Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Swarm Memory

Harry Burge

2021-April

Supervisor: Simon O'Keefe

Acknowledgements

// TODO

Contents

List of Figures

Executive Summary

The aim of the report was to propose and test a solution for directional memory storage policies within a swarm environment, and to improve the authors knowledge on the research of swarms. The proposed solution is to provide redundancy for correlated and non-correlated failures, without pure duplication of data throughout the swarm. Directionality of the policy is defined as having a higher density of duplications nearer to the point that the information is needed.

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

A design is a way that we control an agent's actions in order to get the emergent characteristics of the global swarm. This emergent behaviour will be reviewed using five different global factors. Amount of duplications per data point, 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. Performance will be based on not too many or too little duplications with a roughly even distribution per data point, having a large range in distance with the mean being as close to minimum as possible and for instability to be of lowest magnitude. These factors are tested on both circular motion and semi-static motion in order 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 first 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, left for much improvement in stability, even distribution of memory load and handling of

a volatile connectioned swarm.

The second version of the first design style carried on from the previous by changing the static thresholds to be dynamic, in order to counteract the stability problem of the previous version. To measure instability on a local scale, each agent as a duplication request came in would increase a instability variable, this would be reduced over time. Output of sigmoid functions where then added to the weighted sum, with inputs being time from last suicide for the sucide threashold and instability for replication threashold. The version achieved its target by significantly reducing the magnitude of instability without changing much behaviour of the previous version. We also saw the nature of the instability change in circular motion to being a 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 for the first time. Migration was 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 agents available space, then migrate the data. 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 counter parts. Therefore leading to different use cases for both the second and third versions.

After reviewing 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 therefore leads to using the computational power of neural networks in order to solve for these complex behavioral characteristics. A genetic algorithm was used to train said neural network, using a global fitness function after a simulation was run with each agent using the same model. Unfortunately the learning rate was very slow due to selected hyperparameters and poor judgement on fitness function evaluation being on only one simulation run. This therefore led to a non-testable version due to the extremely poor performance.

The field of swarms are riddled ethical and moral issues. Specifically, swarms have the high potential to be abused in surveillance, breaching people's privacy. Within the military domain, is it ethical to have systems that not only have the choice over human life, but are also so redundant that 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 Introductury Material

1.1 The Problem

The problem this report will try and 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.

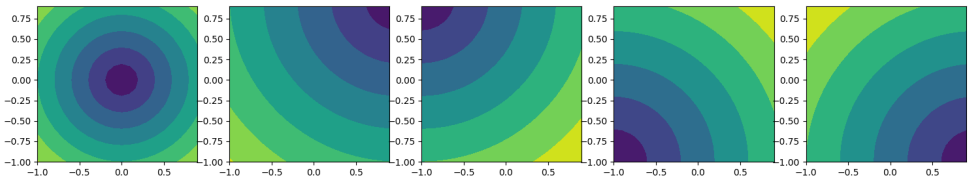

Figure 1.1: Data duplication density based off distance to datas desired location

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

A real life example of where this type of problem arises, is in mine field 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 can only permit the knowledge of a minimal amount of mines. Therefore agents closer to the mine should be prioritized in knowing that mine's location. Agents further away from the mine do not need to know about its location as they will need knowledge of other closer mines.

Complexity is increased when 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 which can withstand the high locality and dynamic behaviour of a swarm. Analyses will then be performed on generated policies as 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 give us an indication of how effective a policy of this type can be. The second will use a multi-agent interaction approach using a neural network. This will then 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. Therefore the report will be structured as follows, firstly we bring the reader up to date with relevant literature and explain key concepts that will be 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 of which will delve into more of a background behind swarms, specifically relating to the problem and possible solutions. We'll then 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. This then leads to analysing 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 heads more towards the robotics side and can be seen as the study of practical implementation of a swarm, whether that be movement or communication within the swarm. This is on the rise in industry, as society's pace increases and manual labor is automated out. Whether its drone delivery to inpatient customers or mapping areas in dangerous environments [1].

