POLITECNICO DI MILANO

Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Swarm resiliency in collective decision making

Supervisor: Prof. Francesco Amigoni
Cosupervisor: Prof. Andreagiovanni Reina
Cosupervisor: Prof. James A.R. Marshall

M.Sc. Thesis by:
Francesco Canciani
id number 883658

Academic year 2018-2019

Swarm resiliency in collective decision making

Francesco Canciani

Abstract

A swarm is a self-organised group of agents that locally interact with each other and with the environment to perform a common task. Individual agents are typically simple, with limited capabilities and partial knowledge. Numerous studies investigated the problem of collective decision making in swarms which consists in the process of how a swarm reaches an agreement. However, little attention has been devoted to studying swarm resiliency. Resiliency from a swarm point of view is the ability of the swarm to achieve its goal under disturbance. In this study disturbance is considered as an attack characterised by malicious agents introduced in the swarm. In this thesis we analysed swarm resiliency in collective decision making processes. We focused on best-of-n decisions defined as the task of finding a collective agreement over the most favorable option among a set of alternatives. We investigated the resilience of four different behaviours taken from the literature through computational analysis. We focused on how different kinds of attacks affect the performance of the four behaviours. To do so, we implemented DeMaMAS, a novel multi-agent simulator for comparison of decentralised decision making strategies. DeMaMAS is modular and allowed us to quickly re-implement the studied literature behaviours through the composition of a sequence of reusable unitary modules. In fact, we described all the investigated behaviours through a common structure which consists in a sequence of phases. All behaviours are composed by the same phases which can have different implementations. A specific implementation of a phase is what we call module. Thanks to its modular architecture DeMaMAS allowed us to compare and quantify the resiliency provided by each module. A further advantage of DeMaMAS's modular architecture is the possibility to implement malicious agents by changing just a few modules. To test the resiliency, our tool is able to simulate heterogeneous swarms, in which agents composing the swarm can have different behaviours. We decided to implement three types of malicious agents called wishy washy, sect, and contrarian, respectively performing denial of service, slow down, and wrong addressing. We analysed the performance on the four selected literature behaviours exposed to the three investigated attacks in different conditions varying the number of options, the difficulty of the decision problem, and the percentage of attackers in the swarm. Our first result showed that in simple decision problems, characterised by a low number of options and easily distinguishable alternatives, listening less is the best strategy for being more resilient. However, this strategy does not work to make more difficult collective decisions. Our results also revealed the importance of two modules to increase the swarm resiliency: the majority module, in which an agent picks the most frequent option from its neighbourhood, and the cross-inhibition, a module inspired by the nest-site selection process in honeybees that allows the swarm to break decision-deadlocks in complex best-of-n problems. Thanks to the modularity of DeMaMAS, once we identified the most resilient module, we created a new behaviour combining them. The new behaviour showed the best resiliency performance in terms of both speed and accuracy. Finally, we validated the performance of the novel behaviour through a set of experiments on

physical devices. We implemented and tested two behaviours under a slow down attack in a swarm of 50 simple robots called Kilobots. The results from the robot experiments qualitatively matches the simulation results and confirms the benefits of the proposed behaviour. The outcomes of this thesis have been useful to support the analyses of a few research projects whose results will be disseminated through the following manuscript:

- Swarm resiliency in collective decision making
  (**F. Canciani**, M.S. Talamali, J.A.R. Marshall, A. Reina) in preparation for the ICRA 2019 Workshop on Resilient Robot Teams, Montreal, Canada, May 20-24, 2019.

Additionally, DeMaMAS, which I designed and implemented, is currently used for other research studies of which I will be acknowledged as a co-author.

KEYWORDS: resilience, collective decision making, swarm robotics

Swarm resiliency in collective decision making

Francesco Canciani

Sommario

Uno sciame è identificato come un gruppo di agenti auto-organizzati che interagiscono con l'ambiente e localmente l'uno con l'altro per conseguire un obiettivo comune. Gli agenti sono solitamente semplici, caratterizzati da capacità limitate e da una conoscenza parziale. Numerosi studi hanno esaminato il problema relativo al processo collettivo decisionale negli sciami, tuttavia sono stati fatti pochi studi riguardo la resilienza negli sciami, ossia l'abilità dello sciame di raggiungere il proprio obiettivo sotto l'influenza di un disturbo. In questo studio il disturbo viene considerato come un attacco caratterizzato da un agente (o più agenti) malevolo introdotto nello sciame. In questa tesi analizziamo la resilienza dello sciame in processi decisionali collettivi. La nostra attenzione si concentra nell'area riguardante il best-of-n definito come il compito di scegliere l'opzione migliore tra n alternative. Abbiamo studiato attraverso l'analisi computazionale la resilienza di quattro comportamenti proposti in letteratura. La nostra attenzione si è focalizzata su come diversi tipi di attacchi influenzano le prestazioni dei quattro comportamenti. Per far ciò abbiamo implementato DeMaMAS, un nuovo simulatore multi-agente per la comparazione di strategie di decisione collettiva decentralizzate. DeMaMAS è modulare e ci ha permesso di re-implementare velocemente i comportamenti studiati tramite la composizione di una sequenza riutilizzabile di moduli unitari. I comportamenti presi in considerazione possono essere descritti tramite una struttura comune, che consiste in una sequenza di fasi. Tutti i comportamenti selezionati sono composti dalle stesse fasi, che possono avere diverse implementazioni, chiamate moduli. Grazie alla sua architettura modulare DeMaMAS ci ha permesso di comparare e quantificare la resilienza fornita da ciascun modulo. Un altro importante vantaggio della struttura modulare di DeMaMAS è la possibilità di implementare agenti malevoli solamente cambiando alcuni moduli. Per testare la resilienza, la nostra applicazione è capace di simulare sciami eterogenei, in cui gli agenti che compongono lo sciame possono avere diversi comportamenti. Abbiamo implementato tre diversi tipi di agenti malevoli chiamati wishy wasy, contrarian e sect che eseguono rispettivamente denial of service, slow down e wrong addressing. Abbiamo analizzato le prestazioni dei quattro comportamenti scelti sotto l'influenza dei tre attaccanti in condizioni differenti, variando il numero delle opzioni, la difficoltà del processo decisionale e la percentuale di attaccanti nello sciame. Il nostro primo risultato ha mostrato che nei problemi decisionali semplici, caratterizzati da un basso numero di opzioni e alternative facilmente distinguibili, ascoltare meno è la strategia migliore per essere più resilienti. Tuttavia questa strategia non è più utilizzabile in problemi decisionali complessi. I nostri risultati hanno rivelato l'importanza di due moduli che incrementano la resilienza: il primo detto majority, in cui un agente sceglie l'opzione più frequente dal suo vicinato; il secondo detto cross-inhibition, ispirato dal processo di selezione di un nuovo sito per il nido dalle api, permette allo sciame di rompere le decisioni soggette a dead-lock in problemi best-of-n complessi. Grazie alla modularità di DeMaMAS, una volta identificati i moduli più resilienti, tramite l'utilizzo di questi ultimi abbiamo creato un nuovo comportamento, il quale ha migliorato la resilienza

sia in velocità di decisione che accuratezza. Infine, abbiamo validato le prestazioni del nuovo comportamento per mezzo di un insieme di esperimenti su dispositivi fisici. Abbiamo implementato e testato due diversi comportamenti sotto l'effetto di un attacco slow down in uno sciame di 50 robot semplici, chiamati Kilobots. Il risultato degli esperimenti sui robot combacia qualitativamente con i risultati delle simulazioni e conferma i benefici del comportamento proposto. L'esito della tesi è stato utile per il supporto di analisi di alcuni progetti di ricerca, che verranno diffusi tramite il seguente articolo:

- Swarm resiliency in collective decision making
  (**F. Canciani**, M.S. Talamali, J.A.R. Marshall, A. Reina) in preparazione per ICRA 2019 Workshop on Resilient Robot Teams, Montreal, Canada, Maggio 20-24, 2019.

In aggiunta, DeMaMAS, è correntemente usato per altri studi in cui verrò riconosciuto come co-autore.

Parole chiave: resilienza, processo collettivo decisionale, sciame di robot

Dedication

I dedicate this dissertation to my family, who have been my source of inspiration and gave me strength when I thought of giving up.

To one of the things that count the most in life: my dearest friends.

To my lifelong friends Oj, Piè, Trabu, and Giulia to support me during the past 20 years.

To my gazeria friends GasGas and Tommy that always push me to the limit, to the 100000 backflips done and to the future 900000 backside rodeo to come.

To Smith, Fari, Bonny, Colo, Albe, and Ivan to the many adventures had so far and the many others on the book.

To my classmate Lore, Dippi, Ame, Alle, Bruce, Adri, Agne and Dibe who supported me during the past 5 years.

To music, snowboarding, surfing and skating, some of the reasons that make this life worth living.

Acknowledgements

I want to thank my co-supervisor Giovanni for his cooperation, feedback, and friendship. In addition I would like to express my gratitude to Salah and Leon for the last minute favor and for the laughs. Last but not least I would like to thank professor Francesco Amigoni and James A.R. Marshall for giving me this amazing opportunity to work in the University of Sheffield.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
  - 1.1.1 Swarm . . . 1
  - 1.1.2 Collective decision making . . . 2
  - 1.1.3 Best-of-n . . . 2
  - 1.1.4 Resiliency in swarm . . . 3
- 1.2 Problem statement and contributions . . . 3
- 1.3 Structure of thesis . . . 5

2 State of the art **6**
- 2.1 Collective Decision Making (CDM) . . . 6
- 2.2 Swarm resiliency . . . 9

3 Multiagent simulator DeMaMAS **10**
- 3.1 Main components . . . 11
  - 3.1.1 Option . . . 11
  - 3.1.2 Totem . . . 12
  - 3.1.3 Agent . . . 12
  - 3.1.4 Message . . . 14
  - 3.1.5 Environment . . . 14
- 3.2 Architecture . . . 15
  - 3.2.1 Modulation self-social . . . 17
  - 3.2.2 Update model . . . 18
  - 3.2.3 Create message . . . 20
  - 3.2.4 Update option . . . 21
- 3.3 Documentation . . . 24
  - 3.3.1 Experiment . . . 25
  - 3.3.2 Model and update . . . 27
  - 3.3.3 Message . . . 29
  - 3.3.4 Movement . . . 29
  - 3.3.5 Agent and totem settings . . . 30
  - 3.3.6 Decay . . . 30
  - 3.3.7 Interaction function . . . 31
  - 3.3.8 File . . . 32
  - 3.3.9 Graphic and plot . . . 33

4 Attacks **34**
4.1 Classification . . . 35
4.1.1 Attack definition . . . 35
4.1.2 Attacks in best-of-n . . . 35
4.1.3 Attackers in best-of-n . . . 39
4.2 DeMaMAS and heterogeneous swarms . . . 44

5 Experiments **45**
5.1 Test description . . . 45
5.1.1 Literature behaviours parameters . . . 46
5.1.2 Attacker behaviours parameters . . . 48
5.1.3 General test parameters . . . 50
5.2 Analysis . . . 51
5.2.1 Metrics . . . 51
5.2.2 Data representation . . . 52
5.3 Results . . . 57
5.3.1 Contrarians perform slow down . . . 58
5.3.2 Sect executes wrong addressing . . . 59
5.3.3 Wishy Washy accomplishes a denial of service . . . 62
5.3.4 In simple problems, listening less is better . . . 64
5.3.5 Majority contribution . . . 68
5.3.6 Cross-inhibition contribution . . . 72

6 Attacks in real robots **78**
6.1 Kilobot . . . 79
6.1.1 Specifications . . . 80
6.2 ARK system . . . 81
6.3 Testing resilience in Kilobots . . . 82
6.4 Results . . . 85

7 Conclusions and future works **86**

List of Figures

1.1 Examples of swarm 1

3.1 DeMaMAS logo. 10
3.2 DeMaMAS graphical simulation 11
3.3 Agent status motif 13
3.4 Torus representation 14
3.5 Execution flow DeMaMAS 15
3.6 DeMaMAS detailed architecture 16
3.7 Modulation self-social motif 17
3.8 Update model motif 18
3.9 Direct switch motif 18
3.10 Cross-inhibition motif 19
3.11 Create message module motif 20
3.12 Update option motif 21
3.13 Discovery phase motif 21
3.14 Process social motif 22
3.15 Experiment parameters overview 25
3.16 Model and update parameters overview 27
3.17 Message parameters overview 29
3.18 Movement parameters overview 29
3.19 Agent and totem settings overview 30
3.20 Decay overview 30
3.21 Interaction function overview 31
3.22 File overview 32
3.23 Graphic and plot overview 33

4.1 Attack components motif 35
4.2 Goals of an attack motif 36
4.3 Attacker parameters motif 37
4.4 Attack vector motif 38
4.5 Attacker types motif 39
4.6 Error categories motif 40
4.7 Repeater categories motif 41
4.8 Jammers categories motif 42
4.9 Hostile environment categories 43
4.10 Killer motif. 43
4.11 Heterogeneous run 44

5.1 General used plots 53
5.2 Detailed used plots 55

5.3 Bar charts - slow down . . . 58
5.4 Bar charts - comparison sect and zealot . . . 59
5.5 Bar charts - sect . . . 60
5.6 Plots varying composition . . . 61
5.7 Bar charts - wishy washy . . . 62
5.8 Bar charts - wishy washy varying number of options . . . 63
5.9 Bar charts - 3-unanimity varying difficulty . . . 64
5.10 Bar charts - 3-unanimity varying number of options . . . 65
5.11 Bar charts - 3-unanimity varying composition . . . 66
5.12 Bar charts - 3-unanimity and sect . . . 67
5.13 Bar charts - DMVD and DMMD comparison . . . 68
5.14 Bar charts - DMVD and DMMD comparison (contrarian) . . . 69
5.15 Bar charts - CDCI and CDMCI comparison . . . 70
5.16 Bar charts - CDCI and CDMCI comparison (contrarian) . . . 71
5.17 Cross-inhibition motif . . . 72
5.18 Bar charts - CDCI and DMVD comparison . . . 73
5.19 Bar charts - CDMCI and DMMD comparison . . . 74
5.20 Bar charts - CDMCI comparison (contrarian) . . . 75
5.21 Bar charts - CDMCI comparison (wishy washy) . . . 76

6.1 Kilobot sketch. . . . 79
6.2 A Kilobot experiment captured with a video camera. . . . 83
6.3 Four frame of an experiment . . . 84
6.4 Box plots - Kilobots and ARGoS . . . 85

Chapter 1

Introduction

1.1 Context

Animals started to organise in large groups in the aim of increasing their chances of survival [15]. As a result, this activity increases their ability of foraging, investigating the environment and reproducing in addition to the ability of surviving. For a long time, different disciplines have been interested in social group dynamics such as biology, psychology and physics. In the last years thanks to a rapid development of information technologies, computer scientists have started to raise their interest in the field of swarm dynamics and swarm robotics [9]. In this work we focus in analysing swarm resiliency in collective decision making processes. For a better comprehension of the problem below we describe what we meant for swarm, collective decision making and resiliency. The first element that needs to be explained is the concept of swarm.

1.1.1 Swarm

Figure 1.1: There exist natural swarm such as (top row) School of fish (image by joakant on Pixabay), locustus swarm (photograph by Pierre Holtz), flock of starlings (photo by Wolstenholme images) and artificial swarm such as swarm of Colias robots (photo by University of Lincoln), swarm of kilobots (image by University of Sheffield) and a swarm of Alice micro-robots.

As showed in Figure 1.1 a swarm can be composed of animals such as honeybees, ants, starlings or can be composed of simple robots as Colias, Kilobots and Alice. We focus on swarm in artificial environment such as swarm of robots.

According to the definition of swarm robotics given by Dorigo and Sahin [4], the goal of swarm robotics is the study of large number of relatively simple robots, designed such that a desired collective behaviour emerges from the local interactions among robots and between the robots and the environment. The collective behaviour analysed in this thesis is called collective decision making and it is described in details in the next section.

1.1.2 Collective decision making

We define collective decision making as follow:

"Collective decision making is a process concerned with how groups reach an agreement"[2].

Collective decision making is one of the most important process for any artificial system or living organism. Animals constantly have the need of making decisions for surviving, reproducing, and foraging. In artificial systems such as robotic swarms, the interactions among robots, based on simple behavioural rules, requires to use a collective decision making process for solving a problem. The use of this process allows the swarm to accomplish tasks that without a correct collaboration cannot be done by single individuals. Extensive research has been carried out in the field of collective decision making as discussed in detail in Section 2.1. Due to the frequent use of this process in different scenarios a vast cases of problems are connected to it. The chosen category of problems for this thesis is called best-of-n and a formal definition is given in the following section.

1.1.3 Best-of-n

The best-of-n problem [44] related to self-organized systems can be described in summary as follow:

"The problem of finding a collective agreement over the most favourable option among a set of alternatives".

The term option is used to abstract domain-specific concepts that are related to particular application scenarios. The different options of the best-of-n problem are referenced by using natural numbers from 1 to n. An agent is defined as an individual of the swarm such as social insects and robots that is able to explore the environment and communicate with other agents. Each option i is characterised by a quality related to a function of one or more attributes of the environment, and the value associated to the quality is represented by a number $q_i \in [0, \infty)$. Agents are able to estimate the quality of the options. Considering a swarm of M agents, the problem is solved as soon as the swarm makes a collective decision for the best option, represented as the one characterised by the highest quality, among the set of alternatives $i \in \{1, \ldots, n\}$. A collective decision is represented by the achievement of a quorum threshold $\delta \in [0, 1]$ defined as the percentage of agents that have to favour the same option. The problem of best-of-n can be viewed in details here [44].

The last definition needed to understand the considered problem is the concept of swarm resiliency.

1.1.4 Resiliency in swarm

Swarms during their process of decision making, both in nature and artificial environment, can find some problems related to intruders. The scope of this thesis is to analyse if a swarm with particular social behaviour is able to resist to the attackers and reach the prefixed goal. This task is called resilience and it is analysed in a collective decision making perspective focused on best-of-n problems. From a swarm point of view resilience is defined as

"the ability of the swarm to achieve its goal under disturbance".

This kind of property needs to be tested under different conditions as the quality level of the options in the environment, the level of noise and the portion of the swarm affected by faults or attackers. This thesis tries to analyse various kind of attackers over several collective decision making behaviours. A detailed description of resiliency in swarm robotics it is explained in Section 2.2.

1.2 Problem statement and contributions

In the current literature lots of effort has been spent proposing collective decision making behaviours in swarm robotics, but less in analysing resilience in the same processes. This resilience analysis can help to identify which factors influence more the performance of a collective decision behaviour under disturbance. In our work we defined disturbance as an attack that can be described as malicious agents introduced in the swarm. In this thesis we investigate swarm resiliency in collective decision making, focused on the best-of-n decision problem.

We analysed the resilience of four different behaviours taken from the literature under the influence of three different types of attackers performing respectively a denial of service, slow down, and wrong addressing through computational analysis. Our analysis was possible thanks to the implementation of a computational tool called DeMaMAS. The modular structure of this tool allowed us to implement the four behaviours taken from the literature by combining a sequence of reusable unitary modules. All the three malicious agents can quickly be implemented in this tool by correctly setting some of the modules that compose a behaviour. Another important characteristic of DeMaMAS is the possibility to create heterogeneous swarms, in which agents in the swarm have different behaviours. Due to the fact that resilience can be tested under several different conditions, this works is focused in a restricted area of experiments characterised with the following list of **statement**:

1. The number of options n and their relative qualities q_{i} are finite and constant for the whole duration of the experiment. Each option i is referenced by using natural numbers from 1 to n. Options are localized in the environment.

2. We focus in problems with one and only one superior-quality option and $n-1$ inferior-quality distractor options. A difficulty level is fixed and constant for the whole duration of the experiments. The difficulty level is defined as the

ratio between the highest quality relative to the best option over the lowest quality associated to the worst one. With lower value we characterised simple problems (e.g., 0.4).

3. A quorum threshold δ defined as the percentage of agents that has to favour the same option to reach a decision is set to 0.8.
4. The number of agents M and the proportion between the number of agents and malicious agents is fixed and constant for the whole duration of the experiment.
5. Agents are free to move in a bi-dimensional space.
6. Agents are able to detect an option in the environment and noisily estimate its quality when they are in the proximity of the option location.
7. Agents have limited memory and can store only one option i and its quality q_{i}.
8. Agents are allowed to communicate with each others within a communication radius r.
9. Agents can exchange only their option with their neighbours.
10. Agents are not able to detect attackers.

The **contributions** of this thesis are:

1. Creation of a modular multi-agent simulator for comparison of decentralised decision-making strategies and testing resilience.
2. Definition of a taxonomy for attacks in collective decision making problems related to best-of-n.
3. Implementation, description and test of three different types of attacks each one characterised by a different goal (Denial Of Service, Wrong Addressing and Slow Down, for more details see Section 4.1.2).
4. From our experiments about resiliency we discovered that in simple problems, characterised with a low number of options (2) and a low difficulty level (0.4), listening less is the best strategy for being more resilient.
5. Through the comparison of literature methods under attacks, we discovered two important modules that help in increasing the resiliency of a behaviour. The first one called majority, in which the agent selects its new option according to the one shared by the majority of individuals in their neighbourhood and cross-inhibition, a module that allows the behaviour to break deadlocks in complex problems (high number of options and high number of difficulty).
6. With the previous contribution we were able to create a new behaviour that combines the two previous modules.
7. We validated the new behaviour through preliminary tests with 50 simple robots called Kilobots.

8. The outcomes of this thesis have been useful to support the analyses of a few research project whose results will be disseminated through the following manuscript:

    - Swarm resiliency in collective decision making
      (**F. Canciani**, M.S. Talamali, J.A.R. Marshall, A. Reina) in preparation for the ICRA 2019 Workshop on Resilient Robot Teams, Montreal, Canada, May 20-24, 2019.

    Additionally, DeMaMAS, which I designed and implemented, is currently used for other research studies of which I will be acknowledged as a co-author.

1.3 Structure of thesis

The thesis has the following structure:

- Chapter 2 introduces the **state of the art** in collective decision making and swarm resiliency fields.
- Chapter 3 explains in details the development, the usage and the powerful modularity characteristics of **DeMaMAS multi-agent simulator**.
- Chapter 4 provides a detailed **taxonomy of attacks** and define the **implemented attackers**.
- Chapter 5 describes how to set **experiments** with heterogeneous swarms in DeMaMAS and the main **results** obtained through simulations.
- Chapter 6 describes in details the main components used in **kilobots experiments** and the obtained results.
- Chapter 7 reports **conclusions**, main contributions and **future works** related to the usage of DeMaMAS.

Chapter 2

State of the art

This chapter provides an overview of the literature in the fields of collective decision making and swarm resiliency.

2.1 Collective Decision Making (CDM)

Inspired by natural swarm approaches in problem solving, in the past years, the scientific community increased its interest for collective decision making [2]. Collective decision making processes are useful in a lot of different problems, below we present a selection of some interesting cases.

- *Aggregation and clustering*: the challenge to position each component of the swarm close to each others in one location that is unspecified. Some solutions are inspired by different natural flock such as ladybugs [17] and young honeybees [38]. Clustering is a subsection of aggregation tasks that includes collecting objects due to particular feature (e.g., colour, size or age) in specific locations, one example of this behaviour is related to the termite collection of resources. From a collective decision making point of view the swarm needs to decide where to aggregate and how to aggregate the objects. In this context some of the most important works are the papers by Schmickl et al. [33] [34] and Kernbach et al. [13], about the BEECLUST algorithm.

- *Collective construction*: a collective construction goal is defined as a construction of buildings or any other architectural objects. From a CDM perspective the swarm needs to decide where to build the artifact. Possible solutions are inspired by how social insects, like wasps, ants and termites decide where and how build their own home [47]. Relative to collective construction some interesting works are proposed by Theraulaz and Bonabeau [41] [42], in which it is proposed a building behaviour modelled as cellular automata.

- *Collective motion and flocking*: the task of collective motion is to synchronise the movement of the whole swarm. This behaviour is observed in flocks of birds. From a collective decision making stand point at any time all the individuals composing the swarm need to decide the direction and the speed of the flight. An important model of flocking is described and proposed by Reynolds [27]. In this work three basic rules for simulating a bird flock are suggested: (1) Alignment: adaptation of direction based to the one of your

neighbours. (2) Cohesion: stay close to your neighbours. (3) Separation: avoid collision, trying to stay not to close to your neighbours.

Lots of effort has been spent on the previous tasks and various interesting models have been proposed. In order to check the reliability of the proposed algorithms and models, scientists decided to use simulators and swarms of robots as real-world models. A lot of the previous and following described works use simulation and swarms of robots.

Related to the scope of this thesis on the best-of-n category, defined as the set of problems of finding a collective agreement over the most favourable option among a set of alternatives, different models can be used for solving these tasks and some interesting cases are discussed in details. Below we discuss in details only models considering discrete options and assume a spatial environment. For completeness we also briefly report models using different assumptions.

Models adopting a non-spatial environment also called urn models are explained by Hamann [8] [9] and by Eigen and Winkler [5].

In the paper of Hegselmann and Krause [10] and Kuramoto [16] there are models handling a continuum option assumption.

The most relevant works related to the scope of the thesis are reported and described in the following list.

- The *Voter model* is a simple but powerful model for collective decision making problems proposed by Clifford et al. [3]. It has very useful properties adopted in several other algorithms. The following definition describes the behaviour of the Voter model:

  an agent 'i' considers the options of its neighbourhood and picks one of them at random. The agent 'i' switches to the selected option if and only if its previous option is different from the new one.

  This behaviour that seems apparently counterproductive helps the swarm to converge to the correct decision. A presence of a majority in the flock permits to converge to the best option, represented as the one with the highest quality, because it is more likely for the majority-option to be picked by a random selection [3].

- *Majority*: The behaviour of the majority model is explained as follows:

  "the agent selects its new option to the one shared by the majority of individuals in their neighbourhood"[7].

  In case of a tie, the new option is chosen at random from the most m-frequent picked from its neighbourhood. There are several different versions of the majority model, two of the most significant are the following:

  - Simple-Majority: an agent i, after checking its neighbourhood, counts the occurrence m_j of each option O_j and switches to the most frequent one O_k where $k = argmax\ m_k$.
  - Quorum-Majority: instead using a most frequent approach in this variant a quorum that needs to be satisfied in order to pick the new option is added[18].

- *DMMD* stands for Direct Modulation of Majority-based Decisions, it is a model developed by Valentini et al. [45]. This model combines the behaviour of the majority with the modulation of positive feedback, the individuals advertise their options for a time proportional to the assessed quality. In the analysis reported in their paper [45] it is demonstrated that DMMD is faster but less accurate than the voter model.

- *DMVD* stands for Direct Modulation of Voter-based Decisions [46], it is characterised by a combination of voter model and modulation of positive feedback as described in the previous model.

- *K-unanimity*: The behaviour of k-unanimity is summarised as follow:

  "an individual selects its new option O_j if and only if all the last k-observations in its neighborhood are identical to O_j, otherwise it does not select any"[32].

  The "k" parameter represents the number of neighbours that are evaluated for choosing the new option.

- *CDCI* stands for Collective Decision through Cross-Inhibition [26], this model is inspired by experimental and theoretical studies of the nest-site selection behaviour observed in honeybee swarms (Apis mellifera) [35] [26] [25]. The main idea behind that method is to break deadlocks thanks to a cross-inhibition behaviour, like the one that bees use during their decision making process for the selection of the best nest-site. This strategy copies the option of a random neighbour and if its status is uncommitted or the copied option is equal to its past one it switches to the new option, otherwise it flips uncommitted.

Another interesting model that is not used for our scope is the Sznajd model [37], also called USDF ("Unite we Stand Divided we Fall"). Its principle is described as follow: *The option of an individual 'i' can assume only two values* $O_i = 1$ *or* $O_i = -1$ *and only two neighbours. In each time step two individuals options* O_{i+1} *and* O_i *are picked randomly. The opinion of the two neighbors* O_{i-1} *and* O_{i+2} *are updated as follow:*

- *If* $O_iO_{i+1} = 1$ *then* $O_{i-1} = O_{i+2} = O_i$
- *If* $O_iO_{i+1} = -1$ *then* $O_{i-1} = O_{i+1}$ *and* $O_{i+2} = O_i$

This model can end in a stalemate. A stalemates is a situation that we want to avoid in our models, because in most of CDM problems one of the main goal is to reach consensus that permits to take a decision.

Also, in natural processes some deficiencies have been discovered in collective decision making approaches. One of the most representative problems is performed by the pheromone trails, released by a group of ants while searching for resources. It can happen that the trails form a loop which traps them. The swarm is not able to detect the situation and keep circling until they eventually die. After noticing several issues, the scientific community starts to propose new solutions. Another problem related to swarms and collective decision making is called swarm resiliency.

