirect information (obtained from others) and aimed to achieve information coherence across agents as a result. However, rather than dealing with unreliable information, this chapter aims to evaluate common exploration techniques, identify the causes behind their performance differences, and explore solutions that combine their most favourable characteristics.

Agents are called “termites” because they follow social organization and their feeding is carried out via trophallaxis, meaning that food is stored in their stomach and it is transferred among nestmates through mouth-to-mouth feeding [69]. In this chapter, local information is also inside each agent and local exchange of information is performed between neighbors that look for more information present in other nestmates using stigmergy. Trophallaxis is important in the nutritional dynamics and communication of many social insects: individual foragers return from a food source and transfer a portion of their gut material to one or several nestmates. Subsequently, these recipients become donors to others, and the process continues [91].

In this chapter, a solution to the sharing information problem is proposed and deals with some of the challenges introduced before. A simulated world is defined with some data of interest distributed in the environment. To get information in this world, the agents called termites are defined with the task of exploring and obtaining this desired data. A termite can move, sense data in its current location and share its local information with other neighbours. However, termites are unreliable and can crash with a given probability \( p_f \). Given these conditions, the problem is how to get at least one termite to collect the whole information of a world before they all crash.

This section uses stigmergy for guiding termites in their search of new information enabling them to explore a terrain, to get data information from other nestmates and to synchronise local information with others. This approach inspired by trophallaxis and swarms allows termites to get the whole data before they crash in a decentralized way. A swarm features self-organisation, which allows it to continue working even if some termites die looking for new information. Some experiments are performed with this feature by adding failures to termites, aiming to determine if our proposal can achieve failure resistance and to establish limits in a simple way inspired by nature. The remaining of this section is organized as follows: the next subsection presents a detailed description of the problem and some approaches to solve it including this proposal of sharing information based on stigmergy and trophallaxis. Finally, a result analysis is performed.

2.1.1 The Problem: World exploration

A world is a bidimensional toroidal space defined as a matrix of properties \( Props^{width \times height} \). Props is a collection \( Props = \{ \tau_w, data \} \), defined with the following values: amount of pheromone in the world \( \tau_w : \tau_w \in \mathbb{R} \land \tau_w \in [0,1] \) and
data : \( \text{data} \in \mathbb{R} \land \text{data} \in [0, 1] \). data represent some information of interest in this world (e.g. temperatures, altitudes, distance to a determined objective). For now, data values do not change because the main objective is to find a way to get all data information and share it in a fast way. At the beginning, all the world positions have a pheromone value of 0.5 and data is generated for each location in a random fashion.

### 2.1.1.1 Termites Definition

Multi-agent systems define agents capable of independent actions with the ability of interacting with others. In order to achieve their tasks, they are required to cooperate, coordinate and negotiate which each other [8]. Each termite senses information from the environment and acts by using actuators [78]. For now, agents are reactive, i.e. each termite operates to respond to changes and to satisfy its design objectives [78]. Objectives are defined in a termite program that determines the action to be executed by an agent according to its local knowledge. Algorithm 1 presents a termite program. While an agent is alive (line 4), this agent senses its environment (line 10), then chooses an action based on its perceptions (line 11) and finally the action has an effect on the environment (line 12).

**Algorithm 1** Example of agent program

1: Percept p  
2: Action action  
3: round ← 0  
4: while Agent.status ≠ Fail do  
5: \( \lambda \leftarrow U[0, 1] \) \quad \triangleright \text{uniform random number}  
6: if \( \lambda < p_f \) then  
7: Agent.status ← Fail  
8: break  
9: end if  
10: p ← environment.sense()  
11: Action ← computeAction(p)  
12: environment.act(Agent, Action)  
13: round ← round+1  
14: end while

In this chapter, each termite is designed with the objective of getting data information in a simulated world of size width \( \times \) height. data values are represented as a matrix of continuous values \( \mathbb{R}^{\text{width} \times \text{height}} \). The idea is that termites cooperate and coordinate among them to get the global data information starting from local perceptions. Each agent looks for, senses new data locally, and shares its collected data\( ^{\text{width} \times \text{height}} \) at the same time.

Different perceptions have been defined for termites: \( \text{Percept} = \{ \text{pheromone}, \text{data}, \text{socialStatus}, \text{neighbor}, \text{msg}, \text{loc} \} \). pheromone is a vector \( \mathbb{R}^n \) with values in \([0, 1]\) representing the amount of pheromone that a termite has in its vicinity (Moore neighborhood \( r = 1 \) with center in the termite location [35]). data is the information in the current location of the termite; socialStatus = \{SEEKER, CARRIER\} indicates the status of a termite; neighbor returns the id of a nestmate randomly selected from its Moore neighbourhood,
if existing; \( msg \) stores new messages received from other nestmates; and \( loc \) returns the current termite location as a matrix position \((\text{row, column})\).

A termite has the following \( \text{Actions} = \{\text{none, down, left, right, up, upleft, upright, downright, downleft, Die, Collect, Send, Receive}\} \). The first nine actions refer to movements in the world and \( \text{Die} \) stops an agent’s thread to simulate failure. \( \text{Collect} \) is performed in each round and means to store the data collected in the local memory of a termite. A termite \( s \) sends its data collected \( I_s \) to a termite \( r \) in a \( msg \) encoded as a collection \( msg = [I_s] \) in a process defined as \( \text{Send}(r, msg) \). Sending this information is inspired by the traditional asynchronous distributed systems where there are FIFO channels (messages received first are processed first by each agent [66]), local communication (each agent receives and sends messages only to its neighbors), and there is no timing assumptions regarding message delay, clock drift or time taken to send a message. The control returns to the invoking process after the data is copied in the buffer of the process that receives \([20] [42]\). To implement this mechanism, there is a FIFO queue mailbox for each neighbour in the world and an internal queue in each agent that loads new data using the \( msg \) perception. In an analogous way, each time a termite \( r \) receives new information from a neighbour \( \text{Recv}(msg) \), it decodes the received message \( msg = [I_s] \) and takes decisions based on this information, depending on the solution approach.

\( \text{Trophallaxis} \) inspires the information interchange of this approach. Figure 2.1 depicts an example of stigmergy-based agent exploration in a simulated world. Agents start from random locations and communicate via \( \text{trophallaxis} \): when two agents occupy adjacent locations, they exchange their information (green circles in Fig. 2.1). In this way, and just like in nature, termites are donors to other neighbors in a cascade scheme called \( \text{trophallactic cascade} \). This transfer pattern may prove to be more efficient and result in more equitable distribution than the direct transfer in nature [91]. Communication is added so each time that a termite \( s \) senses a neighbour \( r \), it gets its \( \text{id} \), and exchanges its local data information with \( r \) by using \( \text{Send} \) and \( \text{Receive} \) in the following way:

- Termite \( s \) performs \( \text{Send}(r, msg) \), where \( msg = \{I_s\} \). In this way, a termite \( s \) sends its current information \( I_s \) to \( r \).
- Termite \( r \) receives \( \text{Recv}(msg) \) and completes its information \( I_r = I_r \cup I_s \).

### 2.1.1.2 Failure Definition

One known type of distributed system failure is a crash, that is, a process of a distributed system is working fine until it halts. When a process fails, nothing is heard from it. This failure can be identified because the node stops sending messages and does not report a failure [93] [83]. In the proposed model, a crash is equivalent to the death of a termite.

Computation is executed in rounds: in each round the agent senses its environment, computes an action and effects it onto its environment. This cycle is repeated until
the agent completes the exploration or fails \[8, 78\]. This kind of failure is implemented by defining a probability of failure (\(p_f\)). In Algorithm 1, a pseudo-random number is generated in each round (line 5). If this pseudo-random number is less than \(p_f\), then the agent fails (lines 6-7). For example, \(p_f = 0.1\) means that a process has a probability of failure in 1 of 10 rounds.

### 2.1.2 Solution Approaches

In this subsection, some approaches are proposed to solve the aforementioned problem. First, a sequential solution is defined with one termite. After that, more termites that communicate among them are added into the environment featuring a random strategy of exploration. Finally, a solution based on stigmergy is proposed. For experiments, a fixed size of the environment is defined (\(width = 50, height = 50\)) and several values of \(p_f\) are defined. The idea is to estimate the amount of information that agents can get from the environment.

#### 2.1.2.1 Sequential Exploration with One Termite

Sequential exploration with one termite is modeled as point of reference given the definition of the world as a matrix. Sequential exploration is a good strategy to explore the world because it implies exploring each location in the world only one time. The termite \(i\) knows its current location as \(loc_i = (x, y)\), where \(x\) corresponds to rows and \(y\) corresponds to columns. Given this location, a simple movement program of the termite is depicted in Algorithm 2 and Fig 2.2.

One termite is set to get data in the world of \(Props^{50 \times 50}\) (size of the world equal to 2500). The termite is located in a random location and the world is explored in a sequential way. Considering the size of the world, a maximum of 2500 movements is enough for completing information. Sequential exploration experiments were performed 30 times.
Algorithm 2 Sequential exploration for termite $i$

```
while (status != Action.DIE) {
    ...
    if (x != width -1) {
        action = right;
    } else {
        action = downright;
    }
    ...
}
```

Figure 2.2. Sequential Exploration Algorithm.

for several values of $p_f$ and results were averaged to determine how much information a
termite can collect in each case.

Table 2.1 presents the summary of sequential exploration experiments. Column Data
Collected displays the average and standard deviation of the data collected in the 30
experiments performed. As expected, a greater value in the $p_f$ parameter implies a lower
probability of exploring the entire world. For instance, for $p_f \geq 10^{-3}$, exploration of the
world was never achieved. For the experiments performed with $p_f = 4 \times 10^{-4}$, a full
exploration of the world was achieved only in 12 of the 30 executions. For $p_f = 10^{-5}$, it
was achieved in 28 of 30 experiments. The single termite could explore the entire world
in a reliable way only if $p_f \leq 10^{-6}$.

2.1.2.2 Random Exploration with Communication

In these experiments, more than one termite is added to explore the world in a random
fashion starting in random locations in the same environment ($Props^{50 \times 50}$). Termites
move randomly in the environment, but have the same $p_f$. For the same world, experiments
were performed with populations of 10, 30 and 50 termites. A maximum of 3000 iterations
were defined to get the complete information and each experiment was performed 30 times.
Termites can communicate and exchange information with neighbors as specified in the
subsection 2.1.1.1.
### Table 2.1. Summary of Experiments for Sequential Exploration \((Population = 1, width = 50, height = 50\). A \(p_f = 4 \times 10^{-4}\) is added because \(1/(width \times height) = 4 \times 10^{-4}\).)

<table>
<thead>
<tr>
<th>(p_f)</th>
<th>Data Collected</th>
<th>Successful Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-1})</td>
<td>6.8 ± 6.17</td>
<td>0/30</td>
</tr>
<tr>
<td>(10^{-2})</td>
<td>113.27 ± 90.36</td>
<td>0/30</td>
</tr>
<tr>
<td>(10^{-3})</td>
<td>792.2 ± 560.57</td>
<td>0/30</td>
</tr>
<tr>
<td>(4 \times 10^{-4})</td>
<td>1532.93 ± 906.9</td>
<td>12/30</td>
</tr>
<tr>
<td>(10^{-4})</td>
<td>2047 ± 825</td>
<td>22/30</td>
</tr>
<tr>
<td>(10^{-5})</td>
<td>2496.2 ± 14.85</td>
<td>28/30</td>
</tr>
<tr>
<td>(10^{-6})</td>
<td>2500 ± 0</td>
<td>30/30</td>
</tr>
<tr>
<td>0</td>
<td>2500 ± 0</td>
<td>30/30</td>
</tr>
</tbody>
</table>

#### 2.1.2.3 Lévy Walk Exploration With Communication

A Lévy walk is a movement process in which a particle makes a sequence of movements in the same random direction during a time length that depends on another uniform random variable. Foraging mechanisms present in some animals appear to obey Lévy walks [15]. Lévy Walks have been used for solving the networking coverage problem in robots by moving them until they find a location with an acceptable number of neighbors and make connections [12]. In this work, the motion mechanism of [12] is adapted in the following way (Alg. 3): \(\text{randomDir}()\) returns a random direction \(\text{dir} \in \{\text{down, left, right, up, upleft, upright, downright, downleft}\}\); \(\alpha\) is a uniform random number that represents the increment rate of \(\text{accumulator}\); \(T\) defines a threshold of accumulator for generate a new direction \(\text{dir}\); and \(\Delta t\) defines an increase of \(\alpha\) in terms of time. Several experiments using Lévy walks were performed using \(T = 1\) and \(\Delta t = 1\) and varying the other parameters as in random exploration.

```
Algorithm 3 Reactive Lévy walk for an agent [12]

\[
\text{while} \quad \text{Agent.status} \neq \text{Fail} \quad \text{do}
\quad \text{dir} \leftarrow \text{randomDir()}
\quad \text{accumulator} \leftarrow 0
\quad \alpha \leftarrow U[0,1] \quad \triangleright \text{uniform random number}
\quad \text{repeat} \quad \triangleright \text{repeat is executed at each round}
\quad \quad \text{move(dir)}
\quad \quad \text{accumulator} \leftarrow \text{accumulator} + \alpha
\quad \text{until} \quad \text{accumulator} \geq T \lor \text{neighbour_sensor()} \lor
\quad \quad \text{proximity_sensor()} \lor \text{Agent.status} = \text{Fail}
\text{end while}
```

#### 2.1.2.4 Pheromone-based Exploration

In this approach, each termite determines its movements by using stigmergy inspired by swarms and the Ant Colony System algorithm [23]. As main difference, in this proposal
termites are looking for new information (present in other nestmates) instead of looking for food. In this way, termites will have a status determined by the amount of local information that each one has: SEEKERS are termites that look for others to get new information and to explore locations with more pheromone, and CARRIERS are termites that believe they have more information than others and explore world locations with less pheromone. Pheromone value $\tau_w$ defined in an environment is used. At the beginning, all the world locations have a pheromone value of 0.5.

A termite $i$ chooses a direction $dir$ that corresponds to the application of a biased exploration or an exploitation rule depending on a random variable $q \in [0, 1]$ by applying Eq. 2.1:

$$
\text{dir} = \begin{cases} 
\text{exploitation rule} & \text{if } q \leq 0.9 \\
\text{biased exploration} & \text{otherwise}
\end{cases}
$$

(2.1)

Exploitation rule generates a direction depending on the termite status:

- **SEEKERS**: If a termite is SEEKER, this termite will choose the direction with the maximum amount of pheromone in its vicinity looking for CARRIERS. If all values in the vicinity are the same, a random direction is chosen.

- **CARRIERS**: If a termite is CARRIER, this termite will choose the direction with the minimum amount of pheromone in its vicinity. If all values in the vicinity are the same, a random direction is chosen.

Biased exploration is a random-proportional rule [23] that gives to a termite $i$ the probability of choosing a direction $p_d(x,y)$ depending on the amount of pheromone $\tau_w$ in its vicinity $\text{neighborhood}(i)$ (Eq. 2.2). $\text{neighborhood}(i)$ represents the locations in the Moore neighbourhood of $i$ with $r = 1$:

$$
p_d(x,y) = \frac{\tau_w(x,y)}{\sum_{(k,l) \in \text{neighborhood}(i)} \tau_w(k,l)}
$$

(2.2)

Each time a termite $i$ performs a movement, it updates its local amount of pheromone $\tau_l(i)$ (local update rule of Eq 2.3) and updates the pheromone in this world location $\tau_w(x,y)$ (global update rule of Eq 2.4):

$$
\tau_l(i) = (\tau_l(i) + 0.01 \cdot (0.5 - \tau_l(i)))
$$

(2.3)
\[ \tau_w(x, y) = \tau_w(x, y) + 0.01 \ast (\tau_t(i) - \tau_w(x, y)) \] (2.4)

If a termite \( i \) turns into a SEEKER or finds new information, its pheromone value is updated to 0 (\( \tau_t(i) = 0 \)). Equation 2.3 is based on the local update rule of ACS [23] and represents a local update rule which makes possible that the termite pheromone value increases with the time until a certain point. Simultaneously, it reduces the amount of pheromone of the world at the same time (global update rule of equation 2.4). This influence is reduced gradually and makes pheromone in the world converge to its default value of 0.5. A termite in SEEKER state can reach a maximum value of pheromone of 0.5.

If a termite \( i \) becomes a CARRIER or finds new information, the pheromone of the termite gets a value of 1 (\( \tau_t(i) = 1 \)). Local update rule of Equation 2.3 produces a decrease in the local amount of pheromone until reaching a minimum value of 0.5. At the same time, a CARRIER increases the amount of pheromone in the world (global update rule Eq. 2.4). This influence is reduced with the time making pheromone in the world converge to its default value (0.5). A termite in a CARRIER state can reach a minimum value of pheromone of 0.5.