These areas often are highly integrated rather than disjoint from each other, 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 gone down the route of optimization on distributed problem-solving algorithms, as 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.

This relative lack of research into collective memory, seems to the author 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, like how invaluable it would be to 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 swarms to be used.

An explanation for why swarm based memory management solutions are an under developed 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 to 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, too storage on an effectively a highly dynamic network.

The main use cases nowadays 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 for the future when technology can support and utilize the fullest potential that swarms could offer.

2 Litreture Review

2.1 Cloud/Backup storage policys/schemes

Like most things in computer science, cloud storage started off relatively simple (In nowadays terms). 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 to 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 needed 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 and with no relation to other failures with the system as a whole. For example, a server node can shut down from a software failure, therefore connection is lost, typically this is completely independent of other servers in the rack. The opposite, correlated failure, is when a device fails with other devices with a relation linking the failure. A typical example is on-mass restarts from a power surge in a data-center. The power surge event is what links the failures together. To be correlated doesn't require close geographical location, however, when mentioned later on it should be regarded as such.

To control for these two types of failures there are multiple defensive strategies. Focusing on strategies that are reactionary rather than preventative. For local redundancy, typically RAID is used to ensure safety in a storage failure. Then a replication policy [9] is used for internode redundancy. Working in tandem provides 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 too. This duplication of data onto another storage device means that if one fails then 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 alright policy for handling failures, however, without tracking of global duplications it can

lead to over used 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 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. An example, If a video is hosted in one country and replicated within said country then international viewers might have delays to their streaming. If we then tried to compensate for that by hosting it in multiple countries, those other countries might not view the video as much. This therefore means we are wasting storage space of which we could use for other more popular videos. This therefore leaves us trying 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 you have key-value pairs on multiple devices on a network where 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 a drastic change. Servers running over a network generally have complete connectivity e.g. Global scope. Also servers have a constant power supply compared to the average swarm agent. When restricting the masters scope we have to rely on messages bouncing from agent to agent. Which will first of all reduce processing capability, power loss will be more significant to agents closer 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 usally be because of too many duplicates. 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, if we ran them sequentially it could possible be that if one suicided the other would not have. If they were the only two left of the duplicated data then the policy would've failed. 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 essitially 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 not needed 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 thus that if a tolerant amount of disks are lost the data that was lost can be reconstructed. 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 go over that then the data is lost. RAID is predominantly used internally within a storage node to provide redundancy against disk failures and increase speed of writes that are typically on hard disks.

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

2.2 Swarm robotics

As explained broadly in the Section 1.2, the study of swarms are 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, compared to their counterparts. From papers read by the author, predominantly the topics undertaken are testing distributed solvers and comparing them to already researched global scope solutions. An example of this within the TSP domain is a genetic algorithm versus AS-TSP [5]. Another good example of where these algorithms excel is 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 we have 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.

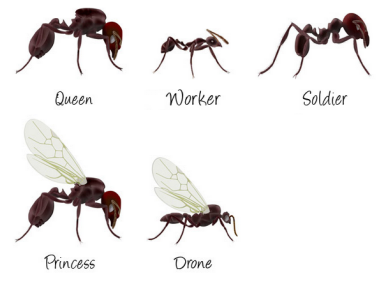

Figure 2.1: Example of a hetrogenus ant colony

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 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 for the slack of the swarm by being able to transport them longer distances than the swarm could normally cope with and carry more complex hardware.

The argument against heterogeneous swarms is there 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 travel longer distances and have a recharge point. Without the boat the swarm fails after some time. The decision to use a heterogeneous swarm in this case is valid because if the naval ship is lost, then something has already gone horrendously wrong.

