Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Swarm Memory

Harry Burge

2021-April

Supervisor: Simon O’Keefe

Acknowledgements

I would like to thank my supervisor, Dr. Simon O'Keefe, for all his guidance, feedback, and support. I would also like to thank the university for providing me with resources, of which this project wouldn't have been possible. And my family for helping me down this path.

Contents

Executive Summary **vi**

1 Introductury Material **1**
1.1 The Problem 1
1.2 Introduction 2
1.3 Background and Motivation 2

2 Literature Review **5**
2.1 Cloud/Backup storage policys/schemes 5
2.2 Swarm robotics 8
2.3 Approach and Justification 11

3 Design **12**
3.1 Experiments 12
3.2 Static Heursitic 13
3.3 Dynamic Heuristic 16
3.4 Dynamic Heuristic with Migration 17
3.5 Neural Network Heuristic 18

4 Analysis **21**
4.1 Static Heuristic 22
4.2 Dynamic Heuristic 24
4.3 Dynamic Heursitic with Migration 25
4.4 Neural Network Heuristic 26

5 Conclusion **28**
5.1 Conclusion 28
5.2 Futher Improvements 29

A Code apendix **31**

B Results apendix **32**

List of Figures

1.1 Data duplication density based on distance to a datas desired location 1

2.1 Example of a hetrogenus ant colony 8

3.1 Example of changing of replication and suicide threshold on a uniform agent density 14
3.2 Neural network design - Thresholded argmax output 19

4.1 Static Heuristic on semistatic movement swarm, with noncorrelated failures 21
4.2 Genetic Algorithm fitness over generation 26

B.1 Static Heuristic on semistatic movement swarm, with threshold changes 32
B.2 Static Heuristic on semistatic movement swarm, with correlated failures 33
B.3 Static Heuristic on circular movement swarm, with noncorrelated failures 33
B.4 Static Heuristic on circular movement swarm, with correlated failures 34
B.5 Dynamic Heuristic on semistatic movement swarm, with threshold changes 34
B.6 Dynamic Heuristic on semistatic movement swarm, with noncorrelated failures 35
B.7 Dynamic Heuristic on semistatic movement swarm, with correlated failures 35
B.8 Dynamic Heuristic on circular movement swarm, with noncorrelated failures 36
B.9 Dynamic Heuristic on circular movement swarm, with correlated failures 36
B.10 Dynamic Heuristic with Migration on semistatic movement swarm, with noncorrelated failures 37
B.11 Dynamic Heuristic with Migration on semistatic movement swarm, with correlated failures 37
B.12 Dynamic Heuristic with Migration on circular movement swarm, with noncorrelated failures 38

B.13 Dynamic Heuristic with Migration on circular movement swarm, with correlated failures 38
B.14 Discouraging results from Neural Network Heuristic, semi-static movement and noncorrelated failures 39

Executive Summary

This report proposes and tests solutions for directional memory storage policies within a swarm environment. The proposed solutions are to provide redundancy for correlated and non-correlated failures, without pure duplication of data throughout the swarm. Directionality of the policies is defined as a higher density of duplications nearer to the point that information is needed.

A simulated homogeneous swarm is used to provide results on two different designs of policies. Each swarm agent can perform four actions, Replication, Suicide, Migration, or Nothing. These actions can be performed on data stored in their memory to gain redundancy from both types of failures. Replication duplicates data to other agents. Suicide deletes the data. Migration passes the data onto another agent and nothing is as the name suggests.

A design is the way we control an agent's actions to get emergent characteristics. This emergent behaviour will be reviewed using five different global factors. Amount of duplications per data point, the distance between a duplication and its desired location, instability of the system, agents memory usage, and a visualisation of duplications ratio over the 2D plane. The performance will be based on levels of duplication with a roughly even distribution per data point, having a large range of distances with a mean close to the minimum, and for instability to be of the lowest magnitude. These factors are tested on both circular motion and semi-static motion to see how behaviour changes over different swarm environments e.g. lots of volatility in connections or stable connections. Within each movement, style correlated and non-correlated failures are tested to see the effectiveness and characteristics of the proposed method.

The initial version of the first design style used a weighted sum with a threshold for both suicide and replication. The input factors are; duplications ratio in the locality of the data selected, distance to the desired data point of selected data, and average spare memory left on agents in the locality. This version performed adequately in most factors, however, stability left room for improvement, uneven distribution of memory load, and poor handling of a volatile connection swarm.

The second version of the first design style carried on from the previous by changing the static thresholds to be dynamic, to counteract the stability problem of the previous version. To measure instability on a local scale, as each agent received a duplication request, would increase an instability variable, this would be reduced over time. The output of sigmoid functions was added to the weighted sum, with inputs being time from last suicide for the suicide threshold and instability for the replication threshold. The version achieved its target by significantly reducing the magnitude of instability without changing the behaviour of the previous version. The nature of the instability changes within a circular motion swarm to being roughly constant instability compared to the periodic nature of the static heuristic. However, there was still the issue of duplication load over the swarm.

The third version of the first design style implemented migration. Migrations threshold is controlled by, if we would statically replicate this item and the agent in the locality with the most available space is higher than the current agent's available space. As expected this increased the instability due to more movement of data. The spread of workload was significantly better in semi-static swarms compared to the circular counterparts. Therefore leading to different use cases for both the second and third versions.

Having reviewed the first design style even though performance, as defined, was good, it still lacked effectiveness when compared to a theoretical best solution. This is because the algorithm doesn't take into account many edge cases or complex behavioural changes based on multiple thresholds. This led to using the computational power of neural networks to solve for these complex behavioral characteristics. A genetic algorithm was used to train the neural network, using a global fitness function post-simulation run with each agent using the same network. Unfortunately, the learning rate was very slow due to selected hyperparameters and poor judgement on fitness function evaluation being only one simulation run. Therefore, it led to a non-testable version due to the extremely poor performance.

The field of swarms is riddled with ethical and moral issues. Specifically, swarms have a high potential to be abused in surveillance, breaching people's privacy. Within the military domain, is it ethical to have systems having choice over human life, but also are so resilient the opponent doesn't have any chance? Socially, is it acceptable to have self-piloting drones flying around delivering packages at all times of day and night? Environmentally, a swarm solution will nearly always use more energy and cost more CO2 to produce in any significant size, when compared to a single robot solution. In terms of security, allowing multiple more access points to a network also increases risk, especially if each agent/node has locally stored accessible data.

1 Introductory Material

1.1 The Problem

The problem this report tries to solve is the creation of a directional memory storage policy for agents in a swarm, without complete duplication. This can be split into two separate sub-problems. The first is the handling of data duplications throughout the swarm to control for the failure of agents and provide a mechanism for recovery.

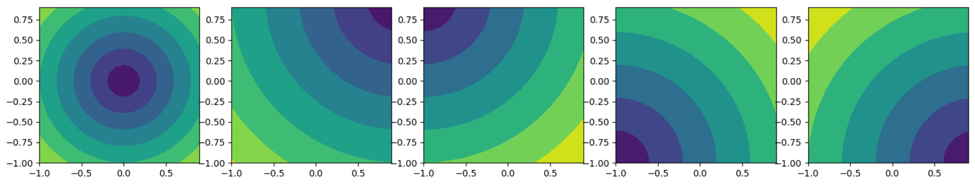

Figure 1.1: Data duplication density based on distance to a datas desired location

The second is to apply directional characteristics to the above method. Visualization can be seen in Figure 1.1, where the density of duplication decreases as the distance from the desired location is increased. Each piece of data should have its own desired location.

A real-life example of this type of problem arises in minefield operation [2]. Agents need to map out potential mines and record their location. Once an agent locates a mine it wants to spread the mine's location. However, memory restrictions in agents only permit the knowledge of a minimal amount of mines. Agents closer to the mine should be prioritized in knowing the mine's location. Agents further away from the mine do not need to know about its location, because they will need knowledge of other closer mines.

Complexity is increased taking into account a practical implementation of this problem. Firstly the policy has to withstand fluctuations of a swarm, which is a hard task in and of itself. But secondly, handle correlated (Mine detonation) and non-correlated (Individual power loss) failures, with minimal loss of data.

1.2 Introduction

The objective of this report is to solve the problem defined in Section 1.1, with an effective and reliable policy that can withstand high locality and dynamic behaviours of a swarm. The analysis will then be performed on generated policies to gauge their real-world capabilities and effectiveness.

Two design styles will be undertaken, with an emphasis on the first. This design will be handcrafted and will indicate how effective a policy of this type can be. The second will use a multi-agent interaction approach using a neural network, which will be compared against the handcrafted policy. Policies will be rated based on a few key characteristics; ability to handle both types of failures, over or under saturation of duplication, and stability of the policy.

To complete this objective, we will programmatically break down the problem described in Section 1.1 into solvable tasks. The report will be structured as follows, firstly bringing the reader up to date with relevant literature and explaining key concepts required for a complete understanding of how the proposed solutions have been derived. This will be undertaken in sections, Section 2.1 of which goes into detail about current cloud-based storage technologies, and Section 2.2 which delves into more of the background behind swarms, specifically relating to the problem and possible solutions. We'll go through the methodology of the proposed solutions' designs, how it is supposed to act and react in different scenarios, and the reasons for why. Leading to an analysis of how effective the proposed policies have kept to standards derived in the methodology section, and whether they effectively solve the problem defined in Section 1.1.

1.3 Background and Motivation

Swarms are an increasingly important area of research for society, as the world moves towards a distributed technology future. The research of swarms within a technology setting can be broadly divided into two partitions, these are intelligence and mechanics.

Swarm intelligence can be viewed as the research into highly distributed problem solving[2, 5]. This is ever more becoming relevant as computer systems start to level out in sequential performance [7] and parallelism is embraced, satisfying the demand of the age of big data [9].

Swarm mechanics leans towards the robotics side and can be seen as the study of practical implementation of a swarm, whether that be movement or communication. This is on the rise in industry, as society's pace increases and manual labor is automated. Whether its drone delivery to inpatient customers or mapping areas in dangerous environments [1].

These areas often are highly integrated and are rarely seen in their pure form. An example of a pure form of swarm intelligence can be seen in [5] with network routing protocols. This project focuses predominantly on swarm intelligence to deal with a practical problem.

Most research on memory within a swarm has followed the route of optimization on distributed problem-solving algorithms, compared to practical applications of storage of abstract ideas as a collective. As one of the key reasons for using a swarm is redundancy, which is often assumed and boasted about, rather than proven, specifically within the memory domain.

The relative lack of research into collective memory appears to be a glaring hole in the foundations of a complex and interchangeable subject. The need for more research into collective memory can be seen in examples, such as how invaluable it would be in the mapping of dangerous areas [2]. By being able to handle the loss of agents combined with the redundancy of sufficient memory policy, provides a much wider scope for swarm usage.

An explanation for why swarm-based memory management solutions are an underdeveloped area of study is the existence of cloud-based storage research. The argument for these two subjects being partially separated is the nature of a swarm's locality and ever-changing network style, compared to a typical server network.