2.2 Swarm resiliency

An individual is resilient if it is able to resist disturbance. According to the definition of resiliency provided in Section 1.1.4, from a collective decision making point of view, a swarm is resilient if it is able to achieve its prefixed goal also adding disturbances, such as intruders. An intruder is able to manipulate and alter the behaviour of the swarm. Several works investigated swarm robotics relative to fault detection [14] [40] and the concept of security itself [12] [11] but limited effort has been devoted to swarm resilience.

Most studies about swarm resilience assume that is not possible for the agents to detect an intruder. It is also assumed that individuals composing the swarm are connected by a static network topology and they are not allowed to move freely. The intruder is characterised by a physical failure, due to a component malfunctioning or depletion of energy, or malicious behaviour. Another assumption is that the number of adversaries is not known a priori.

An example of work using the previous assumptions is the paper by Hyongju et al. [21]. In this paper a resilient version of the classical rendezvous problem is described with a strictly relation to the Byzantine problem in distributed computing [43]. The proposed algorithm is called DRY-Sync and is based on the computation of interior Tverber points for each fault-free robot in the swarm. The strength of the previous algorithm is supported by accurate analysis and simulations.

Kelsey et al. [31] assume a time-varying network instead of a statistic one and describe an accurate analysis of an alternative version of Weighted-Mean Subsequence-Reduced (W-MSR) [48]. The main contribution is related to the addition of a control policy that guarantees the connectivity of the agent above a given critical threshold. Thanks to the combination of the control policy and the W-MSR algorithm a resilient flocking behaviour is preserved.

An interesting paper assuming time-varying networks with a fixed number of intruders F is the one by Saldana et al. [29]. The proposed method relies on network typologies satisfying the well-known property called (2F + 1)-robustness. The novelty in the algorithm is given by the addition of a sliding window approach extending the classical W-MSR. The main difference with the classical W-MSR is that the agent during the time interval of the window phase takes into account all values received by its neighbours. This approach guarantees a better resilience than the classical one.

Finally, Strobel et al. [36] present a blockchain solution for managing byzantine attacker in a swarm robotics collective decision making scenario. This approach increases considerably the resiliency thanks to the establishment of a secure swarm coordination mechanisms able to identify and exclude intruders.

No studies about swarm resiliency are conducted assuming a fixed number of intruders, a spatial bidimensional model in which agents are allowed to move freely and no detection of adversaries is performed. One of the main contributions of this thesis is to test and explore resiliency in best-of-n problems under the previous assumptions.

Chapter 3

Multiagent simulator DeMaMAS

Figure 3.1: DeMaMAS logo.

This chapter presents the multi-agent simulator, its architecture and usage.

The purpose of the project is to provide a modular tool that is simple to use and extendable for future work in the field of decentralised collective decision making problems. DeMaMAS is the name of the application, an acronym that stands for Decision Making Multi-Agent Simulator, whose goal is the comparison of decentralised collective decision-making strategies. The logo used to represent the tool is shown in Figure 3.1. The application is accessible through the GitHub platform on the following link: https://github.com/DiODeProject/DeMaMAS

3.1 Main components

In this section we present the principal components that constitute the multi-agent simulator. A simulation is composed basically of a finite number of agents M, options n, totems T, an environment (torus), a set of messages, a time step limit $\bar{ts} \in [0, \infty)$ and a quorum value $\delta \in [0, 1]$. As described above, the main goal of a simulation is to try to solve a best-of-n problem. A screen shot of simulation can be seen in Figure 3.2. The parameters (50 in total) that have to be set in order to run a simulation are discussed in details in Section 3.3. An execution ends when the quorum value δ or the time step limit $\bar{ts}$ is reached. Each principal component involved in a simulation is composed of a set of parameters, that represent its own characteristics and define its behaviour. In the following five sections we explain in details how each principal component works and its main characteristics.

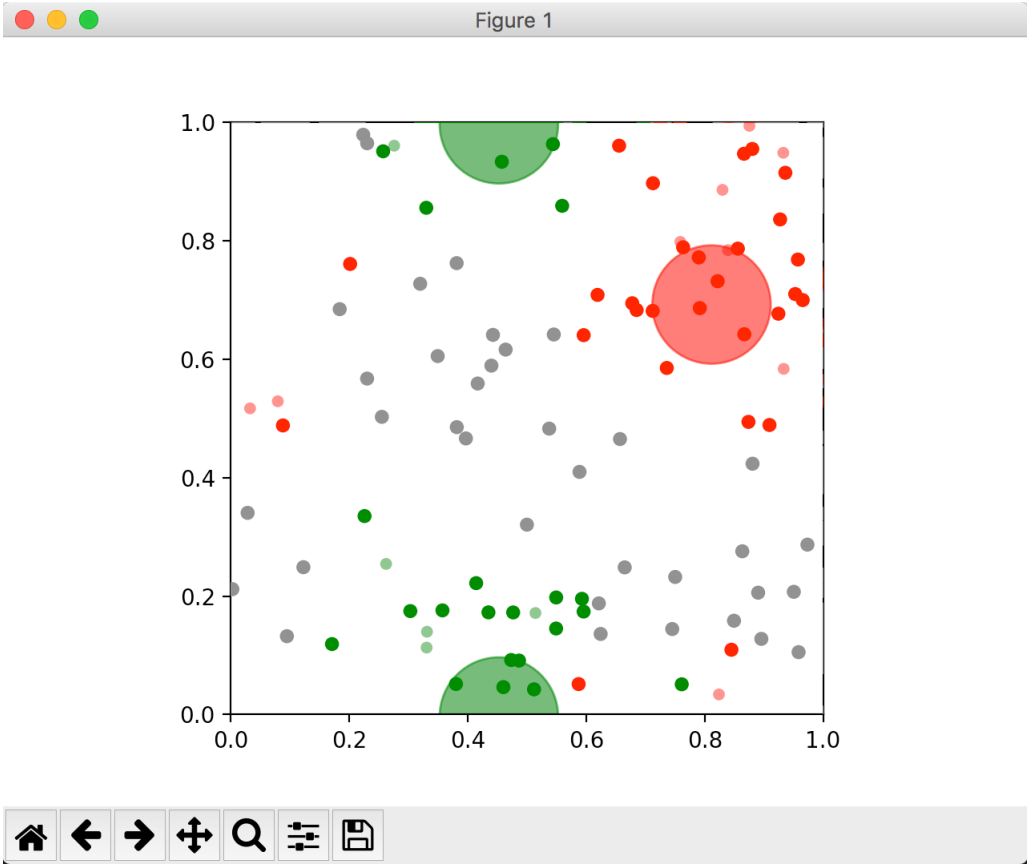

Figure 3.2: A graphical execution of DeMaMAS with 100 agents, 2 different options represented with the 2 different totems (red and green) and a quorum set to 0.8.

3.1.1 Option

As said in Section 1.1.3 we focus on the best-of-n task, described as the problem of finding a collective agreement over the most favorable option among a set of alternatives. We use the term option to abstract domain-specific concepts that are related to particular application scenarios. In our simulator each different option i is characterised by an unique identifier id and a quality q_{i}. The identifier of the option is an integer number varying from 1 to the number of options n in the environment. Each quality q_{i} is characterised by a float value $q_{\text{i}} \geq 0$.

3.1.2 Totem

The totem is another main components of a simulation. A totem is an instance of an option i and it is visualised as a big dot with a colour related to the corresponding option i as showed in Figure 3.2. The main attributes of those components are listed below:

- *id*: it is a unique identifier (integer) that identifies a totem (static).
- *message*: it is a class that contains information that are shared through agents (static).
- *option*: it is an integer that represents the option i related to the totem (static).
- *quality*: it is an integer that represents the quality q_{i} related to the option i of the totem (static).
- *position*: it is a parameter that represents the x and y coordinate that describes the position of a totem (static).
- *radius*: it is a float value r that represents the communication radius (static).

The main difference between a totem and an agent is characterised by a static status for the totem and a dynamic one for the agent. A totem never changes its option i, quality q_{i} , and position for the entire execution of the simulation, while an agent can change its option i, estimated quality $\tilde{q}_{\mathrm{i}}$ and position each time step.

3.1.3 Agent

The agent is the main component of our system, in our graphical simulation in Figure 3.2 it is represented as a little dot with a colour that represents its current option. Agents are allowed to communicate with each others within a communication range with radius r, to move freely in the space using a random walk, and to detect an option i in the environment and estimate its relative quality $\tilde{q}_{\mathrm{i}}$. The estimation introduces a noise factor ζ taken from a normal distribution. They have a finite memory and can store only one option i per time with its relative estimated quality $\tilde{q}_{\mathrm{i}}$. Agents cannot collide with other agents, because their bodies are not material. In our case of study agents can share only their option i without its estimated quality $\tilde{q}_{\mathrm{i}}$. Its principal behaviour is represented by a list of parameters, each one of them can be static or dynamic. A static parameter never changes its value during the whole execution of the simulation, otherwise is dynamic. The most important parameters are listed and described below:

- *id*: it is a unique identifier (integer) that identifies an agent (static).
- *option*: it is an integer from 1 to n that represents the option of the agent (dynamic).
- *message*: it is a class that contains information that are shared through agents (dynamic).
- *neighbours*: it is a list composed by all the agents that is in the radius range of the agent (dynamic).

- *position*: it is a parameter that represents the x and y coordinates that describe the position of an agent, this value can change through the execution of the simulation (dynamic).
- *quality*: it is a float that represents the estimated quality perceived by the agent, this value can change during the simulation (dynamic).
- *radius*: it is a float value that represents the communication radius of an agent (static).
- *create message*: it is a string that represents the method used for communication purpose (dynamic).
- *update algorithm*: it is a parameter that represents the rule that computes how to select the new option for the agent (static).
- *update model*: it is a parameter that represents how to compute the update of the new option (static).

Each agent of the swarm can be in four different states, as showed in Figure 3.3.

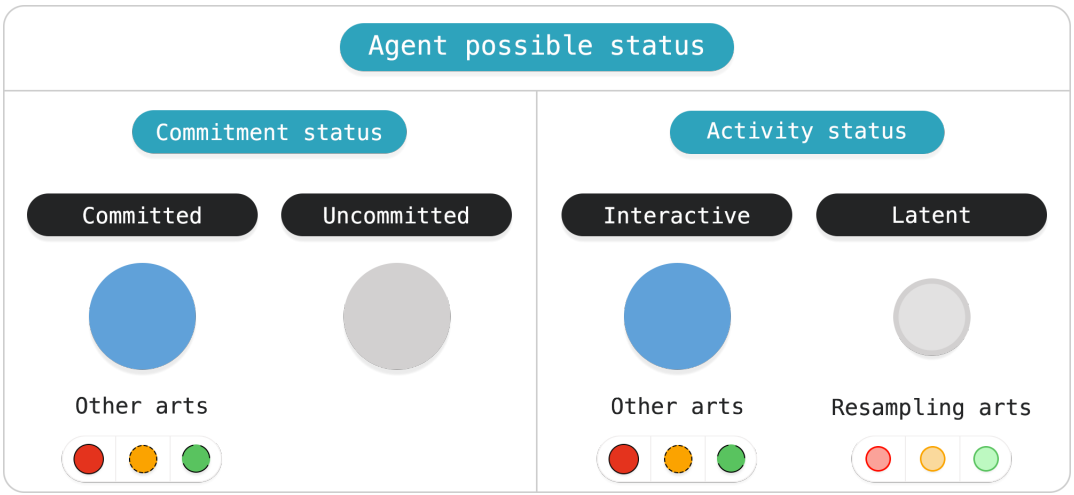

Figure 3.3: Possible agent configurations motifs, each one of the different circles represents how an agent in the relative status is represented in the simulation.

- *Commitment status*: describes the level of information that an agent has.
  - *Committed*: The agent has an option i with its estimated quality $\tilde{q}_i$ and the colour of the agent represent its current option.
  - *Uncommitted*: The agent does not have an option and a quality. From a graphical point of view it is represented with a grey little dot.
- *Activity status*: describes which activity an agent can perform.
  - *Interactive*: The agent is allowed to communicate with others. It can share its own option i and can use the information given by its neighbours. It has the same graphical representation of the committed status.

  - *Latent*: The agent is not allowed to share and gather any kind of information with its neighbours and with the environment. One example of a latent state is when an agent is in a resampling state.
    * *Resampling*: the agent has an option i, received by one of its neighbours, but needs to estimate the relative quality $\tilde{q}_{\mathrm{i}}$. To estimate $\tilde{q}_{\mathrm{i}}$ it requires to go to the related totem where the option was previously sampled by its neighbours. The colour showed in the graphical simulation represents the current option i, but it has a smaller dimension and it is darker.

3.1.4 Message

The message represents the communication packet that can be shared among agents and totems. A message is defined as a class in the DeMaMAS simulator and its main parameters are:

- *id*: it is a integer representing the identifier of the sender of the message.
- *option*: it is a integer that describes the option of the sender.
- *totemPos*: it is a list of length two that displays the x and y position of the totem where the option came from.
- *visible*: it is a Boolean set to False if the sender is in latent status, otherwise is set to True.

3.1.5 Environment

The simulation runs over an environment characterised by a single parameter called environment size that describes the dimension of the x and y axes considered identical. The environment is a torus. A torus is defined as a surface or solid formed by rotating a closed curve, especially a circle, about a line which lies in the same plane but does not intersect it. An example of torus can be viewed in Figure 3.4. In DeMaMAS the 2D visual representation of the torus allows agents and totem to cross the boundaries.

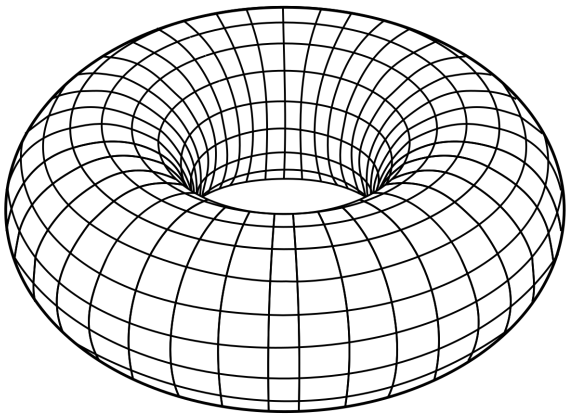

Figure 3.4: Torus representation in 3D, public domain via Wikimedia Commons.

3.2 Architecture

The novel modular structure of DeMaMAS is realised after a deep study and comprehension of different behaviours relative to decentralised collective decision making problems. After a long investigation we noticed that all the investigated behaviours have a common structure which consists in a sequence of phases, represented in blue in Figure 3.5. All behaviours are composed of the same phases which can have different implementations. A specific implementation of a phase is what we called module. More generally we divided the execution of a behaviour in three different sections called respectively sensing, thinking and acting as showed in black in Figure 3.5. In the sensing section, the agent perceives the options and receives the information from its neighbours. During the thinking section an agent is able to select the new option and update it. It is composed of four different phases called discovery, process social, modulation self-social and update model. In the acting section the individual creates the message with its new information and spreads it, this section is composed by the create message phase.

Thanks to its modularity DeMaMAS allowed us to quickly re-implement the investigated behaviours through the composition of a sequence of reusable unitary modules.

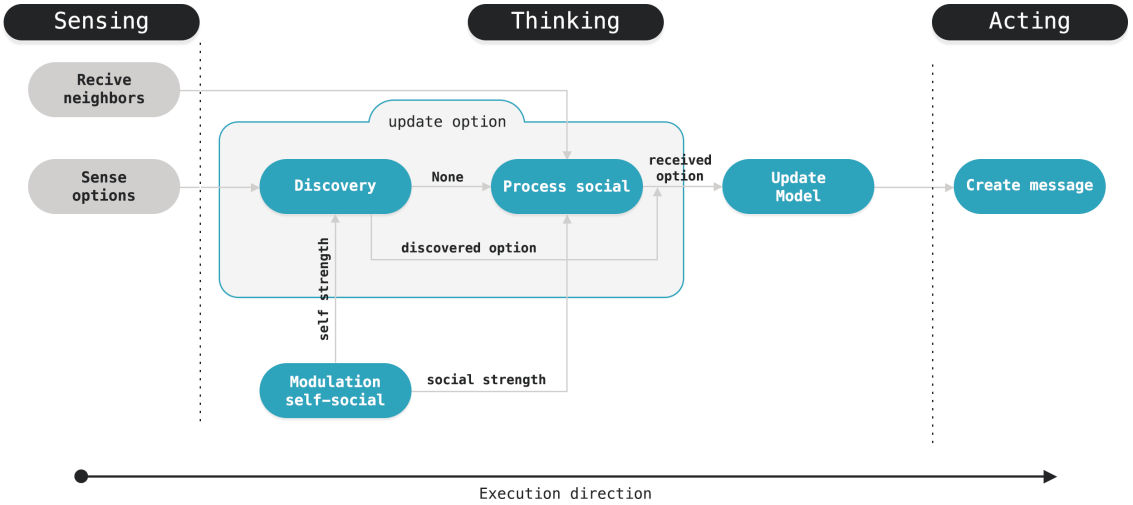

Figure 3.5: Execution flow graphical representation, in black we represent the three different sections and in light blue the different phases.

In the following sections we describe in details each one of the previous phases and modules depicted in Figure 3.5 and with more details in Figure 3.6, in which we show all the already implemented modules. For each one of these phases we describe the already implemented modules.

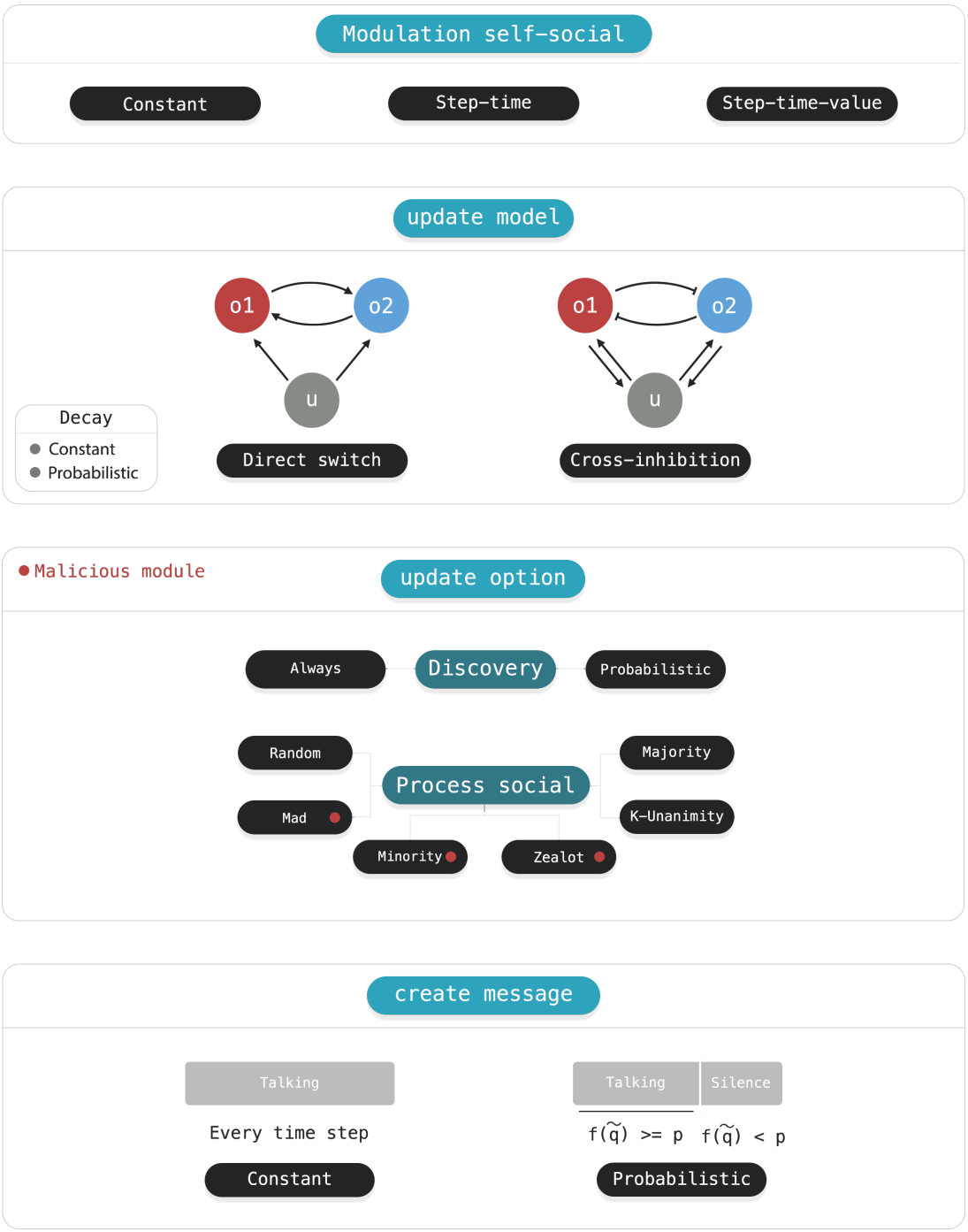

Figure 3.6: DeMaMAS detailed architecture, with the light blue rounded rectangles we represented the phases and with black rounded rectangles the implemented modules.

3.2.1 Modulation self-social

The phase called modulation self-social allowed the user to update the agent discovery and social strength. Each agent has two values called social-strength ($\rho \in [0, 1]$) and self-strength ($\varrho \in [0, 1]$), which allow the agent to modulate the balance respectively in process social and discovery. If ρ is set to 0 the agent is not able to perform process social, while if ϱ is set to 0 the agent is not able to perform discovery. If $\rho \in (0, 1)$ or $\varrho \in (0, 1)$ it means respectively that the social-strength and self-strength modulate the probability of applying process social and discovery. If ρ and ϱ are set equal to 1, it means that they do not influence how the agent performs process social and discovery phases. In DeMaMAS as showed in Figure 3.7 we already implemented three different module:

- Constant: The agent never updates its social-strength value ρ and its self-strength value ϱ.
- Step-time: The agent updates its social ρ and self ϱ strength after a fixed time step $\hat{t}$.
- Step-time-value: The agent updates its social ρ and self ϱ strength according to a function f($\tilde{q}$) defined as follow:

$$\mathrm{f}(\tilde{q}) = \frac{\hat{t} * Q}{\tilde{q}}$$

  Where $Q \in [0, \infty)$ is the maximum quality, $\hat{t}$ is a fixed time step and $\tilde{q}$ is the estimated quality. And it is used as follow:

$$\rho, \varrho = \begin{cases} \text{update}, & \text{if } ts \geq \mathrm{f}(\tilde{q}) \\ \text{not update}, & \text{otherwise} \end{cases}$$

Where ts is the current time step of the simulation.

Figure 3.7: Modulation self-social graph, with the light blue rounded rectangle we represented the phase and in black rounded rectangles the implemented modules.

3.2.2 Update model

In this section it is explained in details how the update module works and which modules are already implemented.

The main goal of that phase is managing the options received by the agent and updates its own option according to the selected module. In the update model two different modules called direct switch and cross inhibition are already implemented, as showed in Figure 3.8. They are explained in details in the relative subsections below.

In this phase in addition to the previous modules it is already implemented a submodule called decay that allowed the agent to forgive its option i and its estimated quality $\tilde{q}_i$.

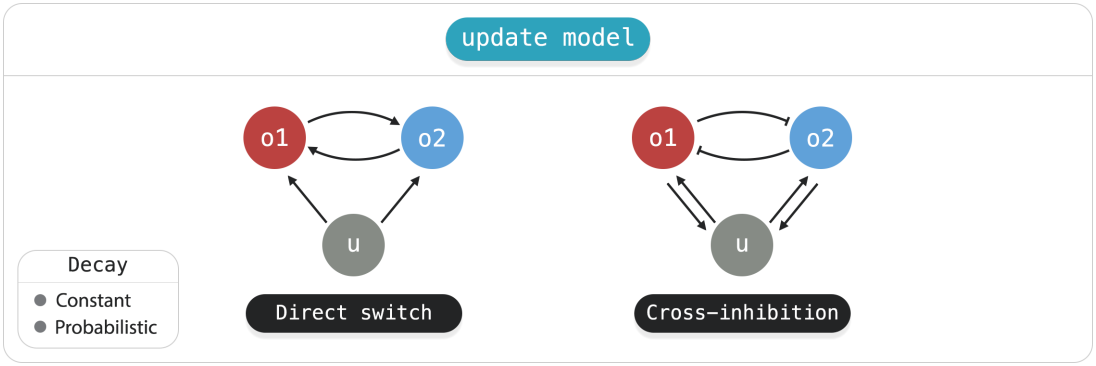

Figure 3.8: Update model phase motif, with the light blue rounded rectangle we represented the phase and in black rounded rectangles the implemented modules.

Direct switch

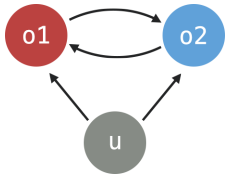

Figure 3.9: The motif is a graphical representation of how the direct switch module works. In this figure O_1 represents an option with identifier 1 represented in red and O_2 with identifier 2 represented in blue. The gray circle identified with the letter u depicts the uncommitted status.

The direct switch module allows the agent to change its own option with the selected new one j (that is different to its previous one i) without pass through the uncommitted phase. An explanation figure is shown in Figure 3.9. When an agent is in an uncommitted phase as soon as it selects a new option this module granted the agent to switch directly to it. If the agent already has a own option i and selects a new one $j \neq i$ it is allowed to substitute its option from i to j.

Cross-inhibition

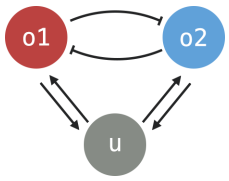

Figure 3.10: The motif is a graphical representation of how cross inhibition module works. In this figure O_1 represents an option with identifier 1 represented in red and O_2 with identifier 2 represented in blue. The gray circle identified with the letter u depicts the uncommitted status.

The cross inhibition module is inspired by the nest-site selection process in honeybees that allows the swarm to break decision-deadlocks in complex best-of-n problems. An agent that uses this update module cannot switch directly from its option i from the selected new one $j \neq i$ without pass through the uncommitted phase. When the agent pass through the uncommitted phase it loses both option i and its estimated quality $\tilde{q}_\mathrm{i}$. In case the agent is in an uncommitted phase it is allowed to switch directly to the new selected option j.

Decay

The decay sub-module allows the agent, at the end of the model update phase, to forgive the values of its current option i and the estimated quality $\tilde{q}_\mathrm{i}$, and go back to the uncommitted state, in which the agent has no option, if and only if either discovery and process social are not performed. In DeMaMAS two different modules concerning decay that use a parameter of the agent called decay-strength $\gamma \in [0, 1]$ are already implemented:

- *Constant*: the agent performs decay according to the following rule:

$$\begin{cases} \text{perform decay}, & \text{if } \gamma > U(0,1) \\ \text{not perform decay}, & \text{otherwise} \end{cases}$$

  Where $U(0, 1)$ identifies a function that returns a random value from uniform distribution with variance equal to 1 and average equal to 0.

- *Probabilistic*: the agent performs decay according to the following rule:

$$\begin{cases} \text{perform decay}, & \text{if } (\gamma \ / \ \tilde{q}_\mathrm{i}) > U(0,1) \\ \text{not perform decay}, & \text{otherwise} \end{cases}$$

  Where $U(0, 1)$ identifies a function that returns a random value from uniform distribution with variance equal to 1 and average equal to 0, and $\tilde{q}_\mathrm{i}$ is the estimated quality related to the option i of the agent.

3.2.3 Create message

In this paragraph we discuss in details how the create message module works and which modules are already implemented. The create message phase is responsible for the creation and spread of the message containing the information gathered by each agent. Up to now we implemented a constant and a probabilistic modules. In the following two sections we discuss in details how the constant and probabilistic modules work.

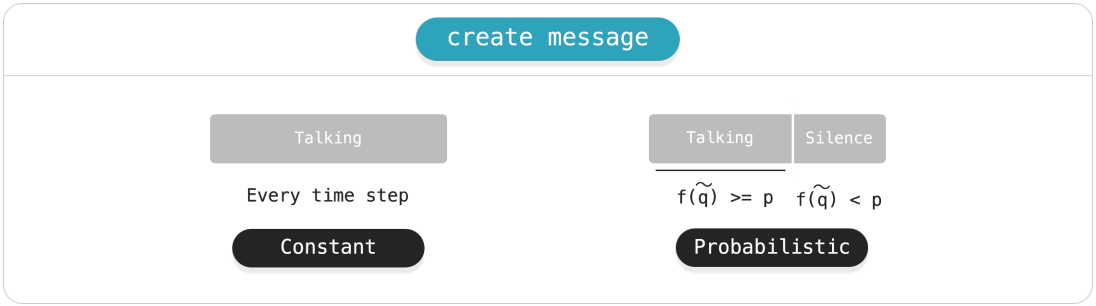

Figure 3.11: Create message module, with the light blue rounded rectangle we represented the phase and in black rounded rectangles the implemented modules.