To have physical differences in agents, is to breed efficiency. However this can only hold true if the vulnerability of losing too much of a key class is mitigated. Common rules for mitigation are found in nature's swarms. These are; jobs need to be either interchangeable between all types of agents however some agents are more efficient at that job [22] or the jobs that are specific need to be non-essential for the colony's survival. These swarms are under research because human implementations currently don't have the adaptability of biology, therefore 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 that will specialise 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 at the microbial level. Biology's natural adaptability compared to current standards of robotics, semi-heterogeneous swarms are exploited better [1, 5]. Therefore we maximise for the floors 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 nano size.

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

Within the practical implementation of swarms, everything gets a bit messy. Usually there is not a clear cut framework that 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, but the key point is that a miner could become a farmer if it was required. The reader might wonder why physical hybrid approaches aren't regarded as the best of all the approaches. This is because when bringing a theoretical solution into the real-world we gain massive complexities. How can we guarantee job type distribution? What about the complexity of changing between job type hardware? These are all questions that are needed to be explored, by the creator. Usually 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 labbelled distributed intelligence. This therefore 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 certain agents control other agents' decision making. This can be handled in different ways, two promoniante ways are hivemind control [18] and royalty/hierarchical control. Hivemind is where one agent controls the entire swarms decision making e.g. The master controller will give abstract ideas as commands and the agents will execute them with their own decision making. A hierarchical approach follows the same style as hive mind, however deals with scalability better. Where we have a power structure of certain agents being sub leaders of leaders. Both control structures leave themselves vulnerable in its physical implementation due to bad actors and power loss distributions over the swarm.

Hybrid control of a swarm is a heterogeneous policy that can adapt to changes of leaders and are usually designed for physically homogeneous/hybrid swarms. 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 for homogenous style agents to act in heterogeneous fashion. This can create more deterministic behaviours compared to emergent behaviors of homogeneous control, and can also help with the power loss distribution problem 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 better concept solution became apparent after conducting more research into cloud based storage.

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

- Implementation of directionality would be tricky and not consistent
- If the two agents die/leave the locality at the same time then the parity cannot be restored
- The algorithm relies on a swarm that breaks connections often to spread data
- There is no duplication reduction, therefore 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 allows us to implement 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 will 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 said 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, then we might spend more time assigning a leader rather than doing our actual task. Scaling of a hybrid system is harder because of the partitioning of agents to leaders. This justifies the authors choice to focus on distributed homogeneous replication policies.

3 Design

3.1 Experiments

To understand and evaluate the proposed solutions we need to see how they react to different scenarios. The scenarios are designed as 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 1, is used to test the ability of the solution on mostly stable networks, therefore allowing us to see the solutions inherent stability. In the opposite case we have circular movement which adds physical instability. This allows comparison of stability with internal and external factors. Correlated and non-correlated failures should be self explanatory as of the problem, defined in Section 1.1.

The proposed solutions will be simulated and critiqued based on five factors. Amount of duplications, this will allow us to see whether the policy over or under staurates duplications and how they are affected by loss of agents. Distance from desired point, will show acceptable ranges for correlated failures therefore the larger the range the better. Instability, as most swarms will be battery powered we need to reduce the amount of communications between agents to have a practical solution. Storage load across the swarm, this will allow us to see how effective our policy is at distributing load over the agents so as to not oversaturate an agent causing possible exclusion of agents to the swarm.

3.2 Static Heursitic

This solution takes heavy inspiration from [10]. We take the methodology of each agent controlling its own data and how it wants to distribute it, done through actions of Replication, Suicide and Nothing. Migration is not applied, therefore we can piggyback on natural movements of the swarm to get duplicates out further.

Thresholded heuristics decide whether to do either action, based off these factors:

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

These values once collected are then put through a weighted sum and compared against thresholds. A pseudo code version of this can be seen in 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, because no suitable agents are in the locality, the data will have this action performed 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 the time that it isn't.

For each iteration we focus on one public data point in memory, this is a basic iteration through memory. For a further improvement we could dynamically choose which data point to service, talked about more in Section 5.2. The current implementation doesn't scale well for large amounts of data.