If one termite has more information than another, then the former turns into a CARRIER; or a SEEKER in the other case (Fig 2.3). This is the nature-inspired trophallactic cascade, where termites are donors to other neighbours. Each time a termite \( s \) senses another neighbour \( r \), \( s \) sends its information to \( r \), \( r \) merge its information (\( I_r = I_r \cup I_s \)) and additionally \( r \) calculates the difference between information dif = \( I_s \setminus I_r \). As a result, two following scenarios are possible (Fig 2.3):

- dif = \( \emptyset \) means that \( r \) had at least the same information as \( s \), thus, termite \( r \) turns into a CARRIER and sets its pheromone value in one \( \tau_t(r) = 1 \).
- Otherwise (if dif \( \neq \emptyset \)), \( r \) turns into a SEEKER and sets its pheromone in zero \( \tau_t(r) = 0 \).

Experiments were performed with 10, 30 and 50 termites. All agents at the beginning are SEEKERS, thus, they start exploring the world and when they make contact with another nestmate information interchange starts. Figure 2.4 shows how termites start as SEEKERS (white points) explore the world and turn into CARRIERS (blue points) over time. Red locations in the world represent the gradual variation in the amount of pheromone and how agents explore the world. Squares 3, 4, 5 and 6 of Figure 2.4 depict cases where communication occurs (green circles).

Fig 2.5 shows how exploration influences the states of the termites between SEEKERS and CARRIERS. The Termites axis represents the individuals in the simulation and the Iteration axis represents the average round number. After some iterations, all the population becomes CARRIERS in all the experiments performed. Bigger populations make termites turn into CARRIERS in a fast way.
1. \( s \) senses \( r \) and sends \( \text{inf}_s \):

![Diagram of termite communication](image)

2. \( r \) completes information, calculates \( \text{dif} = l_s \setminus l_r \) and determines its status:

\[
\text{dif} = \emptyset \\
\text{dif} \neq \emptyset
\]

**Figure 2.3.** Communication and Status Determination of Termite \( r \).

![Pheromone exploration with 10 termites](image)

**Figure 2.4.** Pheromone exploration with 10 termites, white points are \text{SEEKERS} and blue points are \text{CARRIERS}. 
Table 2.2. Summary of average of information collected and number of agents with complete information (pop = 10, 30, 50, width = 50, height = 50, maxiter = 3000).

![Figure 2.5. SEEKERS (red line) vs. CARRIERS (blue line), $p_f = 0$ a) Pop = 10, b) Pop = 30, c) Pop = 50.](image-url)
CHAPTER 2. FAILURE RESISTANT MOBILE DISTRIBUTED SYSTEMS: SELF-ORGANISED TERRAIN EXPLORATION

<table>
<thead>
<tr>
<th>Pop</th>
<th>Average Best Round Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>-</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>-</td>
</tr>
<tr>
<td>$4 \times 10^{-4}$</td>
<td>-</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>2500.81 ± 282.07</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>2681.50 ± 219.28</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>-</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>-</td>
</tr>
<tr>
<td>$4 \times 10^{-4}$</td>
<td>2038.16 ± 460.05</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>1571.30 ± 269.89</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>1401.70 ± 233.84</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>-</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>-</td>
</tr>
<tr>
<td>$4 \times 10^{-4}$</td>
<td>1172.59 ± 570.89</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>1908.00 ± 181.80</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>874.00 ± 134.37</td>
</tr>
<tr>
<td>$3 \times 10^{-6}$</td>
<td>863.50 ± 144.88</td>
</tr>
<tr>
<td>0</td>
<td>823.57 ± 96.36</td>
</tr>
</tbody>
</table>

Table 2.3. Summary of averages of the number of rounds required for the best agents to collect all information ($pop = 10, 30, 50, width = 50, height = 50, maxiter = 3000$).

2.1.3 Results Analysis

Tables 2.2 and 2.3 present the results for the experiments of Random Exploration (column Random Expl.). The column Inf. Col. shows the average and standard deviation of data collected for agents in the 30 executions. Column Ag. Compl. is the average and standard deviation of agents with complete data for the 30 executions of each experiment, Table 2.3 presents the average round number of the agents that completed information at first place in each experiment.

In the experiments performed, a smaller value of $p_f$ means more data obtained and more termites with complete information. However, with 10 termites ($Pop = 10$) it was impossible to get the complete data for random exploration. For the given size, random movements and small populations do not warrant exploration of the whole data in the 3000 iterations specified because it is difficult for termites to meet and communicate and they tend to repeat paths. With 30 and 50 termites, for a $p_f \geq 4 \times 10^{-4}$, it is observed that more than one agent got all the data information (Agents Complete).

Columns Levy Walk Expl. and Ph. Expl. of Table 2.2 present the results for Lévy walks and pheromone exploration, respectively. Results show that both strategies work even with 10 termites exploring the world with $p_f \leq 10^{-4}$ in Lévy walks and $p_f \leq 4 \times 10^{-4}$ in some experiments of pheromone exploration. For 30 and 50 termites, the entire information is obtained even with a $p_f = 10^{-3}$. More data is collected as $p_f$ decreases. Bigger populations produce more exploration and a faster information dissemination.

Lévy walks are a good technique for exploring new information because the average of information collected is higher than in pheromone exploration (Table 2.2). However, the number of agents with complete information before 3000 iterations is bigger in pheromone exploration with 10 and 30 agents and a $p_f = 10^{-3}$. Table 2.3 shows that pheromone
exploration allows some individuals to collect information in a faster way than the other two methods, because the best agents require a smaller number of rounds for collecting all the information.

In the experiments performed, communication is important to reduce the time necessary for getting and disseminating information. Even in random exploration, communication tends to reduce the number of rounds necessary for a termite to get all the information: from 2500 of sequential exploration with one termite to 823.57 for 50 termites (Table 2.3). Results with pheromone exploration are even better: an average number of 451.13 rounds were needed for the best termite to collect the complete data without failures ($p_f = 0$).

As expected, bigger populations provide a better performance for exploration and for sharing information in a decentralised, scalable and simple way, even with unreliable termites. Additionally, Table 2.2 shows that Lévy walks and pheromone-based exploration imply more resistance to failures compared to random exploration and that a faster data synchronization implies more resistance to failures.

Pheromone exploration and Levy walks improve world exploration compared to random exploration. Figure 2.6 shows how a termite explores the world (a trace is added on visited locations). As expected, pheromone exploration tends to avoid path repetition during exploration versus random exploration where a termite can explore a determined location more than once. Additionally, Lévy walks cover an area in a better way than random by maintaining the same direction for some time.

![Figure 2.6. Random, Levy Walk and Pheromone Exploration.](image)

Changes in the status of a termite, between SEEKER and CARRIER, restart the amount of local pheromone that a termite has, reinforcing the trace and rewarding communication. Next section intends to test other methods to relate local information with the local update rule of pheromone (e.g. addition of passive evaporation) and study how a termite status influences data synchronisation.
2.2 Foraging-Inspired Self-Organisation for Terrain Exploration with Failure-Prone Agents

In the previous section, we studied how basic exploration techniques and foraging-inspired self-organisation help mobile agents to achieve such collaborative tasks [72]. The obtained results allowed us to identify two key aspects impacting agent performance and success rates. First, exploration of new paths which addresses the problem of covering unexplored territory. Second, finding and exchanging with agents that detain new information speeds-up global data collection by merging results of local explorations. Results also showed that increasing agent population densities produced better results, even when relying on random walks.

In this section, we explore some alternatives for the two aspects mentioned above, their most promising combinations and the addition of passive pheromone evaporation. We focus on decentralised techniques inspired from foraging behaviours in animals: Random motions and Lévy walks (for the exploration aspect), and stigmergy-based strategies (for the agent search and exchange aspect, and for avoiding explored territory). The main contributions of this section consist in proposing various exploration approaches and analysing them with respect to speed, failure resistance, message exchanges and scalability, which impact their applicability to targeted domains. Speed relates to the rapidity of data collection; failure resistance to the ability of covering the entire terrain in the presence of increasing agent failures; the number of messages relates to communication overheads that have to be minimized in sensor networks for improving battery life [39]; and scalability shows the impact of terrain sizes on the other performance results (at the same agent density – density = number of agents/world size).

As in the previous section, each agent is designed to explore a simulated world, to sense some desired data throughout the environment, and to share its collected data with agents that it encounters. Agents fail with a certain probability and the objective is to allow at least one agent to collect all the information before failing. The remaining of this section is organized as follows. The subsection 2.2.1 provides some basic exploration solutions and the subsection 2.2.2 discusses experiments and results for these solutions. This allows to identify the best candidates to consider in more advanced hybrid solutions. Section 2.2.3 explores such hybrid alternatives and also considers the impact of pheromone evaporation. Results are analysed in Section 2.2.4 discussed in Section 2.2.5 and conclusions are drawn in Section 2.3.

2.2.1 Basic Approaches

The agents success in completing their collective task relies critically on the motion process of each agent. The motion strategy impacts both the exploration of uncharted terrain and the encounter of other agents – both essential for data collection. In this subsection, four motion processes are defined based on two main criteria (Fig 2.7): two purely ex-
ploratory processes – random and Lévy walks – and two stigmergy-based processes that use pheromones to improve collective exploration and agent encounters. Random Exploration, Lévy walks and Seekers and Carriers are introduced in sections 2.1.2.2, 2.1.2.3 and 2.1.2.4, respectively.

The input of all algorithms is the agent perception $Percept$ and the output is the agent direction of movement $dir$ (Cf. Section 2.1.1).

![Figure 2.7. Overview of Basic Solution Approaches](image)

**Seekers and Carriers with Lévy Walks**

In the previous experiments presented in section 2.1, the average amount of information collected by agents was higher for Lévy walks compared to random walks and to seekers and carriers exploration [72]. Lévy walks represent a super-diffusive pattern that appears advantageous to exploration. Therefore, it is assumed that the initial seekers and carriers approach could be improved by replacing its random exploration parts with Lévy walks. The new approach is the same for selecting between an exploitation rule and a biased exploration rule (Eq. 2.1). Yet, biased exploration is replaced from the random-proportional rule of Eq. 2.2 to Lévy walks (Alg. 3), as shown in Eq 2.5. The exploitation rule is updated, so that, if more than one direction contains the same maximum (for seekers) or minimum (for carriers) pheromone amount, and if the last direction given by the Lévy walk algorithm is within these options, then this direction is chosen (otherwise a random one is used as before).

$$
dir = \begin{cases} 
\text{exploitation rule with Lévy walk} & \text{if } q \leq 0.9 \\
\text{Lévy walk} & \text{otherwise}
\end{cases}
\quad (2.5)
$$
CHAPTER 2. FAILURE RESISTANT MOBILE DISTRIBUTED SYSTEMS: SELF-ORGANISED TERRAIN EXPLORATION

Figure 2.8. Box-plot of average information collected by agents for random motions (random), Lévy walks (levywalk), Seekers and Carriers (sandc) and Seekers and Carriers with Lévy walks (sandclw)

2.2.2 Experiments and Results

Experiments for each basic approach were performed considering different failure probabilities $p_f = 0, 1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 7 \times 10^{-4}$ and $9 \times 10^{-4}$. These $p_f$ values were selected to explore the range in-between no failures and failure rates where all algorithms fail. Additionally, different world sizes were considered, while maintaining the same agent density – i.e. agent populations of 10, 20, 30, 40 and 50 for world sizes of $50 \times 50$, $71 \times 71$, $87 \times 87$, $100 \times 100$, and $112 \times 112$, respectively. An agent size is $1 \times 1$. Each experiment stops when an agent collects all the information or if all the agents fail. The results presented are averages from experiments performed 30 times.

Table 2.4 presents the success rates of each algorithm (averaged over the 30 executions). An experiment is successful if an agent manages to collect all the information. Random exploration performs well with low failure rates (until $p_f = 10^{-4}$), yet declines quickly as failure rates increase. Better success rates are observed for Lévy walks and for both seekers and carriers approaches. Successful experiments occurred for 10 and 20 agents up to $p_f = 5 \times 10^{-4}$ and with higher success rates for $p_f = 10^{-4}$ compared to random walks. It can also be noted that among these three algorithms, for $p_f = 3 \times 10^{-4}$ and $5 \times 10^{-4}$, the best success rate varies with the world scale, and the overall success of all algorithms declines as the scale increases. This may occur because the initial agent
locations within the world set randomly; however, further experiments are required in future work to establish this. Figure 2.8 presents a box-plot with the collected information percentages averaged across all agents, at the end of each experiment. It represents the data distribution degree among agents and can provide an indicator of behaviour when failures increase (as the most successful agents may fail before completion). When introducing failures, performance degrades, unsurprisingly, as $p_f$ increases. Different agent populations for each algorithm are shown to assess scalability. In the absence of failures, all agents in the four techniques manage to collect almost 100% of the data by the end of the experiments. Lévy walks perform particularly well, since they promote exploration equally for all agents. When $p_f$ is increased, data collected by agents with random motions decreases faster than in the other methods. For the three other techniques (also with the best success rates in Table 2.4), data collected by seekers and carriers with Lévy walks is slightly higher compared to seekers and carriers with random walks.

Table 2.4. Information Collected and Success Rates

<table>
<thead>
<tr>
<th>Population (World size)</th>
<th>$p_f$</th>
<th>Random</th>
<th>Lévy Walks</th>
<th>Seekers and Carriers</th>
<th>SandC With Lw</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (50 × 50)</td>
<td>9.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>3.33%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>3.33%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>5.0E-4</td>
<td>0%</td>
<td>16.67%</td>
<td>13.33%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>6.67%</td>
<td>83.33%</td>
<td>53.33%</td>
<td>58.97%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>86.67%</td>
<td>100%</td>
<td>96.67%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>20 (71 × 71)</td>
<td>9.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>5.0E-4</td>
<td>0%</td>
<td>6.67%</td>
<td>3.33%</td>
<td>6.67%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>0%</td>
<td>50%</td>
<td>50%</td>
<td>57.89%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>93.33%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>30 (87 × 87)</td>
<td>9.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>5.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>0%</td>
<td>36.67%</td>
<td>43.33%</td>
<td>39.47%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>83.33%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>40 (100 × 100)</td>
<td>9.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>5.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>0%</td>
<td>23.33%</td>
<td>43.33%</td>
<td>31.58%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>73.33%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>50 (112 × 112)</td>
<td>9.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>5.0E-4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>0%</td>
<td>36.67%</td>
<td>26.67%</td>
<td>36.84%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>80%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 2.9 presents the average round number of the agent(s) that complete data collection in the successful experiments for more than one technique (with $p_f = 0, 10^{-4}, 3 \times 10^{-4}$ and $5 \times 10^{-4}$). It is observed that higher $p_f$ values cause an increase in the number of rounds necessary to gather the complete information. When an algorithm failed to complete with a $p_f$ value across all experiments, its number of rounds was not represented in the graph. In all cases, there is little difference between the round numbers.
achieved by the two seekers and carriers approaches. Yet, both approaches finish the task quite faster than Lévy walks (e.g. 1000 rounds faster for \( pf = 0 \) and \( 10^{-4} \)). This may be due to the fact that seekers and carries approaches promote agent encounters and exchanges. When \( p_f \) increases, this exchange aspect seems to outrun the average agent information collection, as this latter aspect was slightly better for Lévy walks (Cf. Fig. 2.8). Round numbers also increase with the world size; however, this should be explored in future work.

Figure 2.10 presents a box-plot of messages sent with the different techniques. It can be observed that a higher amount of messages is sent for random motions without failures (\( pf = 0 \)). This may be explained by the fact that random motion only promotes exploration around the initial location [12], thus, some agents keep exploring the same locations and communicate among themselves, exchanging the same information repeatedly. The Lévy walks technique exchanged the smallest number of messages, as it focused on exploration, but took a higher number of rounds for gathering the information (than seekers and carriers). In each technique, the number of messages decreased with the \( p_f \) increase, as there were less agents to exchange with. The number of messages also increased with the world scales.

In terms of speed and success rates, seekers and carriers with Lévy walks appear to be better than the other techniques. However, Lévy walks feature higher amounts of collected information across agents and smaller message numbers compared to the two
2.2.3 Hybrid Approaches and Evaporation

2.2.3.1 Lévy Walkers and Carriers

This approach redefines the seekers and carriers technique by replacing seekers with Lévy walks, as follows: if an agent $i$ is a carrier, its behaviour is the same defined for seekers and carriers (subsection 2.1.2.4). Hence, carriers will increase pheromone amounts in visited locations and choose paths with minimum pheromone amounts to avoid visited places. Motion is determined by the rules in Eq. 2.1 and Eq. 2.2. Pheromone is deposited as defined in Eq. 2.3 and Eq. 2.4. Otherwise, if an agent $i$ is a seeker, its motion process is a Lévy walk (Alg. 3) and it does not deposit pheromone.