Constant

The constant module allows the agent to create and spread each k-time steps, with $k \in [0,\infty)$, the message containing the collected information. If the current time steps is not a multiple of k the new message is set to not visible and the neighbours of the agent are not allowed to receive its message in this time step.

Probabilistic

The probabilistic module allows the agent to create and share its message with a probabilistic function related to the estimated quality $\tilde{q}_\text{i}$ defined as:

$$g(\tilde{q}_\text{i}) = \frac{(\rho * \tilde{q}_\text{i})}{Q}$$

And used with the follow rule:

$$\begin{cases} \text{allow spreading,} & \text{if } g(\tilde{q}_\text{i}) \geq U(0,1) \\ \text{do not allow spreading,} & \text{otherwise} \end{cases}$$

Where $\rho \in [0,1]$ represents the social-strength, $Q \in [0,\infty)$ means the maximum quality, $\tilde{q}_\text{i} \in [0,Q]$ the estimated quality, and $U(0,1)$ identifies a function that returns a random value from uniform distribution with variance equal to 1 and average equal to 0. Using a probabilistic module an agent having a high estimated quality $\tilde{q}_\text{i}$ has more probability to spread its option through its neighbours.

3.2.4 Update option

In this section we discuss in details the discovery and process social phases that can be viewed as part of a more general phase called update option. A motif is showed in Figure 3.12. The two modules works in an mutually exclusive way. The discovery process is always executed before the process social and if it returns a result different from *none* the process social section is not execute. In the next two subsections, the specifics of the discovery and process social modules are explained.

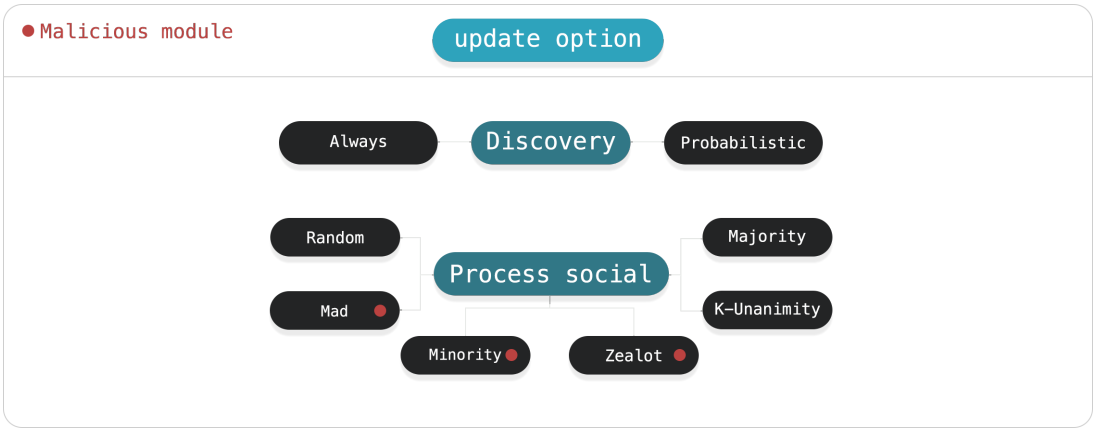

Figure 3.12: Update option module, with the blue rounded rectangles we represented the phases and in black rounded rectangles the implemented modules.

Discovery

The discovery phase allows the agent to specify which module is used during the exploration of the environment, relatively to the detection of totems. It specifies what happens when an agent detects a totem in its radius area. In the discovery phase, as showed in Figure 3.13, two modules labelled as always and probabilistic are already implemented.

Figure 3.13: Discovery phase motif, the two implemented module respectively called always and probabilistic are represented with black rounded rectangles.

- ***Always***: When a behaviour uses the always module if an agent detects a totem, it is allowed to get all the information from the totem including the related option i and the estimated quality $\tilde{q}_i$. This information is returned and used to the next module called update model, while if the agent is not able to detect a totem the module returns *none* and the next module used is the process social. This module allows an agent to always discover the presence of a totem if it is inside its sensing radius.

- ***Probabilistic***: This module allows a probabilistic behaviour when an agent detects a totem inside its radius. The agent uses the information gathered from the totem $(i, \tilde{q}_\text{i})$, in function of $h(\tilde{q}_\text{i})$, defined as:

$$h(\tilde{q}_\text{i}) = \frac{(\varrho * \tilde{q}_\text{i})}{Q}$$

And used with the follow rule:

$$\text{returned values} = \begin{cases} (i, \tilde{q}_\text{i}), & \text{if } h(\tilde{q}_\text{i}) \geq U(0,1) \\ (\text{none, none}), & \text{otherwise} \end{cases}$$

Where $\varrho \in [0, 1]$ indicates the self-strength and $\tilde{q}_\text{i}$ express the estimated discovered quality, Q indicates the maximum quality, and $U(0, 1)$ identify a uniform distribution with variance equal to 1 and average equal to 0.

Process Social

The process social phase described how an agent uses the information received from its neighbours and how it selects the new option received by others agents. As shown in Figure 3.14 many modules are already implemented and they can be divided in two different categories:

- *Malicious module*: mad, minority, and zealot.
- *Literature module*: k-unanimity, majority, and random.

Figure 3.14: Process social motif, the already implemented module called random, mad, minority, majority, k-unanimity, and zealot are represented with black rounded rectangles.

In the following list we depicted how each process social module works.

- ***K-Unanimity***: The first method presented is called k-unanimity.

  The main rule of this process social behaviour can be summarized as follow:

  "an agent computes its new option to i if and only if all the last k-observations stored in its list of neighbours are of option i, otherwise it does not select any"[32].

  The "k" parameter represents the number of neighbours that are evaluated for choosing the new option per each time step.

In the implementation on DeMaMAS of k-unanimity it is assumed that if the number of observations stored in the agent's list of neighbours is less than k, no option is selected.

Due to the fact that DeMaMAS is based on time step and not on real time, it is supposed that each time step the list of messages is shuffled in order to introduce a random order to the list.

- ***Mad***: In accordance to the definition written in the Cambridge dictionary, a mad subject is unable to behave in a reasonable way, due to an unpredictable behaviour this module is part of the attacks. It is modelled as following: *the agent selects a new option i each time step due to a random function.* This kind of conduct can introduce a noise in the system and it is used for testing the resiliency of the swarm.

- ***Majority***: The behaviour of that module can be described as follows: *"the agent selects its new option i to the one shared by the majority of individuals in its neighbourhood" [7].* In case of a tie, the new option i is chosen by random from the most m-frequent (with m $\in$ [1...number of shared option in the list of neighbours]) selected from its neighbourhood.

- ***Minority***: The minority module as an opposite behaviour of the majority one and its definition is the following one: *"the agent selects its new option i to the one shared by the minority of individuals in its neighbourhood" [19].* In case of a tie, the new option is chosen by random from the least m-frequent (with m $\in$ [1...number of shared option in the list of neighbours]) selected from its neighbourhood.

- ***Random***: The agent selects its new option i by choosing a random one from its neighbours, independently on the quality of the options.

- ***Zealot***: A zealot is defined as an entity that has a very strong option about something and tries to make others have them too. In accordance to the previous definition a zealot behaviour is characterised by an agent that never selects a new option and quality. This is an attack module and it is used for testing the resiliency of the swarm.

If after using the chosen module an agent did not pick a new option, the process social phase returns *none* otherwise return the new selected option i.

3.3 Documentation

In this section, we describe in details the different sections of parameters that can be found in the configuration file of the deMaMAS simulator and how to set and use them. The configuration file is divided in nine different subsection:

1. Experiment (Section 3.3.1)
2. Model and update (Section 3.3.2)
3. Message (Section 3.3.3)
4. Movement (Section 3.3.4)
5. Agent ant totem settings (Section 3.3.5)
6. Decay (Section 3.3.6)
7. Interaction function (Section 3.3.7)
8. File (Section 3.3.8)
9. Graphic and plot (Section 3.3.9)

For each subsection the meaning of each parameters is described. A copy of the whole documentation can be also found at the following link:
https://github.com/DiODeProject/DeMaMAS

3.3.1 Experiment

In this part of the documentation we summarise all the basic parameters that an experiment needs in order to set the general components of the simulation. A general summary of parameters contained in the experiment section can be seen in Figure 3.15

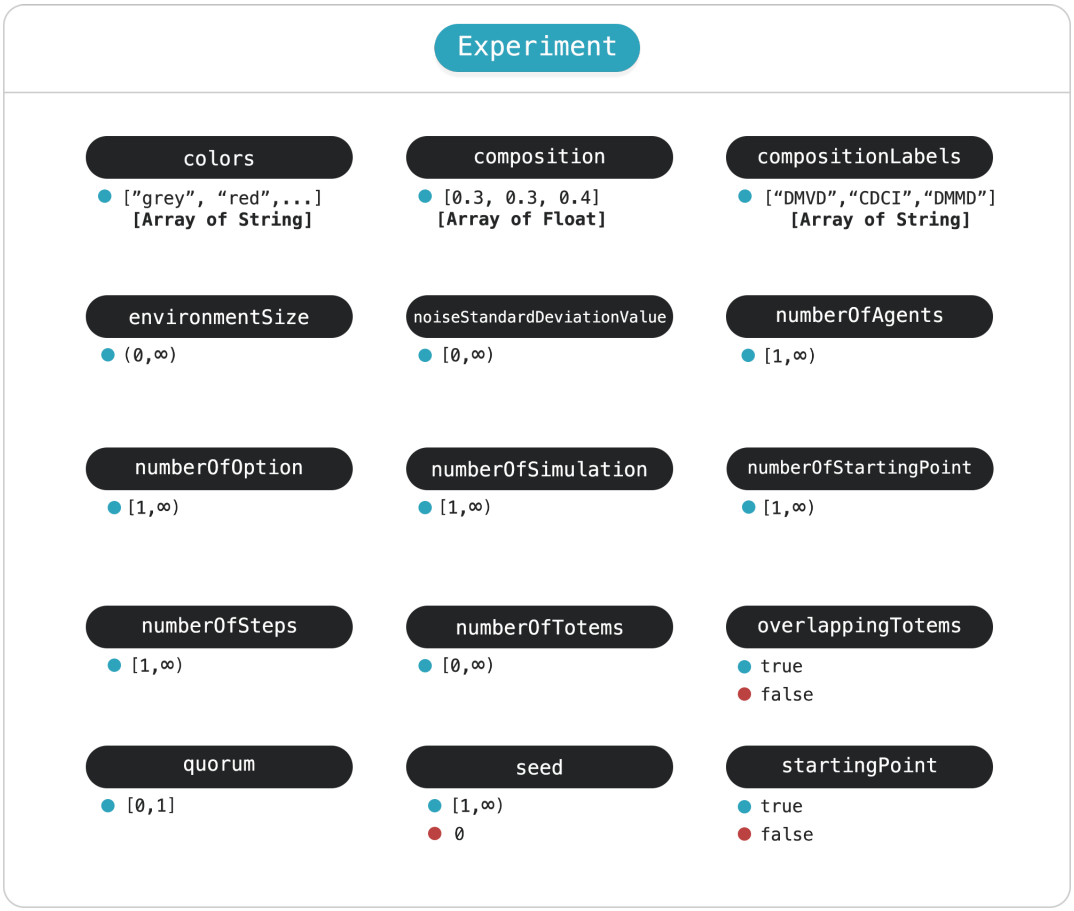

Figure 3.15: Experiment parameters overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *colours*: it is a vector of string that represents the colours that are used to show agents and totems options. All the list of available colour can be found https://matplotlib.org/examples/color/named_colors.html.
- *composition*: it is a vector of floats that represent sub-population proportions in which the swarm is divided. Each element of this vector range in [0,1] and the sum of all the elements need to be equal to 1.
- *compositionLabels*: it is a vector of String used by the simulator to display the used behaviour in the legend that appears during a graphical simulation.
- *environmentSize*: it is a float that represents the dimension of the environment (the environment is a square).

- *noiseStandardDeviationValue*: it is a float that represents the noise (ζ) that the agent adds to its evaluation of the quality from the totem during discovery and resampling.
- *numberOfAgents*: it is an integer that represents the number of agents in the environment.
- *numberOfOption*: it is an integer that represents the number of different options that can be shared by agents and totems.
- *numberOfSimulation*: it is an integer that represents the number of simulations that are executed per configuration file.
- *startingPoint*: it is a Boolean that represents if in the environment there are fixed starting points where agents are created at the initial time step or if they are generated at random positions.
- *numberOfStartingPoint*: it is an integer that represents the number of starting locations (points) where agents are created at the initial time step. This parameter is used only when startingPoint = True.
- *numberOfSteps*: it is an integer $\bar{ts}$ that indicates the maximum duration of the entire simulation in time steps.
- *numberOfTotems*: it is a vector of integers that represents the number of totems in the environment. If the length of the vector is 1, all options have that number of totems (e.g. numberOfTotems = [3] all options have 3 totems), otherwise the vector length must be equal to the number of options (i.e., len(numberOfTotems) == numberOfOption) and each option will have the respective number of totems.
- *quorum*: it is a float that represents the quorum (δ) (i.e., proportion of the swarm) to be reached to finish the simulation, if it is set to 0, the simulation runs until the time step limit $\bar{ts}$.
- *seed*: it is an integer that represents the random seed of this simulation. If 0 the seed is selected randomly. When several experiments are executed, each experiment has a new random seed incremented by 1.

3.3.2 Model and update

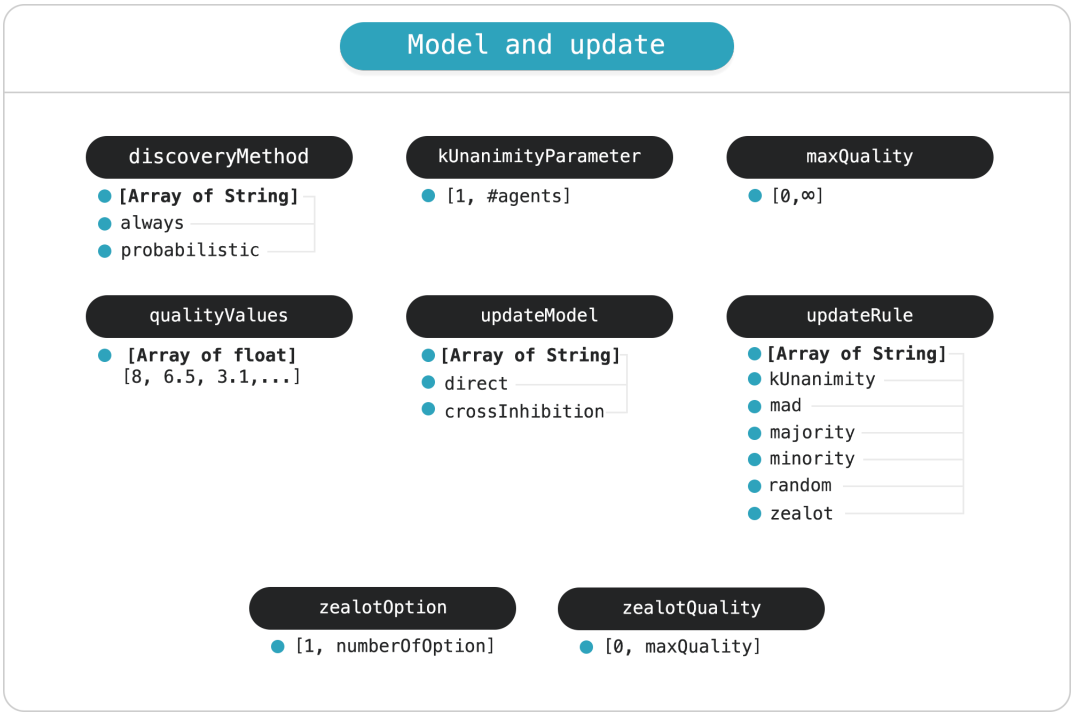

Figure 3.16: Model and update parameters overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *discoveryMethod*: it is a vector of strings that represents the modules used during the discovery phase by the agents. It determines how the agent makes a discovery based on the quality estimated from the totem.
  - *always*: when an agent enters in a totem it always discovers the option given by the totem. This module is described in details in Section 3.2.4
  - *probabilistic*: when an agent enters in a totem it discovers with a probability as a linear function of the estimated quality. This module is described in details in Section 3.2.4.
- *kUnanimityParameter*: it is an integer that represents the factor "k" in the k-unanimity module.
- *maxQuality*: it is a float Q that represents the maximum quality that can be set in the environment.
- *qualityValues*: it is a vector of floats that represent the qualities assigned to the totems. The vector should have a length equal to the number of options (numberOfOption); if the length is longer (len(qualityValues) > numberOfOption), the exceeding values are ignored; if the length is shorter (len(qualityValues) < numberOfOption), the first values len(qualityValues)-1 are assigned to the respective options (totems) and all the remaining options

have the same quality as the last vector value. For instance, for numberOfOption = 5 and qualityValues = [6,5,4] the option #1 will have quality 6, the opinion #2 will have quality 5, and the options #3, #4, #5 will have quality 4.

- *updateRule*: it is a vector of strings that represent the modules that are used by the swarm in order to compute the option to input in the updateModel. The vector length should match the number of subpopulations (i.e., len(composition)).
  - *kUnanimity*: the agent picks from its neighbours the option i if and only if the last k-received-options are identical to i.
  - *mad*: the agent selects its new option i each time step due to a random function.
  - *majority*: the agent picks from its neighbours the option that is more frequent.
  - *minority*: the agent picks from its neighbours the option that is less frequent (light-attacker).
  - *random*: the agent picks a random option from its neighbours.
  - *zealot*: the agent never changes its own option and always share with all its neighbours its option (attacker).
- *updateModel*: it is a vector of strings that represent which modules are used to update the option of the agent. The vector length should match the number of sub-populations (i.e., len(composition)).
  - *direct*: if the agent receives a new option from a neighbour (using the updateRule) it switches to the new option.
  - *crossInhibition*: if an uncommitted agent receives a new option from a neighbour (using the updateRule) it switches to that option. If a committed agent receives a new option different from its previous one from a neighbour (using the updateRule) it switches to uncommitted.
- zealotOption: it is an integer that represents the option assigned to zealots. It can range between 0 and numberOfOption. If it set to 0, the proportion of the population composed by zealots has a equal distribution over the all sub-optimal options in the environment. If it is set to a specific number i excluding the 0 and 1 (in our simulator option 1 always has the best quality) all the zealots have the same option i.
- zealotQuality: it is a float that represents the quality associated to the option used by a zealot.

3.3.3 Message

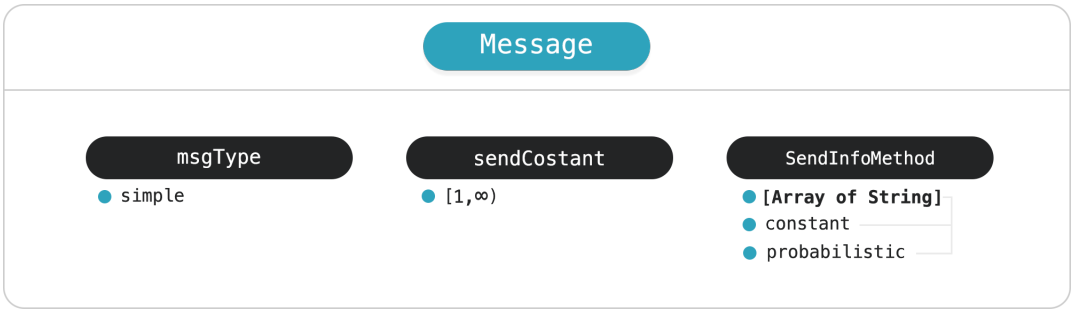

Figure 3.17: Message parameters overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *msgType*: it is a string that represents the structure of messages shared by agents.
  - *simple*: the agent shares only its own option.
- *sendConstant*: it is an integer that represents how many time step the agent needs to wait to send a message.
- *sendInfoMethod*: it is a vector that represents which modules are used by agents to share messages.
  - *constant*: the agent shares its own message each sendConstant parameter timestep. This module is explained in details in Section 3.2.3
  - *Probabilistic*: the agent shares its message with a probability related to the estimated quality $\tilde{q}_i$. This module is explained in details in Section 3.2.3.

3.3.4 Movement

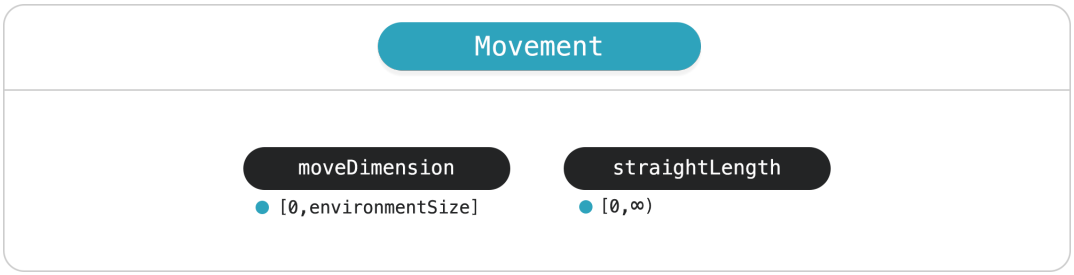

Figure 3.18: Movement parameters overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots are listed the range in which the parameters has to be set.

- *moveDimension*: it is a float that represents the dimension of the movement of the agent.
- *straightLength*: it is an integer that represents how many time steps the agent needs to wait in order to change its direction.

3.3.5 Agent and totem settings

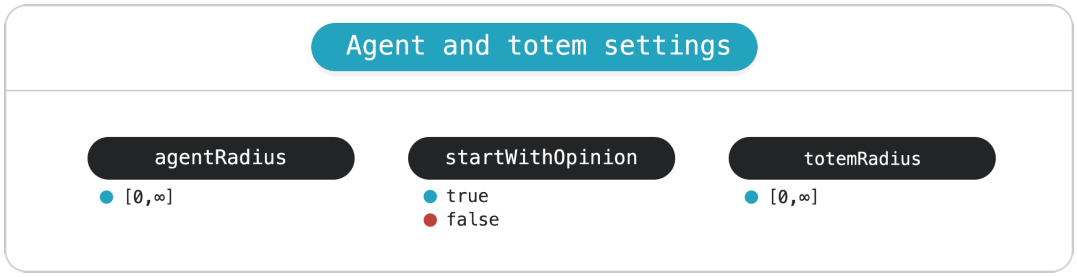

Figure 3.19: Agent and totem overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set.

- *agentRadius*: it is a float that represents the agent communication-radius.
- *startWithOpinion*: it is a Boolean that if True the simulation starts with all the agents in a committed state, while if False it starts with all the agents in an uncommitted state.
- *totemRadius*: it is a float that represents the totem communication-radius.

3.3.6 Decay

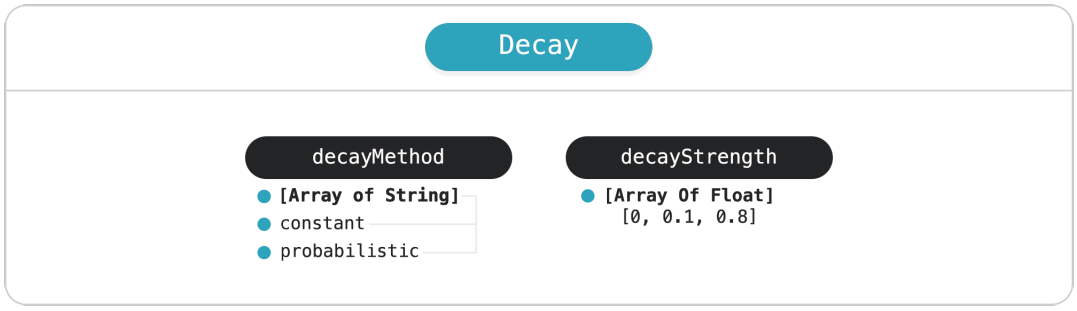

Figure 3.20: Decay overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *decayMethod*: it is a vector of strings that represents the methods that can be used by the decay function by agents.
  - *constant*: the agent forgets its own opinion with a function related to the parameter decayStrength γ. A detailed description is given in Section 3.2.2.
  - *probabilistic*: the agent forgets its own opinion with a probability function related to the parameter decayStrength γ. A deeper description is provided in Section 3.2.2.

- *decayStrength*: it is a vector of floats that is used as a parameter of the decay function, each one of the can be identified with the symbol γ.

3.3.7 Interaction function

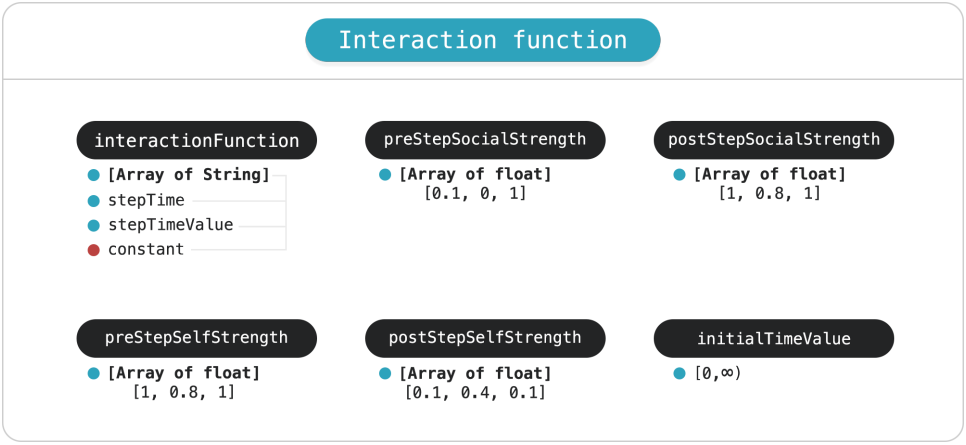

Figure 3.21: Interaction function overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *interactionFunction*: it is a vector of strings that represent the module called modulation self-social (ρ, ϱ).
  - *constant*: the agent never updates its social-strength ρ and self-strength ϱ.
  - *stepTime*: the agent updates its social-stregth ρ and self-strength ϱ after a fixed time step $\hat{t}$.
  - *stepTimeValue*: the agent updates ρ and ϱ according to a function f$(\tilde{q})$. This function is explained in details in Section 3.2.1.
- *preStepSocialStrength*: it is a vector of floats that represents how much an agent can interacts with others before the condition in the interaction function is verified. It represents the value ρ before the update.
- *postStepSocialStrength*: it is a vector of floats that represents how much an agent can interact with others after the condition in the interaction function is verified. It represents the value ρ after the update.
- *preStepSelfStrength*: it is a vector of floats that represents how much an agent can interact with totems after the condition in the interaction function is verified. It represents the value ϱ before the update.
- *posStepSelfStrength*: it is a vector of floats that represents how much an agent can interact with totems after the condition in the interaction function is verified. It represents the value ρ after the update.
- *initialTimeValue*: it is a integer $(\hat{t})$ used by the interactionFunction to calculate when the condition in the function is verified.

3.3.8 File

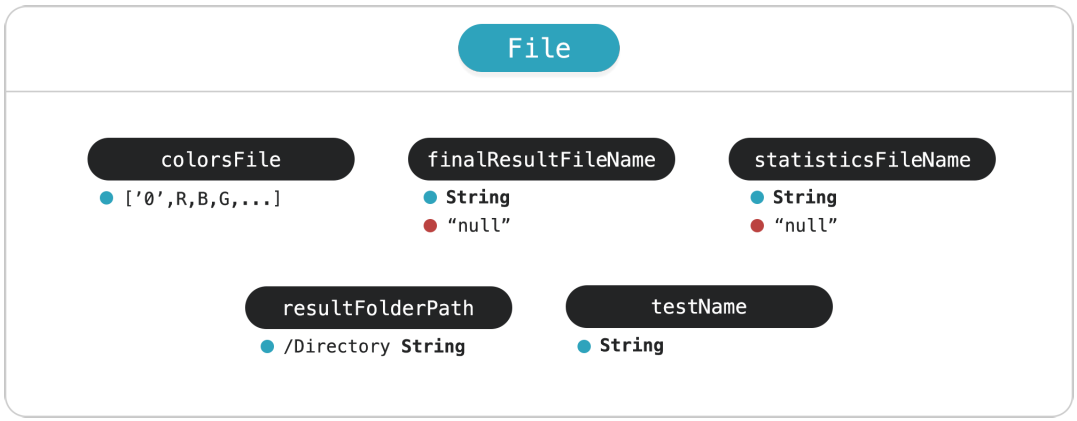

Figure 3.22: File overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots we list the range in which the parameters has to be set or a possible configuration in case of Arrays.