To gain information required an agent broadcasts a packet to agents in the locality. They will then respond with relevant information, e.g. do they have a duplicate of the data point, amount of spare memory. Once collated at the originating agent, it works out the specified values as shown in the code. There are improvements that can be made to the handling of broadcasting/replying and will be explained a bit more in the Section 5.2.

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

Changes to weightings and thresholds applies significant behavioural changes to the swarm. The increase of replication threshold means that

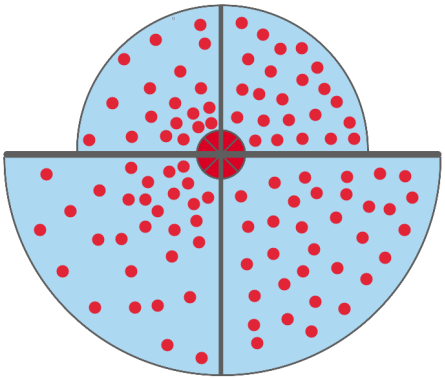

Figure 3.1: Example of changing of replication and suicide threshold on a uniform agent density

duplications will spread out less, avoid higher duplication density areas and not spread when agents are saturated in other data. The higher the suicide factor means there is less of a duplication density diffrence across the network.

For an easier understanding of the heuristics behaviour when changing the threshold values we can look at Figure 3.1. This shows how the heuristic with changing threshold values would influence the spread of duplicated data from a data point on a uniformly space swarm. The smaller red dots represent agents that have a duplicate with data points of the large red circle. Replication threshold increasing in magnitude can be seen by the changing in distance, e.g. blue circle. And suicide threashold when increasing is 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 distance to the datas target point, s is average space 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 lends

the heuristic to be optimised with fitting techniques like genetic algorithms. This would be done for each individual application seeing as there is no dynamic nature to the heuristic. For tests we use values 3.2, none of which have any significance and are purely from tinkering to get good performing results.

Originally “suicide data” in Line 16 of Algorithm 1, was done using Paxos [15]. This ensured that a data point could not go extinct from suicide, and could only from external factors like failures. However from preliminary testing this didn’t make much of a difference in our scenario, but should be added into any practical applications of this algorithm for more guaranteed redundancy.

3.3 Dynamic Heuristic

To control instability of the static heuristic, as described in Section 4.1, we need to enforce a behavior such that when instability increases, factors that contribute to instability are reduced. To keep with an efficient style of homogeneous control, rather than polling agents in the locality for wider information therefore increasing network congestion, computation time and used memory. We instead keep a local track of information on the agents to give a rough guess of local/global behaviour.

Firstly, to control instability of the swarm, it wants to reduce the chance of suicides happening after one has been executed. This in and of itself will not solve instability, however, slows its affects down. The previous heuristic tries to seek a global optimum where it becomes static. In an unstable state the optimum keeps changing at the fringe. Therefore restricting our suicides leads to the possibility of replicating to a key agent outside the usual bubble which could lead to a static optimum. However, this only solves instability at the fringe and not inside. 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, 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 as to use sigmoid functions based on the inputs described abstractly above, which are used to modulate the thresholds as to give them a dynamic coping mechanism with 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 this 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.

Sigmoid functions were used because we can control when the heuristic should be unimpeded or impeded with a grey area in between so if a data needs to duplicate quickly it can still over power the block. α and β can be changed on both heuristics to give different behaviours.

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. This is if an agent learns about a data point it might not be able to spread that information to the rest of the swarm, either because of a physical lack of agents, or that those agents' memories are full, therefore stopping the spread of the data point. There are three possible solutions to the memory being full. Agents repeat duplication requests if 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 for cases of oversaturation, however, in our tests such situations will not occur. Therefore 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].