Since only one agent type (carriers) impacts pheromone amounts, the initial amounts in all the world locations is set to $\tau_w = 0$. The aim of this hybrid approach is to combine the best information collection techniques: Lévy walks by enhanced exploration, and carriers by avoiding already explored territory. Initially, all agents are Lévy walkers. As they encounter other agents, the ones that collected more information become carriers. This
should allow them to explore more uncharted territory by avoiding pheromone locations. Yet, if pheromone-based exploration does not produce more information, then carriers swap back to Lévy walkers.

2.2.3.2 Lévy Walkers and Carriers with Lévy Walks

This approach is similar to the previous one (Lévy walkers and carriers) except for the carriers rules in Eq. 2.1 that are replaced by Eq. 2.5. This means that a carrier will choose the direction with the minimum amount of pheromone, but if several directions exist, and one of them is the same as the current output of the Lévy walk algorithm, then this direction will be selected (if it does not match the direction of the Lévy walk, a random direction is chosen among the minimum ones). Their biased exploration is also a Lévy walk, rather than random. Carriers deposit pheromone as defined by Eq. 2.3 and Eq. 2.4 and the initial pheromone amounts for all the world locations is set to \( \tau_w = 0 \). Seekers are replaced by Lévy walkers (Alg. 3) and they do not deposit pheromone.

2.2.3.3 Carriers and Pheromone Evaporation

This approach only considers carriers exploring the world. It was observed in previous experiments that exploration with pheromone is fast, but some agents collect more information than others (Cf. Fig 2.8). Moreover, it is possible that the agents that collected the most information fail before completing the task, and their pheromone traces are left behind preventing other agents from recollecting the information. A solution to this issue is to also enable passive pheromone evaporation (by the environment, rather than by agents). In this approach, each agent determines its movements by using stigmergy inspired by the Ant Colony Optimization algorithm (ACO) \[24\] and the exploration and exploitation rules of ACS. Each agent explores and updates local pheromone just like a carrier (Eq 2.1). There are no social status changes in this approach. If an agent finds new information from neighbours or from new locations, the pheromone of the agent is set to \( \tau_t(i) = 1 \) (as for carriers in the aforementioned approaches). In the same way, all the world locations are initialised to \( \tau_w = 0 \).

Evaporation is applied to all the world locations, independently of agent movements, by using the definition in \[24\]. Namely, it is defined via (Eq. 2.6), with evaporation rate \( \rho = 0.01 \):

\[
\tau_w(x, y) = (1 - \rho)\tau_{w_{t-1}}(x, y), \text{ for } \forall (x, y) \in \{\text{world locations}\}
\]

As \[13\] indicates, natural pheromone evaporation makes paths less dominant, rewarding exploration while also addressing cases where path-producing agents fail.
2.2.3.4 Carriers with Lévy Walks and Pheromone Evaporation

This model is similar to the previous one (carriers and pheromone evaporation), except for the rules in Eq. 2.1 that are replaced by Eq. 2.5.

2.2.4 Experiments and Results with Hybrid Approaches and Evaporation

Several experiments were performed on these hybrid models, in the same way as for the basic models (Cf. Section 2.2.1): $p_f = 0, 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}, 7 \times 10^{-4}$ and $9 \times 10^{-4}$; agent populations of 10, 20, 30, 40 and 50; and world sizes of $50 \times 50$, $71 \times 71$, $87 \times 87$, $100 \times 100$, and $112 \times 112$, respectively. Each experiment was performed 30 times and stopped when one agent collected all the information or if all the agents failed. The values of $p_f$ were selected because most of our experiments were successful for $p_f = 10^{-4}$ and failed for $10^{-3}$; thus, $p_f$ values between these two extremes were the most relevant to explore.

As can be observed in Table 2.5, success rates are even better for the hybrid algorithms than for seekers and carriers with Lévy walks, which was the best option from the basic approaches (Cf. column “SandC with LW” in both Tables 2.4 and 2.5). This indicates that using seekers to increase agent encounters did not improve success rates. It also indicates that passive pheromone evaporation plays an important role in failure resistance, since both approaches using it were better than the others. Namely, Carriers and Pheromone Evaporation and Carriers with Lévy walks and Pheromone Evaporation achieve some successes for all $p_f$ values, even for $p_f = 9 \times 10^{-4}$, with 10, 20 and 30 agents. Improved success rates were also observed with the other population sizes compared to the other approaches.

Figure 2.11 depicts the average data collected by agents at the end of each experiment. When agents do not fail, they seem to collect more data when using carriers and Lévy walks (the two approaches without evaporation) than when evaporation is used. However, this advantage diminishes as the failure rate $p_f$ increases, with all hybrid approaches performing similarly by the time $p_f$ reaches $9 \times 10^{-4}$. Moreover, Figure 2.11 shows that it could be beneficial to replace the random-probabilistic rule by Lévy walks, since Lévy Walkers and Carriers with Lévy Walks collect slightly more information than Lévy Walkers and Carriers and has somewhat higher success rates (except for 10 and 20 agents with $p_f = 5 \times 10^{-4}$; more study will be performed in future work).

Figure 2.12 presents the round number for the best agent(s) and it is observed that hybrid methods are faster than the basic methods in Section III. It can also be noted that fast information gathering can be linked to higher success rates. Here again, the two approaches using pheromone evaporation are faster than the others (with Carriers with Lévy walks and Pheromone Evaporation, appearing the fastest in most cases). It can also
Table 2.5. Success rates for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C-Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-Lw and Evap)

<table>
<thead>
<tr>
<th>Pop</th>
<th>$p_f$</th>
<th>Success Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SandC with Lw</td>
</tr>
<tr>
<td>10</td>
<td>9.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>58.97%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>20</td>
<td>9.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>6.67%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>57.89%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>30</td>
<td>9.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>3.33%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>39.47%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>40</td>
<td>9.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>6.25%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>31.58%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>50</td>
<td>9.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>7.0E-4</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3.0E-4</td>
<td>36.84%</td>
</tr>
<tr>
<td></td>
<td>1.0E-4</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 2.11. Box-plot of information collected for Seekers and Carriers with Lévy Walks (SandC with Lw), Lévy Walkers and Carriers (Lw and C), Lévy Walkers and Carriers with Lévy Walks (Lw and C-Lw), Carriers and Pheromone Evaporation (C and Evap) and, Carriers with Lévy Walks and Pheromone Evaporation (C-Lw and Evap)
be noted that some techniques no longer occur in the box-plot for higher $p_f$ because they do not manage to get any success before all agents fail.

Figure 2.13 presents the average number of messages sent. When agents do not fail, the hybrid approaches exchange less messages than the basic ones. As $p_f$ increases, messages sent by all approaches decrease (since agents fail), and the difference between approaches also diminishes. For $p_f = 7 \times 10^{-4}$, the two approaches using evaporation are the ones sending most messages. This can also account for their lower round numbers and higher success rates.
2.2.5 Discussion

The above experiments aimed to determine the impact of two main factors on the performance metrics (speed, failure resistance, messages sent and scalability), compared to the basic approaches. These changes were compared to the best of the basic approaches (*seekers and carriers with Lévy walks*).

The first factor consisted in replacing seekers with Lévy walkers (2.2.3.1 and 2.2.3.2). The aim was to favour exploration rather than agent encounters, considering that agents will still meet (as they aim to explore the same uncharted areas). On the contrary, seekers may be inefficient in data collection since they follow already explored paths in search of carrier agents. Hence, agent encounters are more productive between exploring agents, since the data exchange is more complementary. Indeed, this technique managed to improve success rates at increasing failure rates (by lowering the round number), while maintaining the average data collected about the same (compared to *seekers and carriers with Lévy walks*).

The second factor added passive pheromone evaporation (2.2.3.3 and 2.2.3.4). The aim was to address the problem of traces left by failed agents, which prevented other agents from re-exploring them. This technique succeeded in further improving success rates (via even lower round numbers), while average data collection was not significantly impacted. Nonetheless, the applicability of this approach depends on the possibility and cost of implementing this feature in the targeted environment. Besides, the approaches using evaporation appear to exchange more messages than when no evaporation is used, which should be taken into account in terms of energy consumption.

The proposed techniques are applicable in different contexts depending on the domain-specific constrains. For instance, if a targeted domain allows pheromone modelling, either situated or simulated, then hybrid approaches that optimise exploration can be considered – using Lévy walks to improve exploration; pheromone trails to enhance agent information sharing and to avoid already charted territory; and pheromone evaporation to “forget” such trails in case the agents that posed them had failed in the meantime. Otherwise, when pheromone marking and/or evaporation are unavailable, a basic Lévy walk may be the best option. In cases where the agent density is much higher and/or the agent failure rate is much lower, and where the time to completion is not a critical parameter, a basic Lévy walk or even a Random walk may do. Finally, in cases with no pheromone availability and low failure rates, but that require to have many agents collecting the data (rather than a single one), the Lévy walks are the best.

2.3 Summary

In this chapter, we proposed different solutions for the problem of terrain exploration and data gathering via a set of unreliable agents. Since none of the agents can complete the
task on its own, because of limited action time before failure, agents must self-organise and share the partial data they collect to obtain the global result faster. Here, completion speed was an essential characteristic for the success rates. This, in turn, depended on the agents ability to explore uncharted territory, and to meet agents with complementary information.

The best exploration characteristics were achieved in solutions following Lévy walks (better than random walks) and using pheromones to mark (and avoid) already charted terrain. Passive pheromone evaporation was highly beneficial for removing traces left by failed agents and allowing paths to be re-explored by the remaining agents. Attempting to introduce agents with seeker behaviours, which purposefully searched for exploratory agents by following their pheromone paths, did not produce any improvements. On the contrary, seeker techniques seemed to increase only the encounters between agents with similar information. Instead, allowing agents to meet passively during their explorations produced much better results overall. Hence, hybrid approaches that improve exploration while still managing to encounter agents that held complementary information provided the best results. More information on the experiments, including complete measurement tables and source code are available at: \url{http://alife.unal.edu.co/~aerodriguezp/termites/}

Next chapter extends the data-collection problem to complex networks with failure prone-agents. Some motion algorithms are selected and its robustness is studied in different configurations of complex networks.
CHAPTER 3

Self-organised Data Collection in Complex Networks

In previous chapters, the problem of terrain exploration via failure-prone components, defined as agents, is addressed. As main conclusion, a fast data collection implies higher success rates and higher robustness. Distributed data-collection and synchronization is essential in sensor networks and for the Internet of Things (IoT), as well as for data-replication in server farms, clusters and clouds. Generally, such systems consist of a set of interconnected components, which cooperate and coordinate to achieve a collective task, while acting locally and being failure-prone. Hence, an important challenge is to define efficient and robust algorithms for data collection and synchronisation in large-scale, distributed and failure-prone platforms.

This chapter studies the performance and robustness of different multi-agent algorithms in complex networks with different topologies (Lattice, Small-world, Community and Scale-free) and different agent failure rates. Agents proceed from random locations and explore the network to collect local data hosted in each node. Their exploration algorithm determines how fast they cover unexplored nodes to collect new data, and how often they meet other agents to exchange complementary data and speed-up the process. Two exploration algorithms are studied: one random and one using an adaptation of the stigmergy model proposed in previous chapters. Experimental results show that network topologies and agent failure-rates impact data-collection and synchronization, and that a stigmergy-based approach can improve performance and success rates across most scenarios. These results may offer key insights into the suitability of various decentralised algorithms in different networked environments, which are increasingly at the core of modern information and communication technology (ICT) systems.
3.1 Introduction

Sensor networks, server farms and clouds consist of numerous components (e.g. servers, processes, robots) interconnected via complex networks. They cooperate and coordinate their actions towards an overall objective, may share common resources, and appear to the end-user as a single system [93]. An important field of study relates to how fast distributed processes, interconnected via such complex networks, can achieve collective objectives (e.g. data collection, synchronisation or processing); and how the particular topological properties of complex networks impact such performance [18, 98, 34, 58]. These aspects are key in secure communications [34], logging and machine replication in databases [59]; and in information-processing and consensus-making in sensor networks [55]. A particular challenge in this field is data collection from network components, both as a stand-alone objective, e.g. in sensor networks, or as an underlying task for data synchronisation. Important issues must be addressed, as networked components can only act locally and may fail unexpectedly.

Previous works [72, 56] have studied data-collection techniques based on failure-prone agents. Analysed approaches included random walks, Lévy walks and stigmergy. Agents explored a targeted space based on selected algorithms, in order to collect local information and share it with other agents (that they could meet) [72]. [56] presents agents that collect information from distributed sources and can fail and/or provide unreliable information defining collective information as aggregation of information that agents collect individually. [71] showed that data-collection can be speeded-up by algorithms that favour exploration of new paths and the exchange of new information with other agents. It also demonstrated that mechanisms that favour exploration and achieve faster data-collection are more resistant to failure than those that focus on increasing inter-agent communication. Finally, it showed that stigmergy and pheromone evaporation can help to explore new paths, while allowing to re-explore previous paths in order to recover from failure-related data losses.

In this chapter, a data-collection problem within complex networks is studied, rather than within uniform spaces studied previously. It is an important difference, since the topology of the network explored has a significant impact on the agents’ performance, as they explore, collect and exchange information [94]. As before, agents may fail at different rates; yet accurate data-collection is assumed – i.e. when agents are available, their information is reliable (as opposed and complementary to [56]). Two motion algorithms, random and stigmergy, are selected for experiments. These are similar to the ones defined in [72]; as Lévy walks do not apply to non-directional spaces, like networks. The objective is to analyse how agent performance (i.e. how fast all network data is collected) and robustness (i.e. how task completion is achieved in the face of agent failures) depend on the adopted exploration technique, on the network topology and on failure rates.
The remaining of this chapter is organized as follows. Section 3.2 presents the data collection problem in complex networks; the studied network topologies; and the agents’ design. Section 3.3 details the analysed motion algorithms, while section 3.4 presents the experimental settings and discusses obtained results.

3.2 The Problem of Data-Collection in Complex Networks

The problem studied can be summarised as follows. Agents must explore a complex network (simulated), in order to collect desired data present in the network vertices. Agents move among interconnected vertices based on a predefined algorithm (section 3.3), collect data from each visited vertex and exchange their data with any agent that they meet at the same vertex. Additionally, agents can fail over time with probability $p_f$. The aim is to have at least one agent that collects all data from the entire network. The parameters of interest are the speed of task-completion and the success rates in the presence of agent failures, depending on network topologies, agent motion algorithms and failure probabilities.

The agents implementation is based on [71]. Each agent is equipped with a set of perceptions $p = \{\text{pheromone, data, current\_node, neighbors, msg}\}$. *Pheromone* is a vector in $\mathbb{R}^n$ with values in $[0, 1]$ that represents the amount of pheromone in the agent vicinity (i.e. vertices adjacent to the current location) [35], *data* is the information to collect in the agent current vertex. *neighbour* returns the ids of agents in the same vertex. *msg* stores messages from other agents and *loc* returns the agent location (vertex name). Each agent can also perform a set of actions $\text{Actions} = \{\text{Move(vertex), Collect, Send(msg), Recv}\}$. $\text{Move(vertex)}$ moves the agent to the vertex location. $\text{Collect}$ senses data from the agent current location and stores it in its local memory, and $\text{Send}$ and $\text{Recv}$ enable information exchanges with other agents.

Simulation time is defined via discrete rounds. In each round, each agent senses its local environment (e.g. local data, co-located agents and adjacent vertices); decides an action (e.g. collect and exchange data, select a neighbouring vertex to move to); and performs the selected action [8] [78]. The simulation ends when at least one agent completes the exploration (i.e. collects all the data) or if all the agents fail. The environment is the complex network to be explored.

In short, a complex network consists of a large number of interconnected nodes characterized by non-trivial topological properties – i.e. neither purely regular nor completely random, unlike lattices and random networks. Typical features include relative small distances between nodes, high clustering, or power-law degree distributions (i.e. heavy-tailed) [18]. A more formal definition of complex networks is quite difficult to provide as researchers have focused instead on specific topological metrics and on the kinds of node interconnection rules that produce topologies with distinctive properties [94].
In this work, a complex network is defined as a graph $G$ with a set of vertices $V$ and a set of edges $E$: $G = (V, E)$. A probabilistic rule defines the way in which vertices are interconnected when constructing the graph. Hence, complex networks with different topologies can be generated by using different rules of inter-connection. This chapter evaluates the main types of network topologies identified in the literature namely Small-world, Scale-free and Community networks. Additionally, more regular topologies are used for comparison, such as Forest Hub & Spoke, Lattice, Line and Circle.