- *coloursFile*: it is a vector of strings that represents an abbreviation of colours name used by txt files that contains the results of the simulation.
- *finalResultFileName*: it is a string that is used as a name for creating a txt containing only the final results of all simulations for a single configuration file, if it is set to "null" no txt file is created.
- *statisticsFileName*: it is a string (path) that indicates what is the directory path in which all the results have to be saved.
- *resultFolderPath*: it is a string that is used as a name for creating txt files for each simulation, if it is set to "null" no txt file is created.
- *testName*: it is a string that represents the name of the test that we want to run.

3.3.9 Graphic and plot

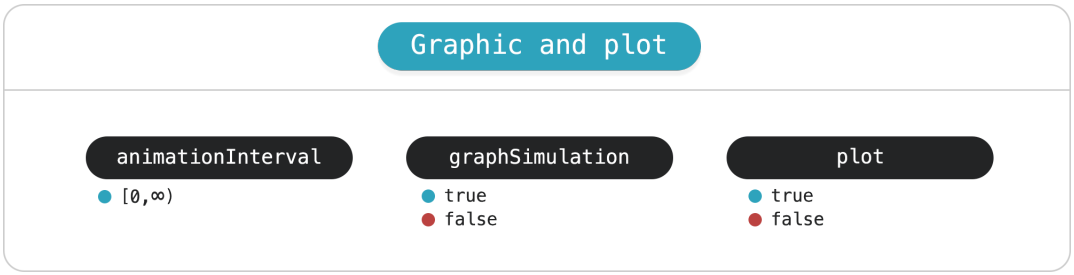

Figure 3.23: Graphic and plot overview, with the black rounded rectangles we indicate the name of the parameters, and next to the blue dots are we list the range in which the parameters has to be set.

- *animationInterval*: it is an integer that represents how many milliseconds the animation function plots an image during the simulation.
- *graphSimulation*: it is a Boolean that represents if the simulation is graphically showed or run in the shell.
- *plot*: it is a Boolean that represents if the user needs a plot of the results or not.

Chapter 4

Attacks

In the previous chapter we showed how the multi-agent simulator works and was provided a detailed description of each parameter that composed it. This chapter is crucial in the identification of the potential threats that can affect a swarm. It starts examining the definition of an attack, then moves to show a taxonomy of all the possible attackers that can influence a swarm and ends with an explanation of heterogeneous swarms.

4.1 Classification

This section opens with a proper definition of attack, then continues with a description of the components that constitute it and ends with a detailed characterisation of the possible adversary.

4.1.1 Attack definition

From a computer security point of view an attack is defined as an information security threat that implicates an attempt to alter, destroy, implant, obtain, remove or reveal information without authorized access or permission. Thanks to the previous explanation in the following section it is discussed in details what are the characteristics of an attack in a best-of-n collective decision problem.

4.1.2 Attacks in best-of-n

Lot of effort has been spent in the classification of attacks in computer security in the past years. In the swarm robotics field less effort has been spent in categorization and definition of attacks [11] [30]. We proposed a description of components that constitute an attack. As represented in Figure 4.1 an attack consists of three main elements:

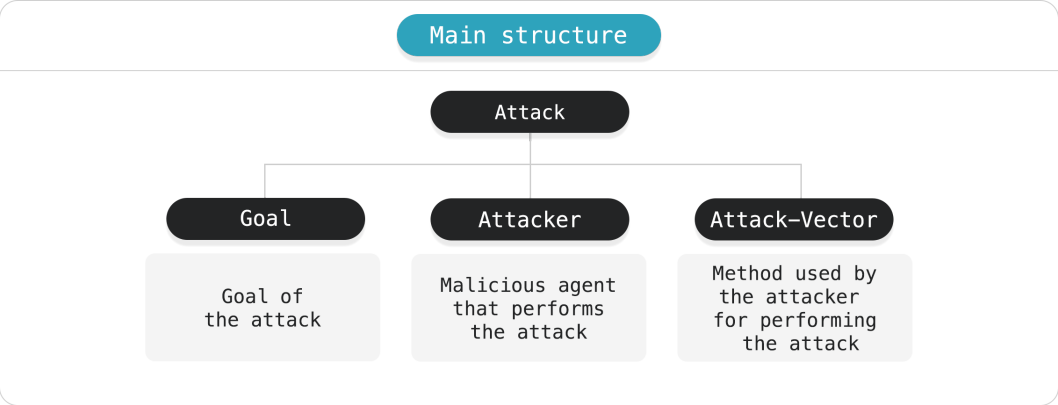

Figure 4.1: Attack components motif, an attack is composed by three different elements called goal, attacker and attack-vector.

1. Goal: it is the main purpose of an attack. The possible different goals of an attack in best-of-n are summarised in Figure 4.2.

2. Attacker: it is the individual or agent (group of agents) that performs the attack. A detailed description of the main characteristics of a malicious agent are reported in Figure 4.3. In Section 4.1.3 we propose and depict five different categories of attackers.

3. Attack-Vector: it represents the method used by an attacker to perform the attack. Different types of attack-vectors are showed in Figure 4.4.

The first component that we present is the goal, that defines the aim of the attack. As shown in Figure 4.2 there are three different categories of goals that can be performed in our scenario:

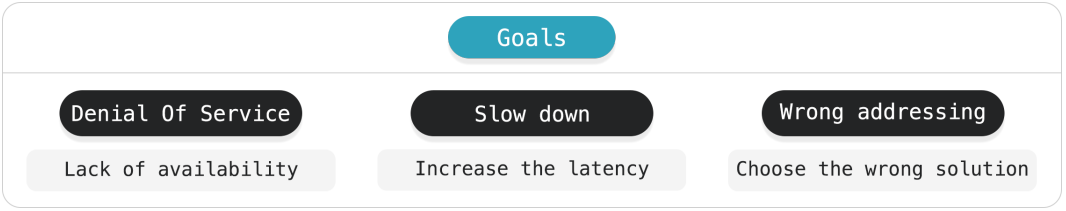

Figure 4.2: Goals of an attack motif, an attack can have three different goals: Denial Of Service (DOS), slow down, and wrong addressing.

1. Denial of service (DOS): according to a computer security perspective it is defined as a threat to the availability, normal functionality and associated services of a system for their intended use by legitimate users. Related to our problem a DOS is verified if the swarm is not able to reach a consensus and as a consequence of that the swarm is not capable to choose one of the possible n-options. This category of aims is the most brutal and dangerous for a swarm, because it compromises the entire system behaviour. A DOS can also be performed by more than one attacker, in this scenario it is called DDOS that stands for Distributed Denial of Service, in our experiments are considered both DOS and DDOS.

2. Slow down: the aspiration of that goal is to affect the system speed evolution by a non-negligible delay in accomplishing the plan of the swarm. The swarm reaches the consensus slower.

3. Wrong addressing: the system goal is redirected to a non-optimal solution, this condition is strictly related to best-of-n problem. E.g., the swarm, instead of choosing the best option over the n possible ones, chooses another one with a quality strictly lower than the best.

An attacker can reach its goal in many different ways, this different paths are characterized by how the behaviour of an intruder is described and implemented. As depicted in Figure 4.3 the main characteristics of a malicious agent are:

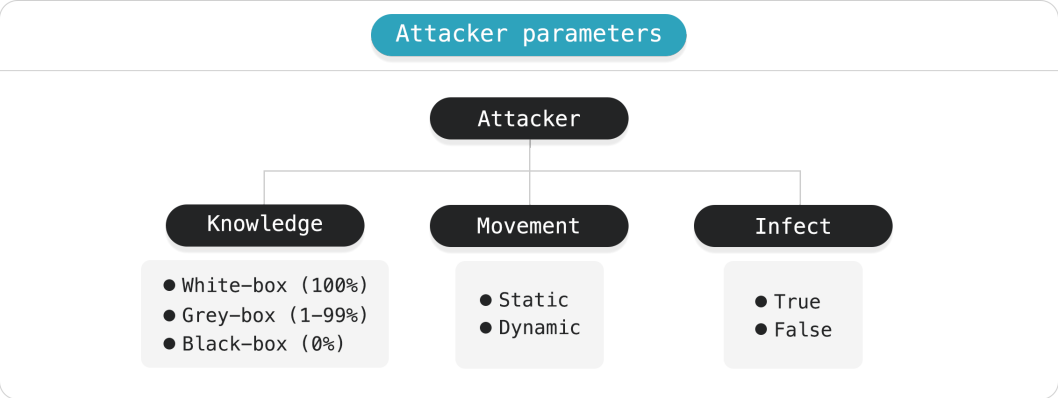

Figure 4.3: Attacker parameters component motif, an attacker is composed by three parameters called knowledge, movement and infect.

1. Knowledge: is defined as a measure of the amount of information the attacker has about the system, from the environment and the swarm.
   - White-Box: an attacker has detailed knowledge of a swarm's implementation that allowed it to undertake tailored attacks against that specific system [100%].
   - Grey-Box: an intruder has a knowledge between the bare minimum and approximately the maximum [1-99%].
   - Black-Box: a malicious agent has no prior knowledge of a swarm that it wishes to attack [0%].
2. Movement: this characteristic represents how an attacker moves through the environment.
   - Static: the malicious agent maintains the same position during the whole execution of the attack.
   - Dynamic: the intruder can change its position during the attack.
3. Infect: it is a property of the attacker, that if set to True the malicious agent can convert the behaviour of every non malicious agent it interacts with.

An attacker to achieve its goal needs an attack strategy, in this context an attack-vector can apply one or more of the four different categories of manipulations:

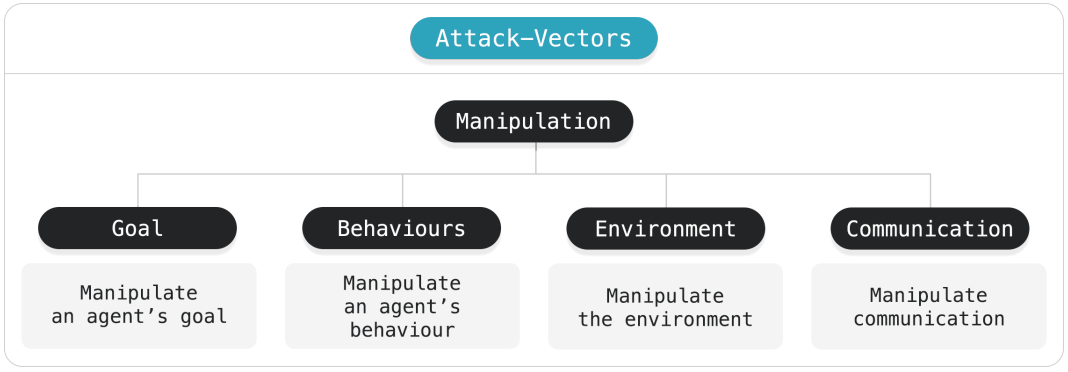

Figure 4.4: Attack vector component motif, an attacker can manipulate the goal, the behaviour, the environment and the communication channel of an agent.

1. Goal: the attacker with its behaviour tries to change directly the goal of the assaulted agent.
2. Behaviours: the intruder attempts to modify the behaviour of an agent (e.g., corruption of movement, update algorithm, quality sensor,...).
3. Environment: through the manipulation of the environment a malicious agent can perform different kinds of damage, from sensor to physical one.
4. Communication: an intruder can disturb the communications between agents or intercept their signals and change them.

After an explanation of the elements that form an attacker, in the next section we classify the different kinds of intruder that can be found in a Collective Decision Making best-of-n problem.

4.1.3 Attackers in best-of-n

As a result of the attack classification, which was established in the last section, this part begins by dividing attackers in five different groups as showed in Figure 4.5. Each category can be explicit or implicit. An intruder is defined as an explicit one, if its rival-behaviour is done by purpose otherwise is called implicit. The five categories are:

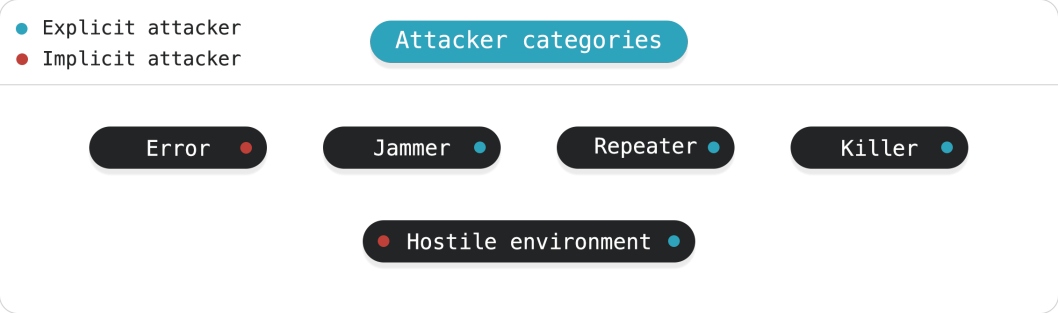

Figure 4.5: Attackers types, an attacker in a best-of-n problems can be in one of the five different categories. Error, jammer, repeater, killer, and hostile environment.

1. Error: an "Error-attacker" is an agent of our system that due to hardware or software failure due to hostile-environment, noise or bad-programming, can be an implicit-malicious-agent. In robot swarms, this kinds of error are very frequent related to the complexity and weakness of the hardware. In DeMaMAS this behaviour can be simulated easily introducing a "bug" in the code or by adding a random noise function that can change the behaviours of the agents. A more detailed characterisation is showed in Figure 4.6.

2. Repeater: it is able to spread a wrong information through the swarm. Given a full knowledge of the system this attacker can perform a wrong addressing attack and in extreme cases a DOS attack. In both robots and simulator cases, this kind of attackers are easy to implement with a little modification of the behaviour of the agent in the code. A more detailed description is given in Figure 4.7.

3. Jammer: it is a malicious-agent that is able to interrupt communication flow between agents in the swarm. This attacker introduces a disturb in the communication between robots in the swarm and does not allow them to use the classical communication channel. In the multi-agent simulator this behaviour can be simulated by introducing a forbidden communication area. A deeper explanation is provided in Figure 4.8.

4. Killer: it is a malicious agent that performs physical tampering. It provokes physical damage to the agents in the swarm. Its aim is to reduce the population, this behaviour can generates a DOS attack.

5. Hostile environment: A hostile environment is a malicious environment that due to its morphology can provoke a wide range of damages to the agents.

Error

As described before an error attacker is characterized by a hardware or software malfunctioning, it can be the cause of different types of malicious agent. The complete list is showed in Figure 4.6 reported below.

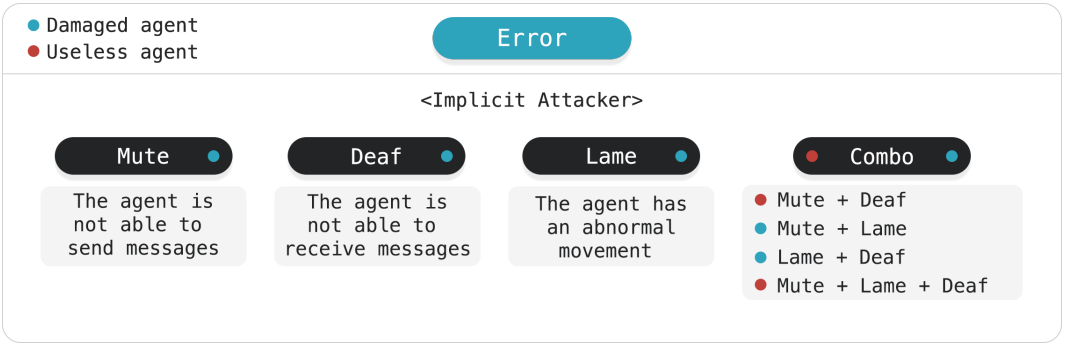

Figure 4.6: Error categories motif, an Error can be divided in four different categories called mute, deaf, lame, and combo.

The Error classification is divided in four different categories:

- Deaf: an implicit-attacker is deaf if it is lacking the power of hearing. A robot is suffering of deafness if it is not able to receive communication from the other robots caused by a malfunctioning in the hardware related to the communication sensing. In our multi-agent simulator an agent is deaf if it is not able to receive any message from its neighbours.
- Lame: a lame is an individual that is not able to move correctly in the environment because of physical fault related to the hardware-movement section or a malfunctioning related to a bad calibration (e.g., Kilobots). In DeMaMAS a lame can be implemented by adding a function that generates an abnormal movement behaviour.
- Mute: a malicious agent is mute if it is unable to speak, due to a fault in the hardware communication section or to a bug in the implementation. In DeMaMAS a mute-agent can be easily implemented by adding a silent function that does not allow an agent to speak.
- Combo: an agent can be affected by more than one of the previous problems.

Repeater

The repeater category is the most interesting to answer our research question. In Figure 4.7 are highlighted three intruder that corresponding to those implemented in DeMaMAS. A Repeater spreads a wrong information through the swarm. We depicted four different types of intruders listed and described below:

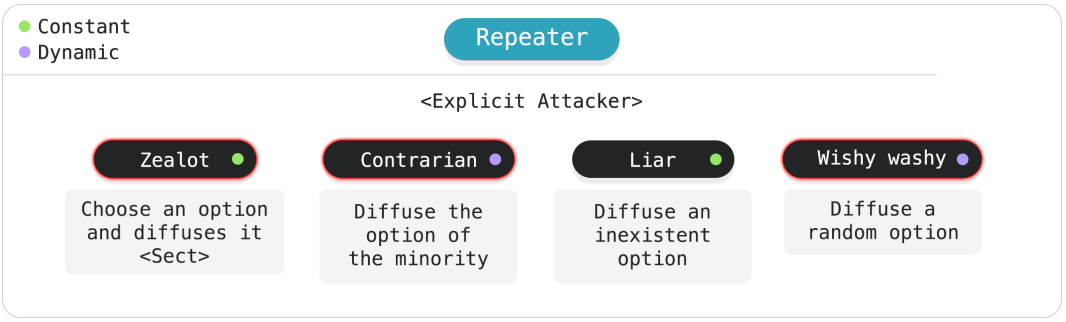

Figure 4.7: Repeater categories motif, a repeater can be divided in four different types called zealot, contrarian, liar, and wishy washy.

- Contrarian: it is a malicious agent that diffuses the option of the minority. With this behaviour it provokes a slow-down to the system and in extreme cases a DOS (or DDOS). This attacker is already implemented in DeMaMAS and a detailed analysis will be provided in Section 5.2.
- Liar: it is an intruder characterized by a diffusion of an non-existent option through the swarm. This attacker-behaviour can generate a DOS. This kind of attacker is not implemented in DeMaMAS.
- Wishy Washy (WW): it is defined as a malicious agent that continuously changes its option every time step. It spreads its option every time step and with this behaviour it introduces a noise factor in the swarm. This behaviour is already implemented in DeMaMAS by allowing the agent to switch option every time step and to spread it. This attacker is very powerful and can generate both slow down and DOS, its strength increases when the number of option in the environment is increased. With an higher number of options more noise is introduced in the system.
- Zealot: A zealot is defined as an intruder that has a very strong option about something and tries to make others have them too. In our simulator it is modelled as an agent that never changes its own option and spreads it through the swarm every time step. Its main goal is to perform wrong addressing and in extreme cases can degenerate in a DOS attack.
  - Sect: it is defined as a group of zealots that spread the same option. This variant is made by the needs of having an attacker that is able to perform wrong addressing.

Jammer

A Jammer is characterized by introducing a noise in the environment that produces an interruption of the communication flow between agents in the swarm. Each category can also perform a filter over specific selected targets. We say that a Jammer performs a targeted filtering if it disrupts only a selected option, otherwise performs an untargeted filtering. As showed in Figure 4.8 this category of attacker is divided in two classes:

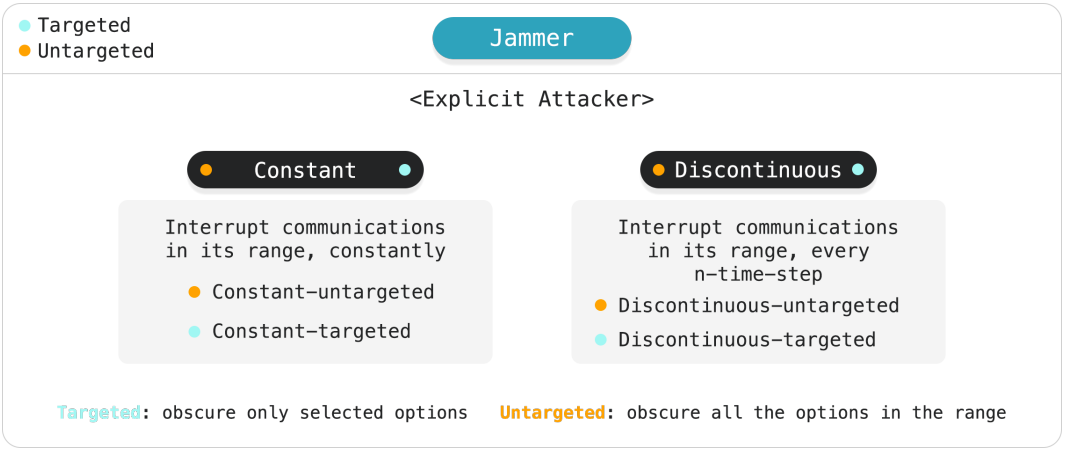

Figure 4.8: Jammer categories motif, a jammer can be of two different types constant or discontinuous.

- Constant: a constant jammer disturbs the communications between agents in a delimited area, characterized by its radius for the whole duration of the attack.
- Discontinuous: a discontinuous jammer interrupts the communications between agents in a delimited area, characterized by its radius. This interruption is not continuous and can alter moment in which the communication in its range is allowed and moment in which is not.

Hostile environment

An hostile environment is a particular case of attackers in which the malicious agent is the environment itself.

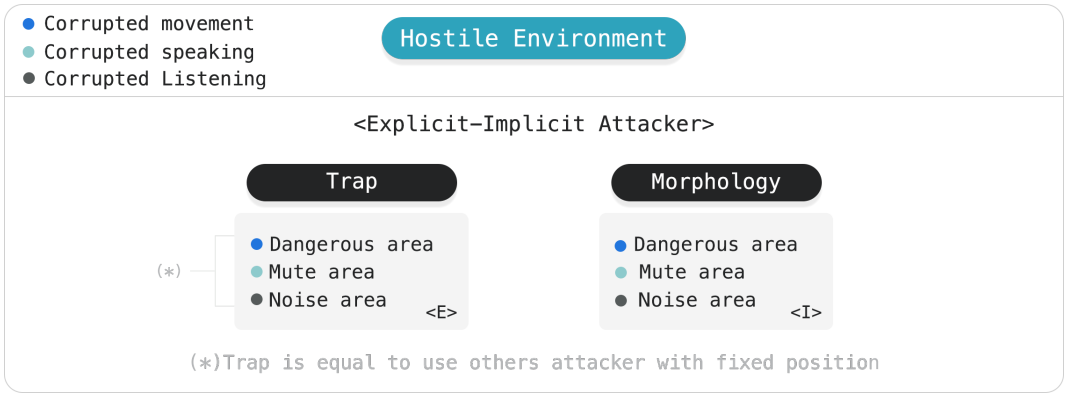

Figure 4.9: Hostile environment categories, an hostile environment can be considered a trap or related to the morphology of the environment.

As showed in Figure 4.9 we categorised two different kinds of hostile environment:

- Trap: It is characterized by the insertion to an initial non-hostile environment of an element that can cause different types of damages to the agents in the swarm. This kind of attackers can be simulated by using the previous listed attacker with a fixed position. The different kinds of traps can generate a dangerous area, in which an agent can be physically damaged, a mute area in which an agent is not able to speak, and a noise area in which an agent cannot communicate.
- Morphology: The environment in this case due to its shape can perform different kinds of attacks. We divided the morphology categories as the same of the trap one with the same meaning, but here the only difference is that the attack is implicit.

Killer

The last kind of attacker that we defined is called killer. Through physical tampering its goal is to perform a reduction of the population that consequently degenerates in a denial of service.

Figure 4.10: Killer motif.

4.2 DeMaMAS and heterogeneous swarms

In order to perform an attack against a swarm our simulator needs to generate two different behavioural strategies. Thanks to its modularity, DeMaMAS allows us to create heterogeneous swarms very quick. An attack is defined as a normal agent that has only a different settings of the module in each different phase of the simulator. In the next chapter we explain in details how to set the configurator file of DeMAMAS in order to generate both literature and attacks methods. A screenshot of a simulation is reported below in Figure 4.11.

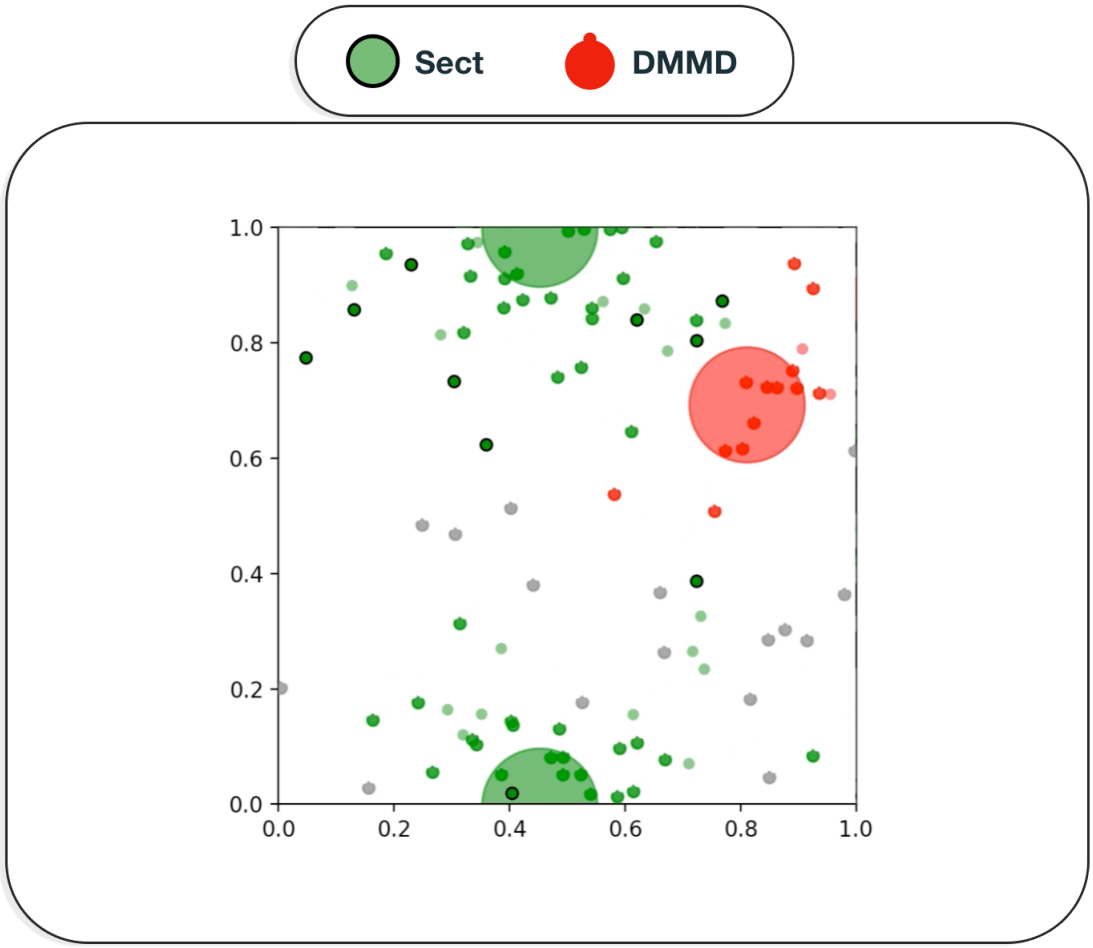

Figure 4.11: A graphical execution of DeMaMAS using an heterogeneous swarm with 100 agents (10% sect and 90% DMMD), 2 different options represented with the 2 different totems (red and green) and a quorum set to 0.8.

Chapter 5

Experiments

In Chapter 4 we described a classification of the possible kinds of attacks that affect a swarm. In this chapter we start with a detailed description of how to set up the investigated scenario, pass through the used analysis metrics and plots and conclude with the obtained results.

5.1 Test description

To investigate the scenarios, we need to describe how to set up all the analysed behaviours for testing the swarm resiliency and how to tune the parameters in order to generate our case of investigation. For a detailed explanation of each behaviour see Section 2.1. In this thesis we compared four behaviours from the literature and a new one made ad hoc, against three different types of attacker. The five decision making approaches are:

- *CDCI* stands for Collective Decision trough Cross-Inhibition, this behaviour copies the option of a random neighbour at each application of the process social and if its status is uncommitted or the copied option is equal to its past one it switches to the new option, otherwise it flips uncommitted.
- *CDMCI* stands for Collective Decision through Majoritarian Cross-Inhibition. This behaviour is created ad hoc for the aim of the dissertation and its behaviour selects the more frequent option shared by its neighbourhood at each application of the process social and if its status is uncommitted or the copied option is equal to its past one it switches to the new option, otherwise it flips uncommitted.
- *K-unanimity*: the agent picks an option from its neighbours if and only if the last k-received options are identical it applies direct switch in order to update its own option with the selected new one.
- *DMMD* stands for Direct Modulation of Majority-based Decisions. The agent picks the most frequent option from its neighbours and if it is different from its previous one it switches to it. A more detailed description is provided in Section 2.1.