We inherently don’t want to limit the speed at which data can be migrated, for example an agent with ten data points meets an agent with none, we want them to both walk away with five. This will inherently increase the instability of the swarm, therefore being a trade off.

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 by replacing lines 11-12 of Algorithm 1 with Algorithm 3.

3.5 Neural Network Heuristic

Results from the handcrafted heuristics show them to be adequate for solving the problem defined, however, they are not close to a theoretically best solution. The problem we face with handcrafted designs is that they are not as expressive and can’t handle mainy different edge cases when compared to the highly expressive nature of a neural network. For example if we are far enough away from the datas desired location we don’t want them to ever suicde unless there is another duplication in the locality. Currently the handcrafted heuristic would have to add extra conditions to satisfy that behaviour, whereas the neural network automatically can produce that behaviour. It would also mean that poorly picked thresholds for different implementations would not be as big of an issue, 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.

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

Where ψ_{app} is approved duplications to this 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 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 actually do that action, in our case the threshold is > 0.05. The lime in Figure 3.2

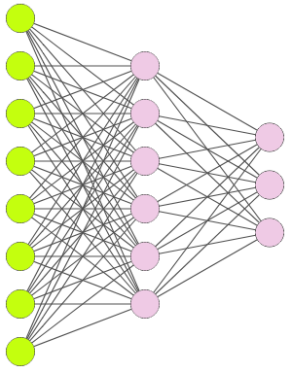

Figure 3.2: Neural network design - Thresholded argmax output

are LSTM nodes and damon are Dense nodes.

This network was picked due to its small size and relatively quick simulation speed and fitness improvement. Most likely a larger network would be better with more LSTM layers, however, training time would be significantly increased. Simulation speed is a concern as we are using a genetic algorithm which involves running multiple simulations. We are using a genetic algorithm because of two reasons. Firstly we don't know the correct output at a specific time so a gradient descendant algorithm is not viable. Secondly doing a move can't give us a definitive reward so reinforcement learning can't be used. Therefore the only way to judge a network's performance is to run a simulation and do a fitness function over the entire run.

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

Where z is the number of iterations ran for. $\overline{d}$ is each agent's average distance to their duplicated data's data points. $\overline{q}$ is the average duplications over all data points, and N is global current active agents. $\overline{\omega}$ 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 that we wanted the algorithm to focus on the most e.g. maximum spread of data and each data point to be duplicated in a fourth of the swarms population. The fitness function would have to be changed for different characteristics you would want 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.

4 Analysis

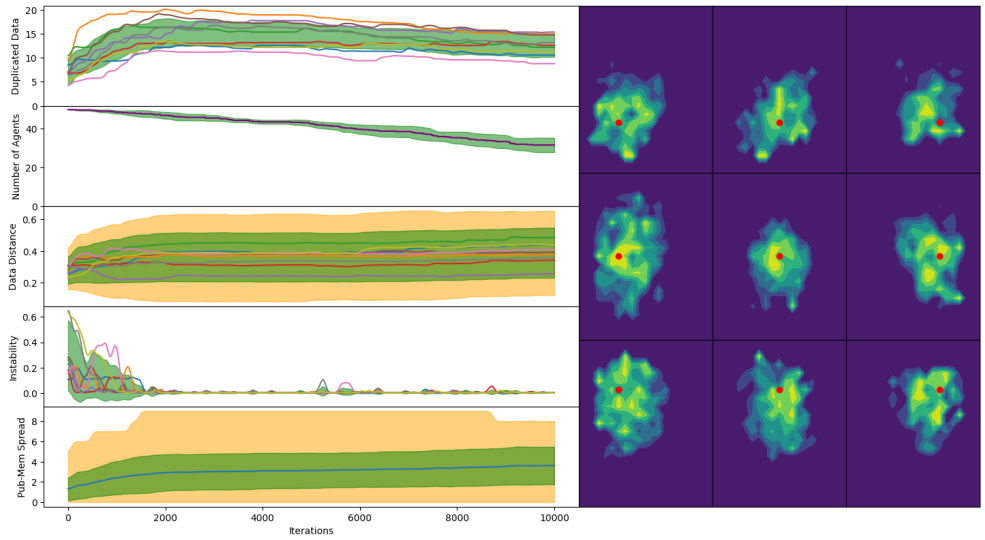

Figure 4.1: Static Heuristic on semistatic movement swarm, with non-correlated failures