### 3.2.1 Small-world Networks

A small-world network is generated by starting from a regular network (in terms of node interconnections) by rewiring some of these connections in a random way. This type of network features relatively short paths between any network nodes, even in very large networks. In this section, a Watts-Strogatz model is used, with different parameters, to generate small-world networks. We start with a regular ring lattice network with $n$ vertices and $k$ edges per vertex, then each edge is rewired with a probability $\beta$. The $\beta$ parameter determines the regularity of the final network: $\beta = 0$ generates a regular network, $\beta = 1$ a random network, and in-between values a small-world network (Fig. 3.1).

![Small-world Networks](image)

**Figure 3.1.** Small-world Networks: $n = 100$, $k = 4$, a) $\beta = 0.3$, b) $\beta = 0.5$, c) $\beta = 0.9$

### 3.2.2 Scale-free Networks

Scale-free networks are characterised by degree distributions that follow a mathematical function known as power-law. The degree distribution is the probability distribution of the node degrees over the entire network, where a node degree is its number of links. A power-law distribution implies that node degrees may differ by magnitudes of scale, and hence that a few nodes (called hubs) have a disproportionate number of links compared to the average degrees. Notable examples of real scale-free networks include the WWW, email or protein interaction networks. They are highly resistant to accidental failures, but rather vulnerable to targeted node attacks.
Scale-free networks can be obtained with \( sn \) starting nodes and \( \eta \) connections. At each step a new node is added and connected via \( \eta \) links to the existing nodes, based on preferential attachment (i.e. nodes with higher degrees are more likely to connect) \[45\]. Namely, the probability to connect to an existing node is defined by \[ p_i = \frac{k_i}{\sum_j(k_j)}, \] where \( k_i \) is the degree of node \( i \) \[101, 89, 92\]. The process is repeated for \( steps \) times \[94\]. Figure 3.2 depicts different configurations, showing how the number of connections increases with \( \eta \).

3.2.3 Community Networks

Community networks feature structures where nodes can be assigned to different groups, or clusters, that are highly interconnected internally, and have relatively few connections among nodes belonging to different groups \[33\]. In this chapter, community networks were generated using a \( n_{\text{clusters}} \) parameter to define the number of groups in the network and adding a single connection between nodes of different groups. Each group was generated as a small world network (with its own \( k, \beta, \) and \( n = m/n_{\text{clusters}} \), where \( m \) is the number of nodes in the network). Figure 3.3 shows a community network with four groups connected either via a central node (selected at random), or via a circle formed by pairs of nodes selected from different groups (also random).

3.2.4 Forest Hub & Spoke

The Forest hub & spoke network is based on the hub & spoke (or Star) configuration, where all nodes are connected (spokes) around a central node (hub). The forest is then formed by connecting pairs of such star structures. This type of network ensures high availability and reliable computing services because it allows the expansion of individual cloud instances \[47\]. In this chapter, 4 Hub & Spoke clusters of 25 nodes each were generated, as shown in Figure 3.4.
CHAPTER 3. SELF-ORGANISED DATA COLLECTION IN COMPLEX NETWORKS

3.2.5 Line, Circle and Lattice

Experimental design includes lattice, line and circle topologies. The purpose of performing experiments with these topologies is to test exploitation and exploration properties of the selected algorithms with a higher diameter (line), a longer path length (line and circle), and regular connections (lattice). Figure 3.5 shows the configurations applied to the experiments, each one with 100 nodes.

3.3 Agent Motion Algorithms and Failures

After selecting different types of complex networks that represent various topologies, the next step is to establish and profile how a determined motion strategy influences important tasks for data collection such as the exploration of unvisited vertices, the encounter of other...
agents and robustness given in terms of completing the task even when agents prone to failure.

Each agent implements an algorithm that determines how to sense data from the environment and select actions such as motions and communication with other agents. The agent program pseudocode is listed in Algorithm 4. Agent failures are also defined in this program, and produced with a failure probability $p_f$ – e.g. $p_f = 0.1$ means the agent fails on average every 1 out of 10 rounds.

**Algorithm 4** Agent program

1: Percept $p$
2: Action action
3: round $\leftarrow 0$
4: while Agent.status $\neq$ Fail do
5: \hspace{1em} $\lambda \leftarrow U[0, 1]$ \hspace{1em} $\triangleright$ uniform random number
6: \hspace{1em} if $\lambda < p_f$ then
7: \hspace{2em} Agent.status $\leftarrow$ Fail
8: \hspace{2em} break
9: \hspace{1em} end if
10: \hspace{1em} $p \leftarrow$ environment.sense()
11: \hspace{1em} Agent.move(motionAlgorithm($p$))
12: \hspace{1em} if Agent.hasNeighbors($p$’s location) then Agent.exchange($p$’s neighbors)
13: \hspace{1em} end if
14: \hspace{1em} round $\leftarrow$ round+1
15: end while

Based on perceptions, agents choose the next vertex to move from two possible motion processes: random, which is exploratory; and pheromone-based, which improves exploitation of new paths and agent encounters (and, thus, enhances collective exploration).

When following a random walk, an agent selects its moving direction randomly from the set of vertices adjacent to the current vertex, at each round. A uniformly distributed pseudorandom generator is used for generating the random sequence.

When following a pheromone-based movement, an agent chooses vertices based on their pheromone load in order to find unexplored vertices, or other agents. This algorithm is
based on the Ant Colony System algorithm (ACS) [23], using stigmergy, and an adaptation of the Carriers algorithm in [71]. Initially, all vertices have a pheromone value $\tau_v = 0.5$. As in ACS [23], a random variable $q \in [0, 1]$ dictates when to apply an exploitation rule or biased exploration (Eq. 3.1):

$$
\text{dir} = \begin{cases} 
\text{exploitation rule} & \text{if } q \leq 0.9 \\
\text{biased exploration} & \text{otherwise}
\end{cases} \quad (3.1)
$$

A carrier agent chooses the direction with the minimum pheromone amount in its vicinity, looking for uncharted vertices. If more than one vertex has the same minimum value, a random direction is picked among these.

**Biased exploration** is a random-proportional rule [23] which gives an agent $i$ the probability of choosing a vertex $p_d(v)$ depending on the amount of pheromone $\tau_v$ in its vicinity $\text{neighbourhood}(i)$ (Eq. 3.2). $\text{neighbourhood}(i)$ includes the vertices connected to vertex $i$. This prevents agents from getting trapped in a confined area (e.g. carriers surrounded by pheromone traces). For carriers $\tau'_{v}(v) = 1 - \tau_v(v)$.

$$
p_d(x, y) = \frac{\tau'_v(v)}{\sum_{(k) \in \text{neighbourhood}(i)} \tau_v(k)} \quad (3.2)
$$

Whenever an agent $i$ moves, at each round $t$, it updates its internal pheromone value $\tau_{a_t}(i)$ (as in Eq 3.3) and also the pheromone amount in its current vertex location $\tau_v(v)$ (as in Eq 3.4).

$$
\tau_{a_t}(i) = (\tau_{a_{t-1}}(i) + 0.01 * (0.5 - \tau_{a_{t-1}}(i))) \quad (3.3)
$$

$$
\tau_v(v) = \tau_{v_{t-1}}(v) + 0.01 * (\tau_{a_{t-1}}(i) - \tau_{v_{t-1}}(v)) \quad (3.4)
$$

If an agent $i$ finds or receives new information, then its internal pheromone value is updated to $\tau_{a_t}(i) = 1$. In this case, Eq. 3.3 decreases the internal pheromone value at each round; and Eq. 3.4 increases the amount of pheromone in the locations that the carrier agent explores.

Passive evaporation is added to make explored paths less dominant and allow re-exploration of routes of agents that fail without sharing information [13]. This type of evaporation is performed by the environment rather than by agents. It is applied in all the vertices of the complex network $G$, using the definition in [24] that corresponds to Eq. 3.5 with an evaporation rate $\rho = 0.01$:
\[ \tau_{v_i} = (1 - \rho)\tau_{v_i(t-1)}, \text{ for } \forall i \in \{V, G = (V, E)\} \]  

3.4 Experiments and Results

Experiments aim to analyse the performance of motion algorithms to solve the data-collection problem in complex networks. The metrics applied were speed, amount of information collected versus time, robustness (in terms of failure resistance) and number of messages sent. Additionally, experimental design provides insights regarding the impact of a selected complex network in the agents performance and the suitability of motion algorithms to achieve the data collection task in a determined topology.

3.4.1 Experimental settings

Each experiment is defined by a combination of a different complex network topology (section 3.2), a different agent motion algorithm (section 3.3), and a different failure probability \( p_f \) for all agents. In all cases, the network consists of 100 vertices and is explored by 10 agents (a relation 10 to 1 from vertex to agent). Each experiment was performed 30 times. Agents start from random locations, selected separately for each topology (but the same ones for all the 30 repetitions in each topology). Each simulation stops when one agent collects all the information from all network vertices, or if all the agents fail. The performance of the two movement algorithms is compared (i.e. Random and Stigmergy-based) within different topologies (listed below), with different failure probabilities starting from zero and increasing \( p_f \) until a value in which most of the experiments fail \((p_f = 0, 0.001, 0.003, 0.005 \text{ and } 0.008)\). The specific parameters used for each complex network topology studied are:

- Lattice: Size \( 10 \times 10 \);
- Small-world: Degree 4, \( \beta = 0.1, 0.3, 0.5, 0.9 \);
- Scale-free: Number of steps \( \text{steps} = 97 \), starting nodes \( sn = 4 \), added links per step \( \eta = 1, 2, 4 \);
- Community: \( \beta = 0.1, 0.3, 0.5, 0.9 \), \( n_{\text{clusters}} = 4 \);
- Community circle: \( \beta = 0.1, 0.3, 0.5, 0.9 \), \( n_{\text{clusters}} = 4 \);
- Forest Hub & Spoke: \( n_{\text{clusters}} = 4 \);
- Line: No specific parameters;
- Circle: No specific parameters.
3.4.2 Results and discussion

Agents are evaluated on different criteria in scenarios with and without agent failure. When no agents fail \((p_f = 0)\), results are analysed in terms of the agents’ performance – i.e. number of rounds before the first agent collects all the data. Figure 3.6 depicts the round numbers for the two algorithms in different network topologies. When no agents fail, all experiments are ultimately successful. In contrast, when agents do fail \((p_f > 0)\), the agents robustness is evaluated instead in terms of success rates – i.e. how often the agents complete the task – and rate of global data collection – i.e. how fast the agents collect the data together (rather than individually). Finally, the number of messages exchanged among agents is also evaluated, as critical in limited resource environments.

An ANOVA test is also performed for failure-less experiments, to determine whether or not the observed differences between the round number means for the two motion algorithms (Random and Carriers) are statistically significant. The null and alternative hypothesis for a determined topology are the following:

- \(H_0\): Round number means of the two algorithms are equal for a network G;
- \(H_1\): Round number means of the two algorithms are different for a network G, indicating a correlation between the algorithm and the round number.

Table 3.1 shows the ANOVA test results. The \(F\)-value represents the F statistics – the variation between the round numbers of the two algorithms, in the given network. The \(p\)-val and \(p\)-wilc indicate the statistical significance between the results of the two algorithms for all the topologies (since \(p\)-val < 0.05 and \(p\)-wilc < 0.05) except for the Community network with a \(\beta = 0.5\) and clusters = 4 (where \(p\)-val > 0.05 and \(p\)-wilc > 0.05; marked as \(*\) in Table 3.1).

Hence, based on the round number box-plots in Figure 3.6 and the ANOVA test, it can be concluded that carriers are faster than random agents for most network topologies, when \(p_f = 0\). However, as observed in Figure 3.6 (and marked as \(*\) in Table 3.1), random exploration is faster than carriers for the forest hub-&-spoke (Figure 3.6e) and the scale-free with \(sn = 4, \eta = 1, steps = 97\) networks (Figure 3.6f). This is probably due to the fact that in these topologies most paths pass through unique large hubs. Therefore, in the carriers case, these hubs are pheromone-marked very often and hence agent movement slow-down across sub-networks.

To quantify the topological features that impact agent performance, a correlation was identified between the round number and topological metrics like network diameter, degree distribution, clustering coefficient, and betweenness centrality. Results showed that the two exception topologies (Scale-free \(sn = 4, \eta = 1\) and \(steps = 97\) and Forest Hub-&-Spoke) feature greater values for the standard deviation of the node betweenness centrality value – \(stdev(betweenness)\) – compared to other topologies, as in Figure 3.6e and f.
Table 3.1. ANOVA and Wilcoxon Test for Carriers vs. Random by Topology

<table>
<thead>
<tr>
<th>Topology</th>
<th>F-value</th>
<th>p-val</th>
<th>Dif</th>
<th>p-wilc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>111.75</td>
<td>3.806e−15</td>
<td>465</td>
<td>1.824e−6</td>
</tr>
<tr>
<td>Circle</td>
<td>248.28</td>
<td>2.2e−16</td>
<td>465</td>
<td>1.822e−6</td>
</tr>
<tr>
<td>Lattice 10 × 10</td>
<td>75.996</td>
<td>2.2e−16</td>
<td>465</td>
<td>1.823e−6</td>
</tr>
<tr>
<td>Scale-free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(⋆) sn = 4, η = 1, steps = 97</td>
<td>91.807</td>
<td>1.476e−13</td>
<td>2.5</td>
<td>2.352e−6</td>
</tr>
<tr>
<td>sn = 4, η = 2, steps = 97</td>
<td>69.922</td>
<td>1.52e−11</td>
<td>465</td>
<td>1.821e−6</td>
</tr>
<tr>
<td>sn = 4, η = 4, steps = 97</td>
<td>128.04</td>
<td>2.2e−11</td>
<td>465</td>
<td>1.822e−6</td>
</tr>
<tr>
<td>Forest Hub and Spoke</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(⋆) clusters = 4</td>
<td>19.755</td>
<td>3.573e−5</td>
<td>36.5</td>
<td>1.355e−5</td>
</tr>
<tr>
<td>Community Network</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>β = 0.1, clusters = 4</td>
<td>47.382</td>
<td>3.952e−9</td>
<td>490</td>
<td>2.21e−6</td>
</tr>
<tr>
<td>β = 0.3, clusters = 4</td>
<td>46.56</td>
<td>5.797e−9</td>
<td>444.5</td>
<td>1.359e−5</td>
</tr>
<tr>
<td>(⋆) β = 0.5, clusters = 4</td>
<td>2.304</td>
<td>0.1345</td>
<td>322.5</td>
<td>0.06561</td>
</tr>
<tr>
<td>β = 0.9, clusters = 4</td>
<td>18.228</td>
<td>6.84e−5</td>
<td>474</td>
<td>8.928e−5</td>
</tr>
<tr>
<td>Community Circle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>β = 0.1, clusters = 4</td>
<td>152.07</td>
<td>2.2e−16</td>
<td>435</td>
<td>2.701e−6</td>
</tr>
<tr>
<td>β = 0.3, clusters = 4</td>
<td>144.48</td>
<td>2.2e−16</td>
<td>422.5</td>
<td>9.77e−6</td>
</tr>
<tr>
<td>β = 0.5, clusters = 4</td>
<td>93.448</td>
<td>1.07e−13</td>
<td>465</td>
<td>1.822e−16</td>
</tr>
<tr>
<td>β = 0.9, clusters = 4</td>
<td>121.477</td>
<td>7.472e−16</td>
<td>465</td>
<td>1.823e−6</td>
</tr>
<tr>
<td>Small World</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>β = 0.1, degree = 4</td>
<td>126.86</td>
<td>3.149e−16</td>
<td>465</td>
<td>1.821e−6</td>
</tr>
<tr>
<td>β = 0.3, degree = 4</td>
<td>65.385</td>
<td>4.394e−11</td>
<td>465</td>
<td>1.822e−6</td>
</tr>
<tr>
<td>β = 0.5, degree = 4</td>
<td>85.378</td>
<td>5.34e−13</td>
<td>465</td>
<td>1.823e−6</td>
</tr>
<tr>
<td>β = 0.9, degree = 4</td>
<td>144.64</td>
<td>2.2e−16</td>
<td>465</td>
<td>1.817e−6</td>
</tr>
</tbody>
</table>
To test this correlation, more scale-free network instances were generated using the same parameters: \( sn = 4, \eta = 1 \) and \( steps = 97 \); and their \( \log(\text{round number}) \) versus \( \text{stddev}(\text{betweenness}) \) was plotted. Indeed, Figure 3.7-a shows the correlation of the betweenness centrality and the round number of all topologies, including the additional Scale-free ones. Since most topologies have relatively low betweenness values (lower than 0.025) compared to the Scale-free cases (greater than 0.05), these cases are only shown in Figure 3.7-b, to highlight that a correlation also exists for these topologies, even at a different scale. Figure 3.7-c shows the same correlation for the Random algorithm, in all topologies; and Figure 3.7-d shows the correlation for both algorithms, in all topologies. The carriers algorithm seems to feature a stronger relation between the round number and the betweenness centrality, compared to the random case, that has greater betweenness values causing larger round numbers (i.e. lower performance). For system designers, this means that selecting the best agent exploration algorithm depends on the network topology (betweenness centrality); and the selected algorithm may have to change over time, for better performance, as the network topology evolves.