- *DMVD* stands for Direct Modulation of voter-based Decisions. The agent picks a random option from its neighbours and if it is different from the previous one it switches to it. A more detailed description is given in Section 2.1.

5.1.1 Literature behaviours parameters

- ***CDCI*** *Collective Decision trough Cross-Inhibition*:
  - update model → cross-inhibition (Section 3.3.2)
  - create msg → probabilistic (Section 3.3.3)
  - update option:
    - discovery → probabilistic (Section 3.3.2)
    - process social → random (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
- ***CDMCI*** *Collective Decision trough Majoritarian Cross-Inhibition*:
  - update model → cross-inhibition (Section 3.3.2)
  - create msg → probabilistic (Section 3.3.3)
  - update option:
    - discovery → probabilistic (Section 3.3.2)
    - process social → majority (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
- ***DMVD*** *Direct Modulation of Voter-based Decisions*:
  - update model → direct switch (Section 3.3.2)
  - create msg → probabilistic (Section 3.3.3)
  - update option:
    - discovery → probabilistic (Section 3.3.2)
    - process social → random (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)

- ***DMMD*** *Direct Modulation of Majority-based Decisions*:
  - update model → direct switch (Section 3.3.2)
  - create msg → probabilistic (Section 3.3.3)
  - update option:
    - discovery → probabilistic (Section 3.3.2)
    - process social → majority (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
- ***K-unanimity***:
  - update model → direct switch (Section 3.3.2)
  - create msg → probabilistic (Section 3.3.3)
  - update option:
    - discovery → probabilistic (Section 3.3.2)
    - process social → k-unanimity → k = 3 (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)

5.1.2 Attacker behaviours parameters

After describing the parameters configurations that allowed us to simulate all the literature behaviour, in this section all the configuration to create the selected attacker for the swarm are listed. The implemented attackers as said before are wishy washy, contrarian, zealot and sect and are explained in details in the previous Section 4.1.3.

- ***Wishy Washy***:
  - update model → direct switch (Section 3.3.2)
  - create msg → constant (Section 3.3.3)
  - update option:
    - discovery → always (Section 3.3.2)
    - process social → mad (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
- ***Contrarian***:
  - update model → direct switch (Section 3.3.2)
  - create msg → constant (Section 3.3.3)
  - update option:
    - discovery → always (Section 3.3.2)
    - process social → minority (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
- ***Zealot***:
  - update model → direct switch (Section 3.3.2)
  - create msg → constant (Section 3.3.3)
  - update option:
    - discovery → always (Section 3.3.2)
    - process social → zealot (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
  - zealot option → 2 (Section 3.3.2)
  - zealot quality → 5 (Section 3.3.2)

- ***Sect***:
  - update model → direct switch (Section 3.3.2)
  - create msg → constant (Section 3.3.3)
  - update option:
    - discovery → always (Section 3.3.2)
    - process social → zealot (Section 3.3.2)
  - modulation self-social: constant → (self-strength = 1, self-social = 1) (Section 3.3.7)
  - decay: constant = 0 (Section 3.3.6)
  - zealot option → 0 (Section 3.3.2)
  - zealot quality → 5 (Section 3.3.2)

5.1.3 General test parameters

For the resiliency tests, we used the high performance computing resource of the University of Sheffield called iceberg (all the documentation of how using it can be found here: https://www.sheffield.ac.uk/wrgrid/iceberg). In order to collect all the needed data we uploaded DeMaMAS on the cluster, set and run different configuration files allowing us to submit thousands of different jobs that run in parallel and gave us the results faster. We collected data on a set of 1440 different experimental conditions, obtained by all the possible combinations for the three values for the parameter numberOfOption = [2,4,6], by the use of four different difficulties proportion corresponding to the parameter difficulty = [0.4,0.8,0.9,1] and six values corresponding to the percentage of attacker in the swarm [0%,1%,5%,10%,15%,20%]. Each one of the previous configurations is performed for 5 different literature behaviours and 4 different attackers. For each one of the different 1440 configurations are performed 100 different runs with different random seeds, for a total of 144000 experiments. In the following list the set up of each test is described:

- ***Literature behaviour vs attackers***
  - agent radius → 0.05 (Section 3.3.5)
  - totem radius → 0.1 (Section 3.3.5)
  - graph simulation → False (Section 3.3.9)
  - plot → False (Section 3.3.9)
  - environment size → 1 (Section 3.3.1)
  - start with option → False (Section 3.3.5)
  - starting point → False (Section 3.3.1)
  - composition → [(1.0, 0.0), (0.99, 0.01), (0.95, 0.05), (0.90, 0.1), (0.85, 0.15), (0.8, 0.2)] (Section 3.3.1)
  - max quality → 10 (Section 3.3.2)
  - number of agents → [100] (Section 3.3.1)
  - number of simulations → [100] (Section 3.3.1)
  - number of steps → 10000 (Section 3.3.1)
  - quorum → 0.8 (Section 3.3.1)
  - msg type → simple (Section 3.3.3)
  - number of options = number of totems → [2, 4, 6] (Section 3.3.1)
  - difficulty → [0.4, 0.6, 0.8, 0.9, 1] (Section 3.3.2)
  - standard deviation → [1] (Section 3.3.1)
  - seed → 7777 (Section 3.3.1)

5.2 Analysis

After the data collection process a deep exploration and understanding of the data are performed. In order to visualise the performance of the behaviours we used three different metrics called accuracy, convergence and time step.

5.2.1 Metrics

Accuracy

A definition of accuracy from a best-of-n point of view is provided below. A simulation composed of a fixed number of agents M, reached a decision if a number of agent m favor the same option i and their cardinality is greater or equal to the quorum threshold δ set by the designer. Given a number of runs per simulation R, we define accuracy as the number of runs $\hat{r}$ that reached a decision to the best option i^* divided by all the runs R.

$$accuracy = \frac{\hat{r}}{R}$$

Convergence

Given a number of agents M per simulation, a number of runs per simulations R, convergence is defined as the number of runs $\tilde{r}$ that reached a decision divided by R.

$$convergence = \frac{\tilde{r}}{R}$$

The convergence value is always greater than or equal to the accuracy one, due to the fact that a simulation for taking a decision to the best option is taking a decision first.

Time step

Time step is defined as the number of time steps $\tilde{t}$ that a run r of a simulation needs to take a decision. If the run reaches the fixed time step limit $\bar{ts}$ without taking a decision, the value of speed is set to the time step limit $\bar{ts}$.

$$speed = \begin{cases} \tilde{t}, & \text{if } r \text{ reached consensus} \\ \bar{ts}, & \text{otherwise} \end{cases}$$

In our analysis when on the axis we plot the time step metrics we meant the average of the time step over all the runs that compose a simulation.

5.2.2 Data representation

The collected data are organised in text files containing a lot of information. Each text file corresponds to the results given from a simulation, in our case it contains the results of 100 different runs. A result relative to a run of a simulation is represented as follow:

SEED: 7777
TS 0 R B Q D S
503 2 80 18 True [4, 4] [178, 28]

In the first line is reported the seed used for the run. In the second line are written the abbreviation corresponding to:

- TS → Time Step
- 0 → number of uncommitted agents
- R → number of committed agent to option Red
- B → number of committed agent to option Blue
- Q → Quorum
- D → Discovery interactions
- S → Social interactions

In the third line are reported the corresponding data to each one of the previous sections. The data of discovery and social interactions corresponds to [discovery interactions for literature behaviour, discovery interactions for attack behaviour] and [social interaction for literature behaviour, social interaction for attack behaviour].

For reading and elaborate all the data a parser that uses regular expressions was implemented. It allowed us to to filter and plot different kinds of graphs through a configuration file. Some of the most useful data representations are described below.

General comparison plots

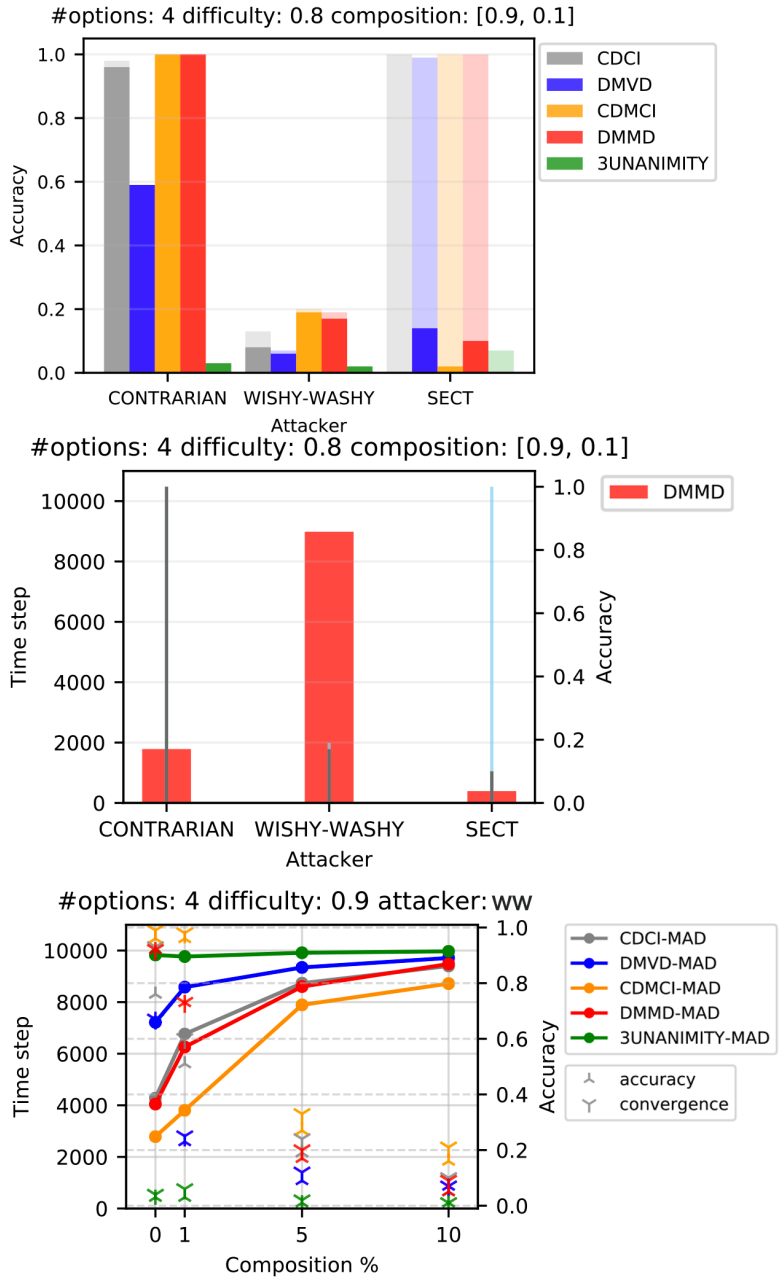

Figure 5.1: Different used plot types for general comparison and visual understanding.

In Figure 5.1 we depicted three important types of plots that are used in our visual representation. In the first graph we compared all the literature behaviours and attackers. On the y-axis we vary the accuracy and the convergence, while on the x-axis we report all the attacks. Convergence values are represented by the less bright bar, brighter bars represent accuracy. In this plot the number of options, the difficulty (the ratio between the option with the highest quality and that with the lowest one, in our investigation we always have a best option and all the remaining has the same quality) and the composition (the first element in the brackets is the percentage of agents using a literature behaviour while the second element represents

the percentage of agents characterised by a malicious behaviour) of the swarm are fixed. This kind of graph summarises a lot of data and allowed us to visualise an overall comparison between attacks and literature behaviours.

In the second plot on the vertical axis we vary both time steps (an average of all the 100 different runs per simulation), accuracy and convergence, on the horizontal axis we vary only the attacks. We fix the number of options, the difficulty, and the composition. In this plot we highlighted how a single behaviour reacts to all the different types of attacks. This graph is useful because in only one graph we described time step, accuracy and convergence. The thin grey lines represent the accuracy, the convergence is represented by thin-light-blue lines, while the different coloured bars represent the time steps related to the compared behaviour.

In the last graph on the y-axis we vary the same metrics as the previous one while in the x-axis we vary the composition in percentage of attackers. In this plot the number of options, difficulty and the attacker type are fixed. The coloured lines represent each compared literature-behaviour, the two different ticks represent accuracy and convergence as reported in the legend. This graph permits to understand how each literature behaviour reacts on varying the same percentage of attacker in the swarm.

Detailed comparison plots

In the following plots more information are plotted in order to obtain a better comprehension of how each behaviour reacts to each different attacker.

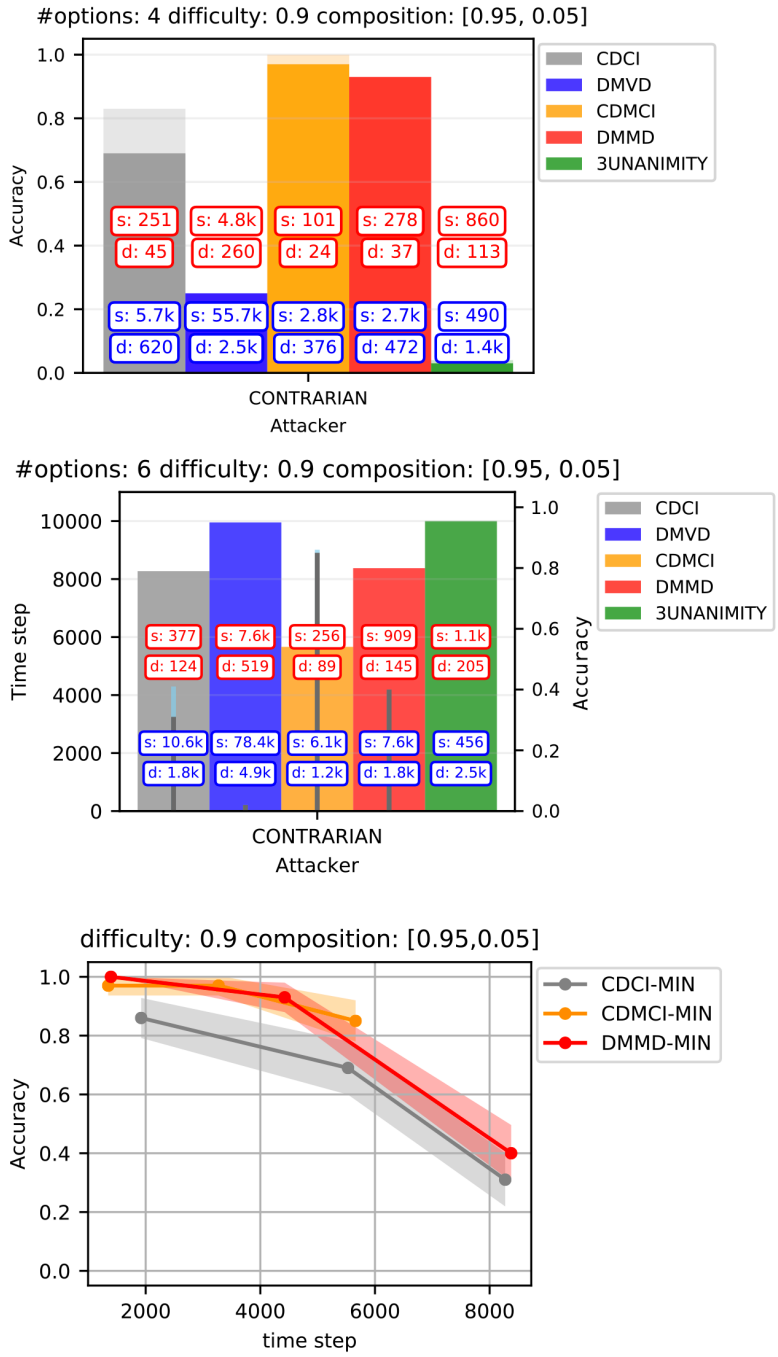

Figure 5.2: Different used plot types for detailed comparison and visual understanding.

In Figure 5.2 we represented three different kinds of plots depicting more detailed information with respect to the more general one described above in Figure 5.1.

In the first graph on the y-axis we vary accuracy and convergence while on the x-axis we vary the types of attacker. Number of options, difficulty values and composition are fixed. The brighter bars indicate accuracy while the less bright

represent convergence. In this plot, per each behaviour and type of agent, we also plot social and discovery interaction. The social interaction counter represents the average number of times that the swarm changes its option due to a social behaviour, while the discovery interaction identifies the average number of times related to a discovery. In the red squares we report respectively the social (s:) and discovery (d:) interactions related to the attacker, while in the blue ones we display respectively the social and discovery interaction of the corresponding literature behaviour. This plot is useful to have a deep comprehension of the interaction mechanism that each behaviour applies to be more resilient.

In the middle plot on the y-axis we report time step, accuracy and convergence represented respectively by the coloured bars, the grey lines and the light-blue lines, while on the x-axis we vary the attackers. In this plot we summarised all the data contained in one text file relative to an experiment. It is very useful because it permits to deeply understand how the behaviours react under attack.

In the last graphs we represent on the x-axis the time-steps while on the y-axis the accuracy metric. In this graph the lines represent the correlation between speed and accuracy. Each dot represents respectively the number of different options [2, 4, 6]. The shadows represent the confidence interval of 95%.

Thanks to the previous charts some interesting results are discovered.

5.3 Results

In this section we explain all the contributions provided by our investigation. The first important result confirmed by our experiments is related to the behaviour of the attackers. Each attacker is modelled to perform a different goal. As explained in Section 4.1.3 a contrarian tries to perform a slow down of the system, a wishy washy attempts to execute a Denial Of Service (DOS or DDOS that stands for Distributed DOS) and a sect strives to accomplish a wrong addressing. The second important result is given by the reaction of 3-unanimity in simple collective decision problems, it has the worst performance in both time step and accuracy when the swarm is not under attacks, but it becomes the best one when intruders are added in simple problems. The third contribution is relative to the impact of using majority as a process social and the last contribution is how cross-inhibition influences resiliency. In the first three sections we report the data results relative to the behaviours of the attackers.

5.3.1 Contrarians perform slow down

As explained before a contrarian attacker aim is to perform a slow down. This kind of attack tries to decrease the convergence speed of the swarm. After analysing the collected data it is discovered that the attacker is able to perform its goal. One example of a quite difficult case is reported in Figure 5.3. In this picture we show four different plots in which on each plot on the x-axis we vary the attacker and on the y-axis we vary the time steps and we fix the number of options, the difficulty and the composition. From one plot to another we vary only the composition of the swarm. Thanks to this visual representation it is highlighted that increasing the number of contrarians in the swarm the number of time steps considerably increases. By adding the interaction and discovery values per each behaviour we also inferred that adding contrarians increases significantly both discovery and social interactions, that it means the agents in the swarm change more frequently their options. By continuously changing their option agents cannot reach consensus as quick as before the introduction of the attackers. An exception is made for the 3-Unanimity behaviour because it is not able to converge without attacker in this level of the problem. Contrarians can also perform a DOS if their percentage is very high, 20% or more, due to the fact that some behaviours, like CDCI and DMVD, are exposed to an high slow down that in some cases reach the maximum number of step per experiments that is set to 10000 and the swarm is not able to make a decision.

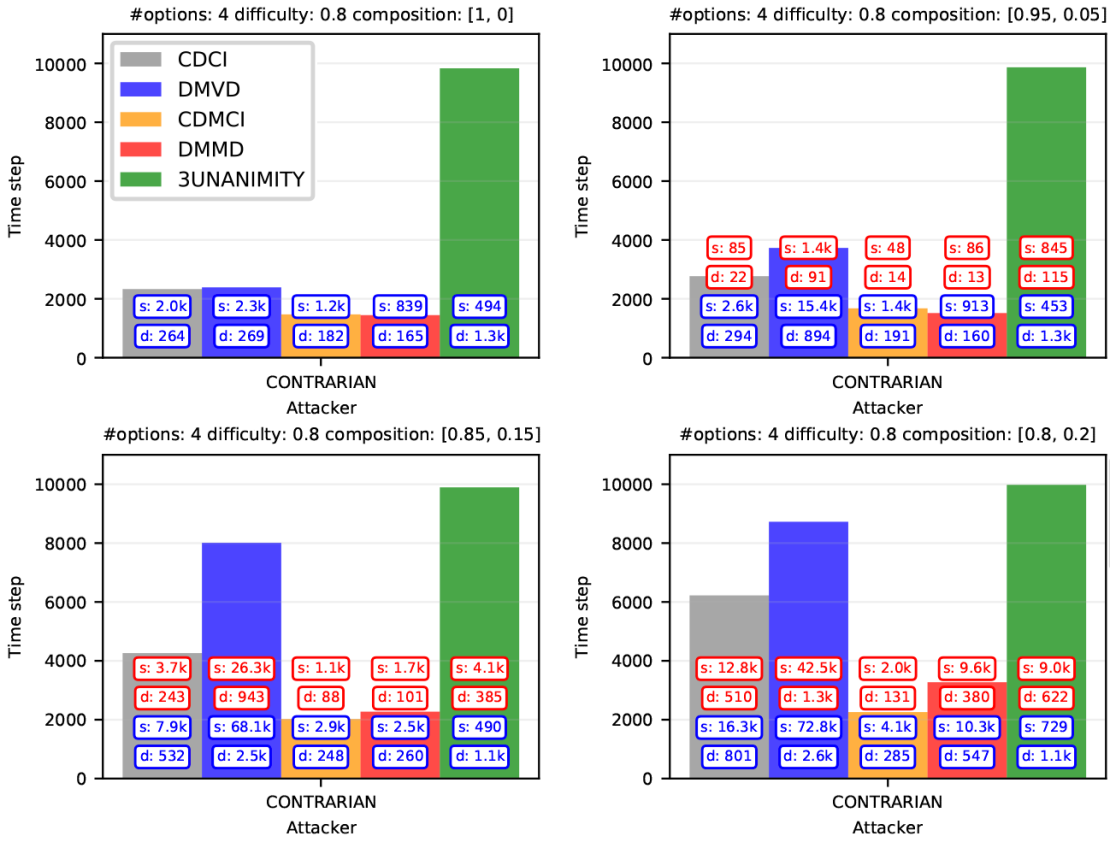

Figure 5.3: Bar charts representing a slow down performed by increasing the percentage of contrarians from 0% to 20%.

5.3.2 Sect executes wrong addressing

During the testing of the different types of attacker we discovered that zealots are performing DDOS instead of wrong addressing. This behaviour is due to the fact that they spread different options, with the correlated consequence of introducing an high level of noise, and are not able to redirect the swarm to a singular option. In order to solve this problem and create an attacker able to perform a wrong addressing a group of zealots that spread the same option is implemented and is called sect. In Figure 5.4 we depicted the differenent results of applying an attack composed by zealots spreading different options and an attack in which all zealots spread the same option (sect). In this plot we vary on the vertical axis accuracy and convergence, represented respectively by the brighter bars and the less bright ones. In each plot we fix the number of options (4), the difficulty(0.8) and the composition. Each different bar represents a literature behaviour as explained in the legend. From one plot to another the only parameter that is varied is the composition. These plots allow to represent the different effect performed by the attackers, when the number of attacker increase to 1% to 5% it is evident that the sect performs a wrong addressing because in each behaviour (excluding always 3-unanimity) the value of the convergence is very close to 1, means that they always reach a decision but to the wrong one, because the level of the accuracy is very low. An attack composed by "independent" zealots on the contrary increases the number of time steps and decreases the accuracy. This double effect means that the behaviours are not able to reach a consensus and they are stuck in the decision process.

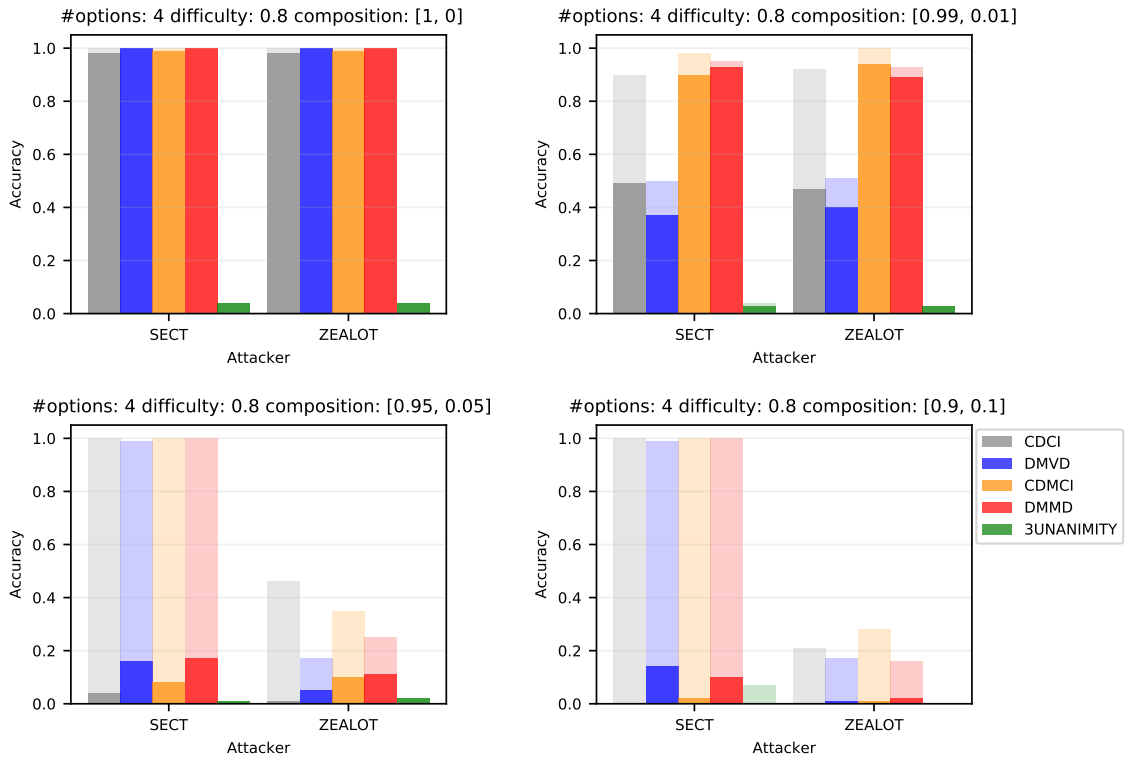

Figure 5.4: Comparison between the effects of sect and zealot. Zealot performs a denial of service, while sect performs a wrong addressing.

In the following Figure 5.5 from the previous Figure 5.4 the only parameters that is changed is the number of options (from 4 to 2) for visualising also the behaviour of 3-unanimity. It is also added the social and discovery interactions represented by the blue rectangles. In this case contrary to the one represented in Figure 5.3, red rectangles are not showed due to the fact that zealots never change their option and they never perform discovery and process social. With this plot a particular behaviour is depicted. From a number of intruders from 0 to 1 we notice a considerable increase in social interaction due to the fact that the agents tries to contrast the attacker by sharing more their options. Observing the differences between the second and the fourth plot the number of social interactions decreases due to the fact that the swarm cannot raise against the influence of the attacker and agents converge quickly to the wrong option.

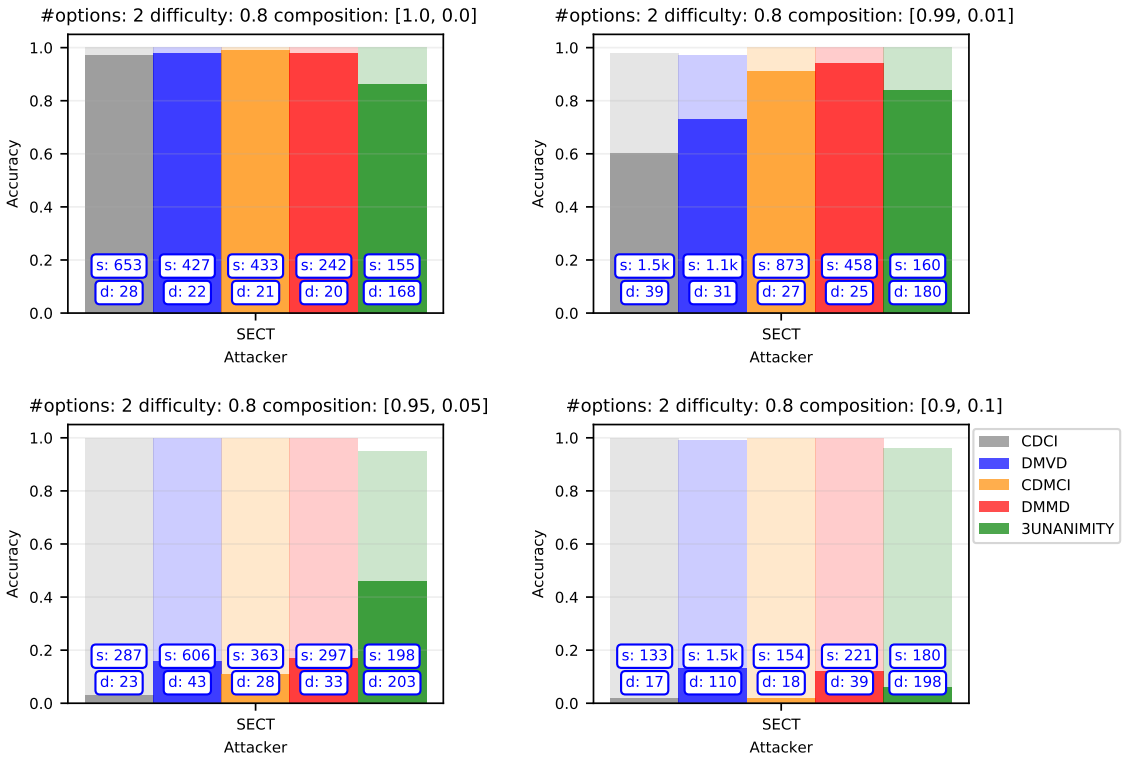

Figure 5.5: Bar charts showing that sect can perform wrong addressing.