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 datapoint. Agents are homogenous, with a small connection radius around them. The world is height and width of 2 and an agents 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 other public. Private data is a record of a data point of 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 also have a gaussian filter applied as to make the graphs readable, kernel of size 50. An example of the results can be seen in Figure 4.1, and all subsections will be explained below.

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

"Number of agents", as the name implies is the number of active agents over all five runs. The purple line is mean over all 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, e.g. Max and min.

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

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

Lastly, there is duplication density spread per data point. This allows the visualization of how density in duplication changes across the physical distance of the swarm. Where the yellower colours signify more duplications to the number of agents than in a blue area. The red dot signifies the data’s point of interest.

We are ideally looking for 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 heavily 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 wide spread with a highly dense epicenter.

4.1 Static Heursitic

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 causes the effectiveness of the suicide threashold to decrease. This is because the effective duplicate area has too few agents inside to make an effective skimming strategy to duplications.

It is also observed that a too high replication threshold and a too low 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 optimum.

Both these observations show a flaw in the proposed design in the fact that the threshold values need to be perfectly chosen in order for the solution to be anywhere near effective in a real world scenario. This flaw will try to be combated in Section 3.3.

Now 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, as of the broken connection nature of the movement. Within the static movement swarm the number of duplications per data point seems to be preset compared to the circular movement. This signifies that the heuristic is relying upon external factors to balance the loads.

The external factor argument can also be brought up in both positional sections 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, thus creating a periodic gain in instability. However even with this accounted for, the heuristic didn't perform adequately as we would like the solution to be able to handle these external factors. We would also like for the magnitude of the stability 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 us a vulnerability. This is that if too much data is in a single area then the learned data might be cut off from the rest of the swarm, this will try to be solved in Section 3.4.

Moving to correlated failures, Figures B.2 and B.4. A correlated failure happens 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 will see we are still suffering from the same problems as described before. A positive observation is made about 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 seems to be struggling to get more duplicates. From repeated tests this seems to be an outlier, however, is shown for transparency. The author believes it might have been to do with random placements of agents, therefore meaning less agents being close to the data duplicate point, therefore minimizing its replication heuristic.

Another interesting observation in Figure B.2 "Data Distance" purple data point, is its reduction before the spike at the failure. This is expected, at the initial expansion phase the data will spread across the swarm quicker than the swarm moves to the center, therefore will slowly bring its data points closer to itself over time, this is exaggerated because the expansion has more directions to move. 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 can see that the spread is the same, however, the instability has been reduced significantly in magnitude. The aim of this heuristic was to try and solve the instability problem rather than just reduce it. As can be seen, the curves are nearly identical. Therefore suggesting that our solution might not behave in the predicted way. We look into this further with more results with good valued thresholds.

From 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 has 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 the internal instability increase is worth it.

An alarming observation is made in Figure B.9 where the increase in duplicates do not seem to slow down. This could lead to 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, therefore leading to less 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 side of the map. This is most prominent in the circular motion swarms. These results are misleading into believing that there are a lot of duplications in that area. On closer inspection of the results data it was visible that these highly dense areas were caused by a few agents with duplicates, due to the minimal size in agents the density will appear large in size even though there is only one or two duplicates there.