Most elements of cloud storage policies at a high level of abstraction could work effectively within a swarm-based environment. However as mentioned above, key adaptations would need to be created for an effective policy. A prime example is “SKUTE” from [10]. “SKUTE” will be the main inspiration for this project's solution, to storage within effectively a highly dynamic network.

Currently, the main issues for swarms are within surveillance [21, 18], delivery or the military [1]. However, to see the full scope of what man-made swarms could do, we look to the future. Whether it be space exploration [20], nano-robots in medicine or in the parallel/distributed software domain [19].

Swarm robotics is limited by the technology of its time. The use of research is limited and far between in our day and age. Research is conducted in mind for the future when technology can support and utilize the fullest potential swarms can offer.

2 Literature Review

2.1 Cloud/Backup storage policys/schemes

Like most things in computer science, cloud storage started relatively simple. As the years have progressed so has the demand for the cloud, varying from everyday people storing files to large businesses storing harvested data. An increase in complexity of solutions came from the Legal Services Act. 2007 [11]. This enforced cloud storage suppliers, to provide reliable, fast, and redundant data storage and collection for all users. In this section, we focus on cloud-based storage policies and some background terminology required for this report.

Firstly the reader needs to understand the difference between correlated and non-correlated failures. A non-correlated failure is when a device fails independently, with no relation to other failures within the system as a whole. For example, a server node can shut down from a software failure, therefore the connection is lost, typically this is completely independent of other servers in the rack. The opposite, correlated failure, is when multiple devices fail with a relational link. A typical example is on-mass restarts from a power surge in a data center. The power surge event is the relational link that binds the events together. Correlation doesn't require close geographical location, however, when mentioned later on in the report, it should be regarded as such.

Multiple defensive strategies can be employed to control for both failure types. Focusing on strategies that are reactionary rather than preventative. For local redundancy, typically RAID is used to ensure safety in a storage failure. A replication policy [9] is used for internode redundancy. By working in tandem achieves extreme redundancy against both types of failures.

A replication policy selects a piece of data and decides whether it needs to be replicated, and if so where it should be stored/located. This duplication of data onto another storage device means that if one fails, we still have a full replica to access. A simplistic approach would be a random replication policy, where data is randomly chosen to be duplicated. This is an acceptable policy for handling failures, however, without tracking of global duplications, it potentially can lead to overused storage. It also doesn't take into account the popularity of certain items. An algorithm like random replication is substantial for long-term storage where the popularity of data and distance to users are averagely the same for all data items.

Cloud-based storage transitioned from a semi-local backup system to a worldwide daily driver. Random replication couldn't withstand the variability of how users interact with files nowadays. For example, a video is hosted in one country and replicated within the said country then international viewers might have delays to their streaming. Alternatively, trying to compensate we host in multiple countries, those other countries might have reduced views. This means storage is wasted, which we could use for other more popular videos. This leaves us attempting to maximise for both situations, and a new replication policy is required.

These new algorithms extracted from papers handling "Distributed key-value store" [14], where key-value pairs are on multiple devices on a network and duplication only leads to more fault tolerance of the data stored. The first approach uses a privileged level of control where the master uses global knowledge to make decisions about whether and how to duplicate items [9, 12]. The same principles can be seen within schema changes [13]. Due to the nature of a privileged user, control of the policy is deterministic and easy to control. This leads to higher guarantees for failures unless on the master node, however, this can be handled dynamically by assigning another master. Touched upon in Section 2.2. Availability and popularity are handled by the policy heuristic controlled by the master node.

Having this master approach doesn't work effectively for a swarm. This is because the change from a server network to a swarm is quite drastic. Servers running over a network generally have complete connectivity e.g. Global scope. Servers also have a constant power supply compared to the average swarm agent. When restricting the master's scope we have to rely on messages bouncing from agent to agent. This will, first of all, reduce processing capability, power loss will be most significant in agents closest to the master, possibly leading to a cut-off from command [1]. It also doesn't fit into the ideology of a swarm, this will be talked about in Section 2.2.

The second approach doesn't rely on a privileged member and can be adapted for locality. Each node (In the case described below its each key-value pair) has its own controller for when and how to replicate, following a distributed control structure. Following the approach used with “SKUTE” as proposed in [10], it can make four decisions per data item. These decisions are; Migration, Suicide, Replication, and Nothing.

Migration transfers itself from the origin to a lower cost or more redundant server. Suicide is the removal of itself, this will usually be due to excess duplications. Replication is when it decides it needs to be duplicated onto another server. Nothing is as the name indicates.

The highly distributed nature lends itself to failures of parallelism. For example, two data points might want to suicide and will do so. Whereas if we ran them sequentially it could be possible that if one suicided the other might not. If only two agents remained with contained duplicated data then the policy could fail to retain data. This is where a consensus algorithm comes into play. Paxos [15] is an example of how both nodes couldn't suicide at the same time. This essentially creates a part-time leader agent to control a suicide action.

Within the server storage domain, an approach like "SKUTE" is less commonly used because it adds complexity which is unnecessary within a global scope and consistently powered network. Most data warehouses would prefer to have one server running as a controller and other servers running at full capacity compared to all servers at a slightly lower capacity due to extra self computation. However, this approach allows for greater hypothetical redundancy because of having no single point of control.

Moving towards local redundancy and optimisation is the stagnated study of RAID. This is where we change orderings of multiple storage discs to gain redundancy and/or performance increases. This grouping of disks is called a RAID array and can be structured in a multitude of ways. Common structures are labelled as RAID levels and give different attributes based on what functionality you are pursuing [17].

One of the key components of multiple RAID levels is the use of parities [8]. This is where a function (Simplistically an XOR) is done on two or more sets of data to create one or more parities. This function has a property enabling a tolerant amount of disks to be lost, and after the event be used to reconstruct lost data. In the case of data A, data B, and A XOR B, if data B is lost then can be reconstructed using data A and A XOR B. With different levels and functions more than one disk can fail and still retain data, however, if disc failures increased over the specified limit then reconstruction cannot occur. RAID is predominantly used internally within a storage node to provide redundancy against disk failures and increase the speed of writes that are typically on hard disks.

The methodology is, so long as the nodes individually are redundant enough and there is control on a higher level with our replication schemes, then it is sufficiently redundant.

2.2 Swarm robotics

Explained broadly in Section 1.2, the study of swarms is split into two subsections, mechanics and intelligence.

The concept of swarm intelligence is creating a solver to a problem using a distributed algorithm that can rely on natural parallelism, doesn't rely on global knowledge, and is adaptable on the fly, in comparison to their counterparts. From papers read by the author, predominantly the topics investigated are testing distributed solvers and comparing them against already researched global scope solutions. Examples of this within the TSP domain is a genetic algorithm versus AS-TSP [5]. These algorithms also excel in the networking domain, because of the high parallelism and need for adaptability [5].

The other subsection can be broadly known as swarm mechanics. Swarm mechanics focuses on problems that are less abstract and are usually in the domain of physical implementation [2, 4]. Swarm robotics is a subset of swarm mechanics, and is justified thusly; within this domain, we focus on the emergent behaviour of a swarm compared to the solution it might give. The second is that swarm robotics doesn't encompass the study of swarm behaviour in nature [5, 22].

Delving deeper into swarm mechanics there are three different methodologies, heterogeneous, homogeneous, and hybrid [1]. These can be adopted in multiple ways, however, we focus upon the adoption of these methodologies in decision making and physical attributes of agents.

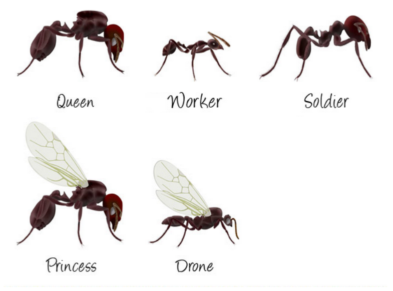

Figure 2.1: Example of a hetrogenus ant colony

A heterogeneous swarm is defined by differences between agents of the same swarm, as in Figure 2.1, whether physically or mentally [1, 5]. These occur commonly in nature and are a less researched sub-field of swarms [5]. For real-world solutions, heterogeneous swarms can be of great use, allowing certain agents to pick up the slack of the swarm, or complete tasks that other swarm members cannot complete. A good example described in

[1] with a mother ship being a navy boat and a swarm of quadcopters. The boat picks up the slack of the swarm by transporting them longer distances than the swarm could independently complete, whilst also carrying complex hardware.

The argument against heterogeneous swarms is a tendency to over-rely on the differences of agents. Running with the example from [1], the agents rely on the naval boat to be able to travel long distances and for a recharge point. Without the boat, the swarm fails at completing its prolonged objective. The decision to use a heterogeneous swarm, in this case, is valid because if the naval ship is lost, then a catastrophic incident has occurred and the swarm is probably not the highest priority.

Physical differences in agents breed efficiency. However, this can only hold true if the vulnerability of a substantial loss of a key class is mitigated. Common rules for mitigation are found in nature's swarms. They are; jobs need to be interchangeable between all types of agents however some agents are more efficient at that job [22] or specific jobs are non-essential to the colony's survival. These swarms are under-researched because human implementations currently lack the adaptability of biology, so this vulnerability has to be taken into account. Some ant species, like Leaf-Cutter Ants, even have subcategories within a category of type. Leaf-Cutters have workers specialising in certain tasks like fungus farming. The best example of showing the interchangeability of these roles is when major ants do worker jobs when there is a significant loss of workers [5].

Homogeneous swarms are defined by each agent being the same. This is found less often in nature, except for the microbial level. Biology's natural adaptability compared to robotics means that semi-heterogeneous swarms are exploited better in nature [1, 5]. Therefore we maximise for the flaws of our current technology. Technically homogenous swarms provide the best platform, but that is getting into speculative futuristic technologies of self-replication, advanced intelligence, and nanorobotics.

Homogeneous swarms benefit from maximum redundancy, if any agent fails there remains an entire swarm's worth of agents to take its place. With the benefit of redundancy, we acquire some possible losses inefficiency which could have been exploited with task-specific hardware. Either the agent is too simplistic, losing efficiency in their tasks, or are too complex, to the point that all agents have the capability for every specialism but may never need them. There is a thin line between the two, either we lose practical power of the swarm or we have to invest more into the swarm than is actually needed.

Within the practical implementation of swarms, things become disorganised. Usually, there is no clear-cut framework of which a practical swarm uses.

This is where engineering and research have a disconnect, and hybrid swarms come into their own.

A hybrid approach tries to exploit the benefits of both heterogeneous and homogeneous designs without the downsides. Using a farmer and miner agent example, a hybrid approach to physicality would be that each agent has exactly the same body and could wield either a pickaxe or a hoe. The key point is that a miner could become a farmer if required. The reader might wonder why physical hybrid approaches aren't regarded as the best approach for all swarms. Bringing this theoretical solution into the real world can cause massive complexity. How can we guarantee job type distribution? What about the complexity of changing between job type hardware? These are all questions that need to be explored, by the designer/creator. Normally it is easier to go for the simpler solution and deal with the possible loss of either efficiency or adaptability.