Another proof of this observation is given by the graphs represented in Figure 5.6. In these plots on the y-axis we vary time steps, accuracy and convergence respectively represented by the coloured lines (each colour refers to a different literature behaviours as it is shown in the legend) and the correspondence ticks shown in the legend. From a plot to another we fix the number of options and the type of the attacker and we vary the difficulty. In the first plot that represents the simplest case in which there are only two options and the difficulty is low an unusual behaviour is highlighted when the number of attackers switch from 1% to 5% and from 5% to 10% this fact as said before can be explained due to the fact that until the number of zealots composing the sect does not reach 10% of the population the various behaviour tries to naturally contrast the attacker. As a result, their accuracy is better than the next case but their speed is far slower. When the percentage of attacker reach the 10% or more the speed of the convergence process increases, but the accuracy dramatically decreases, because the swarm converges very quick to the wrong option. Another showed results is that increasing the difficulty of the problem from 0.4 to 0.8 we notice that the behaviours stop to try to contrast the attacker earlier, due to this evidences the power of the sect is increasing when the difficulty level of the problem increases.

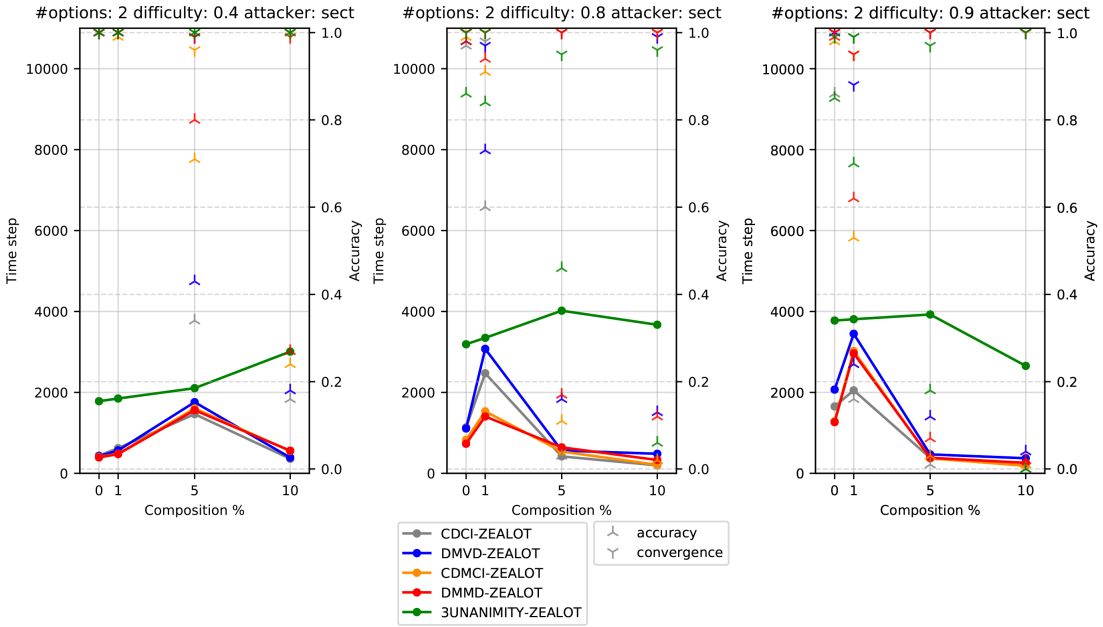

Figure 5.6: Plots showing how each behaviour reacts under different percentages of composition (percentage of attackers in the swarm).

5.3.3 Wishy Washy accomplishes a denial of service

A wishy washy introduces in the swarm an high level of noise. This fact is due to its behaviour of continuously changing its selected option that is constantly spread through its neighbours. Spreading a lot of noise in problems where the difficulty is quite high (0.8) is able to perform a denial of service. In the bar charts showed in Figure 5.7 on the y-axis we vary accuracy and convergence respectively represented by the brighter bars and the darker ones, on the x-axis we vary the attacker, but in this case the only considered one is the wishy washy. In each graph we fix the number of options (4), difficulty (0.8) and we vary the composition between different plots. In the first graph we show that each literature behaviour except 3-unanimity is able to converge with an accuracy value near to 1. In the second plot we introduce only one intruder the DMVD behaviour starts to have problems of convergence. In the graphs of the second row the number of wishy washy increases and reach 5% and 10% of the swarm composition, these experiments demonstrate that just a bunch of wishy washy are able to perform a DOS. In the third graph all the behaviours are affected by the intruder and the best is able to reach a consensus only on the 60% of the simulations. In the last one the number of the malicious-agents reach the 10% and the best behaviour reaches a consensus only on the 20% of the runs, this demonstrates that the wishy washy is able to achieve the goal it is made for.

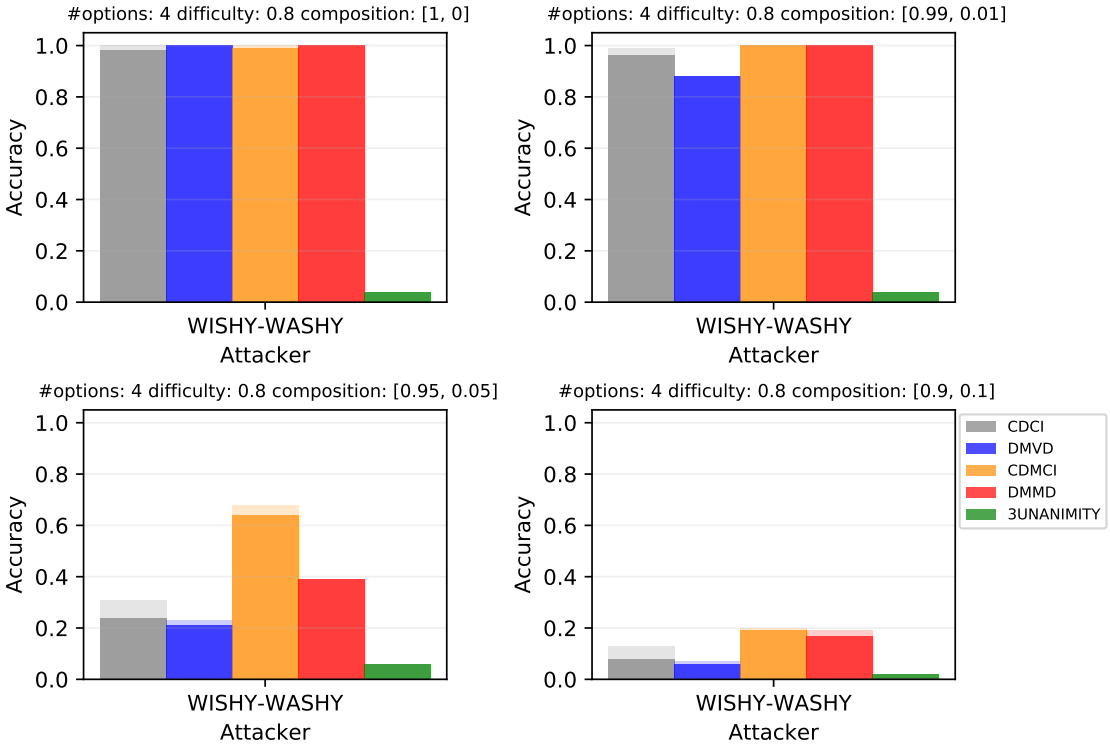

Figure 5.7: Bar charts showing that the attacker type called wishy washy is able to perform a denial of service.

During the analysis on the data we discovered that the power of a wishy washy increases when the number of the options increases, because it can spread more noise. The previous statement is demonstrated in the Figure 5.8. In the x-axis we vary the types of attackers, while on the y-axis we vary accuracy and composition. In this triptych we maintained constant the difficulty and the composition and from top to bottom we increased the number of options. These plots show that also in simple problems where the difficulty is set to 0.4, increasing only the number of options in the environment the intruder increases considerably its power and its able to achieve its goal.

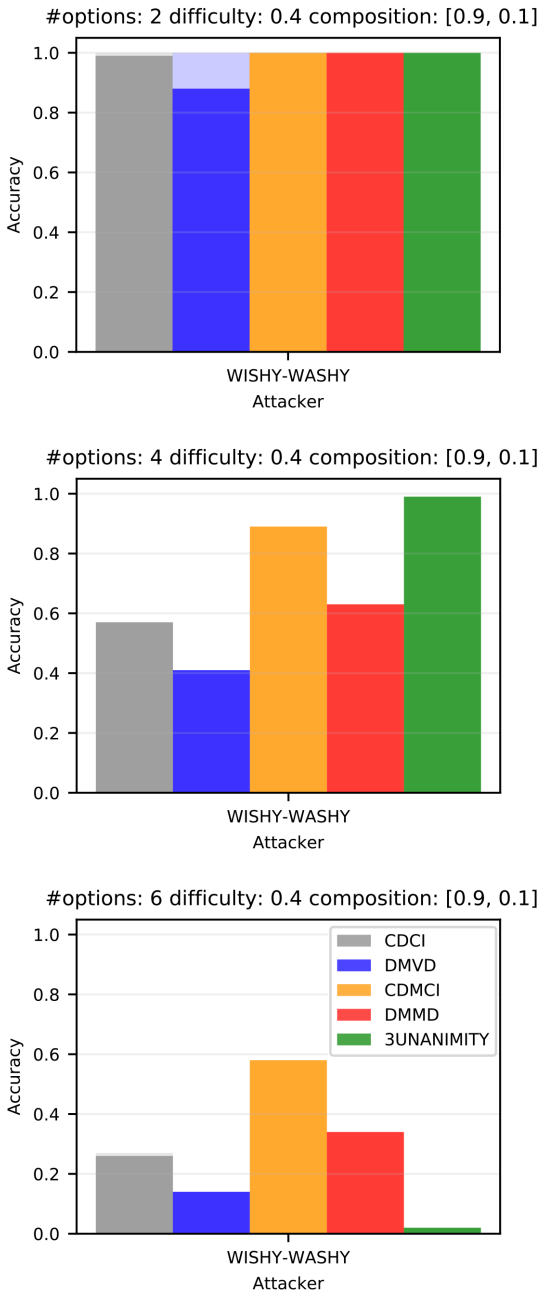

Figure 5.8: Series of plots showing that a wishy washy introduces more disturb and perform a denial of service if the number of options increase.

5.3.4 In simple problems, listening less is better

One of the main results is characterised by the reaction of the k-unanimity behaviour in simple problems. In our experiments setting as described in Section 5.1.1 we decided to set k equal to 3, it means that an agent picks a new option from its neighbourhood if and only if the last 3 received options are equal. This kind of behaviour is the most selective one, because it needs a lot of information for using an information from its neighbourhood, due to this selectivity 3-unanimity is the behaviour that has the worst performance in absence of attackers in both simple and complex cases. These bad efficiency is showed in both Figures 5.9 and 5.10. In both figures on the x-axis we vary the types of attacker and on the y-axis we vary time step, accuracy and convergence, pictured by coloured bars, grey and light-blue lines. In Figure 5.9 from top to bottom we vary the difficulty and it is clear that 3-Unanimity is the slower and the one with less accuracy compared to all the other behaviours. As showed in Figure 5.10 it is the worst also in the case in which instead of changing the difficulty from one plot to another we change the number of options.

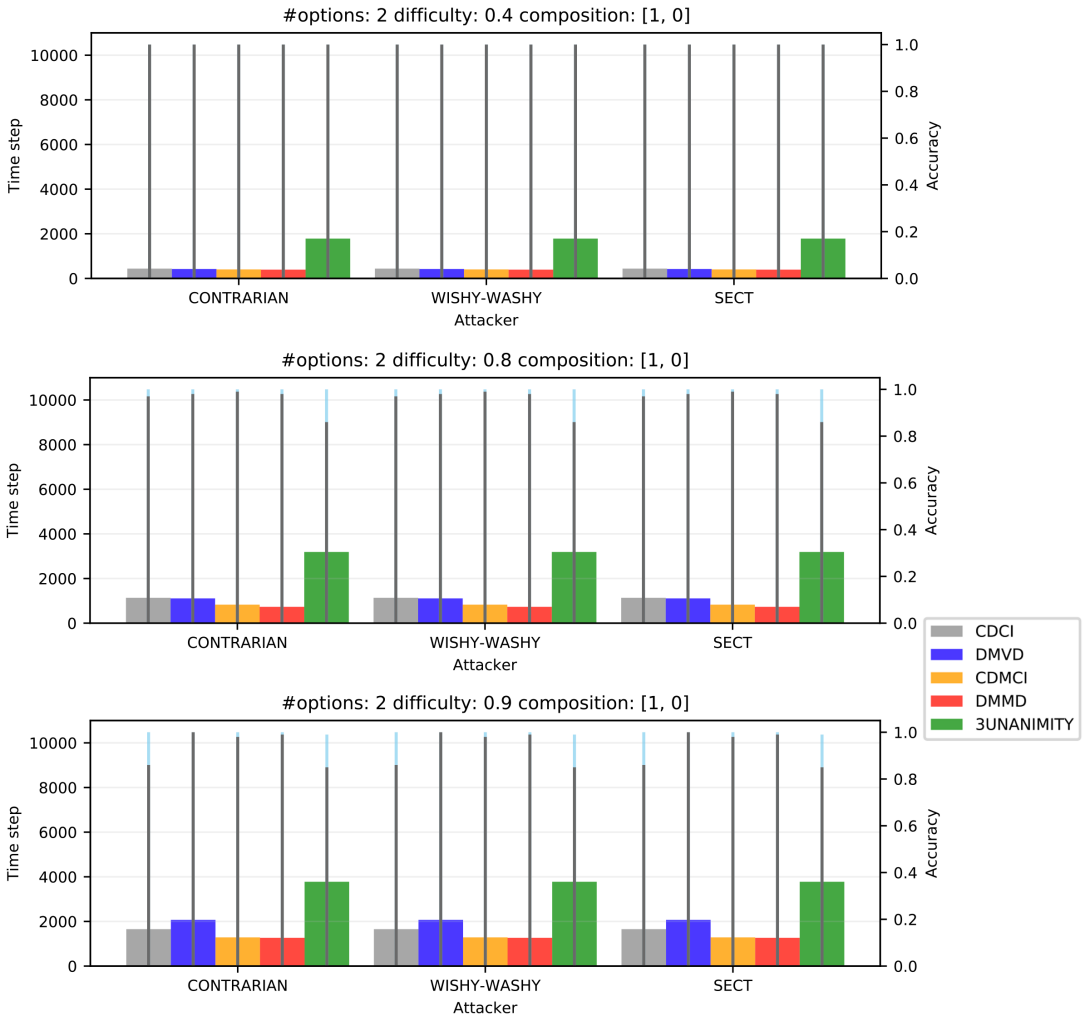

Figure 5.9: Comparison of behaviours varying difficulty. In this plot we show that 3-unanimity is the slower and the one with the lowest level of accuracy in problems with number of options equal to 2 and varying the difficulty from 0.4 to 0.9.

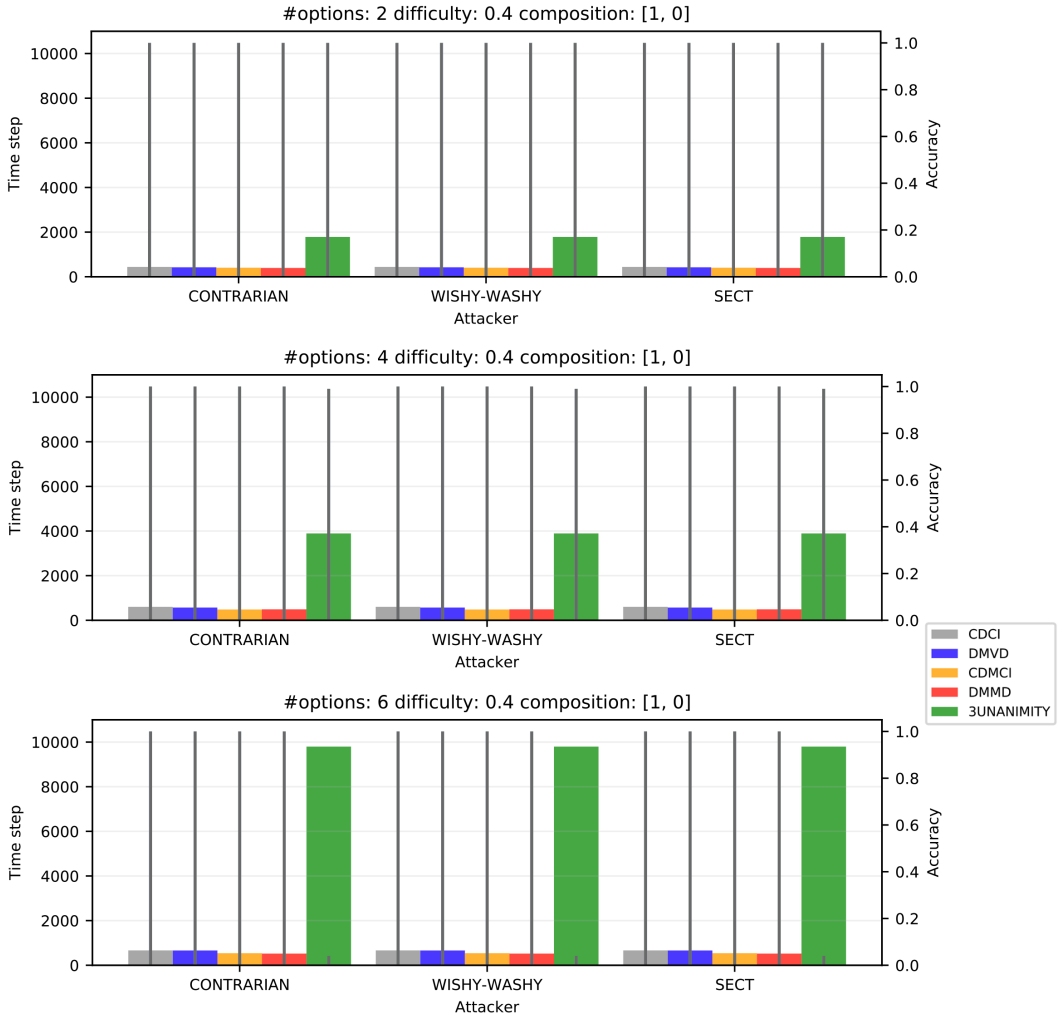

Figure 5.10: Comparison of behaviours varying number of options (2,4,6). In these plots we showed the slowness of 3-unanimity in scenarios with the simplest analysed level of difficulty (0.4).

Analysing the last two graphs we also notice that the only case in which 3-unanimity has the same accuracy of the other behaviours is the simplest case in which the number of the options is equal to 2 or 4 and the difficulty is very low and set to 0.4. Unexpectedly when we plotted and examined the collected data we notice that in this particular scenario 3-unanimity is the only one behaviour that has a very high resilience through all the different types of attacks and it is the only one that is able to face a wrong addressing attacks performed by a sect. In Figure 5.11 we showed the high level of resiliency that characterises 3-Unanimity in these simple cases. In these plots we vary on the vertical axis accuracy and convergence, depicted respectively with brighter and darker bars, and on horizontal axis the different types of attacker. In each plot we fix the number of options (2), the difficulty value (0.4) and the composition of the swarm. The only parameter that we vary from a plot to another is the composition. The plot positioned on the top left corner represents the case in which the percentage of attacker is 0% and on the bottom left we depicted the case in which the number of attacker is maximum and set to 15%. The main difference that is showed is that the only attacker that has an impact on the resiliency in these cases is the sect, that is able to perform a wrong

addressing against all the behaviours except 3-unanimity. This weak behaviour has a very high resiliency in simple scenarios because its ratio between discovery and social interactions is completely different from the other behaviours. In order to show this ratio we need to plot the social and discovery interaction per each behaviour.

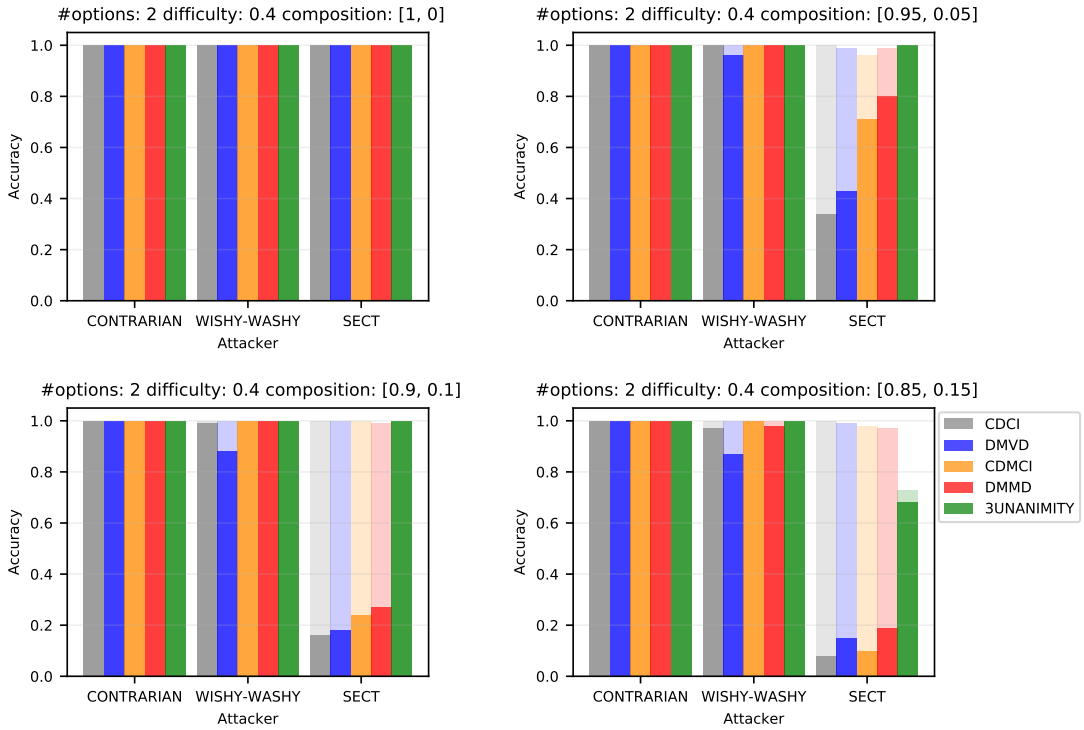

Figure 5.11: In this series of plots we vary the percentage of attacker and it is showed that 3-Unanimity in this simple scenario is able to resist against every kind of attackers.

In the following Figure 5.12 we plotted the discovery and social interaction values per each literature behaviour. On the x-axis we report the attacker and on the y-axis we vary time steps, accuracy and convergence respectively represented by coloured bars, grey and light blue lines. In each one of these plots we fixed the number of options (2), difficulty (0.4) and composition. From one graph to another the only component that is varying is the composition. Thanks to the discovery and social interaction values we highlighted that the discovery interaction of 3-Unanimity is far higher compared to the other behaviours and also the social interactions are way lower than the average of the others. These values mean that 3-Unanimity needs more information in order to apply a process social and due to the noise introduced by the sect it cannot apply most of the time its social behaviour and adopt more frequently discovery. Thanks to the simplicity of the problem, characterised by a lower number of options (2) and a very low difficulty (0.4) and to a probabilistic discovery function related to the estimated quality of the option the probability of discovering the correct options is far higher than the probability of the second option. This fact permits to 3-Unanimity to converge to the correct option only using discovery.

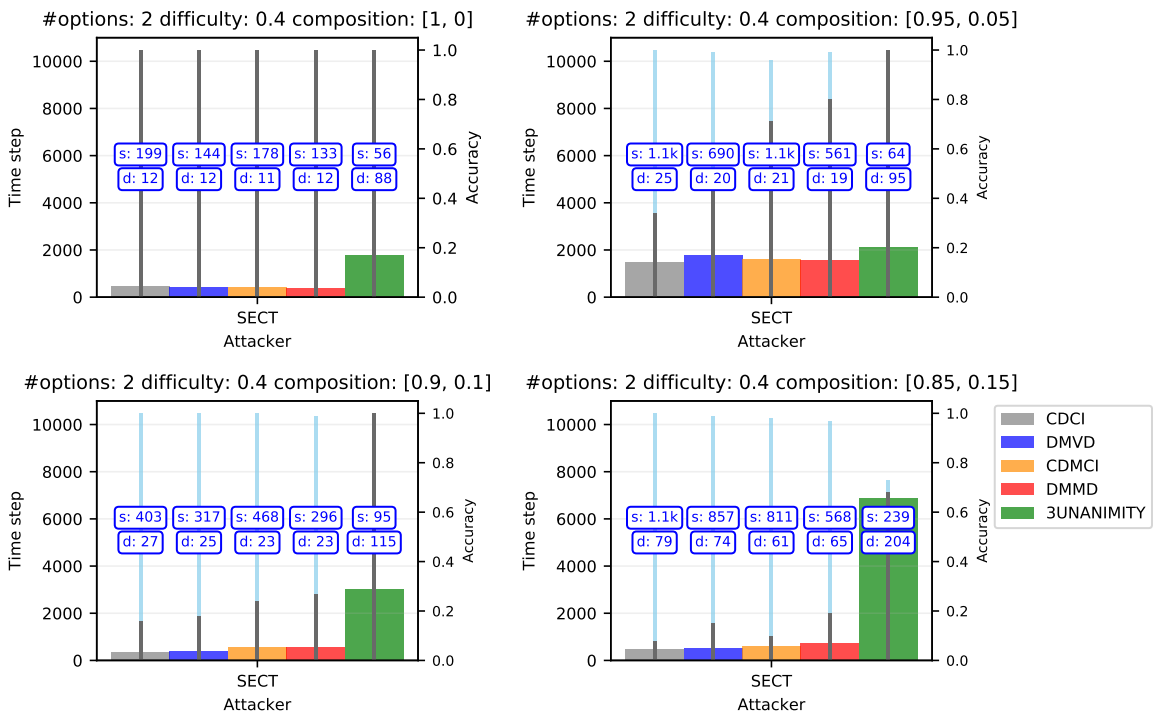

Figure 5.12: In this plots is showed that 3-unanimity is the only behaviour that is able to resist against sect in simple case with number of option equal to 2 and a difficulty level set to 0.4.

5.3.5 Majority contribution

Another important result that we obtained is the contribution given by the majority module in increasing the resiliency of a behaviour. As described in Section 2.1 a majority behaviour is characterised by the selection of a new option as the one shared by the majority of individuals in the neighbourhood of the agent. This behaviour allows the agent to use more information than a random behaviour. Thanks to this advantage of considering more knowledge behaviours using majority are less vulnerable to noise. In the following list of plots we explain in details the differences between behaviours using as a social interaction a random behaviour and behaviours adopting a majority strategy. In the plots of Figure 5.13 we analyse DMVD and DMMD.