An interesting observation about static-movement of non-correlated failures is that the duplicates tend to have varying levels of duplicates for different data points, compared to any other test. Mentioned in Section 4.1, the cause rather than being an outlier or suffocation from other duplicates, it is now believed to be inherent with the movement of the swarm. As the agents gather around the middle they averagly get further away from there data points, in turn making it less likely to replicate and more likely to suicide. This can also explain 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, so is to be expected. We can also see that the method is trying to do the same in the circular movement swarms. However, it lacks actual performance with still the same costs to internal instability. From further investigation the cause was found to be the disconnected nature of movement in the swarm, agents have their own subsection of the swarm where they are roughly equal 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 then we would see similar results, due to the sub swarm nature of the movement.

In the density duplication spread section we can see that the new method significantly spreads data further than before, especially in semi-static movement cases. This shows that the new method is more torrent to larger

correlated failures than previously proposed methods. This characteristic is more inline 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 be 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 talked about in Section 5.1.

4.4 Neural Network Heuristic

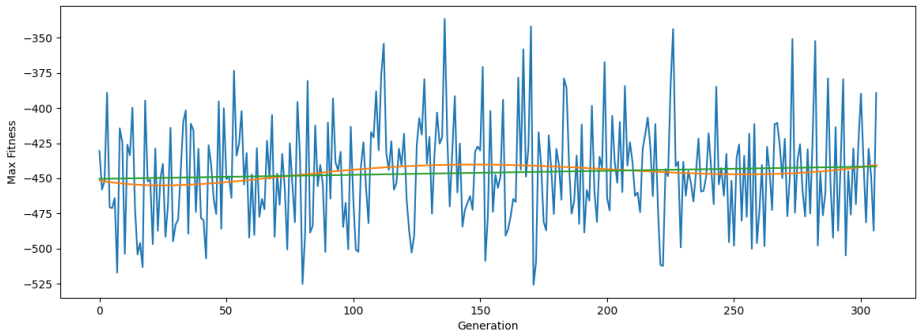

Figure 4.2: Genetic Algorithm fitness over generation

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, after a lot more 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 max fitness of said generation. These results indicate that there is a problem with either the design of the solution or implementation of the genetic algorithm. We therefore run a test as shown in Figure B.14 to find out some more information. Due to time restraints 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 low population size, restricted 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 be 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 was good last round might perform significantly worse in the next. The author believed that this would have been averaged out overtime due to better 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 in order to get a good indication of average performance or to have trained on only certain simulations. Then introduce random simulations in order for the agent to learn a partially overfitted strategy then unlearn slightly to fit other scenarios. The first was not chosen because of computation time and second because of preliminary tests showing 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 in order to nudge the neural network to learn faster, however, this would have lost us potential expressive power. Another interesting proposition is to have the neural network produce the dynamic thresholds. This would restrict the neural networks effectiveness, however, would ensure correct average behaviour and could possibly lead to possible gradient descendant learning 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

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, 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, shows to be effective at combating the problem, however, it lacks 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 handled appropriately. Behaviour is semi-static movement was not desirable, had an over reliance on correct threshold values and didn't have tolerance for large correlated failures.

An updated version of the method was proposed in Section 3.3 in order to combat some of these downfalls. The main target of change was instability of the system, this is where agents would keep replicating and suiciding data at the fringes of the heuristic boundary, occurring with a badly picked suicide threashold value. To combat this effect we used dynamic thresholds in order 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, therefore hopefully 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 ment that we increased internal instability but could significantly reduce external instability, so it would be more effective in different swarm scenarios. However, always out performed the previous 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 completes 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 loss of agents and memory spread is less affected in volatile swarms.

The proposed solutions assume that all data is highly important and that all agents near a data point should know its data. This makes these solutions harder to adapt to other situations where this feature is less needed. An example of this situation is large sensor swarm networks, where it might be good to have a duplication density change over the network but close to the data point it's not necessary for all sensor agents to know the information. A future example of this kind of 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 agents within the damage radius that have time could strap themselves to the ground in order to not acquire damage.