Homogeneous control is where each agent controls its own decisions based on its locality, sometimes labelled distributed intelligence. This creates an emergent/structured global behaviour of the swarm even though each agent is acting of its own fruition. A simple example of this is [23].

Heterogeneous control is where specified agents control other agents' decision-making. This can be handled in different ways, two prominent ways are hivemind control [18] and royalty/hierarchical control. Hivemind is where one agent controls the entire swarm's decision making e.g. the master controller will give abstract ideas as parameters and the agents will execute them with their own decision making. A hierarchical approach follows the same style as a hive mind, however, it deals with scalability better. Enabling a power structure of certain agents being sub-leaders of leaders. Both control structures leave themselves vulnerable in their physical implementation due to bad actors and power loss distributions over the swarm.

Hybrid control of a swarm is a heterogeneous policy that is adaptable to changes of leaders and is usually designed for a physically homogeneous/hybrid swarm. The choice of leader(s) is done through a consensus algorithm [15] rather than based on any form of physical or mental difference. This allows homogenous style agents to act in a heterogeneous fashion. This creates more deterministic behaviours compared to emergent behaviors of homogeneous control, it can also help power loss distribution by re-electing leaders in different locations to help distribute where messages are relayed.

2.3 Approach and Justification

An initial solution was to use an adaptive version of RAID parities. However, a more effective concept solution became apparent after conducting further research into cloud-based storage.

Within the adaptive RAID design, if a subject agent has two agents within its locality with different data, an XOR parity of both data points would be recorded alongside a record of originating agents. If one of those agents left the subject locality then the parity would be used to reconstruct lost data. This approach was disregarded because of the following factors:

- Implementation of directionality would be difficult and inconsistent
- If the two agents die/leave the locality at the same time then the parity cannot be restored
- The algorithm relies on a swarm breaking connections often to spread data
- There is no duplication reduction, over time the data point would spread to all agents

These drawbacks highlighted the need for a new approach, cloud based-storage policies. The proposed solution takes heavy inspiration from "SKUTE" [10]. The design of a replication policy lends itself to heuristic-based control and enables the implementation of duplication density and direction to the spread of data.

A distributed/homogeneous style of control is used, partially because there is an average power loss over the swarm compared to a leader-based control, but mainly because of the methodology of a homogeneous swarm. A hybrid approach, if not accounting for power loss, would almost certainly be better in every single way, due to its global view. A hybrid implementation would be very simple and deterministic compared to its homogeneous counterpart. Both methods could not guarantee data loss would not occur, however, a hybrid approach would be better equipped to handle correlated failures.

With an extensive list of drawbacks to using a homogeneous control over a hybrid control scheme, it would be deemed inappropriate to use homogeneous control. This disgruntlement is valid only when talking about swarms that are partially static in nature. When the swarm is highly volatile, in terms of movement, more time could be spent assigning a leader rather than completing the actual task. Scaling of a hybrid system is harder because of the partitioning of agents to leaders. This justifies the author's decision to focus on distributed homogeneous replication policies.

3 Design

3.1 Experiments

To understand and evaluate the proposed solutions it's necessary to see how they react to different scenarios. The scenarios are designed to test functionality of the proposed solution. These are:

- Semi-Static moving swarm with non-correlated failures, Figure 4.1
- Semi-Static moving swarm with a correlated failure, Figure B.2
- Circular moving swarm with non-correlated failures, Figure B.3
- Circular moving swarm with a correlated failure, Figure B.4
- Changing of replication and suicide thresholds, Figure B.1

Semi-static movement, as described in Algorithm 2 is used to test the solution on mostly stable networks, allowing us to see the solution's inherent stability. In the opposite case, we have circular movement which adds physical instability. This allows the comparison of stability with internal and external factors. Correlated and non-correlated failures should be self-explanatory, as being key to the problem defined in Section 1.1.

The proposed solutions will be simulated and critiqued based on five factors. Number of duplications, allows us to see whether the policy over or under saturates duplications and how they are affected by the loss of agents. Distance from the desired point shows acceptable ranges for correlated failures, therefore, the larger the range the better it is. Instability, as most swarms will be battery-powered, we need to reduce the number of communications between agents, reduce communication power consumption, to have a practical solution. Storage load across the swarm, allows us to see how effective our policy is at distributing the load over the agents to not oversaturate an agent causing potential exclusion of agents.

3.2 Static Heursitic

This solution takes heavy inspiration from Paper [10]. We utilize the methodology of each agent controlling its own data and distribution of data, through actions of Replication, Suicide, and Nothing. Migration is not applied, so we piggyback on the natural movements of the swarm to extend the duplications range.

Thresholded heuristics decide whether to undertake either action, based on these factors:

- Ratio of agents in the locality that does and do not have duplicates
- Distance of the data's point
- Agents average spare memory in the locality

Algorithm 1 Main Control Loop

```
procedure STEP
    move()

    if Learned of new data point then
        Replicate new data point to locallity
        return True

    $d \leftarrow$ Euclidean distance to data current data point
    $r \leftarrow$ (local) Duplicates on agents / Number of agents
    $s \leftarrow$ (local) Average space available / Max public memory

    Replace with insert of choosen heuristic

    if $h_{rep}$ (Equation 3.3) $>$ Threshold$_{rep}$ then
        Replicate current data point to locallity

    if $h_{sui}$ (Equation 3.3) $>$ Threshold$_{sui}$ then
        Suicide current data point

    Iterate to next public data point

    return True
```

Algorithm 2 Static Heuristic Agent - Insert

```
$X_{rep}, X_{sui} \leftarrow$Equation 3.1
$W_{rep}, W_{sui} \leftarrow$Equation 3.2
```

The values collected undergo a weighted sum and are compared against thresholds. A pseudo code version of this can be seen in Algorithm 2 inserted into Algorithm 1. Every step an agent will check whether it has learnt any new private data, if so, then duplicate that data to all agents in the locality. If no duplications happen, it's because no suitable agents are in the locality, the data will have this action performed on the next iteration until it succeeds. This quick replication provides redundancy to non-correlated failures, as fast as possible. Most of the time this action will be redundant, until exceptions happen, calling this into play.

For each iteration we focus on one public data point in memory, this is a basic iteration through memory. A further improvement could be dynamically choosing which data point to service, discussed further in Section 5.2. The current implementation is unable to perform effectively for large data amounts e.g. scales badly.

An agent broadcasts packets to agents in their locality to gain the information required. Agents receiving packets respond with relevant information, e.g. do they have a duplicate of the data point, amount of spare memory, etc. Once collated by the originating agent, it works out the specified values, shown in the equations below. Some improvements can be made to the handling of broadcasting/replying and will be explained further in Section 5.2.

After collating the values we scale them from 0 to 1 and rearrange for them to increase in size when we are more likely to want an action to be taken. For example, a higher density of duplications leads an agent to be less likely to replicate $1 - dupes\_ratio$.

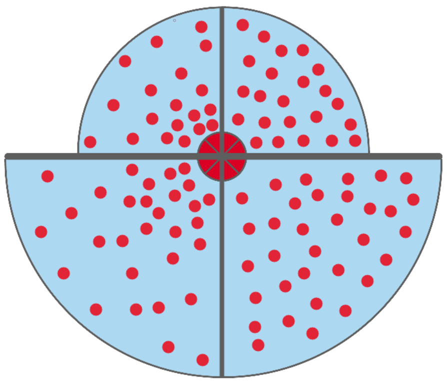

Figure 3.1: Example of changing of replication and suicide threshold on a uniform agent density

Changing weightings and thresholds apply significant behavioural changes within the policy. Increasing the replication threshold means that duplications will spread out less distance, avoiding higher duplication density areas and reducing spread when agents are already saturated with data. The higher the suicide factor indicates there is less of a duplication density difference across the network.

To enable easier understanding of the heuristics behaviour when changing threshold values we can refer to Figure 3.1. This demonstrates how the heuristic changes behaviour with varying threshold values, influencing the spread of duplicated data from a data point in a uniformly spaced swarm. The smaller red dots represent agents that have a duplicate with data points of the large red circle. Replication threshold increasing magnitude can be seen by the changing in distance, e.g. indicated by the blue circle. And suicide threshold when increasing changes the duplication density spread, e.g. from left to right, right being larger in magnitude.

$$\mathbf{X}_{rep} = \begin{bmatrix} 1-r & \frac{\sqrt{8}-d}{\sqrt{8}} & s \end{bmatrix} \mathbf{X}_{sui} = \begin{bmatrix} r & \frac{d}{\sqrt{8}} \end{bmatrix} \tag{3.1}$$

$$\mathbf{W}_{rep} = \begin{bmatrix} 0.45 & 0.45 & 0.1 \end{bmatrix} \mathbf{W}_{sui} = \begin{bmatrix} 0.3 & 0.7 \end{bmatrix} \tag{3.2}$$

We then workout h_{rep} and h_{sui} using multiplication as below:

$$h = \mathbf{W}\mathbf{X}^{T} \tag{3.3}$$

Where r is duplication ratio, d is the distance to the data's desired location, s is average spare storage in agents in the locality. The factors' weights can be adjusted to fit different behaviour styles. For example, you could favour distance from data point over current duplication density. Having the weighted sum enables the heuristic to be optimised by fitting techniques such as genetic algorithms. This would have to be tailored to individual applications because of no dynamic nature to threshold setting. In tests, we used values 3.2, without any significance but purely from tinkering to get good results.

Initially "suicide data" in Line 16 of Algorithm 2, was done using Paxos [15]. This ensured that a data point could not end up going extinct from suicide, and would only occur with external factors (Failures). However, from preliminary tests, this didn't make a significant difference in our scenario but should be added into any practical applications of this algorithm for greater guaranteed redundancy.

3.3 Dynamic Heuristic

To control the instability of the static heuristic, as described in Section 4.1, we need to enforce a behaviour so when instability increases, factors contributing to instability are reduced. Rather than polling agents in the locality for wider stability information increasing network congestion, computation time, and power consumption. We instead use locally tracked information within the agent to gain an understanding of local/global instability.

Firstly, to control the instability of the swarm, it wants to reduce the chance of suicides happening after one has occurred for a period of time. This in and of itself will not solve instability, however, slows its effects down. The previous heuristic attempts to seek a global optimum where it becomes static. In an unstable state, the optimum keeps changing at the fringe. Therefore restricting our suicides enables the possibility of replicating to a key agent outside the usual bubble which could lead to a static optimum. However, this only hopefully solves instability issues at the fringe and not inside the swarm. This is why we restrict the chances of replications based on local instability. Local instability is tracked per agent with a variable that as a request comes in to duplicate data from another agent, a constant is added to the variable, after every iteration, we lose some of the variable's magnitude. This gives the agent a partial picture as to how the locality is behaving. The combination of both feedback mechanisms should decrease instability and smoothen the effects over time.

To achieve these behaviours we modify the current heuristic using sigmoid functions, based on the inputs described abstractly above, which are used to modulate the thresholds to give them a dynamic coping mechanism for instability.