The global information collected by all agents combined (rather than by each agent) was also evaluated. This is useful for analysing algorithm robustness in case of agent failure, especially for applications where all agents can communicate data collected to a central location, and where a percentage of the complete data suffices (e.g. 90%). Hence, the shape of the function describing global information collected in time is important.
CHAPTER 3. SELF-ORGANISED DATA COLLECTION IN COMPLEX NETWORKS

Figure 3.7. Betweenness Centrality (std. dev.) Correlation to the Round Number (log)

a) Carriers for all topologies including several Scale-free cases

b) Carriers for all topologies except Scale-free cases

c) Random for all topologies

d) Carriers vs Random for all topologies
With steeper shapes offering better robustness, data is collected faster, before agents start failing. Global information is measured in each experiment (i.e. given topology, motion algorithm and $p_f$) by reading the local information collected by each agent, at each round, and calculating the total sum. Figure 3.8 presents the increase of the global information with the round number in the Scale-free network (generated with parameters $sn = 4$, $\eta = 1$ and $steps = 97$) with the two algorithms. Each experiment is performed 30 times and the minimum, median and maximum values plotted. Results show that global information is collected faster by carriers than by random agents – e.g. at round 50, the minimum collected by carriers is about 90%, whereas by random is only 70%; at round 100, the minimum for carriers is 97% and for random, about 85%). It also seems that for carriers the longest time is spent for collecting the last 3% of the data, which makes them to be slower than random for collecting all data in this topology (Cf. Figure 3.7-d). This means that in applications where less than 97% of data collection suffices (e.g. some sensor networks), the carriers can outperform random agents even in such topologies. Finally, in all cases, the median values approximate the maximum ones much faster for the carriers than for random (e.g. round 75 for carriers and not before round 125 for random).

Regarding the cases where agents can fail, Figure 3.9 provides a histogram of the success rates (a) and the box-plots of the messages sent (b), with the selected topologies and $p_f = 0.001$. Figure 3.9-a shows that for the carriers, the topologies most impacted by agent failures are the scale-free (with $sn = 4$, $\eta = 1$ and $steps = 97$), where success rates drop to about 60%; and the forest hub-and-spoke, to about 96.66%. For the random algorithm, the only topologies impacted are circle and line. By comparing these success rates with the round number evaluations (Cf. Figure 3.7), we can note that faster data collection favours success rates. Figure 3.9-b shows a higher number of message exchanges among random agents compared to carriers. This could explain the lower success rates for carriers in topologies that infringe agent circulation (e.g. some Scale-free cases), since agents are less likely to meet and their information is lost when they fail.
Figure 3.9. Success Rates and Messages Sent for $p_f = 0.001$

Figure 3.10 shows the global information collected in the topologies most impacted by agent failures (Scale-free and Circle). In both cases, the median reaches a 100% faster for carriers than for random agents. Also, in the scale-free case where success rates suffer, carrier agents actually manage to collect all the information together, yet they never meet to share the information and hence no single agent completes. Random agents are slow to explore line and circle networks as they move around the same vertices and share the same local information. Hence, when they fail, their information is lost and other agents do not reach the same areas before their own failures.

Figure 3.11-a shows that for carriers with $p_f = 0.003$, the success rate for the forest hub-and-spoke topology is further reduced to 43%; and for scale-free ($sn = 4$, $\eta = 1$ and $steps = 97$) to 4%. Random agents also start featuring lower success in this scale-free topology (60%). Community networks start suffering in the random case, while no impacted is registered when carriers are used. For line and circle networks, random exploration becomes severely impaired (less than 10%), while carriers maintain 100% success rates. The success of both algorithms remains intact (100%) for small-world topologies.

Figure 3.11-b indicates that for the community network, random agents exchange more messages than carriers, even with less success rates, thus, they probably exchange redundant data. Figures 3.12-a and 3.12-b show a fast data collection for carriers, indicating that they are better for exploitation of new vertices in these networks, since they feature higher success rates despite exchanging fewer messages. In the forest hub & spoke network, agents exchange more messages via the hubs, and the median of global information converges faster to maximum value for carriers than for random.

Figure 3.13 shows success rates of $p_f = 0.005$ in each kind of network, with different generation parameters. Small-world (3.13-a) and community circle (3.13-c) feature the
Figure 3.10. Global Information Collected for Scale-free $sn = 4 \eta = 1$ and $steps = 97$ and Circle, with $pf = 0.001$

Figure 3.11. Success Rates and Messages Sent for $pf = 0.003$
CHAPTER 3. SELF-ORGANISED DATA COLLECTION IN COMPLEX NETWORKS

Figure 3.12. Global Information Collected for Community Network $\beta = 0.3$ and Forest Hub-and-Spoke, with $p_f = 0.003$

Figure 3.13. Success Rates For Complex Networks $p_f = 0.005$
highest success rates compared to other topologies. Carriers perform well in all small-world networks, whereas random performs the worst yet relatively better with higher beta values – less regular and more random graphs (Figure 3.13-a). Carriers also start to decrease success rates in community networks (Figures 3.13-b and 3.13-c), especially when clusters are less regular (i.e. greater beta values). In community networks, success is lower than in community circle with both algorithms, since the clusters are connected through a single central vertex, impeding movement.

For a $p_f = 0.008$, only the Carriers manage to reach success rates over 70% in the most failure-resistant topologies: lattice, small-world (all configurations), community circle (all configurations) and scale-free (4-4-97).

Additionally, a statistic test was performed for the Community network ($\beta = 0.5$) and $p_f = 0.001$, because no statistically significant difference was found in terms of round number for $p_f = 0$, while the box-plot for the round number did show a difference (Figure 3.14-a). By taking advantage of the 100% success rates of both algorithms, a Wilcoxon test for round number and a $p_f = 0.001$ indicated a $p_{val} = 0.0001538$. This means that there is a significant difference between the round number means, with carriers being faster than random when failures occur. For the other failure probabilities, success rates decrease faster for random agents than for carriers (Figure 3.14-b, c and d).
3.5 Summary

The problem of data-collection in complex networks using failure-prone agents was studied in this chapter. Two agent motion algorithms (random and a pheromone-based) for exploring complex networks with different topologies are evaluated: Small-world, scale-Free and community networks. Several regular network topologies for comparison purposes are selected for experiments: Lattice, forest hub & spoke, line and circle. Experimental results showed that a pheromone-based exploration technique improves the exploitation of new paths and results in faster data-collection for most experiments in the different topologies. Results also indicate a relation between network topology and data-collection performance, where the differentiating factor among topologies can be quantified as the variance of the betweenness centrality among nodes. Namely, the higher the standard deviation of the betweenness centrality of nodes in a complex network, the higher the completion times of the data-collection task in that network.

Next chapter introduces a model to replicate mobile agents and node failures are introduced. In this scenario, agents are recovered by nodes and mobile agents aim to synchronise the network topology allowing nodes to recreate other crashed nodes.
CHAPTER 4

Replication-Based Self-healing of Mobile Agents
Exploring Complex Networks

In previous chapters, a pheromone-based agent exploration algorithm that performs best in most targeted environments is proposed. Additionally, the most sensitive network topology characteristics to agent failure are studied. In this chapter, a replication-based self-healing approach is proposed. The approach enables agents to complete a data-synchronisation task even with high-failure rates, in failure-sensitive network topologies. System nodes can learn and estimate time-outs dynamically, to minimise false positives. Overheads incurred by agent replication are studied in terms of memory consumption and message communication. The reported findings can help to design viable multi-agent solutions for a wide variety of data-intensive distributed systems.

4.1 Introduction

Multi-agent based approaches can offer highly scalable, robust and flexible ways to provide data-collection and synchronisation services in large-scale dynamic distributed environments, ranging from physical terrains to sensor networks, computing clouds, and the Internet of Things (IoT). In the previous chapters, it was shown that both the network topology of the targeted distributed system, and the agents’ network exploration algorithm, are key to service performance, and, consequently, to robustness in case of failure-prone agents. In [71] a pheromone-based agent exploration algorithm was proposed, which performs well for collecting data from physical environments with various topologies. The performance of this algorithm was also confirmed for distributed computing systems, where the exploration space was a complex network (rather than a uniform surface) [73]. These findings are consistent with related works that also correlate complex network topology to distributed task performance [94, 18, 34].
Performance also impacts robustness, since faster agent exploration provides better resistance to agent failure (i.e. the task is completed before all agents fail). In [73], the most sensitive characteristics of network topologies to agent failure were identified, when using the pheromone-based algorithm. Namely, the agents success was inversely correlated to the variance of the betweenness centrality among nodes – e.g. Small World or Hub & Spoke topologies were the most fragile. To improve the robustness of the exploration algorithm in such topologies, a self-healing approach is proposed based on localised agent replication.

Replication is a well-known strategy based on object copy creation (e.g. files, databases, sensor information) [95]. It helps to deal with component failures and hence ensure better performance and dependability. In distributed systems, time-outs are commonly used to detect node failures and trigger replication. Nodes repeatedly exchange ping messages (or heart-beats) and the lack of a node response within a delay indicates the node failure [93, 95]. One difficulty here is to estimate time-out values, so as to avoid false positives and node over-replication. Another difficulty is system scalability, as the number of monitoring messages may grow exponentially with the system size. Several strategies are available to limit communication to local domains [84, 40].

In the presented study, agents aim to complete the collective task of synchronising data across all nodes of a distributed system. Each node is initialised with different data, which is only updated by agents (this is a simplifying assumption that can be lifted in future works). Agents explore the system and progressively collect, merge and deposit data at each node, until all nodes contain the same data aggregate. Agents can fail with a predefined probability, meaning they are removed from the system. The self-healing technique proposed here aims to replicate agents for: i) ensuring task completion, even for relatively high failure rates; and, ii) avoiding false positives and over-replication, when time-outs are initially unknown (e.g. vary dynamically).

In short, each node keeps references of agents that leave the node for a neighbour node. Once arrived, the agent notifies the node that it left from. If the node does not receive a notification from an agent within an expected delay, then, the node creates a new agent and injects the latest local data aggregate into it. The time-out is the estimated delay for an agent’s transfer from the current node to its neighbour. To avoid over-replication, if a notification arrives after a replica has already been created, the node picks one of the next arriving agents, merges its data with the local aggregate, and then removes the agent. The node also updates its time-out delay accordingly, hence learning and adapting to local communication delays on each of its network links.

This replication technique applies to tasks performed by identical agents, where differing agent states (i.e. partial data aggregates) can be reloaded from the local nodes. Possible applications include ad-hoc or sensor networks where agents can be lost during communication and in re-designing services like traditional DNS using peer-to-peer protocols where network partitions can produce disruptions in internet services and high performance is required [22]. Additionally, future work will aim to enable agents to handle
node and link failures in a local fashion (e.g. partial failures, where agents provide altered data [56] or node crashes in cloud environments where the topology must be maintained and node instances can be created and interconnected dynamically).

The remaining of this chapter is organized as follows: Section 4.2 presents the agents data-replication collective task, the complex network topologies studied and the agent design. Section 4.3 presents the replication technique and Section 4.4 details the experimental settings (i.e. complex network topology, failure rates and replication algorithm configuration) and discusses results.

4.2 Data Replication Problem

Agents must explore nodes interconnected via a complex network. Initially, each node holds different data. The agents collect and replicate data across the nodes until all nodes hold the same data aggregate. Agents can fail with a probability \( p_f \) (e.g. \( p_f = 0.1 \) means that an agent can crash in one of ten simulation steps).

The agent implementation is based on [73], which is in turn inspired by previous approaches [79]. Each agent is endowed with a collection of perceptions \( P = \{ \text{pheromone, data, node, neighbors, msg} \} \) and actions \( A = \{ \text{Move}(k), \text{Collect}, \text{Send}(msg), \text{Recv} \} \). In the collection \( P \), \text{pheromone} is a vector of real numbers with values in \([0,1]\) that represent the amount of pheromone in the agent vicinity (i.e. nodes adjacent to the current location); \text{data} stores a data copy on the current node; \text{neighbour} returns the identities of agents in the same vertex; \text{msg} stores messages from other agents; and \text{node} returns the agent location (current node). In the actions, \text{Move}(k) \) moves the agent to a node \( k \) and copies the agent’s data in \( k \); \text{Collect} copies data from the current node in the agent’s memory; and \text{Send} and \text{Recv} exchange information with other agents.

Simulation is performed in rounds. Each round defines the time in which each agent senses its local environment (current node, data in current node, co-located agents and adjacent nodes); decides an action (e.g. move to determined vertex based on the amount of pheromone) and execute it [79].

Complex networks have a large number of interconnected nodes with some special features, like relatively small distances among nodes, power-law degree distributions and non-trivial topological properties [18]. For the experiments, complex networks that were challenging for completing a data-collection task were selected, as shown in [73]: Small-world, Scale-free, Community network and Hub & Spoke (Fig 4.1). For the small-world (Fig 4.1-a), we used the model of Watts-Strogatz [95, 101]. In this model, a regular ring lattice network with \( n \) vertices and \( k \) edges per vertex is created. Then each edge in the network is rewired with a probability \( \beta \). The \( \beta \) parameter determines how regular the final network will be: \( \beta = 0 \) generates a regular network, \( \beta = 1 \) a random network, and in-between values a small-world network [50]. The community
network in this chapter (Fig 4.1-b) were generated using a $n_{clusters}$ parameter to define the number of groups in the network and adding a single connection between nodes of different groups. Each group was generated as a small-world network (with its own $k$, $\beta$, and $n = m/n_{clusters}$, where $m$ is the number of nodes in the network). The community network with four clusters is connected via a circle formed by pairs of nodes selected randomly from different groups. The scale-free network (Fig 4.1-c), was modelled by starting with $sn$ nodes and $\eta$ connections. At each step, a new node is added and connected via $\eta$ links to existing nodes, giving priority to nodes with higher degrees [45]. The probability to connect to an existing node is defined by $p_i = \frac{k_i}{\sum_j(k_j)}$, where $k_i$ is the degree of node $i$ [101, 89, 92]. Hub & spoke corresponds to a star configuration of a central node and $n-1$ adjacent nodes. Hub & spoke networks are applied for obtaining high availability and reliable computing services by expansion of individual cloud instances [47].

Agent exploration strategy

To implement the agents exploration strategy, the pheromone-based approach presented in [73] was adopted. It is an adaptation for complex network exploration of the algorithm in [71], which is in turn inspired by the Ant Colony Systems (ACS). In this strategy, agents choose the nodes with the minimum amount of pheromone as exploitation rule. If more than one nodes have the same minimum amount, the next location is selected at random.

When a simulation starts, the initial value of pheromone in each node is $\tau_0 = 0.5$. As in ACS [23], the decision of exploring or exploiting new nodes is based on a pseudo-random variable $q \in [0, 1]$ (Eq. 4.1):

$$dir = \begin{cases} 
\text{exploitation rule} & \text{if } q \leq 0.9 \\
\text{biased exploration} & \text{otherwise}
\end{cases} \quad (4.1)$$
Biased exploration is a random-proportional rule based on the amount of pheromone in each node adjacent to the current location of an agent \( i \) (Eq. 4.2). The **neighbourhood**(\( i \)) includes the vertices connected to vertex \( i \). Biased exploration prevents agents from getting trapped in a confined area (e.g., agents surrounded by pheromone traces). \( \tau'_v(v) = 1 - \tau_v(v) \) is defined to increase the selection of vertices with a minimum amount of pheromone.

\[
p_d(x,y) = \frac{\tau'_v(v)}{\sum_{(k) \in \text{neighbourhood}(i)} \tau_k(k)} \quad \text{(4.2)}
\]

In each round \( t \), each agent \( i \) updates its internal pheromone value \( \tau_{at}(i) \) (as in Eq 4.3) and the pheromone amount in its current node \( \tau_v(v) \) (as in Eq 4.4).

\[
\tau_{at}(i) = (\tau_{at-1}(i) + 0.01 \times (0.5 - \tau_{at-1}(i))) \quad \text{(4.3)}
\]

\[
\tau_v(v) = \tau_{vt-1}(v) + 0.01 \times (\tau_{at-1}(i) - \tau_{vt-1}(v)) \quad \text{(4.4)}
\]

If an agent \( i \) finds or receives new information, then its internal pheromone value is updated to its initial value \( \tau_{ai}(i) = 1 \). In this case, Eq. 4.3 decreases the internal pheromone value at each round; and Eq. 4.4 increases the amount of pheromone in the locations that the carrier agent explores.