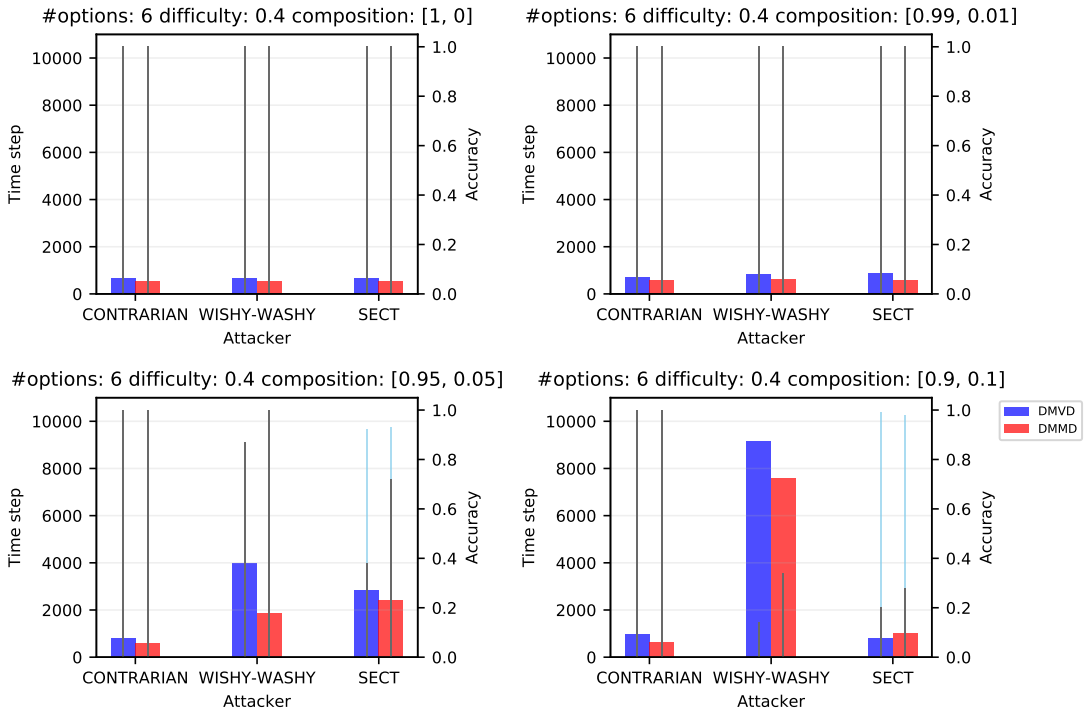

Figure 5.13: DMMD and DMVD comparison bars plot showing that DMMD is more resilient thanks to the use of the majority module.

The only difference between these behaviours is characterised by the different social process phase. DMVD uses a random module, while DMMD selects a majority one. Both behaviours using a probabilistic discovery and sharing of messages related to the estimated quality and direct switch as update model module. In this series of plots in Figure 5.13 on the vertical axis we vary time step, accuracy and convergence represented respectively by coloured bars, where red represents DMMD and blue represents DMVD, grey and light blue lines. On the horizontal axis we vary all the different type of attackers. In each plot we fix the number of options (6), difficulty value (0.4) and composition. From one plot to another we vary only the percentage of attackers represented by composition values. The represented case is characterised by a number of options equal to 6, in which there is only one with an high quality (8) and all the other has lower quality (3.2). In this plot we show that a majority process social helps in increasing resilience resulting in a faster decision process and

in a higher level of accuracy. We can notice that in the second row DMMD is performing in all the cases better than DMVD both from a accuracy and time step point of view. Especially DMMD has a higher performance under the influences of wishy washy, sect and zealot. Increasing the difficulty of the problem, we notice also that DMMD is performing better also in contrarian case. In the following Figure 5.14 on the x and y axis we keep the previous metrics used in Figure 5.13 and from a plot to another instead of varying the composition, here fixed to [0.9, 0.1], we vary the difficulty, from 0.4 to 0.9.

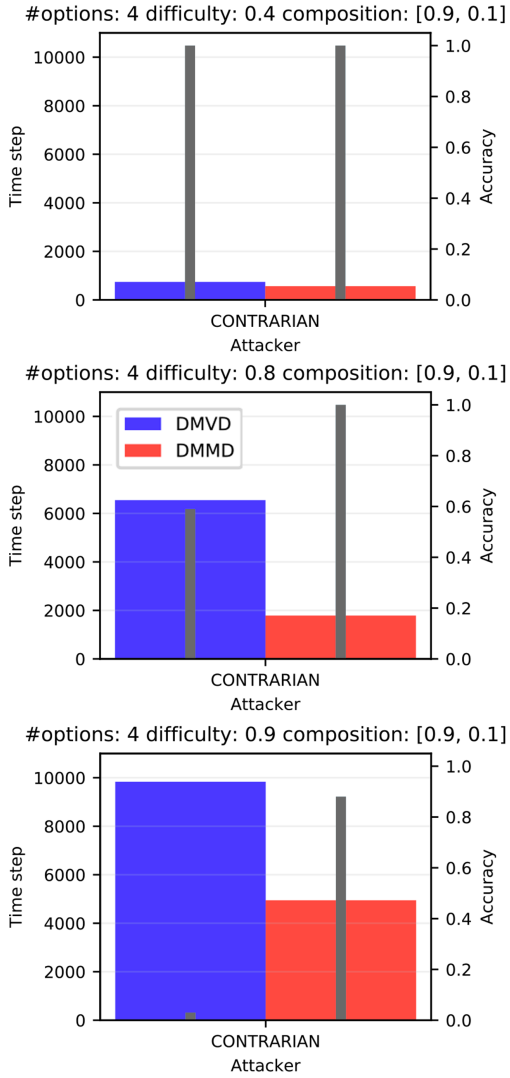

Figure 5.14: DMMD and DMVD comparison bars plot showing that increasing the difficulty level DMMD is able to resist more than DMVD against contrarian.

In this triptych is show that also in the contrarian case if we compared the two behaviours in cases where the difficulty is very high DMMD has a better resiliency.

In this second series of plots we depict the differences between CDCI (represented in grey) and CDMCI (represented in orange). Both behaviours using cross-inhibition as a update model module and a probabilistic module in discovery and create message. The only difference is the process social module. CDCI uses a random, while CDMCI a majority module.

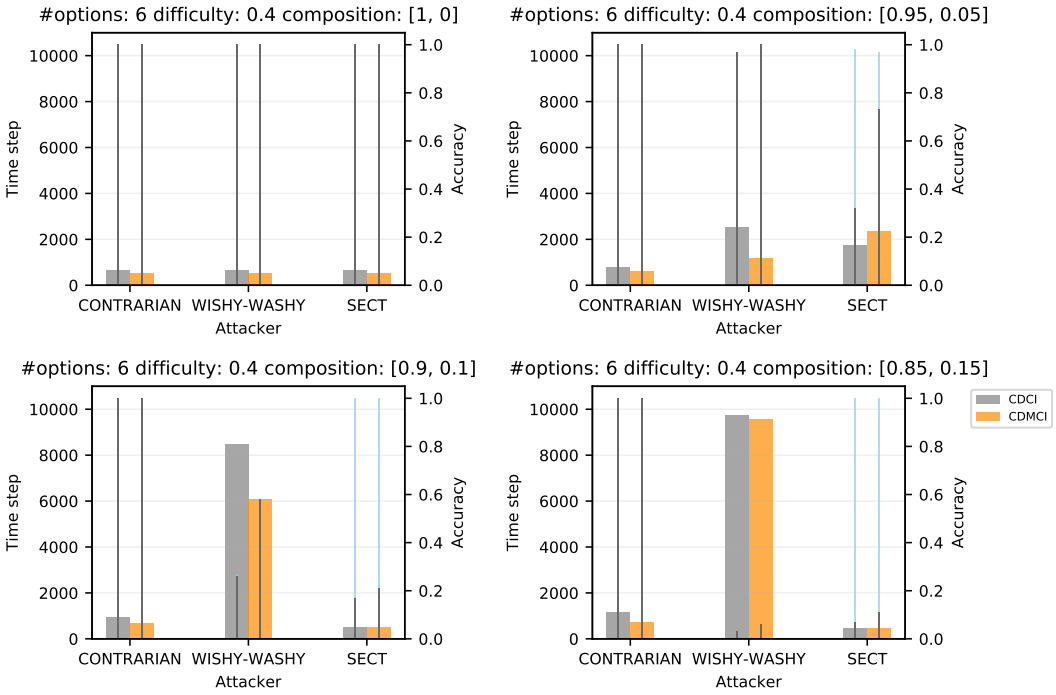

Figure 5.15: CDCI and CMDCI comparison bars chart showing that CDMCI is more resilient in most of the feasible scenarios against contrarian, wishy washy and sect until the number of attackers reaches 10%.

In Figure 5.15 on the vertical and horizontal axis we vary the same values as in Figure 5.13. When the number of attackers reaches the 5%-10% we notice that CDMCI is faster and has a higher level of accuracy. Until the number of zealots composing the sect is below 10% CDMCI is also able to react against sect. In the third graph a considerable difference in accuracy is showed when a sect is introduced in the swarm. When the number of zealots composing the sect is greater than 5% no behaviour is able to react to it in problems when the number of options is greater than 4. In the third plot also wishy washy provokes a decrease in accuracy, because it introduces more noise in the system because the number of options is very high and it can switch over a high number of different options. In this case we show that CDMCI has an accuracy to 60%, while CDCI equal to 23% and CDMCI is also faster.

Due to the fact that contrarians are the weakest attacker because their only goal is to perform a slow down, in order to show that CDMCI is also way better than CDCI in the contrarian case we need to increase the level of difficulty. In Figure 5.16 on the x and y axis we vary the same elements as the previous plots and is fixed the composition to [0.9, 0.1] and number of options equal to 6. From one plot to another we vary only difficulty from 0.8 in the first graph to 0.9 in the second one. As can be noticed by the high level of the difficulty values in these plots we report two of the most complex cases that we analyse. Thanks to this complexity in these plots we are able to detect that CDMCI is faster and has a higher accuracy compared to CDCI, meaning that CDMCI is more resilient also against contrarians. The difference in accuracy between the two behaviours is very high from 30% to 40%. In the first plot CDCI has an accuracy around 77% while CDMCI has 100%, increasing the difficulty to 0.9 CDCI decreases to 22% while CDMCI has 65%. CDMCI is also faster than CDCI, in the case with a level of difficulty equal to 0.8 the average number of time steps is around 2300, while for CDCI is around 6250. In the most difficult explored scenario CDMCI has an average of time steps equal to 6700, while CDCI equal to 9000.

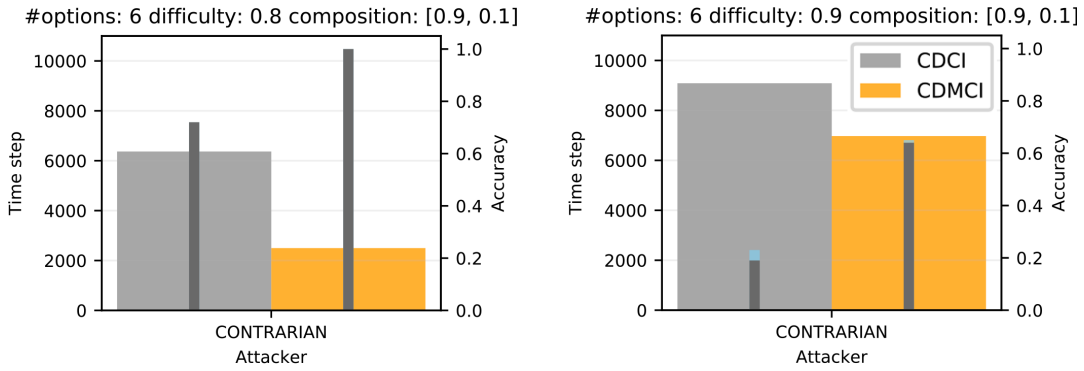

Figure 5.16: CDCI and CMDCI difficulty comparison bars chart showing that in very difficult scenarios characterised by a number of options equal to 6 and a difficulty level equal to 0.8 and 0.9 CDMCI has a higher level of resiliency than CDCI.

5.3.6 Cross-inhibition contribution

As said before in Section 5.3 another important result is characterised by how cross-inhibition module contributes in increasing resiliency. Cross-inhibition in DeMaMAS in considered as an update model module. Its behaviour can be explained through the Figure 5.17. This module is responsible on how an agent updates its own option after the selection of a new one. This module does not permit to directly switching to a new option (different from its previous one), but imposes a constraint to pass through the uncommitted phase u.

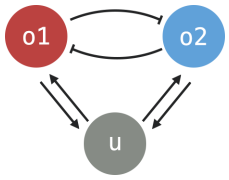

Figure 5.17: The motif represent a macro view of how cross-inhibition module works in an experiment with 2 options, O_1 represent the first option and O_2 represent the second available option, U represent the uncommitted state.

In the following series of plots we show that in our most difficult explored scenarios the cross-inhibition module increases the resiliency against contrarians for all the different percentages of attackers (1%, 5%, 10%) and against wishy washy and sect, only if the percentage is less than 5%, than using a classical direct switch, which allow the agent to change its own option to the new selected one, without passing through the uncommitted state. In each graph on the x-axis we vary the type of attackers, on the y-axis we vary time steps, accuracy and convergence represented respectively by coloured bars, grey and light blue lines. In each graph we fix number of options, difficulty and composition. From one graph to the following one we vary only the composition of the swarm. In these plots we show that in complex problems represented with a high number of options (6) and high level of difficulty (0.8 and 0.9), cross-inhibition helps to perform better, in both speed and accuracy.

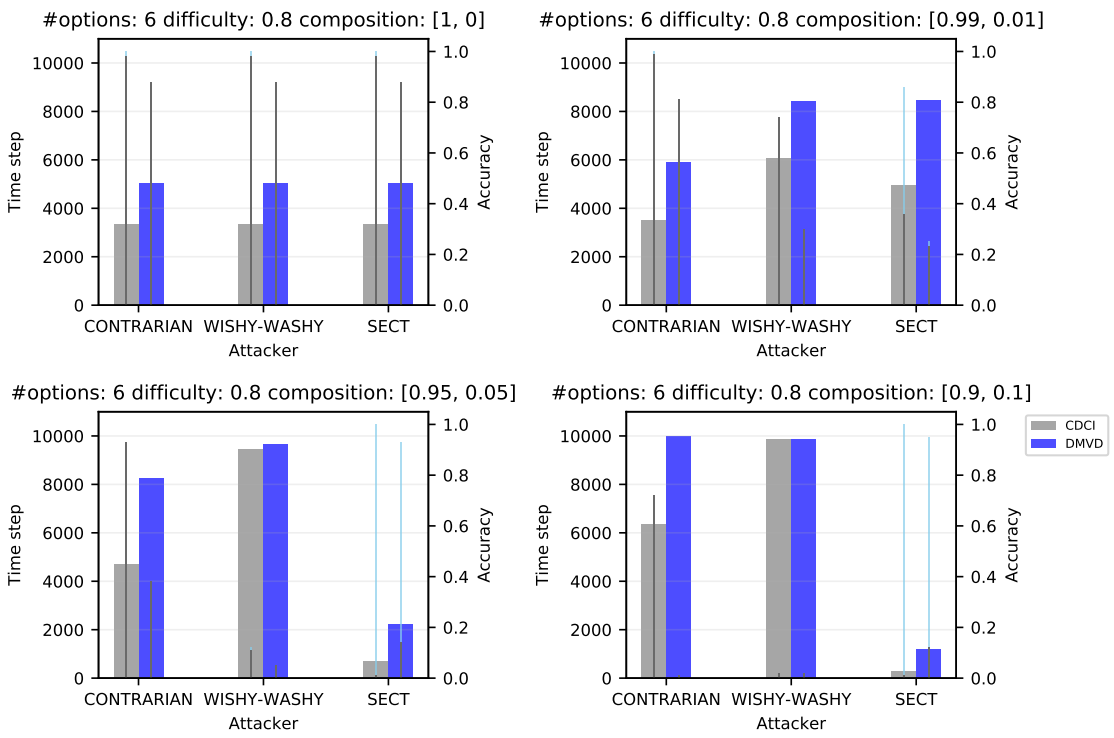

Figure 5.18: CDCI and DMVD comparison bar charts showing that in complex cases with number of options equal to 6 and difficulty level equal to 0.8 CDCI performs better than DMVD.

In the first series of Figure 5.18 we compare CDCI and DMVD, in which the only difference between the two behaviours is that CDCI uses cross-inhibition and DMVD uses direct switch as update model module. They both use a probabilistic discovery module and create message. As showed in the legend CDCI is represented by the grey bars, while the DMVD is displayed by the blue bars. When the percentage of attackers is 1% the CDCI is faster and has a higher accuracy and convergence for each type of the attackers. When the number of attackers increases and reaches 5% and 10% in this type of complex scenario no one of the two behaviours is able to react against wishy washy and sect, but only CDCI is able to have a significance level of accuracy against contrarians, as showed in the second and third plots. Against the other types of attackers in complex cases as showed in this series of plots, both CDCI and DMVD have not a good resiliency due to the fact that in these particular cases as said in Sections 5.3.2 and 5.3.3 these kinds of attacker increases a lot their performance and no literature behaviours is able to resist under attack.

In the following picture we analyse the most complex case, in which the number of options is equal to 6 and the difficulty is set to 0.9. In the first plot in which the number of attackers is equal to 0% we show that our new behaviour CDMCI has a speed that is twice faster than DMMD and has an accuracy that is twice higher compared to the one of DMMD. If we increase the percentage of attackers to 1% we notice that CDMCI is still the best one. Against contrarians CDMCI is faster and has an accuracy of 93% against a 50% of DMMD. Against wishy washy it is faster and has an accuracy equal to 60%, while DMMD has only 20%. In facing sect, CDMCI and DMMD have the same low accuracy equal to 20%, but CDMCI is faster and has a convergence of 78%, while DMMD has only 36%. With the last two plots in which the number of attackers increased to 5% and 10% neither CDMCI and DMMD are able to resist to wishy washies and sect, but CDMCI has an higher accuracy and speed against contrarians. In the case in which the percentage of attackers is equal to 5% CDMCI is faster and has a better accuracy equal to 81% while DMMD has 40%. In the last scenario in which the number of intruders is set to 10% CMDCI is still faster and has a better accuracy equal to 62% while DMMD has a low value equal to 18%.

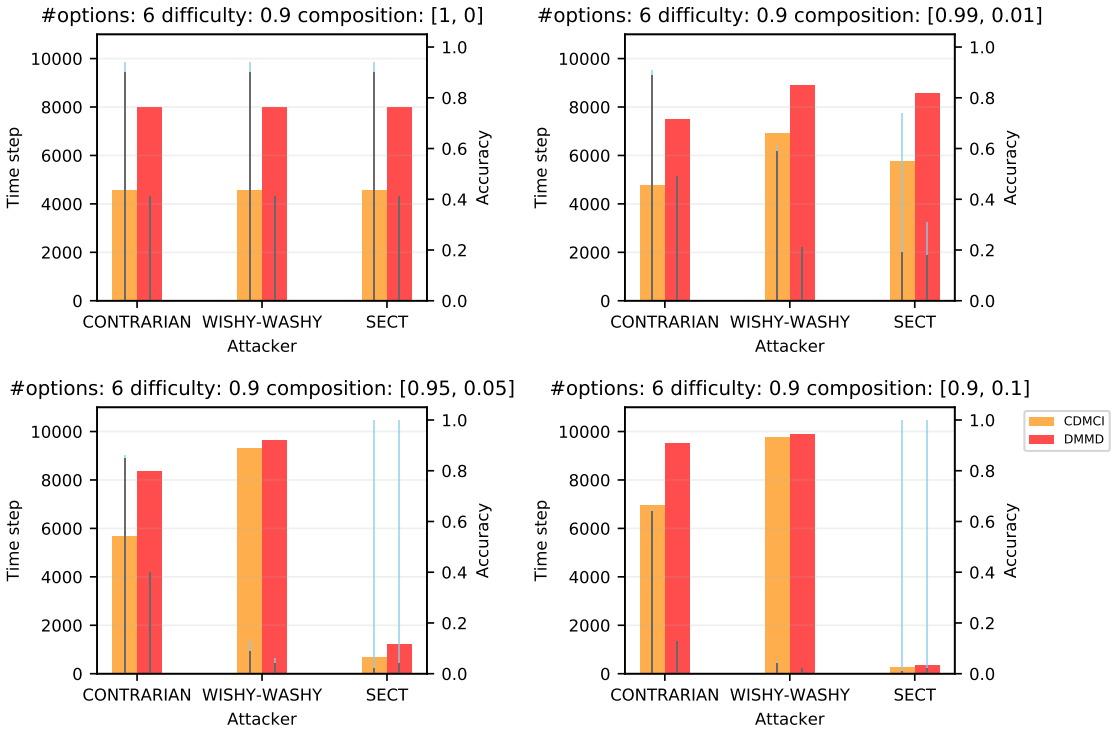

Figure 5.19: CDMCI and DMMD comparison bar charts showing that in the most complex scenario with a number of options equal to 6 and level of difficulty equal to 0.9 in some feasible cases CDMCI performs better than DMMD.

Thanks to all the previous analysis we discover that CDMCI, that uses both cross-inhibition and majority modules, in most of the scenarios where the other behaviours suffer more has a higher resilience. In the following plots we analyse some of the most interesting cases in which we have complex problems, related to the high percentage of attackers, number of options and difficulty and we show that CDMCI has better performance than the other behaviours against contrarians and

wishy washy, because related to this high complex problems no one is able to face with sect. In the first plot of Figure 5.20 we show how CDMCI reacts in different cases against contrarians. On the y-axis we vary the average number of time steps, accuracy and convergence represented respectively by the coloured bars, the grey lines and the light blue lines. On the x-axis we report the analysed attacker. In the two plot we fix the number of options (6), the difficulty level (0.8), and the percentage of attacker equal to 15%. In this two plots we show that CDMCI is the only behaviour that is able to resist to contrarians maintaining an high level of accuracy. In the first plot CDMCI has an accuracy near to 100% while CDCI has an accuracy near to 40%, DMMD near to 78% and DMVD is the one that suffers more, with an accuracy equal to 0%. In the second plot that represents one of the most complex analysed scenario our novel behaviour has an accuracy around 50% while all the other behaviours as an accuracy around 0%.In both plots CDMCI is also the faster. Against contrarians we can confirm that in all our explored cases CDMCI is the one with the highest resiliency.

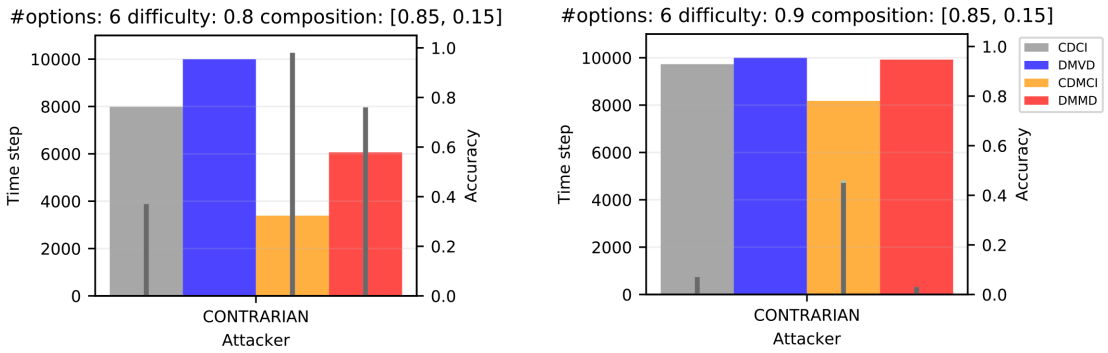

Figure 5.20: Comparison of different behaviours in a complex case with a percentage of attacker equal to 15% showing that CDMCI is the one that is more resilient against contrarians.

In the next triptych of plots in Figure 5.21 we show that in particular complex cases CDMCI is able to resist more than the others behaviours.

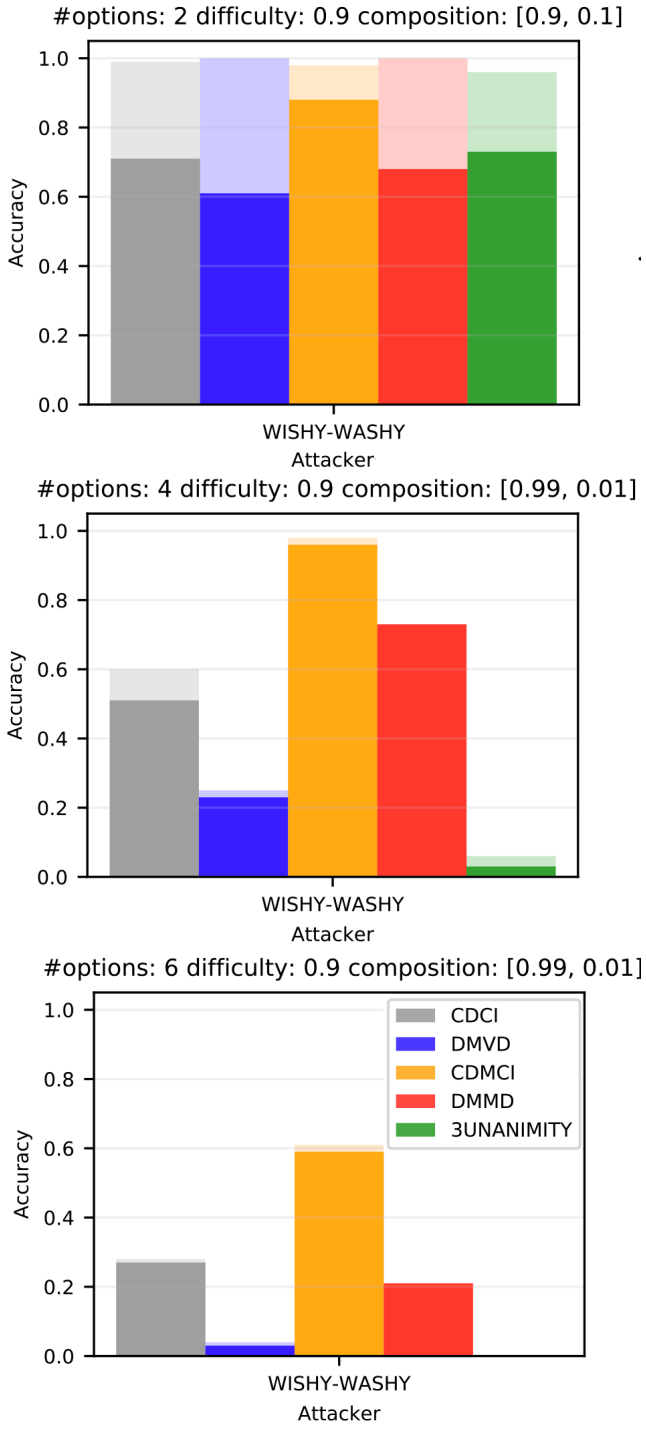

Figure 5.21: Comparison of different behaviours in different scenarios against wishy washy. This plots showing that is some cases where all the behaviours are less resilient, CDMCI is the one with the highest level of resiliency.

On the vertical axis we vary accuracy and convergence depicted respectively with brighter and darker bars. Each bar represents a different behaviour as explained in the legend. In all the different plots we fix the difficulty level, set to the maximum and equal to 0.9. From top to bottom we increase the number of options from 2 to

4 and 6. In the first plot the percentage of attackers is equal to 10% while in the other two plots is 1%. In the first plot CDMCI has an accuracy around 90% while CDCI and 3-Unanimity around 72%, DMMD around 68% and DMVD around 61%. In this first case with the high difficulty level of the problem (0.9) and a low number of options (2) the group of wishy washy is not able to perform a proper denial of service due to the limited number of options and all the behaviours are able to converge. Increasing the number of options to 4 in the second plot a wishy washy (1%) increases its ability to introduce more disturbance in the swarm and all the behaviours except CDMCI are not able to face the disturbance introduced by only one wishy washy. In this second plot CDMCI has a high level of accuracy around 98%, DMMD around 75%, CDCI around 52%, DMVD 21% and 3-Unanimity near 0%. In the last plot all the behaviours suffer on the introduction of only 1 wishy washy, because the number of disturbances introduced is very high related to the number of options equal to 6. CDMCI has an acceptable level of accuracy around 60%, CDCI around 23%, DMMD around 20%, DMVD and 3-Unanimity around 0%. CDMCI is the only behaviour able to resist to the difficulty of the problem and the high level of disturbance introduced by the wishy washy.

Chapter 6

Attacks in real robots

In Chapter 5 we discussed how we set up the experiments and discussed the obtained results. Based on the results we decided to reproduce four different scenarios in real robots. Thanks to the opportunity given by the BET Lab (behaviour Evolution Theory Laboratory, University of Sheffield, UK) we used simple robots called Kilobots for implementing and testing the selected scenarios. In the following sections we explain Kilobots, the Augmented Reality system for Kilobots called ARK, how we set up the experiments, and the obtained results.