For replication:

$$\lambda_{rep} = \left(\frac{1}{1+e^{-\frac{\theta-\alpha_{rep}}{\beta_{rep}}}}\right) \tag{3.4}$$

$$\mathbf{W}_{rep} = \begin{bmatrix}0.45 & 0.45 & 0.1 & -0.6\end{bmatrix} \mathbf{X}_{rep} = \begin{bmatrix}1-r & \frac{\sqrt{8}-d}{\sqrt{8}} & s & \lambda_{rep}\end{bmatrix} \tag{3.5}$$

For suicide:

$$\lambda_{sui} = -\left(\frac{1}{1+e^{-\frac{\psi-\alpha_{sui}}{\beta_{sui}}}}\right) + 1 \tag{3.6}$$

$$\mathbf{W}_{sui} = \begin{bmatrix}0.3 & 0.7 & -0.6\end{bmatrix} \mathbf{X}_{sui} = \begin{bmatrix}r & \frac{d}{\sqrt{8}} & \lambda_{sui}\end{bmatrix} \tag{3.7}$$

θ in Equation 3.4 is iterations since last suicide, once a suicide occurs on the agent $\theta = 0$. ψ in Equation 3.6 is a value based on how many agents asked the current agent to store a bit of data, this naturally increases as instability is increased. ψ is reduced every iteration until 0.

Algorithm 3 Dynamic Heuristic Agent - Insert

```
1: $\theta \leftarrow$ Iterations since last suicide
2: $\psi \leftarrow$ Internal stability
3: $\lambda_{rep} \leftarrow$ Equation$3.4$
4: $\lambda_{sui} \leftarrow$ Equation$3.6$
5: $X_{rep}, W_{rep} \leftarrow$Equation 3.5
6: $X_{sui}, W_{sui} \leftarrow$Equation 3.7
```

Sigmoid functions were used because we can control when the heuristic should be unimpeded or impeded, with a grey area in between so if data needs to be duplicated quickly it can still override the blockade. α and β can be changed on both heuristics to give different behaviours. A pseudo-code example can be seen in Algorithm 3.

3.4 Dynamic Heuristic with Migration

With the success of the dynamic heuristic in Section 3.3, another speculative issue needs to be addressed. A possible issue, mentioned previously, is the inherent problems of connections through a swarm. If an agent learns a new data point it may be unable to spread to the rest of the agents in the swarm, either because there is a physical lack of agents in the locality or those agents' memories are full. There are three possible solutions to memory saturation. Agents repeat duplication requests if memory is full, priorities are used for data duplication or we keep the swarm at a state where information is spread equally around the swarm.

The latter being the solution explored in this project. This solution is inherently worse than the priority solution because the swarm can become oversaturated with duplicates. A combination of the two solutions would be optimal, to reduce cases of oversaturation. The author focuses solely on the swarm distributing data points equally, as a proof of concept. The concept is the same as a migration from “SKUTE” [10].

Inherently we don't want to limit the speed at which data can be migrated, for example, an agent with oversaturated memory meets an agent with no memory used, ideally both walk away with equal memory load. This will inherently increase the instability of the swarm, therefore, it's a trade-off.

Algorithm 4 Dynamic Heuristic Agent With Migration - Insert

```
1: $\theta \leftarrow$ Iterations since last suicide
2: $\psi \leftarrow$ Internal stability
3: $\lambda_{rep} \leftarrow$ Equation3.4
4: $\lambda_{sui} \leftarrow$ Equation3.6
5: $X_{rep}, W_{rep} \leftarrow$Equation 3.5
6: $X_{sui}, W_{sui} \leftarrow$Equation 3.7
7:
8: $SX_{rep} \leftarrow$Equation 3.1
9: $SW_{rep} \leftarrow$Equation 3.2
10:
11: $\delta \leftarrow$ Max avalibale space - Our avaliable space
12:
13: if $h_{rep}(SX_{rep}, SW_{rep})$ (Equation 3.3) $>$ Threshold$_{rep}$ and $\delta > 1$ then
14:     Duplicate item to target agent
15:
16:     if Successful then
17:         Suicide current data point
18:         return True
```

The migration heuristic is a **AND** of the heuristic described in Section 3.2 and whether the proposed agent's available space is significantly lower than the originating agent's. A pseudo-code example can be seen in Algorithm 4.

3.5 Neural Network Heuristic

Results from the handcrafted heuristics show them to be adequate for solving the problem defined, however, they are far from the theoretically best solution. The problem faced with handcrafted designs is that they are not as expressive e.g. can't inherently control for many edge cases or scenarios when compared to the highly expressive nature of a neural network. For example, if we are far enough away from the data's desired location we don't agent to suicide data unless there is another duplication in the locality. Currently, the handcrafted heuristic would need extra conditions to satisfy that behaviour, whereas the neural network automatically can produce that behaviour, dependent on training. It would also mean that poorly selected thresholds for different implementations would not be as problematic, as long as training is undertaken properly. The design is as follows; for each run of a simulation, a single neural network model is used on all agents.

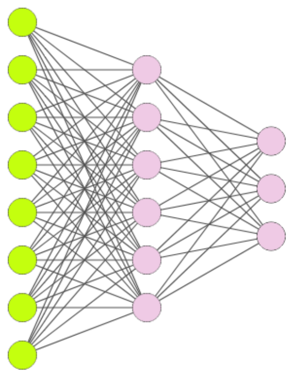

Figure 3.2: Neural network design - Thresholded argmax output

$$input = \begin{bmatrix} \frac{\psi_{app}}{\psi_{tot}} & \frac{n}{50} & r & \frac{s_{self}}{s_{loc}} & \frac{d}{\sqrt{8}} & \frac{\theta_{sui}}{1000} & \frac{\theta_{rep}}{1000} & \frac{\theta_{mig}}{1000} \end{bmatrix} \tag{3.8}$$

Where ψ_{app} is approved duplications to the agent and ψ_{tot} is total duplication requests sent to this agent, both being zeroed after the agent's step. n is the number of agents in the locality, r is the duplication ratio as described before. s_{self} is space left in the agent's memory and s_{loc} is the average space in locality excluding this agent. d is the distance to the data's data point. θ is how many iterations since the last action of a specific type. The three outputs correspond to Suicide, Replication, and Migration. We, therefore, argmax the output and then check whether it is above a threshold to trigger the action, in our case the threshold is > 0.05. The lime nodes in Figure 3.2 are LSTM nodes and Damon nodes are Dense nodes.

This network was selected because of its small size, relatively quick simulation speed, and fitness improvement. It's likely a larger network would be better with more LSTM layers, however, training time would be significantly increased. Simulation speed is a concern because of using a genetic algorithm that involves running multiple simulations. We are using a genetic algorithm because of two reasons; Firstly, it's impossible to know the correct output at a specific time, so a gradient descendant algorithm is not viable. Secondly, an action is unable to give an effective reward so reinforcement learning is also not viable. Therefore the only way to judge a network's performance is to run a simulation and compute a fitness function over the entire run.

$$fitness = \frac{\sum_{i=0}^{z} \left(max(\overline{d_i}) - min(\overline{d_i})\right)^2 - \frac{\left(\overline{q}_i - \frac{N_i}{4}\right)^2}{100} - \overline{\varpi_i} - std(\overline{\gamma_i})}{z} \tag{3.9}$$

Where z is the number of iterations run. $\overline{d}$ is each agent's average distance to their duplicated data's desired location. $\overline{q}$ is the average duplications over all data points, and N is global current active agents. $\overline{\varpi}$ is the mean instability over all data points. $\overline{\gamma}$ is the spread of agent's used memory space.

Square terms were used to emphasize a property we wanted the algorithm to focus on e.g. maximum spread of data and each data point to be duplicated in a fourth of the swarms population. The fitness function would need to be changed for different characteristics you desired for the swarm.

The genetic algorithm is run for 287 iterations with a population size of 40 with a mating rate of 36, therefore 4 completely random agents are created per iteration. Mating selection is made by a probability distribution over the magnitudes of fitness min-max normalized. Mating occurs by picking two agents from the previous population and randomly selecting weights from both models. Parent selection is a probabilistic model based on fitness values, MinMax normalized then set so all normalised fitness values sum to one. This, therefore, means that agents with higher fitness are more likely to be picked.

4 Analysis

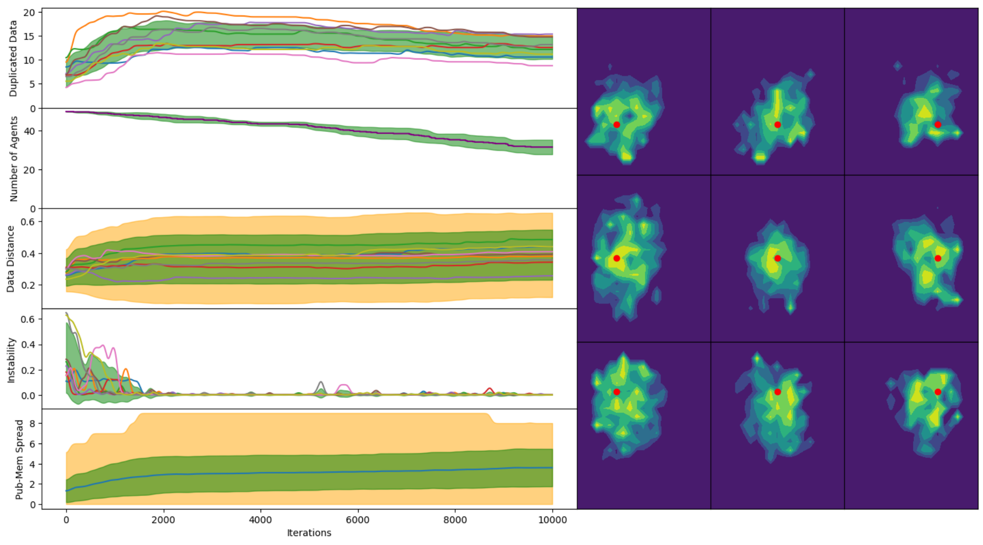

Figure 4.1: Static Heuristic on semistatic movement swarm, with non-correlated failures

The simulation is a 2D representation of a flat surface where agents can move freely around. Agents are randomly located on the surface within a square that is 75% of the environment's area. Then the closest agent to each defined datapoint will learn its location, once learned these agents cannot lose this specific data point. Agents are homogenous, with a small connection radius around them. The world coordinate system goes from -1 to 1 in both height and width, with 0 as the centre. An agent's connection radius is 0.25. Agents are assumed to have perfect connections and each agent can simultaneously respond to incoming packets and send outbound packets.

Agents can have two different types of data, one being private and the other public. Private data is a record of a data point from which it has learnt directly. This data cannot be deleted and will be prioritised over public data. Public data is learnt from interactions with other agents and can be deleted. Both types of data can be passed on to other agents in the form of public data.