Additionally, a passive exploration strategy is added to avoid stagnation, allowing re-exploration of routes of agents that failed without sharing information [13]. The environment performs pheromone evaporation on all nodes \( V \) of the complex network \( G \), using the definition in [24] with evaporation rate \( \rho = 0.01 \): \( \tau_{vi} = (1 - \rho)\tau_{vi(t-1)} \), for \( \forall i \in \{V, G = (V, E)\} \).

### 4.3 Agent Replication Approach

To deal with agent failure, a replication strategy is proposed enabling nodes to create new agent instances. Agents can fail when moving between nodes (e.g., unreliable communication). Node crashes will be addressed in the next chapter. In the proposed model, each node is responsible for a dynamic set of agents, called followedAgents = \( \{a1, a2, a3, ..., a_n\} \), which is monitored and possibly replicated.

When an agent departs from a node, it sends a message *departing* to its current node, including the agent id and the node destination. When an agent arrives at a node destination, it sends a message *freeresp* back to its previous node. Each time a node receives a *departing* message, it assumes responsibility for the corresponding agent, adding it to the
CHAPTER 4. REPLICATION-BASED SELF-HEALING OF MOBILE AGENTS EXPLORING COMPLEX NETWORKS

followedAgents set. When a node receives a freeresp message, it deletes the agent from followedAgents.

Each node has an internal round counter, nodeAge, for calculating the round difference between the departing and freeresp messages for each agent $k$ in its followedAgents set. All differences are stored in a special-purpose data structure, expectedMsgtime, separately from each neighbour node:

\[
\text{expectedMsgtime}(\text{destination}(k)) \leftarrow \text{getRound(freeresp)} - \text{getRound(departing)}.
\]

Algorithm 5 defines how nodes decide when to create an agent replica, for each agent in followedAgents. The variable $wsize$ is introduced to calculate the median and standard deviation of the last $n$ elements in expectedMsgtime, and the expected time-out. When the time-out expires, the node creates an agent replica and injects its local data aggregate into it. When a node detects a false positive – i.e. receives a freeresp after creating a replica – the node deletes the next arriving agent (after collecting its data).

\begin{verbatim}
Algorithm 5: Replication algorithm
1: for each Agent $a \in \text{followedAgents}$ do
2:     $\text{dest} \leftarrow \text{getDestination}(a)$  \hfill \text{\# gets next location of agent $a$}
3:     $m\text{Rounds} \leftarrow \text{median(expectedMsgtime(dest), wsize)}$
4:     $s\text{Rounds} \leftarrow \text{stdev(expectedMsgtime(dest), wsize)}$
5:     if $\text{nodeAge} - \text{getRoundDepartMsg}(a) > m\text{Rounds} + 3 \times s\text{Rounds}$ then
6:         createReplica(a)
7:         followedAgents.remove(a)
8:     end if
9: end for
\end{verbatim}

4.4 Experiments and Results

Experiments aimed to: i) determine whether the proposed replication strategy is able to deal with agent failures, or not, as well as complete the data-replication task; ii) assess the time taken in rounds, and the induced overheads in terms of memory consumption and communication messages; and, iii) show how the system could learn to approximate communication time-outs and avoid over-replication, by alternating simulations with and without agent movement delays.

Each experiment was performed 30 times. Agents start from random locations, selected separately for each topology (but the same ones for all 30 repetitions in any topology). In addition, each network consists of 100 nodes and is explored by 10 agents (in a relation 10 to 1).

An experiment is a combination of: agents that replicate data (10 agents are defined); a failure probability: $p_f = \{0, 1e^{-3}, 3e^{-3}, 5e^{-3}, 7e^{-3}, 9e^{-3}, 1e^{-2}, 3e^{-2}, 5e^{-2}\}$; activation or deactivation of the replication algorithm; and one complex network (small-world $n = 100, k = 4, \beta = 0.5$, community network $n_{clusters} = 4, m = 100, k = 4, \beta = 0.5$, scale-free $sn = 4, steps = 97, \eta = 1$ and hub & spoke: $n = 100$).
CHAPTER 4. REPLICATION-BASED SELF-HEALING OF MOBILE AGENTS EXPLORING COMPLEX NETWORKS

Figure 4.2. $p_f$ vs. success rates for Community network $n = 100$, $\beta = 0.5$, degree = 4 with no replication.

Table 4.1. False Positives with Replication and without Delay

<table>
<thead>
<tr>
<th>Complex Network</th>
<th>Number of False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small-World $\beta = 0.5$, $n = 100$, degree = 4</td>
<td>0</td>
</tr>
<tr>
<td>Hub &amp; Spoke</td>
<td>0</td>
</tr>
<tr>
<td>Scale-free $sn = 4$, $eta = 1$, $numSteps = 97$</td>
<td>1</td>
</tr>
<tr>
<td>Community $\beta = 0.5$, degree = 4, clusters = 4</td>
<td>4</td>
</tr>
</tbody>
</table>

The parameters of the replication algorithm are: time-out of $t = 10$ rounds assigned initially to each element in $expectedMsgtime$ and $wsize = 10$. $expectedMsgtime$ is stored in each simulation to have a history and use this values in the next simulation allowing learning of the time-outs.

4.4.1 Experiments without agent movement delays

These experiments were performed without agent movement delays between nodes. The first comparison is between experiments with and without replication. With replication, all experiments are successful even with a $p_f = 0.5$, as the agents complete the task for all the analysed complex networks. Without replication, success rates drop with the increase of $p_f$ (Fig. 4.2). Hence, there are no successful experiments for a $p_f \geq 0.1$ in the community, small-World and hub & spoke networks; neither for $p_f \geq 0.003$ in the selected scale-free network.

In terms of number of replicas created, the 30 simulations were sorted by date and obtained maximum, median and minimum number of agents in each simulation. As expected, without replication, the median and minimum number of agents reduces when $p_f$ increases (Fig. 4.2). Fig 4.3 shows the results with replication. For a $p_f = 0$, the initial number of agents is maintained (i.e. 10 agents), as there is no need to create replicas. For increasing $p_f$ (i.e. 0.1, 0.3, 0.5), the minimum and median numbers are lower, showing the need for replication. Additionally, the system learns the time-out and, hence, increases the minimum number of agents, making that the median tends to the initial number, for all $p_f$ values. The same behaviour is observed for all the tested complex networks.
Figure 4.3. Simulation sorted by date vs number of agents (max, median, min) in a Small-World network $n = 100$, $\beta = 0.5$, degree $= 4$.

Table 4.1 presents the total number of false positives in the 30 experiments for each topology and all the values of $p_f$. From this table, it is observed that the number of false positives is small. For the scale-free one false positive is presented with a $p_f = 0.5$ and for community network, there are two with a $p_f = 0.007$ and a $p_f = 0.3$.

4.4.2 Experiments with agent movement delay

A delay of 1000 ms is added in each agent movement, and the initial time-out of 10 rounds is maintained. As this time-out is too small compared to the actual agent movement delay, it initially produced a lot of false positives. Fig 4.4 shows how time-out values adapt, reducing the number of replicas from one simulation to another. Time-out learning can be faster if $p_f$ tends to zero. Additionally, in all cases using replication, success rates are 100%. For higher $p_f$ values, results showed that the number of agents is reduced, yet this takes more time since agents fail continuously (Fig. 4.4). As shown in Fig. 4.5a, a higher $p_f$ implies less agents moving in the network and a higher time to estimate the real time-out. However, even with high failure rates, the system adapts and reduces false positives over time.

4.4.3 Results in terms of round number

These results compare the differences in rounds for the experiments with and without replication for $p_f = 0$. The null and alternative hypothesis for a determined topology are the following:
CHAPTER 4. REPLICATION-BASED SELF-HEALING OF MOBILE AGENTS EXPLORING COMPLEX NETWORKS

Figure 4.4. Simulations Sorted by Date vs. Number of Agents (max, median, min) in a Small-world Network $n = 100, \beta = 0.5$, degree = 4 with delay.

Table 4.2. Differences in Rounds of Replication with Delay and No Replication with a $p_f = 0$

<table>
<thead>
<tr>
<th>Network Type</th>
<th>$p - value$</th>
<th>repl</th>
<th>no repl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small-World</td>
<td>0.6891</td>
<td>$2828.034 \pm 742.79$</td>
<td>$2636.448 \pm 250.54$</td>
</tr>
<tr>
<td>Community Network</td>
<td>0.6856</td>
<td>$4927.552 \pm 1126.13$</td>
<td>$5050.655 \pm 1380.60$</td>
</tr>
<tr>
<td>Scale-free</td>
<td>$3.725e-09$</td>
<td>$103529 \pm 46554.29$</td>
<td>$28640.9 \pm 7469.88$</td>
</tr>
<tr>
<td>Hub &amp; Spoke</td>
<td>0.003511</td>
<td>$5371.52 \pm 584.27$</td>
<td>$5855.14 \pm 485.43$</td>
</tr>
</tbody>
</table>

- $H_0$ : the means of the round number with and without replication are equal for a network $G$;
- $H_1$ : the means of the round number for the two algorithms are different for a network $G$ indicating correlation between the application of replication and the round number.

As shown in Table 4.2, for small-world and community networks there is no significant difference between means ($pval > 0.05$). For Hub & Spoke, the mean and median of the round number are lower with replication than without replication, with a significant difference. However, the greatest difference in means is for scale-free networks, as the replication algorithm spends more time with replication compared to other complex networks.

Fig. 4.5.b shows a box-plot of $p_f$ versus round number for a Small-World network. As $p_f$ increases, the round number for task completion increases. This can be because with
CHAPTER 4. REPLICATION-BASED SELF-HEALING OF MOBILE AGENTS EXPLORING COMPLEX NETWORKS

4.4.4 Results in terms of memory consumption

To evaluate memory consumption the scenario with delay was chosen, because it generates a greater number of replicas, and it was compared to the simulation with $pf = 0$. As shown in Fig. 4.6, the difference is not bigger than 2 MB in memory consumption after the system learns the time-out for small-world, Community and hub & spoke. The scale-free network features the highest difference in terms of memory consumption. The reason could be that this network topology takes more time to complete the task, with the agent exploration algorithm proposed.

Figure 4.5. a) Simulation sorted by date vs. false positives b) Number of rounds necessary to successfully complete a simulation versus $pf$ - Small-World network $\beta = 0.5$, degree = 4 with replication.

$p_f \geq 0.1$, agents fail more frequently, thus, requiring more replicas to complete the task (for $p_f = 0.1$ there are no successful experiments without replication).

Figure 4.6. Simulation Sorted by Date vs. Memory Consumption.
4.5 Summary

In this section, we proposed a replication approach to deal with failures in mobile agents. This approach allows a system to complete a distributed task even with high failure rates. Agents communicate between them and perform a data-replication task. Nodes have been modelled to store information and also can create new agents and learn the delay necessary to create a new agent copy, allowing the system to self-recover. The proposed approach only stores in local node memory the identities of the departed agents, rather than entire agent replicas, taking advantage of the monotonically increasing synchronisation of information on each node, over time. When a new agent replica is created, it receives the information available in the local node, which is the same or more up-to-date than the information of the failed agent. This helps to save node memory and increase task competition speed. Experimental results show that the system approximates the number of replicas to create by obtaining the initial number of agents. By replication, it is possible to have 100 percent of success for the task, in performed experiments. However, experiments suggest that the time in rounds taken to complete the task increases with the failure probability (if $p_f \geq 0.1$).

Experimental results show that the system approximates the number of replicas to create by obtaining the initial number of agents. By replication, it is possible to have 100 percent of success in the performed experiments. However, experiments suggest that the time in rounds needed to complete the task increases with the failure probability (if $p_f \geq 0.1$).

Obtained results may provide useful insights into the behaviour of data management applications with different execution environments, and could help design distributed protocols for data replication in server clouds, clusters and different kinds of distributed devices (IoT). More information regarding experiments, including complete measurements, tables and source code are available at [http://www.alife.unal.edu.co/~aerodriguezp/networksim/](http://www.alife.unal.edu.co/~aerodriguezp/networksim/). Next chapter will focus on another important challenge in self-healing: recovering failing nodes from failures.
CHAPTER 5

Self-healing Networks via Self-organising Mobile Agents

After having replication of agents when several links fail in the last chapter, it is necessary to deal with node failures and propose a way to recover the structure of a determined complex network. Given a network structure and some constraints -like a failure speed lower than the speed of the recovery process-, a model to recover nodes from failures is proposed based on collecting information regarding its local topology. In experiments performed, self-healing of a network structure is achieved by using the collection algorithms from Chapter 3 to synchronise topology data. Additionally, the idea of the previous chapter to heal mobile agents is extended recovering the network structure and mobile agents at the same time.

In this chapter, a model to self-heal complex networks from failures in nodes is proposed. As a difference with another works reviewed, the model creates replicas of the existing failing nodes and its links instead of defining new links among surviving nodes to recover network functionality. The proposed approach is based on local information regarding the topology of a determined network. With some assumptions, experiments reveal that it is possible to detect and recover failures in a network if nodes know the network topology. However, there are cases when the topology is unknown to nodes or changes in time. To tackle this, some strategies to collect topology data are studied: multicast and mobile agents. By multicast, nodes are able to recover the topology given a failure probability of 0.001 and maintaining a median similarity of 100% for 20000 rounds. By using mobile agents, results of the experiments look promising because a good median similarity over time is achieved, with the advantage of less overloads in terms of number of messages compared with multicast. Additionally, a strategy to recover mobile agents by extending a replication algorithm in chain is also presented and analysed.
5.1 Introduction

Electric power grids, the Internet, social networks, data centres and clouds are distributed systems with different complex network topologies. The underlying topology has a significant impact on system performance [94]. Hence, several researches have been striving to identify and study different types of complex network topologies [38, 101, 89, 92] and determine their impact on various functions and properties of the system, including the speed and robustness of data collection [73], routing, information replication [99], or generation of networks with resilience based on self-organisation ideas [38].

In the case of server farms and cloud management, maintaining and recovering servers from failures is essential for meeting stringent requirements of availability and reliability. Since these systems serve millions of users in parallel, downtimes can generate losses of thousands of dollars per minute [43]. To tackle this challenge in such large-scale distributed systems, inspiration has been drawn from natural systems, like insect colonies [17]. Self-organising mechanisms are of particular interest as they can achieve global objectives and behaviours based on strictly local interactions, hence seamlessly adapting to external perturbations and fluctuations in the individual population.

In this chapter, a multi-agent approach is proposed for self-repairing the underlying network structure of large-scale distributed systems more than generating resilient networks as in [38]. Similar work has been proposed in the literature for dealing with node failures that can otherwise lead to network disintegration into isolated components and render the system dysfunctional. [30] proposes a healing strategy that deals with node failures by creating new links between the remaining nodes to avoid network splitting. The main difference with our proposal is that in [30] failed nodes are not recovered, but it aims to ensure network connectivity by recreating new connections when a node reaches a threshold of lost neighbours or based on constraints like maintain distances as short as possible compared with the original network or a determined maximum distance. In the model proposed in this chapter, neighbouring nodes monitor each other via mobile agents performing a collective task, and recreate each other in case of failure based on local interactions. The objective of self-healing here is to maintain the network topology as close as possible to the original one.

In our model, the occurrence of node failures is defined via a failure probability, which is then enacted for each node during system simulation. This assumption generally holds in computer networks, which is our target application domain. Nonetheless, it would not hold in certain application domains such as financial markets or organisms that consists of many interacting dynamical networks [48]. [48] defines three main types of failures: internal failures, associated with malfunction or bad management of organs in the body; external failures, that can result via propagation through connections with other failing nodes, within the same network; and, dependent failures, produced by cascading failures from another connected network. Our work aims to self-heal internal failures in nodes because we propose a self-healing approach based on: 1) a decentralised data-collection
mechanism for acquiring network topology information; and, ii) a local node recreation and reconnection mechanism for recovering network structure when nodes fail. As in [48], we assume that the self-repair process is fast enough relatively to the node failure frequency.

Two types of agents are defined: i) static agents that manage the network self-repair process; and ii) mobile agents that collect and distribute information on network topology within local neighbourhoods. The mobile agents developed in chapter 3 [73] are used here for generic data collection and replication, with the specific data concerning the network topology.

This proposal was validated experimentally with the simulation platform presented in [75]. Results show that it is possible for nodes to recover the network structure having knowledge of the network structure or getting information from external sources like via multicast or mobile agents. The proposed approach based on mobile agents is better in terms of communication overloads than multicast because it implies less messages regarding the topology even with the messages overhead of a node to recover mobile agents from cascade failures when produced in nodes. Additionally, a chain protocol to recover mobile agents from failures is proposed. This model recovers agents based on the replication algorithm proposed in Chapter 4 [74].