6.1 Kilobot

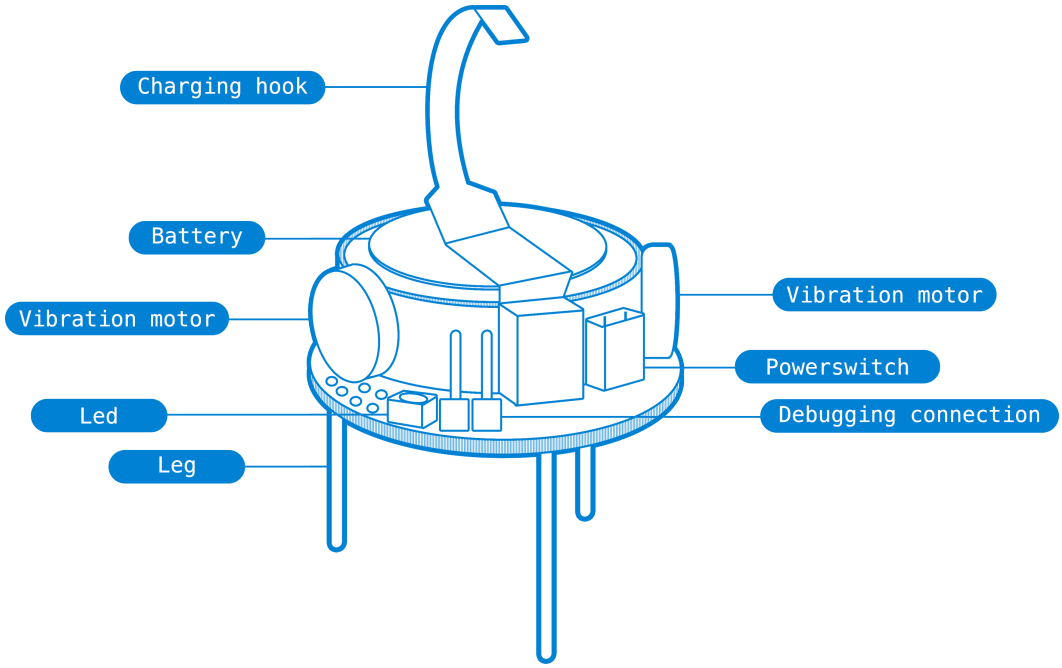

Figure 6.1: Kilobot sketch.

Kilobot is a simple robotic system for the development of swarms of robots. A Kilobot is easy to program and can perform useful functions such as coordination between many individuals. Another of its main characteristics is represented by its low cost. In the field of swarms robotics the cost factor is essential due to the fact that a swarm is composed of several robots. Its low cost is due to its simple and cheap components. As depicted in Figure 6.1 it is composed of three legs that enable the robot to move thanks to the vibrations crated by the two different vibration motors taking the energy from the battery. It also has a debugging connection and a power switch. All the complete details about its components are listed in the next table taken from the Kilobot website: https://www.k-team.com/mobile-robotics-products/kilobot. The Kilobot is designed to provide scientists with a physical testbed for advancing the understanding of collective behaviour [28].

6.1.1 Specifications

Elements	Technical informations
Processor	ATmega 328P (8bit @ 8MHz)
Memory	32 KB Flash used for both user program and bootloader, 1KB EEPROM for storing calibration values and other data and 2KB SRAM
Battery	Rechargeable Li-Ion 3.7V. Each Kilobot has a built-in charger, which charges the on board battery when +6 volts is applied to any of the legs, and GND is applied to the charging tab.
Charging	Kilobot charger. Time for charge is about 3 hours.
Communication	Kilobots can communicate with neighbours up to 7 cm away by reflecting infrared (IR) light off the ground surface.
Sensing	When receiving a message, distance to the transmitting Kilobot can be determine using received signal strength.
Movement	Each Kilobot has 2 vibration motors, which are independently controllable, allowing for differential drive of the robot.
midrule Light	Each Kilobot has a RGB LED pointed upward
Dimensions	The diameter is 33 mm and the height is 34mm
Software	The KiloGUI interface is available for controlling the controller board, sending program files to the robots and controlling them.
Programming	The open source development software WinAVR combined with Eclipse gives a C programming environment.
Debug	A serial output header is available on each robot for debugging via computer terminal.

6.2 ARK system

Due to the Kilobots low cost and their restricted range of capabilities, based on a single sensor, a lot of experiments encountered several problems, in order to avoid those issues an Augmented Reality for Kilobots system is implemented by the BET Lab (Behaviour Evolution Theory Laboratory) group in 2017 [24] [6] (another augmented reality system is proposed by Antoun et al. [1]). ARK increases the Kilobots capabilities, thanks to an augmented reality environment these simple robots are allowed to communicate among each other and access to their location and state. It also automates some necessary steps for the configuration phase of an experiment; e.g., positioning, motor calibration, ID assignment, these operations are typically laborious and time consuming when done manually. ARK permits also advanced functionalities e.g., log and record experimental data for subsequent analysis. The system is composed of three different components:

- *Base control station (BCS)*
- *Modified overhead controller (OHC)*
- *Overhead camera tracking system*

The BCS allows to manage tracking, communication, virtual environments and Kilobots. Its behaviour is divided in four sequential phases. In the first step the control system receives camera images from each of the four cameras placed on top of the Kilobot arena, due to a distortion of the images the system needs to correct and interpolate them in order to obtain a single image that covers the entire environment. Secondly the software permits the Kilobots to be identified and tracked independently. Thirdly BCS allows the user to send unique messages to each Kilobot related to their individual state (or the same message in broadcast). Last but not least, the software shows a Graphical User Interface (GUI) to allow the user to view the progress of the experiment and to configure the entire system. All the necessary instructions for implementing experiments can be found in the DiODe website: http://diode.group.shef.ac.uk/kilobots/index.php/ARK.

The OHC [20] provides full communication coverage of the arena, it allows the transmission from the BCS to the Kilobots and both from and to the Kilobots. The communication is guaranteed through infrared signals (IR). This implementation works well when robots run on a smooth, large and glossy table that helps IR signals reflect off the surface and reach the robot light sensor. In order to solve the shadows and communication uncertainties challenge another light source directed on the arena is added. The processing stage to send information to Kilobots is permitted by the construction of packets that are sent to the OHC.

The tracking phase is guaranteed through a four overhead cameras which are related to the BCS for processing purposes. The number of cameras can be changed according to the modularity of the software. The software is able to find the Kilobots in the arena and to perform an interpolation of the four images to create a single image that covers the whole arena thanks to openCV panorama stitching module. For more detailed information about each components of ARK please read [24].

6.3 Testing resilience in Kilobots

Before starting the experiments with Kilobots and testing resilience, due to the complexity of the system, we chose which literature behaviour and attacker need to be tested. We decided to implement CDCI and CDMCI as literature behaviours and contrarian as an attacker. We selected CDCI, CMDCI and contrarians to have a preliminary validation of our novel behaviour, CDMCI, and for demonstrating that contrarians perform a slow down. Before implementing the selected behaviours on Kilbotos we prototyped them in a Kilobot simulator called ARGoS [23] [22]. For implementing both literature and attacker behaviours on Kilobots we used the open source code available online:
https://github.com/DiODeProject/Time-Varying-CDCI/blob/master/KilobotCode/CDCI_HighInteractionRate(r%3D100)/agentCDCIlocal_r100.c.

We decided to use a swarm of 50 Kilobots in a squared environment (1mx1m) and reproduce 4 different experiment conditions with a level of difficulty set to 0.4 (with an highest quality of 8 and a lowest one of 3.2), a number of options in the environment equal to 2 and a quorum set to 80%. The 4 different conditions are represented by the following settings:

1. 50 Kilobots using a CDCI behaviour in absence of intruders.
   [Percentage of intruders equal to 0%]
2. 45 Kilobots using CDCI and 5 applying minority (contrarians).
   [Percentage of intruders equal to 10%]
3. 50 Kilobots using CDCMI in absence of intruders.
   [Percentage of intruders equal to 0%]
4. 45 Kilobots using CDMCI and 5 applying minority (contrarians).
   [Percentage of intruders equal to 10%]

The swarm of Kilobots apply the same behaviour we used in DeMaMAS. The Kilobots are free to move in the Kilobot arena using a random walk and are allowed to communicate with each other exchanging only their option within a communication radius set to 10cm. They store only one option per time, and if their new option is picked from their neighbours they need to estimate the quality of the new option by go resampling to the corresponding totem. The go resampling action is possible tanks to the use of ARK that every 5s sends to the corresponding Kilobot the correct direction for reaching the totem. Due to the calibration problems of Kilobots this continuous updating of the direction is necessary. During the resampling time, as in DeMaMAS, they are not allowed to speak and listen. ARK allows also the discovery phase by sending the corresponding quality to the Kilobot in the communication area of the option. After received the quality from ARK the Kilobot estimates it by adding a noise factor. During the experiment Kilobots can collide, this situation cannot happen in DeMaMAS. The main differences between DeMaMAS and Kilobots experiments are listed below:

- Environment:
  - DeMaMAS: torus (1x1)
  - Kilobots: square with solid border (1mx1m)
- Number of agents:
  - DeMaMAS: 100
  - Kilobots: 50
- Time
  - DeMaMAS: discrete (time steps)
  - Kilobots: continuous (seconds)
- Number of simulation:
  - DeMaMAS: 100 per experiment
  - Kilobots: 5 per experiment (20 in total)
- Communication rate:
  - DeMaMAS: every time step
  - Kilobots: the system sometimes needs to stop the communication between Kilobots for updating the direction during the resampling phase and for checking the id correspondence.
- Communication radius:
  - DeMaMAS: 0.05 (relative to simulation unit of measurement)
  - Kilobots: 10cm
- Totem radius:
  - DeMaMAS: 0.10 (relative to simulation unit of measurement)
  - Kilobots: 20cm

In Figure 6.3 we showed respectively 4 different frames of one of our 20 different experiments, corresponding in a scenario with CDMCI and 5 attackers as showed in Figure 6.2.

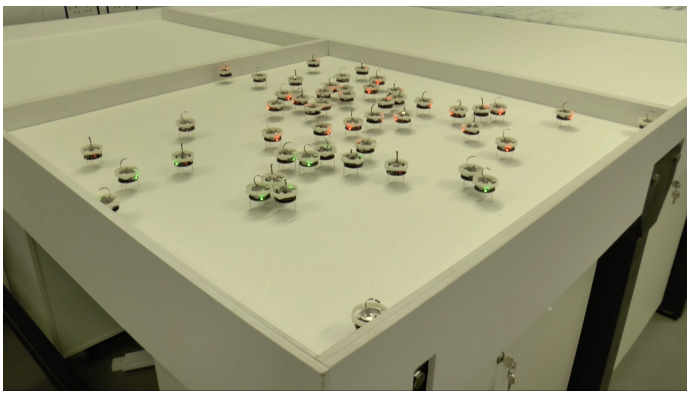

Figure 6.2: A Kilobot experiment captured with a video camera.

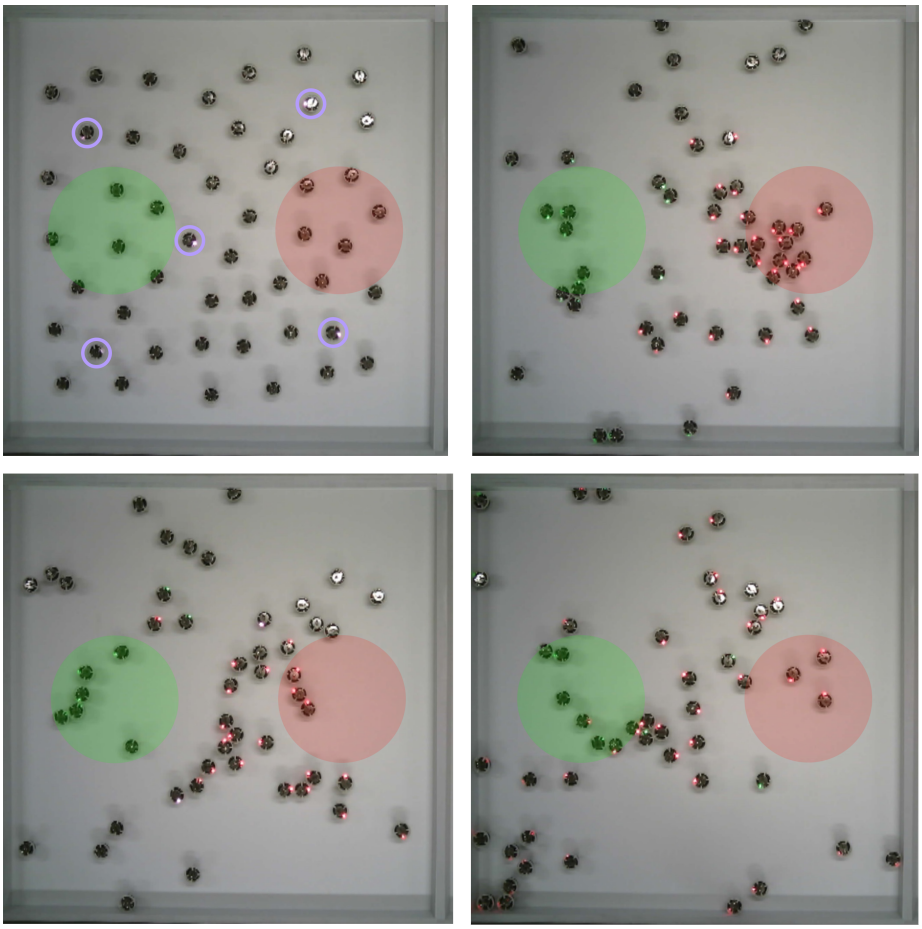

Figure 6.3: Four frames of an experiment given by ARK with 2 options, 50 Kilobots (5 contrarians and 45 using CDMCI), quorum set to 80%, and difficulty set to 0.4. The green and red circle represents the two options. The best option is represented in red. The highlighted Kilobots in the first picture represents the 5 contrarians. In the following three frames we showed how the consensus is reached. In the last frame the swarm reached the consensus on the best option.

For running an experiment as showed in Figure 6.3 we followed these steps:

- **Charge** all the 50 Kilobots
- Use the auto **calibration** provided by ARK
- Do a **visual check** that the calibration was done correctly
- **Position** each Kilobot in the environment
- **Upload** the code corresponding to the two different behaviours
- Assign a different ID per each Kilobot using the **ID assignment** function provided by ARK
- **Run** the experiment

In the following section we described the obtained results.

6.4 Results

In Figure 6.4 we showed the comparison between the results obtained between Kilobots experiments and ARGoS simulations. On the x-axis we vary the used models, in our case CDCI and CDMCI, on y-axis we vary the time in minutes. The red box-plots represent the speed performance of each behaviour in presence of intruders, while the green box-plots display the speed performance in absence of intruders. Outliers if present are represented with circles. All the 20 different runs of our Kilobots experiments and the 100 different runs with ARGoS were able to reach the consensus to the best option, due to the simplicity of the problem. As showed in these plots we obtained that CDMCI is faster than CDCI in both cases without and with intruders. In the first plot representing the Kilobots experiments we showed that the average of the decision time of CDMCI in both cases in which the percentage of attackers is equal to 0% and 10% is lower. During our experiments only one CDCI run with 0% of attackers can be considered as an outliers and is depicted with a circle. The difference between the Kilobots experiment and ARGoS are related to the difference based on the number of runs per experiment (100 for ARGoS and 5 for Kilobots). We showed that the results range in the same decision time, that is between 1 and 4.5 minutes. These outcomes are a preliminary confirmation of our previous result obtained in Chapter 5.3, in which we stated that CDMCI is performing better in speed and accuracy compared to CDCI. In this plots we also showed that a percentage of contrarians equal to 10% is able to perform a slow down of the system. Under attacks the behaviours present an higher level of variance and an higher average.

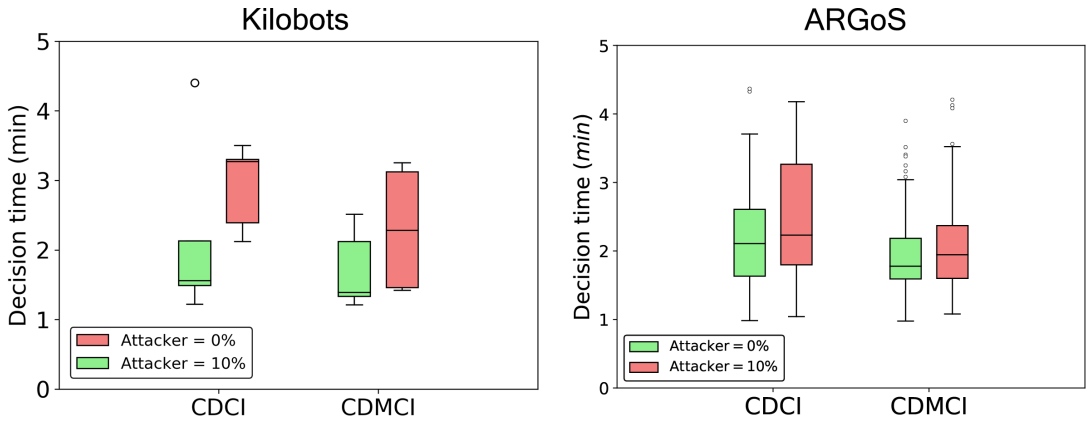

Figure 6.4: Box plot relative to Kilobots experiments and ARGoS simulations (100 runs per simulation).

Chapter 7

Conclusions and future works

We provided a novel modular tool called DeMaMAS for comparison of decentralised collective decision making strategies. Thanks to its modular architecture we were able to simulate heterogeneous swarms and test resiliency in collective decision making problems focused on best-of-n. We compared four different literature behaviours (CDCI, DMMD, DMVD, and 3-unanimity) and our new behaviour CDMCI with three different types of attackers (contrarian, wishy washy and sect). The implemented attackers perform different kinds of disservices. A contrarian causes a slow down, a wishy washy causes a denial of service, and a sect causes a wrong addressing. We tested four literature behaviours in different scenarios varying the percentage of attackers, the difficulty of the problem, and the number of options. As a result of this analysis we obtained three main results. The first one is that the 3-unanimity behaviour in very simple scenarios, characterised by few options and easily distinguishable alternatives, is the behaviour having a better resiliency. This result is related to the high selectivity of 3-unanimity in selecting a new option, that turns into a deafness if the information received is not matching the stringent 3-unanimity's requirements. The deafness allowed the behaviour to avoid the noise introduced by the attackers and be more resilient in simple problems where individual actions (i.e., individual discovery) were sufficient to reach a consensus. The second main result is that using a majority module in the selection of the new option increases the resiliency in most of the studied cases, excluding the simplest one. The third main result is that using a cross-inhibition module in complex cases increases the resiliency of the behaviour especially in complex cases. This kind of module was originally created for breaking deadlocks in symmetric quality problems and this behaviour makes the swarm more resilient to the malicious influences of an attacker. After discovering which of the tested modules increase the resiliency of a behaviour, we implemented a new behaviour called CDMCI that used both majority and cross-inhibition. Comparing this novel behaviour with the four literature ones, we realised that CDMCI has better performance in most of the analysed scenarios. In order to have a preliminary validation for the CDMCI, we implemented and tested it in a swarm of 50 Kilobots against a contrarian attack. The experiments on the physical robots gave us results in agreement with the simulations, the CDMCI showed the best performance in terms of speed, accuracy, and resiliency. The explored tests are only a little part of the possible scenarios that DeMaMAS can simulate. Thanks to its modularity, this tool allows a user to compare different behaviours and to discover which module affects more the performance of each behaviour. In future

this tool can help in testing quickly new behaviours for collective decision making problems. In DeMaMAS some additional modules are already implemented that are not used for the purpose of this thesis, but they are created for ongoing studies. The already implemented modules that can be used and compared in future are:

- message → super
  - This new module allows the agent to share in addition to the option the estimated quality.
- modulation self-social → step-time and step-time-value
  - This two modules allowed the agent to update the social-strength (ρ) and self-strength (ϱ) during the execution of the experiment [39].
- decay → probabilistic
  - This module allowed the agent to forget its option i and the relative estimated quality $\tilde{q}_\mathrm{i}$ with a probabilistic function.
- update model → self-inhibition
  - This module allowed the agent to apply the following described behaviour. If an uncommitted agent receives a new option from a neighbour it switches to that option, if a committed agent receives a new option from a neighbour that it is equal to its own option it switches to uncommitted.
- process-social → max-quality
  - This module allowed the agent to pick from its neighbours the option with the highest quality. This module can be used only if the agents use the message module type set to super [39].

In addition of the previous modules another future work that can be done with DeMaMAS is to combine more than one behaviour per time against attackers for creating more resilient behaviours.

Bibliography

[1] Anthony Antoun, Gabriele Valentini, Etienne Hocquard, Bernát Wiandt, Vito Trianni, and Marco Dorigo. Kilogrid: A modular virtualization environment for the kilobot robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3809–3814. IEEE, 2016.

[2] Thomas Bose, Andreagiovanni Reina, and James A. R. Marshall. Collective decision-making. *Current Opinion in Behavioral Sciences*, 6:30–34, 2017.

[3] Peter Clifford and Aidan Sudbury. A model for spatial conflict. *Biometrika*, 60(3):581–588, 1973.

[4] Marco Dorigo and Erol Sahin. Guest editorial: Swarm robotics. *Autonomous Robotics*, 17(2-3):111–113, 2004.

[5] Manfred Eigen and Ruthild Winkler. *Laws of the game: how the principles of nature govern chance*, volume 10. Princeton University Press, 1993.

[6] Anna Font Llenas, Mohamed Salaheddine Talamali, Xu Xu, James A. R. Marshall, and Andreagiovanni Reina. Quality-sensitive foraging by a robot swarm through virtual pheromone trails. In M. Dorigo et al., editor, *Swarm Intelligence (ANTS 2018)*, volume 11172 of *LNCS*, pages 135–149. Springer, Cham, 2018.

[7] Serge Galam. Majority rule, hierarchical structures, and democratic totalitarianism: A statistical approach. *Journal of Mathematical Psychology*, 30(4):426–434, 1986.

[8] Heiko Hamann. *Collective Decision-Making*, pages 129–162. Springer International Publishing, Cham, 2018.

[9] Heiko Hamann. *Swarm Robotics: A Formal Approach*. Springer, 2018.

[10] Rainer Hegselmann and Ulrich Krause. Opinion dynamics and bounded confidence models, analysis, and simulation. *Journal of artificial societies and social simulation*, 5(3):2–29, 2002.

[11] Fiona Higgins, Allan Tomlinson, and Keith M Martin. Survey on security challenges for swarm robotics. In *2009 Fifth International Conference on Autonomic and Autonomous Systems*, pages 307–312. IEEE, 2009.

[12] Fiona Higgins, Allan Tomlinson, and Keith M Martin. Threats to the swarm: Security considerations for swarm robotics. *International Journal on Advances in Security*, 2(2&3):288–297, 2009.

[13] Serge Kernbach, Ronald Thenius, Olga Kernbach, and Thomas Schmickl. Reembodiment of honeybee aggregation behavior in an artificial micro-robotic system. *Adaptive Behavior*, 17(3):237–259, 2009.

[14] Adil Khadidos, Richard M Crowder, and Paul H Chappell. Exogenous fault detection and recovery for swarm robotics. *IFAC-PapersOnLine*, 48(3):2405–2410, 2015.

[15] Jens Krause and Graeme D Ruxton. *Living in groups*. Oxford University Press, 2002.

[16] Yoshiki Kuramoto. *Chemical oscillations, waves, and turbulence*. Courier Corporation, 2003.

[17] Richard E Lee Jr. Aggregation of lady beetles on the shores of lakes (coleoptera: Coccinellidae). *American Midland Naturalist*, 104(2):295–304, 1980.

[18] James AR Marshall, Ralf HJM Kurvers, Jens Krause, and Max Wolf. Quorums enable optimal pooling of independent judgements in biological systems. *eLife*, 8:e40368, 2019.

[19] M Mobilia and S Redner. Majority versus minority dynamics: Phase transition in an interacting two-state spin system. *Physical Review E*, 68(4):046106, 2003.

[20] Eleftherios Nikolaidis, Chelsea Sabo, James A R Marshal, and Andreagiovanni Reina. Characterisation and upgrade of the communication between overhead controllers and Kilobots. Technical report, Figshare, The University of Sheffield, May 2017.

[21] Hyongju Park and Seth Hutchinson. A distributed robust convergence algorithm for multi-robot systems in the presence of faulty robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2980–2985. IEEE, 2015.

[22] Carlo Pinciroli, Mohamed Salaheddine Talamali, Andreagiovanni Reina, James A. R. Marshall, and Vito Trianni. Simulating Kilobots within ARGoS: models and experimental validation. In M. Dorigo et al., editor, *Swarm Intelligence (ANTS 2018)*, volume 11172 of *LNCS*, pages 176–187. Springer, Cham, 2018.

[23] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012.

[24] Andreagiovanni Reina, Alex J Cope, Eleftherios Nikolaidis, James AR Marshall, and Chelsea Sabo. Ark: Augmented reality for kilobots. *IEEE Robotics and Automation letters*, 2(3):1755–1761, 2017.

[25] Andreagiovanni Reina, James A. R. Marshall, Vito Trianni, and Thomas Bose. Model of the best-of-N nest-site selection process in honeybees. *Physical Review E*, 95(5):052411, 2017.

[26] Andreagiovanni Reina, Gabriele Valentini, Cristian Fernández-Oto, Marco Dorigo, and Vito Trianni. A design pattern for decentralised decision making. *PLoS One*, 10(10):e0140950, 2015.

[27] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21, pages 25–34. ACM, 1987.

[28] Michael Rubenstein, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014.

[29] David Saldana, Amanda Prorok, Shreyas Sundaram, Mario FM Campos, and Vijay Kumar. Resilient consensus for time-varying networks of dynamic agents. In *2017 American Control Conference (ACC)*, pages 252–258. IEEE, 2017.

[30] Ian Sargeant and Allan Tomlinson. Review of potential attacks on robotic swarms. In *Proceedings of SAI Intelligent Systems Conference*, pages 628–646. Springer, 2016.

[31] Kelsey Saulnier, David Saldana, Amanda Prorok, George J Pappas, and Vijay Kumar. Resilient flocking for mobile robot teams. *IEEE Robotics and Automation Letters*, 2(2):1039–1046, 2017.

[32] Alexander Scheidler, Arne Brutschy, Eliseo Ferrante, and Marco Dorigo. The k-unanimity rule for self-organized decision-making in swarms of robots. *IEEE Transactions on Cybernetics*, 46(5):1175–1188, 2016.

[33] Thomas Schmickl and Heiko Hamann. Beeclust: A swarm algorithm derived from honeybees. *Bio-inspired computing and communication networks*, pages 95–137, 2011.

[34] Thomas Schmickl, Ronald Thenius, Christoph Moeslinger, Gerald Radspieler, Serge Kernbach, Marc Szymanski, and Karl Crailsheim. Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18(1):133–155, 2009.

[35] Thomas D Seeley, P Kirk Visscher, Thomas Schlegel, Patrick M Hogan, Nigel R Franks, and James AR Marshall. Stop signals provide cross inhibition in collective decision-making by honeybee swarms. *Science*, 335(6064):108–111, 2012.

[36] Volker Strobel, Eduardo Castelló Ferrer, and Marco Dorigo. Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 541–549. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[37] Katarzyna Sznajd-Weron and Jozef Sznajd. Opinion evolution in closed community. *International Journal of Modern Physics C*, 11(06):1157–1165, 2000.

[38] Martina Szopek, Thomas Schmickl, Ronald Thenius, Gerald Radspieler, and Karl Crailsheim. Dynamics of collective decision making of honeybees in complex temperature fields. *PloS one*, 8(10):e76250, 2013.

[39] Mohamed S. Talamali, James A. R. Marshall, Thomas Bose, and Andreagiovanni Reina. Improving collective decision accuracy via time-varying cross-inhibition. In *Proceedings of the 2019 IEEE International Conference on Robotics and Automation (ICRA 2019)*, page in press, 2019.

[40] Danesh Tarapore, Anders Lyhne Christensen, and Jon Timmis. Generic, scalable and decentralized fault detection for robot swarms. *PloS one*, 12(8):e0182058, 2017.

[41] Guy Theraulaz and Eric Bonabeau. Coordination in distributed building. *Science*, 269(5224):686–688, 1995.

[42] Guy Theraulaz and Eric Bonabeau. Modelling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology*, 177(4):381–400, 1995.

[43] Nitin H Vaidya, Lewis Tseng, and Guanfeng Liang. Iterative approximate byzantine consensus in arbitrary directed graphs. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 365–374. ACM, 2012.

[44] Gabriele Valentini, Eliseo Ferrante, and Marco Dorigo. The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI*, 4:1–15, 2017.

[45] Gabriele Valentini, Eliseo Ferrante, Heiko Hamann, and Marco Dorigo. Collective decision with 100 kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-Agent Systems*, 30(3):553–580, 2016.

[46] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. Self-organized collective decision making: The weighted voter model. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 45–52. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[47] Karl Von Frisch and Otto Von Frisch. *Animal architecture*. Harcourt Brace Jovanovich New York, 1974.

[48] Haotian Zhang and Shreyas Sundaram. Robustness of information diffusion algorithms to locally bounded adversaries. In *2012 American Control Conference (ACC)*, pages 5855–5861. IEEE, 2012.