A control loop runs for 10,000 iterations, where agents are threaded for a single step per iteration. All tests will be an average of 5 simulations. Data will have a Gaussian filter applied to make the graphs readable, using a kernel size of 50. An example of the results can be seen in Figure 4.1, and all subsections are explained below.

"Duplicated Data", is the number of duplicates in the swarm of a specific data point, represented by the individual line. The green area is the standard deviation around the mean of all data points.

"Number of agents", as the name implies is the number of active agents overall five runs. The purple line is the mean overall runs and the green area is the standard deviation from the mean.

“Data Distance”, is the distance from a duplicate to its respective point. The lines and green area are the same as before, and the yellow area is the range of results, from maximum to minimum.

“Instability”, is how many agents changed from having a duplicate to not or vice versa, per data point.

“Pub-Mem Spread”, is the number of duplicates per agent.

Finally, there is duplication density spread per data point. This enables visualization of how density in duplication changes across the physical distance of the swarm. The yellower colours signify more duplications to the number of agents than in blue areas. The red dot signifies the data's point of interest.

Ideally, we want a suitable amount of duplicated data with a close spread, no matter the starting location. The distance of data points to desired locations to have the widest range possible. Instability to be low in magnitude and not react significantly to certain changes in other factors and the spread of memory load to have the smallest range possible. In the visualisation section, we should see a widespread with a dense epicenter.

4.1 Static Heuristic

Firstly, we check whether the solution in Section 3.2 works as intended with different threshold values, as in Figure 3.1. Looking at Figure B.1, the reader will see that partially correct behaviour is observed. Tests were performed with semi-static movement at Threshold$_{rep} = \begin{bmatrix} 0.7 & 0.9 \end{bmatrix}$ and Threshold$_{sui} = \begin{bmatrix} 0.4 & 0.5 \end{bmatrix}$. These values are extreme and thus will highlight defects in the proposed solution.

It is observed that values of higher replication thresholds cause the effectiveness of the suicide threshold to decrease. This is because the effective duplicate area has too few agents inside to make an effective skimming strategy to duplications.

Too high a replication threshold and too low a suicide threshold causes massive instability at the fringe of the normal replication boundary. This is because the solution doesn't have a stable mapping, therefore switches between multiple optimums.

Both these observations show a flaw in the proposed design, in the fact that the threshold values need to be perfectly selected for the solution to be anywhere near effective in a real-world scenario. Compensation for this flaw will be attempted in Section 3.3.

Moving onto performance under ideal threshold conditions, Figures 4.1 and B.3. We start by looking at non-correlated failures on both movement-type swarms. The reader will see that the gradient of data duplications and loss of agents are fairly correlated, this is a good sign to show that the proposed heuristic is adapting to the environment as it changes, after the initial expansion phase. As expected the expansion phase takes longer in the circular movement swarm, due to the broken connection nature of the movement. Within the static movement swarm, the number of duplications per data point seems to be constant compared to the circular movement. This signifies that the heuristic is relying upon external factors to balance the loads, compared to actively trying to keep each data point balanced in the number of duplicates.

The external factor argument can also be considered in both positional subsections where the circular movement has further reach, as expected from the less cohesive nature of movement e.g. less chained connections.

Stability, in the circular network, is very erratic and periodic in nature. The periodic nature of instability can be explained by the circular motion of the agents, all the agents that start facing outwards will all converge back into the swarm in a periodic manner, creating a periodic gain in instability. However, even with this accounted for, the heuristic didn't perform adequately as the designed solution should've been able to handle these external factors. We would also like the magnitude of the instability to be decreased.

In "Pub-Mem Spread" we can see there is a large disparity between agents levels of used memory. This is not effective when thinking about scaling of the swarm and leaves a vulnerability. The vulnerability is if there is too much data in a single area then the learned data could be cut off from the rest of the swarm, Section 3.4 tries to solve this.

Moving onto correlated failures, Figures B.2 and B.4. A correlated failure occurs in the middle of the swarm at the 3000th iteration with a failure radius of 0.25 (Same as an agent's connection range).

As the reader can observe we are still experiencing the same problems as described previously. A positive observation is made regarding the recovery of the duplication values, after the correlated failure. An interesting observation is the lime green data point in Figure B.4, where it appears to be struggling to gain duplicates. Repeated tests indicate this to be an outlier, however, it is shown for transparency. The author believes it might have been associated with random placements of agents, meaning agents being close to the data's desired location, therefore minimizing its replication heuristic.

Another interesting observation in Figure B.2 "Data Distance", the purple data point, is its reduction preceding the spike at the failure. This is expected, during the initial expansion phase, the data will spread across the swarm quicker than the swarm moves to the center, therefore it will slowly bring its data points closer to itself over time. This is exaggerated because the expansion has more directions to move in. The opposite can be seen in other data points.

4.2 Dynamic Heuristic

Following the same approach as in Section 4.1, with the values α_{rep} = 15, β_{rep} = 2, α_{sui} = 150, β_{rep} = 3. $\psi + = 50$ every time a duplication request is made to this agent and $\psi - -$ every iteration till 0. These are in accordance with the equations in Section 3.3.

Testing the threshold value changes, we observe the spread is the same, however, the instability has been reduced significantly in magnitude. This heuristic aimed to try and solve the instability problem, rather than just reduce it. As can be seen, the curves are nearly identical. Suggesting our solution might not behave in the predicted way. Further investigation is required with results from correctly picked thresholds.

Comparing Figures B.9 and B.8 compared against Section 4.1 we can see that the heuristic has significantly improved the stability of the external factors, and stopped its periodic nature. However, the change induced instability when comparing Figure B.6 to its former self. This indicates that the solution should only be used in cases where high external instability exists, therefore making the internal instability increase is worth it.

An alarming observation is made in Figure B.9 where the increase in duplicates does not appear to slow down. This could lead to an over-saturation of duplications. The logical explanation for this behaviour is due to the movement of the swarm, as Figure B.7 doesn't share the same behaviour. This is most likely caused by agents breaking apart from the swarm, leading to fewer suicides of duplicates.

When comparing all the results of the dynamic heuristic versus the static heuristic, we see a new tendency in the duplication density spread per data point section. The tendency is to have higher duplication densities to the sides of the map. This is most prominent in the circular motion swarms. These results are misleading us into believing that there are a lot of duplications in that area. On closer inspection of the results data, it was apparent that these highly dense areas were caused by a few agents with duplicates. Due to the minimal size of agents, the density will appear large in size even though there are only one or two duplications present.

An interesting observation regarding static-movement of non-correlated failures is, the duplicates tend to have varying levels of duplicates for different data points, in comparison to any other test. Mentioned in Section 4.1, the cause, rather than being an outlier or suffocation from other duplicates, is now believed to be inherent with the movement of the swarm. As the agents gather around the middle they averagely get further away from their data points, in turn making it less likely to replicate and more likely to suicide. This also explains the neatly spread-out duplicate lines in static-movement correlated failure tests.

4.3 Dynamic Heursitic with Migration

The new method proposed in Section 3.4 was created to promote positive load sharing between the agents. Figures B.10 and B.11 both show this positive affect at the cost of instability. This internal instability is created by the movement of migrating data points and is to be expected. We can also see that the method is attempting to do the same in the circular movement swarms. However, it lacks actual performance, with the same costs to internal instability. From further investigation, the cause was discovered to be the disconnected nature of movement in the swarm. Agents have their own subsection of the swarm where they are roughly equal when globally they are not. The semi-static movement handles this better because they are all drawn to the middle, if this was not the case, we would see similar results, due to the sub-swarm nature of the movement.

In the density duplication spread section we can observe the new method significantly spreads data further than before, especially in semi-static movement cases. This shows that the new method is more tolerant to larger correlated failures than previously proposed methods. This characteristic is more in line with the best conceivable solution to the problem defined in Section 1.1. However, the increase in instability is an unwelcome sight.

Figure B.13 continues the uptrend mentioned in Section 4.2. This shows that this behaviour is most likely not coincidental and is as mentioned previously.

The changes made to convert the dynamic heuristic to using migrations were intended to give a mild improvement without any significant side effects. However, even though the behaviors are similar, the use cases for each differ dramatically. This will be discussed in Section 5.1.

4.4 Neural Network Heuristic

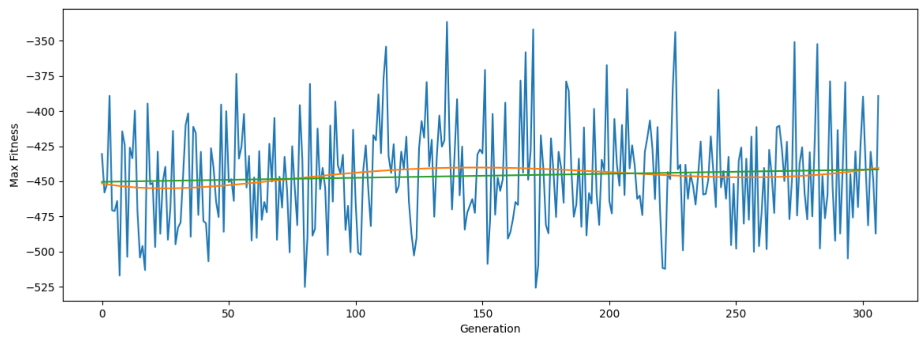

Figure 4.2: Genetic Algorithm fitness over generation

The results of the genetic algorithm described in Section 3.5 are disappointing. Initially, the fitness on average was increasing and from running two separate tests seemed to have an improvement in action-based performance. However, following significant numbers of generations it became apparent that the solution was not improving effectively.

Figure 4.2 shows an improvement with a gradient of 0.03. However, this is the max fitness of said generation. These results indicate that there is a problem with either the design of the solution or the implementation of the genetic algorithm. Therefore, run a test as shown in Figure B.14 to find out some additional information. Due to time constraints on training and poor performance the genetic algorithm was only trained for semi-static motion with non-correlated failures.

The results are not promising, other than nearer the end from about the 8000 iterations mark, where our duplicated data fits quite well to the target of a quarter of the population.

A factor that could have led to the failure of the genetic algorithms learning is the low population size, constricted by training time and hardware. Another limitation could be the mating selection allowing for worse agents to reproduce. A better solution would be to have a top number of agents being able to reproduce and have a slightly larger mutated percentage of the population. Running along these lines, the mutated agents relied on Tensor Flow's clone feature which assigns biases to zeros and uses "glorot_uniform" for weights. These values could have been too small in the weights to get past the 0.05 threshold and we could've lost expressive power from not allowing biases. I believe one of the biggest contributions to the slow learning rate is large variations in the simulation environment, meaning that a model that performed effectively last round might perform significantly worse in the next. The author believed that this would have been averaged out over time due to more effective models performing better on average. However, it didn't act as expected and made the genetic algorithm essentially a bad random search. An improvement on this design would be to have the model run for multiple simulations to get a good indication of average performance or to have trained on only certain simulations. Then introduce random simulations for the agent to learn a partially overfitted strategy then unlearn slightly to fit other scenarios. The first was not selected due to computation time and the second because of preliminary tests indicating good results from the used design.