In the remainder of this chapter, section 5.2 presents the problem of self-repairing a network structure with its assumptions and constraints. Section 5.3 presents the proposed approach to address this problem by detecting missing nodes with information provided from mobile agents and via multicast. By using agents, a chain strategy is also proposed based on [74] to tackle cascade failures in agents located in crashed nodes. Section 5.6 presents experiments and results obtained when it is possible to achieve self-healing of the network by synchronising topology information from a external source and profiling, in terms of network similarity, messages consumption, number of agents to achieve the task and analysis of the chain protocol.

5.2 Network Self-healing Problem

This work intends to solve the following question: how to achieve self-healing of a determined complex network structure by using mobile agents? To solve this question there are some assumptions: 1) The first one is that the number of interconnected nodes that fails at the same time is limited. 2) As a living organism, a network can self-repair from some distributed small failures but not from several critical failures. 3) Nodes cannot be recovered, thus, a new replica of a crashed node must be created to recover a network structure. 4) If a node fails, all the history about the topology and agents in this node are lost. 5) A node can recreate and/or connect to any other node in order to find a way to rebuild the network structure. 6) There are no failures in communication because if two nodes exchange messages there are no integrity problems and all messages sent are received by its destination.
To recover failing nodes as long as any other nodes stays alive, it is required that each node collects and stores information about the topology of the network. This might incur too high overheads for collecting and storing this information, and even not scale for large networks that change frequently and/or if the topology is unknown. Additionally, it is required a trade-off between the amount of information collected by each node about other nodes and links in the network and the extent of the failure that can be self-repaired completely in terms of sub-networks size or number of interconnected nodes that can fail at once.

Figure 5.1 presents some complex networks challenging for data collection in [73] that are selected for experiments: a small-world, scale-free, community and forest hub & spoke. The Watts-Strogatz model [98] is used to generate the small-world network in Fig 5.1-a. A probability $\beta$ is defined to rewire a lattice network of $n$ vertices with degree $k$. A $\beta = 0$ generates a regular network, otherwise $\beta = 1$ a random network, and in-between values a Small-World network [50]. Fig 5.1-b corresponds to a community network generated by defining 4 clusters each one with an small-world with its own $k$, $\beta$, and $n = m/n$ clusters, where $m$ is the number of nodes in the network. Connection among clusters is performed via a circle of pairs of nodes selected randomly for each cluster. Fig 5.1-c, is an scale-free network designed to start with $sn$ nodes and $\eta$ connections. A new node is added and $\eta$ links are created to connect existing nodes by using a preferential attachment given by the degree of node $i$ $k_i$ and a probability $p_i = \frac{k_i}{\sum_j (k_j)}$ [101, 89, 92]. Forest Hub & Spoke corresponds to $n$ clusters of a Star configuration of a central node and $n-1$ adjacent nodes (Fig. 5.1-d). Hub & Spoke are applied to provide expansion of individual cloud instances in cases where high availability is required [47].

Nodes can fail given a failure probability $p_f$. Each node generates a uniform random number in $[0,1]$ by round. If the random number is lower than $p_f$, the node fails with all the mobile agents in this node. A failure in this case implies that a node crashes. The graph similarity metric in [57] and implemented in [82] was used to evaluate the changes in the structure of the failing network versus time in rounds. This metric provides the advantage of estimating the similarity of two graphs, generating a score in $[0,1]$. This
score is calculated solving the assignment problem between the nodes of the two graphs using a weight based on the degree of each node in the two networks and normalizing based on the maximum local degree of the two nodes compared. In this work, this value is written as percentage having a 100% of similarity in case of two isomorphic networks.

In the first experiments performed, a failure probability of $p_f = 0.001$ was defined for all the nodes and tested for the selected complex networks. Simulations were performed 30 times and executed until all nodes fail. Figure 5.2 presents box-plots of the network similarity versus time in rounds. Firstly, all similarity values are calculated and results go to zero for all the networks. Experiments were performed with a maximum round number of 20000, but stopped before in all cases. After round 1000, the network presents a median of similarity lower than 40% and in round 2000, a median similarity lower than 15% for all the networks. For the Community network and the Forest Hub & Spoke, after round 3000 no nodes are alive and for the Scale-free Network and Small World, no nodes are alive after round 5000.
5.3 Proposed Approach

The first part of the model rebuilds the network structure. To achieve this, it is assumed that each node is capable of sensing its immediate neighbours. This perception is compared with previous information about the neighbours so that a difference corresponds to a failure and a crashed node can be re-created by another one. Consider a network with 2 nodes and 1 agent -or even n agents-. If a node fails, agents located in this node also crash and if the topology data is not synchronised, the system will fail. For this reason, it is required that each node senses its neighbours in the proposed model. Considering a network of 3 nodes: \(a\ldots b\ldots c\), if nodes \(a\) and \(b\) fail, \(c\) can recreate \(b\) but not \(a\). To tackle this problem, it is necessary that \(c\) has information about node \(a\), then, \(c\) can recreate \(b\) and also send information about neighbours to \(b\). In this way, \(b\) can recreate \(a\) and nodes are able to learn in a local way recovering a determined network topology.

If \(b\) fails in \(a\ldots b\ldots c\), it is required that \(a\) or \(c\) recreate \(b\). In this case, the node with the minimum id is able to recreate a neighbour, thus, \(a\) recreates \(b\). Algorithm 6 presents how a node detects missing nodes based on the topology information obtained previously from external sources. Algorithm 7 presents the failure detection process as a local program in each static agent. The node with the minimum id tries to connect to the node before creating it. With algorithm 6 and the short time taken to recreate new nodes, no additional replicas of a node were obtained in the experiments performed. In algorithm 7 line 6 is important because a creator node shares its local topology information with the new node replica increasing the recovery capability.

### Algorithm 6 Failure Detection Program by Node

```
1: neighbours ← node.senseNeighbours();        ↦ sense adjacent neighbours
2: data ← node.getTopologyDataFromExtSource(); ↦ gets topology data obtained from external sources
3: if neighbours \(\cap\) data = ∅ then
4:   for each dif \(\in\) neighbours \(\cap\) data do        ↦ for each node id dif
5:     neighDif = getNeighboursFromLocalInfo(dif)        ↦ Obtain the neighbourhood of neighDif
6:     if getMinimumId(neighDif) = node.id then
7:       createNewNode(dif, neighbours)                   ↦ Send a message to connect dif with its other neighbours
8:     end for
9:   end if
10: end if
11: end for
12: end if
```

### Algorithm 7 Node Creation: CreateNewNode(nodeId, neighbours)

```
1: nod ← findNode(nodeId);        ↦ Try to connect with the other node before create
2: if nod \(\neq\) null then
3:   connect(nod)         ↦ Node reference already exists connecting instead create
4: else
5:   nod = createNode()    ↦ provide topology information to new node
6:   nod.setNetworkdata(neighbours)     ↦ Node connect with recently created new node
7:   connect(nod)
8: end if
```

Although an assumption of the problem is that the network topology is unknown, a second experiment was performed assuming that all the nodes have initial information
about the network topology and applying the algorithms 6 and 7 to recover the selected networks. Each experiment runs for 20000 rounds and it is repeated 30 times to collect information. As a result, median similarity for all networks is 100%. This means that the network is failing and recovering in time and all the steps of the healing algorithm are successful. The healing algorithm is able to tackle a failure probability of $p_f = 0.001$. This result is promising compared with the same networks without self-healing in Figure 5.2 when network similarity has a median value below 20% after round 2000.

With the previous results and algorithms 6 and 7 the problem addressed now is how to collect data about the network topology because this information is used to detect and recreate the missing nodes and their links. Two main strategies are studied in this chapter to collect and synchronise topology information: Multicast and Mobile Agents.

### 5.4 Data Collection Techniques

In this section, two techniques to collect the topology information in the nodes are compared and the proposed healing algorithm is applied. The first one is multicast based on the local communication between nodes and the second one is based on mobile agents applying a random exploration or a movement strategy inspired in swarm intelligence.

By multicast, each node senses and shares its local information about the topology with other nodes. In this way, nodes progressively acquire knowledge of a determined network structure. As assumed before, there are no errors in communications in terms of loss of messages or integrity problems. It is multicast, because messages about topology are sent by each node to its neighbours of degree one [66].

Mobile agents are defined using an agent program [78] and computation is performed in rounds. Algorithm [8] presents how each mobile agent calculates its next location using a movement algorithm (line 6). Lines 7 to 9 mean that in each round, each agent obtains a list of the identifiers of the adjacent nodes (line 7), information is stored and trunk for the last hops $n_{hops}$ (line 8) and finally each agent communicates with nodes using the $SendMsg$ function. $SendMsg$ has as parameters the type of message ($networkdata, freeresp, departing$); data to send depending on the type of message and finally the destination node. For the collection task, an agent sends the $networkdata$ message containing the topology information obtained to its current location (line 9). In line 12, an agent moves to its next location and increase its internal round number.

Each agent explores and collects data regarding the topology progressively applying a movement strategy (line 6 in Algorithm [8]). In [73], data collection with failure prone agents in complex networks was studied. As a main result, the speed of data collection in agents is related to a higher failure resistance and successful rates in data collection –at least one agent collects data from the selected complex networks before it fails–. Additionally, two motion algorithms were profiled in different complex networks: one is random exploration...
Algorithm 8 Mobile Agent program

1: Percept p
2: Action action
3: round ← 0
4: while Agent.status ≠ Fail do
5:   p ← environment.sense()
6:   newLocation ← movementAlgorithm(p)
7:   Agent.topologyInfo.add(Agent.getLocation(getLocalNeighbours()))
8:   Agent.trunkTopologyInfo(Agent.topologyInfo, n_hops)
9:   SendMsg(networkdata, Agent.topologyInfo, Agent.getLocation())
10:  SendMsg(freeresp, Agent.getLocation(), Agent.getPrevLocation())
11:  SendMsg(departing, newLocation, Agent.getLocation())
12:  Agent.move(newLocation)
13:  round ← round+1
14: end while

and the other is an algorithm based on swarm intelligence that showed to be promising also in bi-dimensional environments [71].

By random walks, an agent selects via a pseudo-random generator its next location from the set of nodes adjacent to the current node at each round. When following a pheromone-based movement, an agent chooses the next node based on its pheromone load to exploit unexplored nodes [73]. In [73], the pheromone movement algorithm is based on ACS (Ant Colony System) of [23]. The idea is to use pheromone as a trace that agents avoid. In this way, it is expected that an agent chooses the node with a minimum pheromone amount from the adjacent nodes in its current location to explore new locations. Passive evaporation and a random proportional rule allow re-exploration of nodes [13] and can be useful to re-explore routes previously explored by agents before they fail.

5.5 Agent replication algorithm

When a node fails, all the agents located in this node also crash also. To tackle with kind of failures, a model based on [74] that generates new replicas of agents is applied with some modifications. Initially, the replication Algorithm 9 of [74] intended to deal with link failures that cause agent crashes. This model approximates time difference—in rounds—between the round in which an agent departs from a node (departing message) and the arrival of an agent to its next destination (freeresp message). When an agent is close to depart, it sends the departing message including its next destination to its current node. In consequence, this node adds this agent to a collection called followedAgents. When a node receives the freeresp message, it computes the median and standard deviation of the number of rounds taken to arrive to a determined destination and deletes this agent from followedAgents. In algorithm [9] median and standard deviation of time expected between the two messages is obtained (lines 3 and 4). If the difference between the internal node age counter and the reception of the departing message exceeds a normal distribution (line 5), a new replica of an agent is created and removed from followedAgents. Although
CHAPTER 5. SELF-HEALING NETWORKS VIA SELF-ORGANISING MOBILE AGENTS

Table 5.1. Example of chain protocol and followedAgents vector by node

<table>
<thead>
<tr>
<th>Movement</th>
<th>followedAgents by node in format {hop, {id : agentId, dest : newLocation}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{1, {id : 100, dest : b}}</td>
</tr>
<tr>
<td>a → b</td>
<td>{2, {id : 100, dest : bc}}, {1, {id : 100, dest : c}}</td>
</tr>
<tr>
<td>a → b → c</td>
<td>{3, {id : 100, dest : bcd}}, {2, {id : 100, dest : cd}}, {1, {id : 100, dest : d}}</td>
</tr>
</tbody>
</table>

this model works well when communication failures occur, information about time-outs is stored locally in each node, hence making impossible for a node to tackle agent failures because local information is lost.

In this chapter, the idea of [74] was extended by providing a collection of replication algorithms in each node that create a chain of departing and freeresp messages. In this approach, each node stores a list of replication algorithms and, in consequence, a list of followedAgents by hop. This is implemented as a dictionary structure with a hop value as key. A maximum of nhopsChain indicates the maximum number of hops. Each time an agent moves from a node to another, a node retransmits the departing and freeresp messages to the previous visited nodes. In the same way, it computes algorithm 9 for each hop.

Algorithm 9 Replication algorithm of [74]

1: for each Agent a ∈ followedAgents do
2:     dest ← getDestination(a)  \rightarrow gets next location of agent a
3:     mRounds ← median(expectedMsgtime(dest), wsize)
4:     sRounds ← stdev(expectedMsgtime(dest), wsize)
5:     if nodeAge − getRoundDepartMsg(a) > mRounds + 3 * sRounds then
6:         createReplica(a)
7:     end if
8: end for

As an example, Table 5.1 presents a set of possible movements for one agent with id = 100 going from node a to node d in the network a—b—c—d and nhopsChain = 3. The first column of Table 5.1 shows a trace of the agent in an incremental way and the second column presents the followed agents vector in format \{hop, \{idAgent, destination\}\}. In the first row, it is observed that the initial followedAgents vector for node a is hop 1 and id = 100 and agent departs from node a to node b. Once agent 100 moves from a to b, a receives a freeresp message, computes the expectedMsgTime to b and deletes the reference in node a = \{1, \{id : 100, dest : b\}\}. By applying the chain, b receives the departing message from agent 100 and adds agent 100 in hop 1 to destination c, then resends the departing message to node a with destination bc. Node a adds agent 100 to followedAgents in hop 2 with destination bc, as shown in row 2. In row 3, once agent 100 arrives to node c, followedAgents vector is updated for c in hop 1, deleted from b in hop 1 and added in hop 2 and deleted from node a in hop 2 and added in hop 3 defining three references to agent 100 distributed in different nodes. Initial time-out for each node in the chain is duplicated by each hop: hop 1 has an initial time-out of 50 rounds, hop 2 had 100 rounds and so on.
### Table 5.2. Example of chain protocol and followedAgents by node

<table>
<thead>
<tr>
<th>Movement</th>
<th>FollowedAgents by node in format {hop, {id : agentId, dest : newLocation}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{1, {id : 100, dest : b}}</td>
</tr>
<tr>
<td>a → b</td>
<td>{2, {id : 100, dest : bc}}</td>
</tr>
<tr>
<td>a → b → c</td>
<td>{3, {id : 100, dest : bcb}}</td>
</tr>
<tr>
<td>a → b → c (loop)</td>
<td>{1, {id : 100, dest : a}}</td>
</tr>
<tr>
<td>a → b → c (noloop)</td>
<td>{1, {id : 100, dest : a}}</td>
</tr>
<tr>
<td>b</td>
<td>{1, {id : 100, dest : c}}</td>
</tr>
<tr>
<td>c</td>
<td>{1, {id : 100, dest : b}}</td>
</tr>
</tbody>
</table>

#### 5.5.1 Emergence of loops in agents replication

It is possible that an agent returns to a previous visited node making possible that a node have references of the same agents in different hops. As an example, Table 5.2 presents a set of possible movements for one agent with \(id = 100\) going from node a to b to c returning to b with a as next destination in the network a—b—c—d and \(nhopschain = 3\). Table 5.2 presents the followedAgents vector by each node. The first two rows are the same movement that in Table 5.1. When agent 100 is in node c and departing to b, it is observed that in row 3, c resends the departing message in hop 1 to node b producing the reference in hop 2 with destination cb and b sent this message to node a producing reference of agent 100 with destination bcb. Once agent 100 arrives and send the departing message to node b, In row 4 (loop) of table 5.2 it is observed that \(nhopschain = 3\), thus, reference of agent 100 is deleted in node a. Additionally, node b has now two references to agent 100, first in hop 1 indicating the next movement of agent b to node a, sent to node c generating the reference in node 2, and updating the reference of agent 100 in node b hop 3 with destination cba. Experiments are performed to see how does the chain replication protocol deal with this loops.