The model in and of itself could have been flawed, in that it could be too large in input size for the length of time allowed for training. To restrict the size of the input we could have reused functions from previous sections to nudge the neural network to learn faster, however, this would have lost potential expressive power. Another interesting proposition is to have the neural network produce dynamic thresholds. This would restrict the effectiveness of the neural network, however, would ensure correct average behaviour and could potentially lead to a possible custom gradient descendant learning method using a mapping of the fitness function to a predicted value of where we want each threshold to be at certain times.

5 Conclusion

5.1 Conclusion

The handcrafted algorithms proposed in Sections 3.2, 3.3, 3.4 solve the problem proposed in Section 1.1, with varying degrees of effectiveness and behaviour. When comparing them to a theoretical hybrid solution, we see obvious flaws, partially due to the nature of a fully homogeneous approach. In a real-world application, these approaches would not be as viable as the alternative of the hybrid model, unless under very specific circumstances. These circumstances include tight power consumption limits, large population swarms, and high natural instability in connections. The method proposed in Section 3.5 was a fail, for reasons described in Section 4.4.

The method proposed in Section 3.2, is shown to be effective at combating the problem, however, it lacks the multiple characteristics needed for real-world applications. We saw promising results with duplication spread in more volatile movement swarms with both correlated and non-correlated failures being dealt with appropriately. Behaviour presented in the semi-static movement was not desirable, had an overreliance on correct threshold values, and lacked tolerance for large correlated failures.

An updated version of the method was proposed in Section 3.3 to combat some of these downfalls. The main target of change was the instability of the system. This is where agents would keep replicating and suiciding data at the fringes of the heuristic boundary, occurring with a poorly selected suicide threshold value. To combat this, we used dynamic thresholds to slow the instability. This was done by using a time-based slow on suicides and a local instability slow on replications. The combination of which would give us fast growth but a slow release of duplicated data, leading hopefully to finding stable solutions. These additions significantly improved characteristics of stability especially in naturally volatile swarms, e.g. Circular movement. We can see that instability issues were not solved by this method, but reduced in magnitude. The nature of the method meant that internal instability was increased but there was a significant reduction in external instability, so it would be more effective in certain swarm scenarios. However, it always outperformed the previously proposed method from

Section 3.2.

The additions proposed in Section 3.4 are the most controversial in terms of use case. This is because the method achieves the objective of spreading memory load over the swarm, at the cost of higher instability. As a byproduct of the migration action, the method has more tolerance towards larger correlated failures. However, the amount of duplications seems to be affected more by the loss of agents, and memory spread is less affected in volatile swarms.

The proposed solutions assume that all data is highly important and all agents near the desired location should know that data. This makes these solutions harder to adapt to other situations, where this feature is less required. An example of this situation is a large sensor swarm network, where it might be good to have a duplication density change over the network but close to the data point, all sensor agents don't need to know the information. A future example of this potential use is moonquake detection. Agents close to the moonquake epicenter wouldn't need to know there was one coming because they wouldn't have time to react but may have time to relay a message. Agents within the damage radius that have time could strap themselves to the ground to prevent damage. Agents further away are not in harm's way so should be less likely to strap down, as strapping down may cause failures in and of itself.

The project achieved two handcrafted solutions to the problem proposed, both with different use cases. If improvements were made from the suggestions made in Section 5.2, then I believe that these solutions could be very successful both in adaptability and performance, even when compared to a theoretical hybrid approach.

5.2 Futher Improvements

There are further improvements that could be made to improve the performance of the suggested designs. Either computationally or methodologically. Looking at methodology two design changes could drastically improve the characteristics of the algorithms. One is to allow for message hopping between agents to increase symbolic locality size. This would in turn give agents wider knowledge of the global swarm, however, would increase computation time and power draw per agent. The second is moving to a hybrid control which would allow for a more deterministic policy, however, this leads to downsides both of message hopping and problems described in Section 2.2.

Keeping with the same methodology we can reduce power consumption from messaging, however, it could lead to erroneous results. Rather than each agent asking a question and awaiting a response from the locality, meaning that each agent has to respond to every agent in its locality. Instead at the start of each agent's step, they broadcast their relevant information. Agent's in the locality then update their internal tables of the agents in their locality, therefore computation time is decreased because we have less IO time and no power loss due to not replying to other agents' requests as before. Tables would need to release an agent's information if they haven't broadcast in a while to know that an agent has left the locality. This leads to an issue of possible executing actions based on old information either from agents running at different speeds or being out of the locality.

The currently proposed solutions lack control for larger correlated failures. The handcrafted heuristics only sum together certain values and do a comparison on the output value, rather than making different decisions based on the values of each characteristic like the neural network can. This means that we get a linear behaviour, lacking cases that could help with increasing the max distance of duplicate data. The nonlinear nature of the neural network will improve the chances of being able to change heuristic behaviour when certain values surpass certain thresholds. This behaviour would be trivial to implement within a hybrid swarm, e.g. make sure at least two duplicates are as far away from their data point as possible. We would also have a higher guarantee that the data wouldn't be lost because of the global view of the hybrid approach.

Currently, the effective storage of the swarm is far less than the amount of memory the swarm has as a whole. A priority system could be used to ensure a larger practical collective memory size compared to the smaller size current proposed heuristics provide. To make the priority system even more effective if an agent needs to lose data to learn a higher priority piece of data, that data could be migrated to other agents in its vicinity to make room for the higher priority data. Running along the same line as a priority system, a scheduler would be an effective way to handle the scalability of storage size per agent. As currently, the looping nature through memory works as a round-robin approach. An example of this is using the tables approach mentioned above, each public memory could stop itself from being picked for suicide if agents in the locality don't have any duplications of that data point.

A Code apendix

Algorithm 5 Semi-Static movement

```
procedure MOVE
    $dirforce \leftarrow$ Define vectors from other agents to self
    $forces \leftarrow dirforce$ with magnitudes $(0.24 -$ Current magnitude$)$

    Apply small force to center of map

    $face \leftarrow$ Angle of resultant force on $forces$
    Point at angle $face$ and move forward by 0.0002

    return True
```

B Results apendix

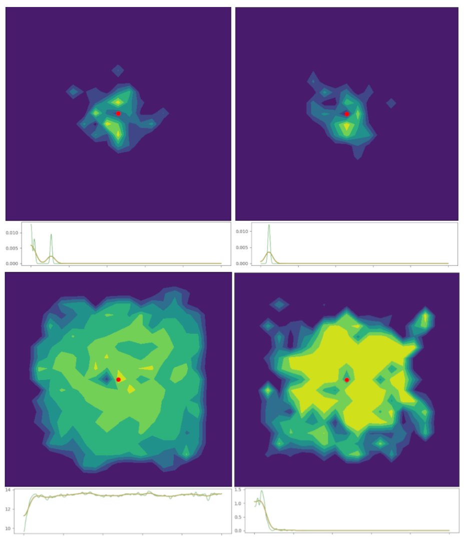

Figure B.1: Static Heuristic on semistatic movement swarm, with threshold changes

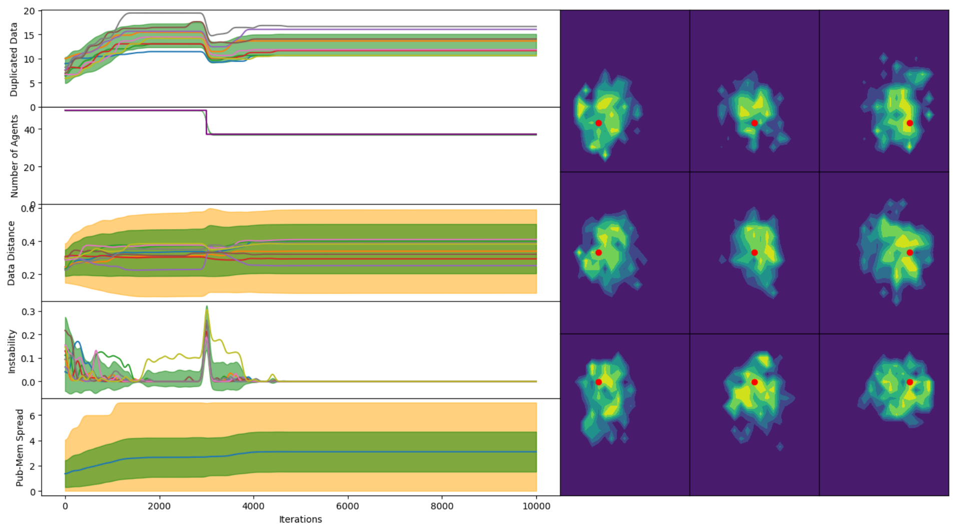

Figure B.2: Static Heuristic on semistatic movement swarm, with correlated failures

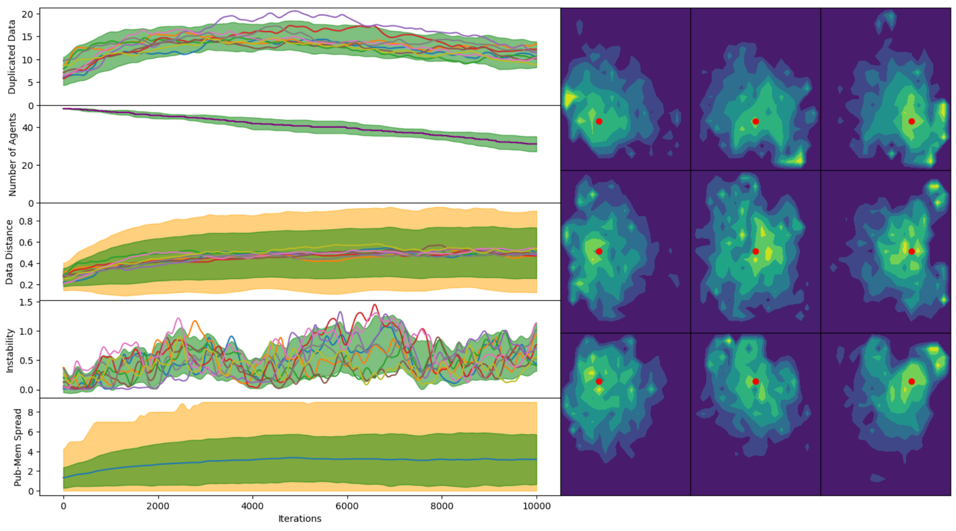

Figure B.3: Static Heuristic on circular movement swarm, with non-correlated failures

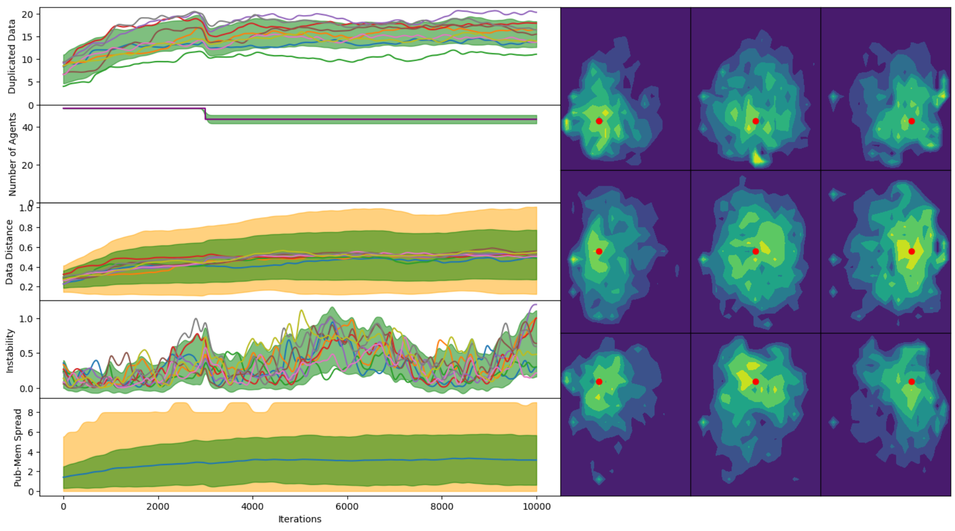

Figure B.4: Static Heuristic on circular movement swarm, with correlated failures

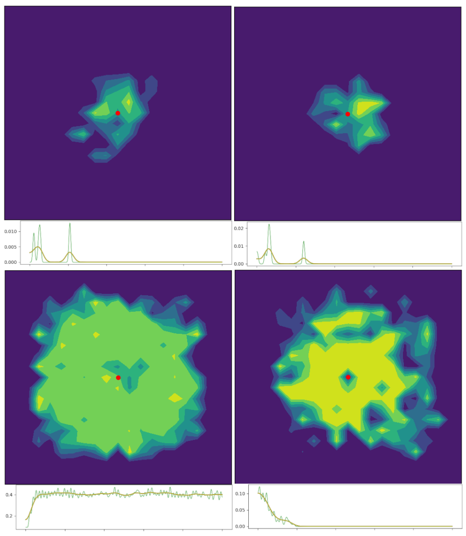

Figure B.5: Dynamic Heuristic on semistatic movement swarm, with threshold changes

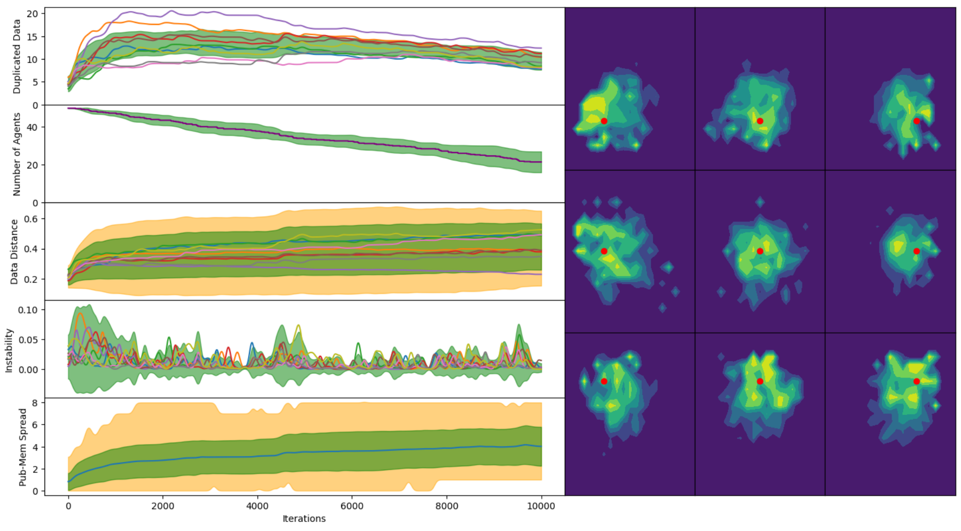

Figure B.6: Dynamic Heuristic on semistatic movement swarm, with non-correlated failures

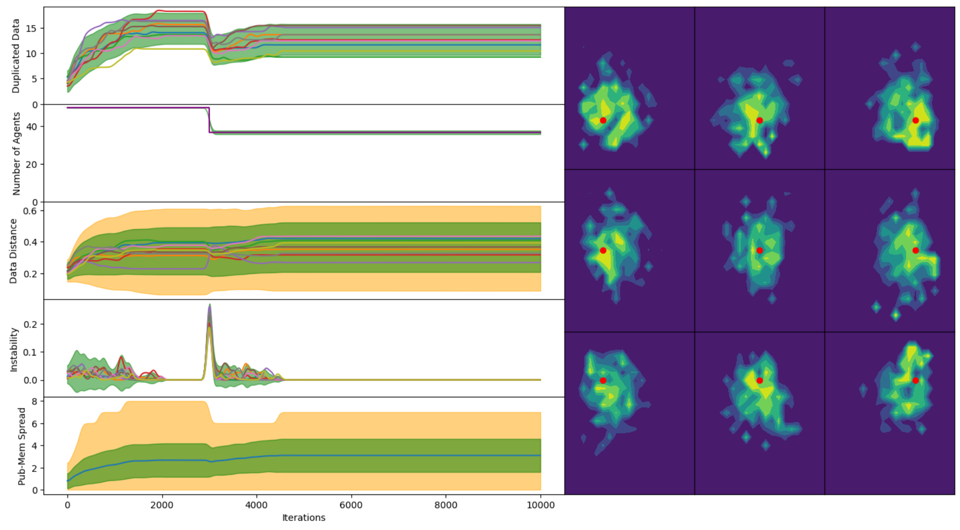

Figure B.7: Dynamic Heuristic on semistatic movement swarm, with correlated failures

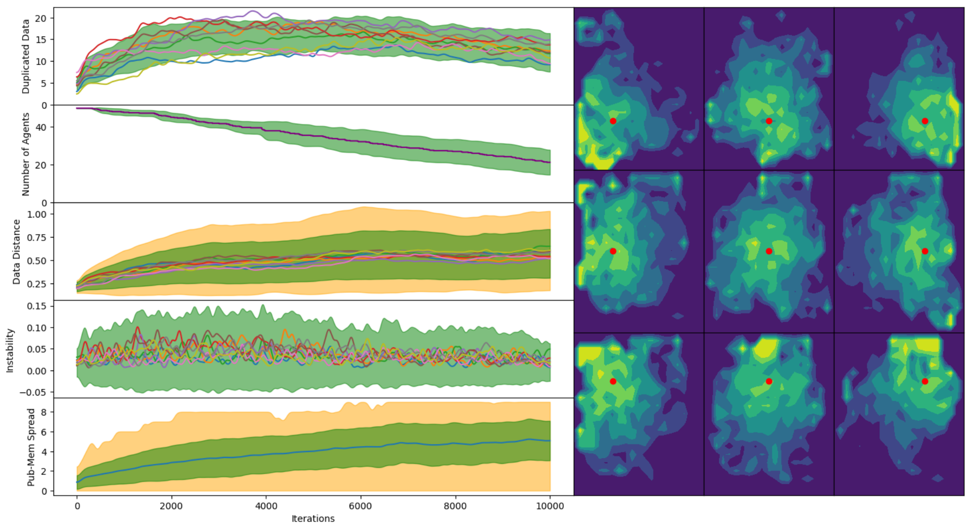

Figure B.8: Dynamic Heuristic on circular movement swarm, with non-correlated failures

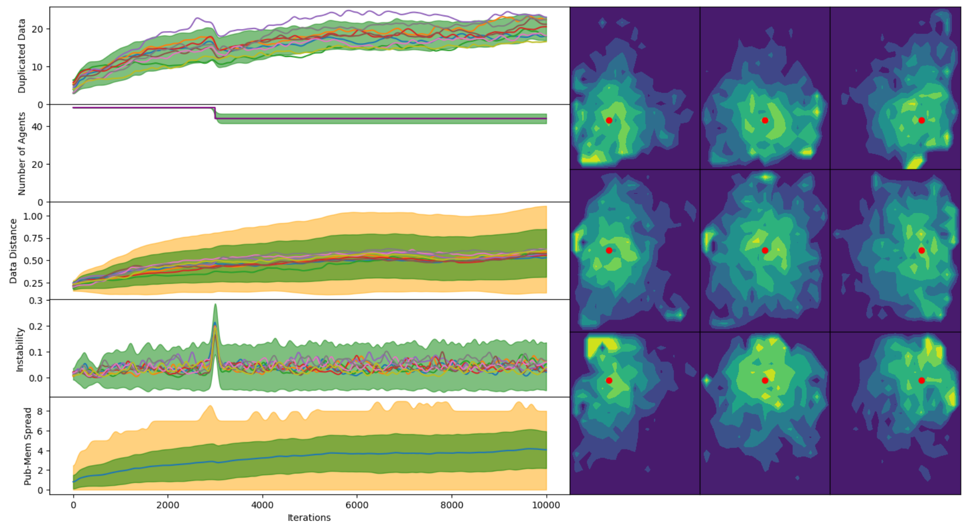

Figure B.9: Dynamic Heuristic on circular movement swarm, with correlated failures

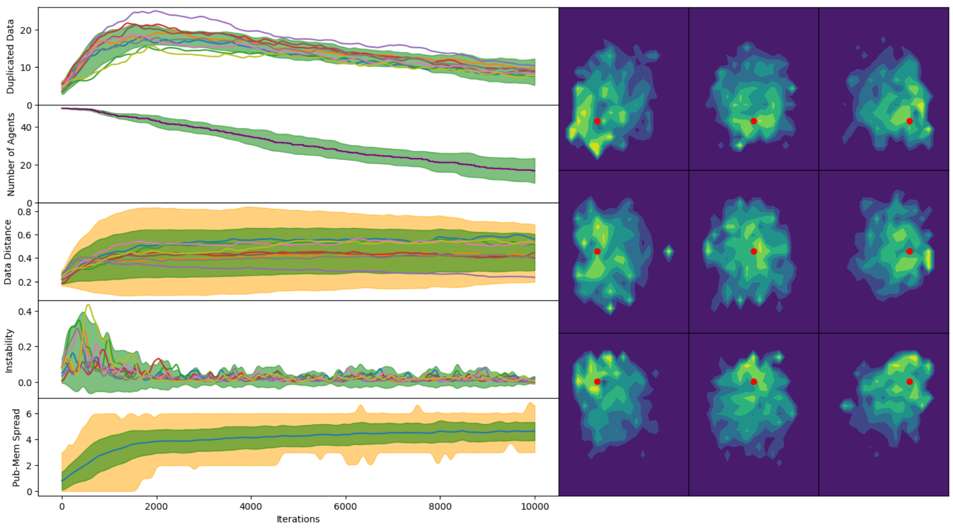

Figure B.10: Dynamic Heuristic with Migration on semistatic movement swarm, with noncorrelated failures