#### 5.5.2 Modification of algorithm to produce experiments with no loops

In topologies like forest hub & spoke it is common to have loops due to agents moving among outer vertices and returning to central hubs. This can generate additional overloads in terms of messages to be processed by the same node. To tackle this overloads, a modification in the replication algorithm named no loops is added. When a node returns to a previous visited location, all the references of this agent in the chain are deleted and restarted for this node in hop 1. In the previous example, when agent 100 moves in a—b—c—d from a to b to c to b and it is departing to a, Table 5.2 in row 4 shows that node b has two references of agent 100 in hops 1 and 3 respectively. By adding the modification with no loops, in row 5 (no loop) it is observed that when agent departs to b by the second time, node detects that another reference of this agent is present, thus it deletes this reference and the chain is restarted, being the only node with reference to agent 100.
5.6 Experiments and Results

Experiments are organised in multi-cast and with agents applying different configurations of the chain replication algorithm with and without loops. Purpose of experiments is to achieve an scenario that maintain the network structure and also agents if they are used. Each experiment is a combination of: a selected topology, 30 executions by experiment, a time limit of 20000 rounds and a failure probability \( p_f = 0.001 \).

Results for each experiment are profiled in terms of the similarity metric presented in section 5.2, number of agents in the scenarios where they are used, and overloads in terms of messages number. Number of messages is calculated by counting all the messages sent among nodes.

For all networks it was defined a \( nh_{\text{lopes}} = 5 \). Parameter \( nh_{\text{opschain}} \) was defined by each network separately to obtain a configuration that maintains living agents during the 20000 rounds of the experiments performed in the configuration with agent loops. For the small-world and the hub & spoke it is a \( nh_{\text{opschain}} = 4 \); a \( nh_{\text{opschain}} = 5 \) is defined for the community and the selected small-world network.

Figure 5.3 presents a collection of box-plots for the experiments performed. Selected networks are displayed in rows and network similarity, agents number and messages number are presented as columns. In this way, it is possible to analyse some behaviours common to all networks in experiments. The column Network Similarity in Figure 5.3 includes box-plots of network similarity sorted in three categories: multicast, loop and noloop. loop and noloop correspond to the chain algorithm for replicating agents and its modification for producing experiments without loops, respectively.

As shown in Fig 5.3 multicast presents the best similarity values for the experiments performed with a median of 100% in all the networks. This can happen because nodes are sharing its local information with its neighbours making possible a fast recovery. Additionally, a similarity closer to multicast is obtained by using agents. For 10 agents and loop there is a median of 97% for small-world; over 98%, for the selected community network; 97.5%, for scale-free; and 95%, for the forest hub & spoke network. It is observed that with more agents in the initial rounds, it is possible to synchronise network information in more nodes and, in consequence, obtaining a higher similarity as shown in the box-plots.

The second column in Figure 5.3 presents results in terms of number of agents. On the one hand, the experiments performed with loop option maintain agents for the 20000 rounds that the experiment takes place. On the other hand, median number of agents in the experiments with noloop is zero. Figure 5.4 presents the difference between simulation without (Fig. 5.4-a) and with chain loops (Figure 5.4-b). As main difference, it is observed that references of agents in nodes are lost without loops compared to the model with loops. A loop occurs because an agent returns to a previous visited node in the chain of replication, making possible for a node to have references of the same agent in different
hops. This compensates the fact that a failure in a node implies lost of followedAgents vector in this node, hence making impossible to recreate some new agents.

Better similarities are obtained in experiments with loops in the selected small-world, community and scale-free networks. This occurs because having agents increases the probability of synchronising network topology. However, in the forest hub & spoke, it is observed better similarity values with noloop. This can occur because during simulation the amount of messages generated by the chain protocol is carried out mainly by central hubs having more loops (and more replicas of agents) and producing delays in the implementation that imply additional time for a determined node to recreate another. Additionally, it is observed that this topology produces a higher amount of loops in the movement of agents.

Figure 5.3 presents in the column messages number that experiments with agents generate lesser messages than multicast and even less with noloop. It is also interesting that, in some cases, similarity is maintained so well even if agents are failing and even if agents are lost in time having a median of zero agents for noloop. This can be explained because before agents fail, they have time to collect information about the network topology and to disseminate this knowledge to nodes. Hence, if the network topology does not change, the initial knowledge provided to the nodes suffices to preserve the network. Additionally, it is assumed that the time needed for the agents to collect the knowledge is important (should be faster than the complete agent failure). In future work, ways can be explored to generate agents whenever the network topology changes purposefully (e.g. when a node is added, then it also generates agents to disseminate knowledge about it to the neighbouring nodes and vice versa).

5.6.1 Analysis of Mobile Agents Replication

Each time a new agent replica is created, this replica corresponds to a predecessor in the followedAgents vector. Location when the replicas are being generated was obtained from simulations and also the distributions of agent ids to analyse the behaviour of replicas and its predecessors. Firstly, the number of replicas looks to be related with the degree of the network.

Figure 5.5 presents a box-plot of number of replicas generated by each node sorted by nodes degree. As observed in Fig 5.5a, the four hops in the forest hub & spoke network have a higher number of replicas. In Figure 5.5b, it is observed that nodes with higher degree generate more replicas than nodes with less neighbours in a scale-free network where the preferential attachment rule rewards connections towards nodes with higher degree.
Figure 5.3. Summary of Results for the Selected Networks
Figure 5.4. Box-plots of round vs number of agents generated for the selected Small-world network with an initial population of 10 agents a) no loop b) with loop
As an additional result, information was collected about number of replicas with the same parent agent id and it was observed that some agents generate more replicas than others. By analysing each replica id and having into account that each agent departs from the same location in experiments, it was observed that ids are not the same and different replicas ids are distributed over all the experiments for each network. This means that movement of agents are in fact more important than initial location in terms of replication because a determined agent movement can generate loops.

5.6.2 Defining an initial round for failures

As shown in section 5.3, a network is learning about neighbours by copying its neighbour information from a node that recreates another previously crashed node. In consequence, the first rounds during the collection step are important to provide self-healing to the network. Several experiments were performed to determine what happens with the similarity metric if the network does not fail in the first 100, 250 and 1000 rounds with initial populations of 10, 20 and 30 agents.
Figure 5.6 presents a summary of similarities for the different networks with different initial rounds without failures. For the selected networks, it is observed that increasing the initial number of rounds without defining failures implies a higher similarity. After 100 rounds, for the small-world, a median similarity of 99.2 is obtained for 10 agents and 100%, for 20 and 30 agents and after 250 rounds. For the selected community network with 100 rounds without initial failures, a box-plot that looks like a line is obtained, which shows that similarity even with failures have values of 100%. For the scale-free network, after 100 rounds it is observed a median similarity of 99.5% for 10 agents and 100% for 20 and 30 agents. After 250 rounds, the median similarity is 100% and for 1000 rounds without failures, the box-plot looks like a line for a value of 100%. For the forest hub & spoke median similarity increases with time having a median of 97.5% and showing that failures in the central hubs impact the similarity metric. After 250 rounds, it is observed that the median similarity has a value of 100% for 10 agents, and 99.5% for 20 and 30 agents. For 1000 rounds without initial failures, the median similarity is 100%.

5.7 Summary

A self-healing model to recover a complex network structure is presented. The main difference with existing researches, is the creation of nodes additional to links recreation. The proposed mechanism recreates an existing structure based on an algorithm that rebuilds most of the entire network structure with some assumptions like nodes know the network topology and frequency of failures is lower than the time required to detect failures and heal the system. In cases when the network topology is unknown or changes dynamically, results show that it is possible to synchronise the network topology information by using mobile agents or multicast. Multicast has the advantage of providing network recovery just by defining messages among nodes, and mobile agents provide less overheads than multicast in terms of amount of messages. In case of using mobile agents, the time needed for the agents to collect information should be faster than the time taken for all agents to fail. Additionally, a chain algorithm to deal with mobile agent failures is proposed. By local interactions, it is possible to maintain references of mobile agents in the nodes and recover the network structure.
Figure 5.6. Consolidated similarities of experiments without failures for 100, 250 and 1000 rounds
Conclusions and Future Work

Solving the problem of failure detection and recovery in distributed systems is a difficult task because components can act independently from each other, are subject to local conditions and can fail unexpectedly. From this problem three sub-problems were identified: 1) determining distributed system behaviours to simulate, 2) defining different types of failures in distributed systems and 3) defining several self-healing processes for countering the identified failure types.

To solve these sub-problems, the main goal of the proposed work in this dissertation was to develop a self-healing model for distributed environments based on artificial life techniques. The proposed model comprises two collections of decentralised mechanisms: one is based on swarm intelligence and animal foraging to achieve robustness in a data collection and replication task in distributed environments and the other relies on local interactions and roles to achieve self-healing.

Main Contributions

The proposed self-healing model concentrates on four main points: Firstly, to model distributed system behaviours using a simulated multi-agent environment. Secondly, to evaluate robustness of mobile agents collecting and replicating data in different distributed environments. Thirdly, to propose techniques to replicate mobile agents that fail during communication. Finally, to design an algorithm to recover the topology of complex networks from node failures, as well to replicate agents that failed with these nodes.

Simulated multi-agent environment

This work proposed an a multi agent-based simulator for modelling distributed tasks. Two types of agents were defined: mobile agents and static agents. Mobile agents are stateful (i.e. have internal states), explore their environments (i.e. collect local data), perform local data-management tasks (e.g. exchange their collected data with data available at the visited locations) and communicate with other agents that they meet (e.g. exchange
collected data). Two types of distributed environments were considered (for simulation experimentation): continuous spaces and complex networks. Static agents define the nodes in the distributed environment (i.e. either a bi-dimensional surface or a complex network). They store pheromone, provide the mechanism of passive evaporation and store local information. Nodes in a network can additionally communicate with mobile agents or with other nodes and create new replicas of other agents in case of failures.

The main contribution of this simulator is to help in the design and evaluation of different decentralised data-management solutions, applicable to various distributed environments, with different characteristics (e.g. diverse tasks, resource constraints, performance requirements, or agent failure rates). The simulator collects metrics that enable statistic analysis, which are critical for profiling new agent designs. Initially, this allowed to determine the best agent exploration strategy for performing a distributed task in different types of terrains and network topologies, with different agent failure rates.

**Robustness of Mobile Agents Collecting and Replicating Data in Distributed Environments**

In particular, a solution for obtaining global information via a set of unreliable mobile agents is proposed. The communication mechanism allows neighbours to interchange information, enabling agents to acquire global data as a result of local interactions and cooperation. Limits were established in terms of robustness by determining the success of certain movement algorithms based on animal foraging, while increasing failure rates. Experimental results showed that mechanisms that favour exploration are more resistant to failures than those focused on increasing communication between agents.

For applications in real environments with mobile agents, the number of message exchanges must be considered, since it impacts energy consumption and battery life. The approaches with the best success rates also performed well in this respect. Scalability is also essential and the results obtained indicated linearity of solutions with increasing terrain sizes at the same population density.

Mechanisms like passive evaporation to avoid stagnation and defining a repulsive trace could be promising to achieve a faster data collection and, in consequence, a higher robustness. Results showed that the proposed carriers algorithm was the most robust in bi-dimensional environments (combined with Lévy walks) and in most of the selected complex networks.

In networks like Forest Hub-&-Spoke and Scale-free with addition of only one edge by step, a random exploration is faster than using carriers. These networks have the highest standard deviation in the betweenness centrality and results suggested it could be related with a larger round number for experiments using the proposed carriers algorithm. This could occur because nodes with high local degree are avoided by carriers, but these nodes require re-exploration in order to have all the data from a complex network.
On a global level, carriers collect information faster than random agents in all scenarios (even with failures). Small-worlds, Community Networks, Lattice and Scale-free with a degree of 4 (4-4-97) are faster than the other topologies analysed, for both algorithms (random and carriers). The Small-world is the fastest topology.

Obtained results provide key information on the characteristics of different decentralised data-collection algorithms, depending on their application context (e.g. network topology and failure rates). This, in turn, allows system designers to select the best option for their particular application and execution environment, covering a broad spectrum of applications like sensor networks, swarm robotics, server clusters, clouds, systems of systems and the Internet of Things (IoT).

**Self-healing Approaches**

Three monitoring and self-healing approaches were proposed based on messages propagation and local interactions. Firstly, an agent replication model to deal with communication failures (losing agents). Secondly, a model to recover nodes and links from failures maintaining a network structure. Finally, a model to deal with cascade failures in mobile agents when nodes fail.

**Replication of Mobile Agents that Fail while Exploring Complex Networks**

This work defined agents to notify nodes upon their departure, and upon their arrival at other nodes. This, in turn, enabled static node agents to estimate approximate time-outs for mobile agents, and hence to determine when mobile agents failed, in order to replicate them. Results show that the system learns to approximate communication time-outs and avoid under and over-replication making the median number of agents to tend to its initial value. Besides robustness, experimental results were successful in the data collection task even with high failure rates $p_f = 0.5$, while experiments with a $p_f \geq 0.1$ require more time in rounds to complete the data collection task.

Results showed that for a Small-world, Community and Hub-&-Spoke network, the memory consumption reduces as time-outs are learned. However, a high memory consumption and the highest round numbers were observed in the experiments with the Scale-free network where agents normally take more time to complete the task. This topology is characterized by difficult exploration because there are hubs with a higher degree of connection that slow-down agent exploration. However, even with this configuration, success is achieved in all the experiments performed with agent replication.

The obtained results can provide useful insights into the behaviour of data management applications with different execution environments, and help to design distributed protocols for data replication in server clouds, clusters and different kinds of distributed devices (IoT).
Self-healing Complex Networks with Self-organised Mobile Agents

Nodes are able to recover other nodes by comparing information about the topology of the network with its local perception of adjacent neighbours in each round. The main assumption to consider this kind of scenarios is that the speed of failure occurrence is lower than the speed of the healing process. As a main difference with other approaches in the literature reviewed, the proposed model is able to create new replicas of failing nodes instead of rewiring the network. This kind of model could be applied to maintain a network structure of virtual servers in a cloud.

One motivation for maintaining the network structure when self-healing the network was found in the experiments performed in the data-collection part of this work. The main factors that impact robustness in the defined data collection task are the movement algorithm and the network topology. In a determined application, the optimal agent exploration algorithm depends on the network topology. Hence, in order to maintain a good performance, it is recommended to keep the network structure when algorithms have been selected to be optimal for that structure.

By defining a failure detection and node creation algorithms, experimental results show a median similarity of 100% with a $p_f = 0.001$. This result contrasts with the same experiments without self-healing, when the median similarity is below 20% after round 2000. If the system does not fail in the first rounds (while the network information is synchronised), higher similarity is obtained in networks. Results also show that if a network does not fail before synchronising topology data, it presents a higher similarity over time.

In particular, the proposed technique to detect failures in nodes is able to rebuild all the selected complex network in experiments when nodes know the topology data for a $p_f = 0.001$. In consequence, if the topology is unknown, the problem can be solved by applying the data collection techniques outlined in this dissertation.

Replication of Mobile Agents when Nodes Fail

A model to deal with cascade failures in mobile agents when nodes fail was proposed. Analysing this model brought to the fore a relation between the node degree and the number of replicas created -the higher the degree, the more replicas will be created. In the experiments performed, loops in the movement of agents are key to have new replicas of agents during the 20000 rounds.

It was observed that even without loops, the agents are able to disseminate the knowledge about a determined network topology. In this case, the similarity is lower than with the chain protocol because the median number of agents in a determined configuration is zero.
Experimental results show that loops in the movement of the agents are key to maintain references of the same agent in different nodes, compensating the fact that references of mobile agents are lost in case of node failures.

**Future Work**

The proposed contributions and results obtained in this thesis open several perspectives for further research in this important field.

One direction for future work is to model and simulate new strategies for recovering from node failures and corrupt data collection. Our objective is to provide a theoretical and experimental base for developing real applications for different distributed environments – e.g. data collection and replication in clouds, clusters and the Internet of Things.

Findings on the carriers algorithm could be applied quite reliably in network coverage since this algorithm was the fastest in exploration and exploitation of bi-dimensional environments, even out-performing the Lévy Walks algorithm characterized by having super-diffusive patterns. When considering carriers in network coverage, some issues are related with the determination of the type of pheromone (situated or simulated) and new local features specific to network coverage, such as communications ratio.

Complementary to this work, some existing data integrity models can be studied in future research. In the experiments performed, communication channels have no failures in terms of data integrity or loss of data and information is complementary. Future research should be proposed to study different existing data integrity models and its compatibility with the proposed model.

If a network topology does not change, the initial knowledge provided to the nodes suffices to preserve (self-heal) the network. Future work should explore ways to only generate agents when the network topology changes, in order to re-explore the updated topology and re-inform the nodes about local structural changes.

Future work will intend to determine the limits of the node self-healing model in terms of failure probability. Finally, future work will deal with dynamic changes in the networks and its ability to recover from failures.
Bibliography


[34] Carsten Grabow, S. M. Hill, Stefan Grosskinsky, and Marc Timme, *Do small worlds synchronize fastest?*, EPL (Europhysics Letters) 90 (2010), no. 4, 48002.