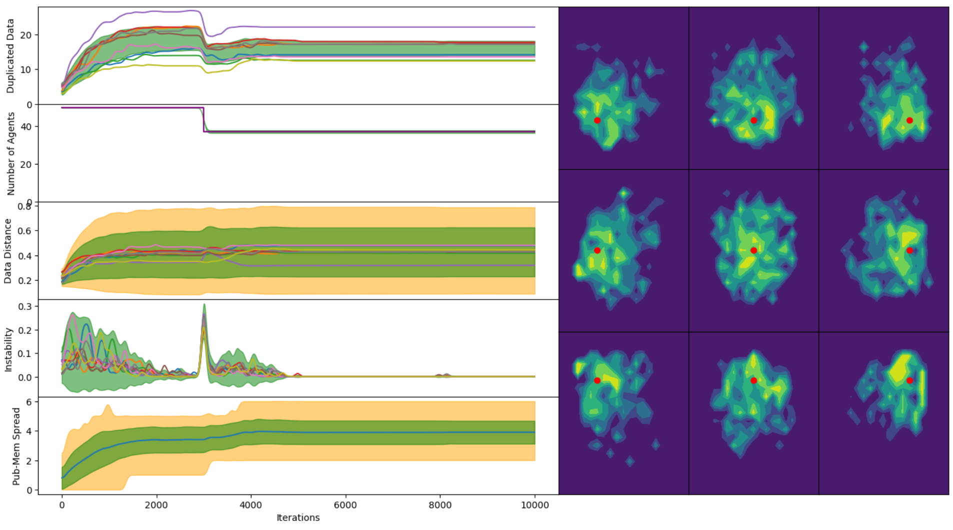

Figure B.11: Dynamic Heuristic with Migration on semistatic movement swarm, with correlated failures

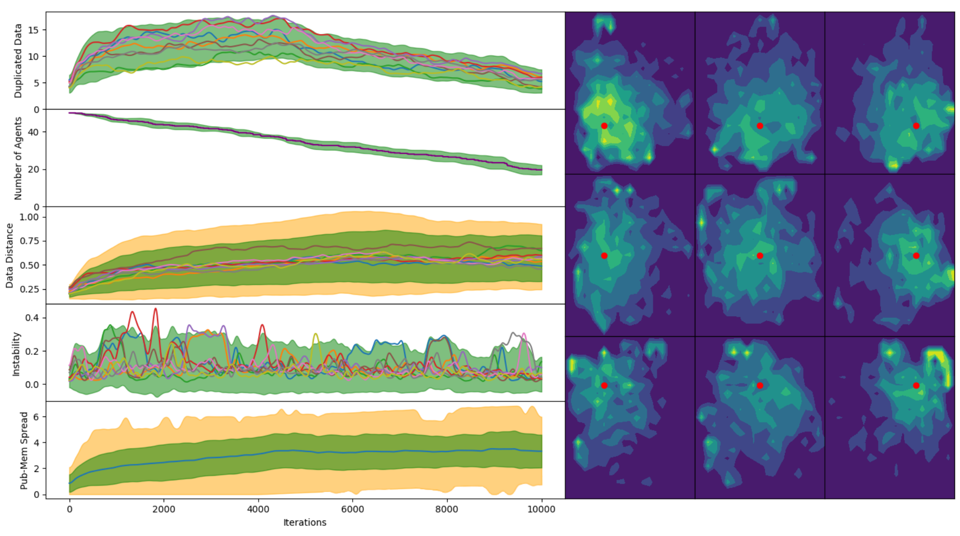

Figure B.12: Dynamic Heuristic with Migration on circular movement swarm, with noncorrelated failures

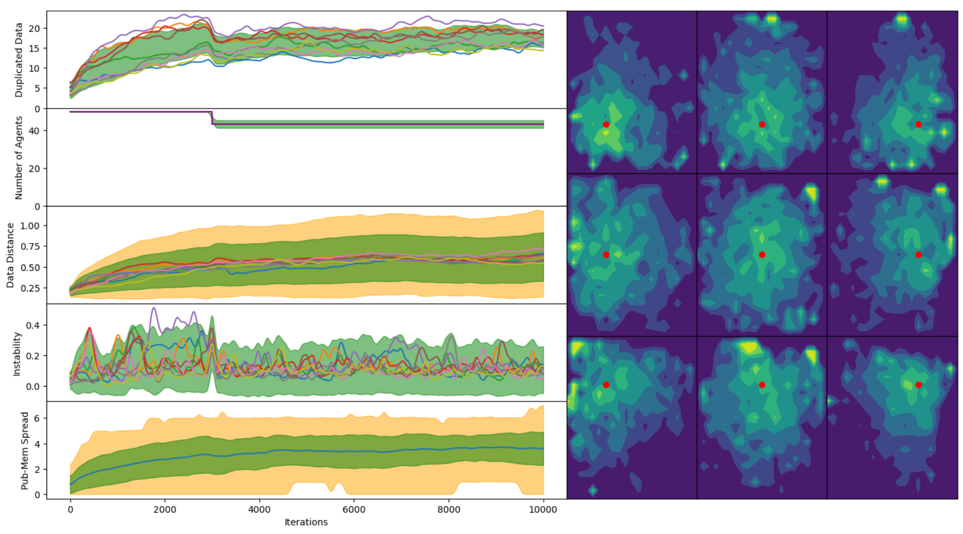

Figure B.13: Dynamic Heuristic with Migration on circular movement swarm, with correlated failures

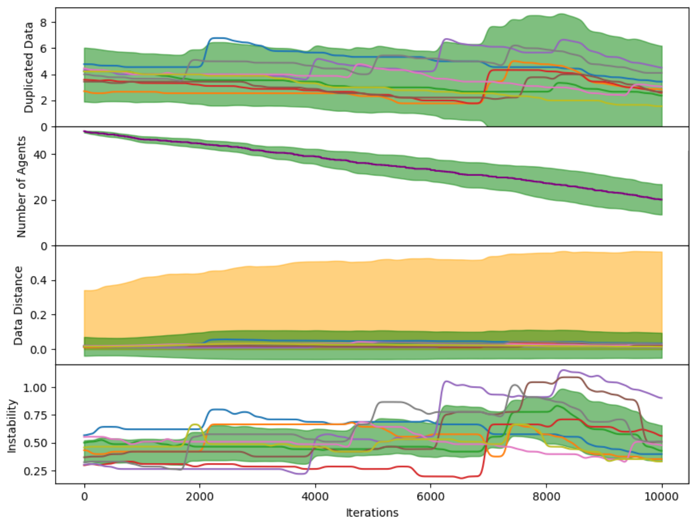

Figure B.14: Discouraging results from Neural Network Heuristic, semi-static movement and noncorrelated failures

Bibliography

[1] J. C. Barca and Y. A. Sekercioglu, “Swarm robotics reviewed,” Robotica, vol. 31, no. 3, pp. 345–359, 2013.

[2] V. Kumar and F. Sahin, "Cognitive maps in swarm robots for the mine detection application," SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483), Washington, DC, 2003, pp. 3364-3369 vol.4, doi: 10.1109/ICSMC.2003.1244409.

[3] H. Wang, D. Wang and S. Yang, “Triggered Memory-Based Swarm Optimization in Dynamic Environments,” in Applications of Evolutionary Computing, M. Giacobini, Ed. Berlin, Germany: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2007, pp. 637–646.

[4] D. A. Lima and G. M. B. Oliveira, "A probabilistic cellular automata ant memory model for a swarm of foraging robots," 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, 2016, pp. 1-6, doi: 10.1109/ICARCV.2016.7838615.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems. Cary, NC, USA: Oxford University Press, 1999.

[6] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li, ‘A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation’, Association for Computing Machinery, vol. 7, p. 11, 2011

[7] C. Mims, ‘Why CPUs Aren't Getting Any Faster’, MIT Technology Review, 2010. [Online]. Available: https://www.technologyreview.com/2010/10/12/199966/why-cpus-arent-getting-any-faster/. [Accessed: 01-Dec-2020].

[8] U. Troppens, W. Müller-Friedt, R. Wolafka, R. Erkens, and N. Haustein, ‘Appendix A: Proof of Calculation of the Parity Block of RAID 4 and 5’, in Storage Networks Explained: Basics and Applic-

ation of Fibre Channel SAN, NAS, ISCSI, InfiniBand and FCoE, U. Troppens, Ed. Chichester: Wiley United Kingdom, 2009, pp. 535–536.

[9] J. Liu and H. Shen, "A Low-Cost Multi-failure Resilient Replication Scheme for High Data Availability in Cloud Storage," 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), Hyderabad, 2016, pp. 242-251, doi: 10.1109/HiPC.2016.036.

[10] N. Bonvin, T. G. Papaioannou, and K. Aberer, A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage. New York, NY, USA: Association of Computing Machinery, 2010.

[11] Legal Services Act. 2007.

[12] A. Prahlad, M. S. Muller, R. Kottomtharayil, S. Kavuri, P. Gokhale, and M. Vijayan, ‘Cloud gateway system for managing data storage to cloud storage sites’, 20100333116A1, 2010.

[13] B. Czejdo, K. Messa, T. Morzy, M. Morzy, and J. Czejdo, ‘Data Warehouses with Dynamically Changing Schemas and Data Sources’, in Proceedings of the 3rd International Economic Congress, Opportunieties of Change, Sopot, Poland, 2003, p. 10.

[14] ‘Key-Value Scores Explained’, HazelCast. [Online]. Available: https://hazelcast.com/glossary/key-value-store/. [Accessed: 02-Dec-2020].

[15] L. Lamport, ‘The Part-Time Parliament’, in Concurrency: The Works of Leslie Lamport, New York, NY, USA: Association of Computing Machinery, 2019, pp. 277–317.

[16] D. Agrawal and A. E. Abbadi. The tree quorum protocol: An efficient approach for managing replicated data. In VLDB’90: Proc. of the 16th International Conference on Very Large Data Bases, pages 243–254, Brisbane, Queensland,Australia, 1990.

[17] S. Lynn, ‘RAID Levels Explained’, PC Mag, 2014. [Online]. Available: https://uk.pcmag.com/storage/7917/raid-levels-explained. [Accessed: 06-Dec-2020].

[18] J. Hu et al., Eds., HiveMind: A Scalable and Serverless Coordination Control Platform for UAV Swarms. ArXiv, 2020.

[19] D. Calvaresi, A. Dubovitskaya, J. P. Calbimonte, K. Taveter, and M. Schumacher, Multi-Agent Systems and Blockchain: Results from a Systematic Literature Review. Cham, Switzerland: Springer Interna-

tional Publishing, 2018.

[20] L. A. Nguyen, T. L. Harman and C. Fairchild, "Swarmathon: A Swarm Robotics Experiment For Future Space Exploration," 2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR), Houston, TX, USA, 2019, pp. B1-3-1-B1-3-4, doi: 10.1109/ISMCR47492.2019.8955661.

[21] M. Y. Arafat and S. Moh, "Localization and Clustering Based on Swarm Intelligence in UAV Networks for Emergency Communications," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8958-8976, Oct. 2019, doi: 10.1109/JIOT.2019.2925567.

[22] D. Jackson and F. Ratnieks, ‘Communication in ants,’Current Biology,vol. 16, pp. 570–574, 2006.

[23] C. W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model. ACM, 1987.