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 a bunch of improvements that could be made in order to improve the performance of the suggested designs. Either computationally or methodologically. Looking at methodology there are two design changes that could drastically improve characteristics of the algorithms. One is to allow for message hopping between agents in order 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, gaining the downsides both of message hoping and problems described in Section 2.2.

Keeping with the same methodology we can reduce power consumption from messaging, however, could lead to erroneous results. Rather than each agent asking a question and awaiting a response from the locality, therefore meaning that each agent has to respond to every agent in it's

locality. Instead at the start of step per agent, they broadcast their relevant information. Agent's in the locality then update their internal tables of the agents in the locality, therefore computation time is decreased because we have less IO time. Tables would need to release an agent's information if they haven't broadcast in a while in order to detect that they are out of the locality. This therefore leads to a problem of possible executing off old information either from the agent running slower or out of locality.

With current proposed solutions we 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, therefore not having cases that could help with increasing 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 cross certain thresholds. This behaviour would be trivial to implement in a hybrid network, 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 in order to insure 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 in order to learn a higher priority piece of data, said 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.

A Code apendix

Algorithm 1 Static Heuristic Agent

```
1: procedure STEP
2:     move()
3:
4:     if Learned of new data point then
5:         Replicate new data point to locallity
6:         return True
7:
8:     d ← Euclidean distance to data current data point
9:     r ← (local) Duplicates on agents / Number of agents
10:    s ← (local) Average space available / Max public memory
11:    X_rep, X_sui ←Equation 3.1
12:    W_rep, W_sui ←Equation 3.2
13:
14:    if h_rep (Equation 3.3) > Threshold_rep then
15:        Replicate current data point to locallity
16:
17:    if h_sui (Equation 3.3) > Threshold_sui then
18:        Suicide current data point
19:
20:    Iterate to next public data point
21:
22:    return True
```

Algorithm 2 Dynamic Heuristic Agent - Insert

```
1: θ ← Iterations since last suicide
2: ψ ← Internal stability
3: λ_rep ← Equation3.4
4: λ_sui ← Equation3.6
5: X_rep, W_rep ←Equation 3.5
6: X_sui, W_sui ←Equation 3.7
```

Algorithm 3 Dynamic Heuristic Agent With Migration - Insert

$\theta \leftarrow$ Iterations since last suicide
$\psi \leftarrow$ Internal stability
$\lambda_{rep} \leftarrow$ Equation3.4
$\lambda_{sui} \leftarrow$ Equation3.6
$X_{rep}, W_{rep} \leftarrow$Equation 3.5
$X_{sui}, W_{sui} \leftarrow$Equation 3.7

$SX_{rep} \leftarrow$Equation 3.1
$SW_{rep} \leftarrow$Equation 3.2

$\delta \leftarrow$ Max avalibale space - Our avaliable space

if $h_{rep}(SX_{rep}, SW_{rep})$ (Equation 3.3) $>$ Threshold$_{rep}$ and $\delta > 1$ **then**
    Duplicate item to target agent

    if Successful **then**
        Suicide current data point
        return True

Algorithm 4 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

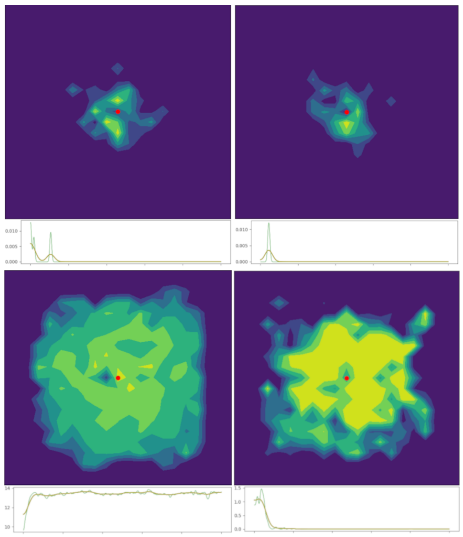

Figure B.1: Static Heuristic on semistatic movement swarm, with threshold changes

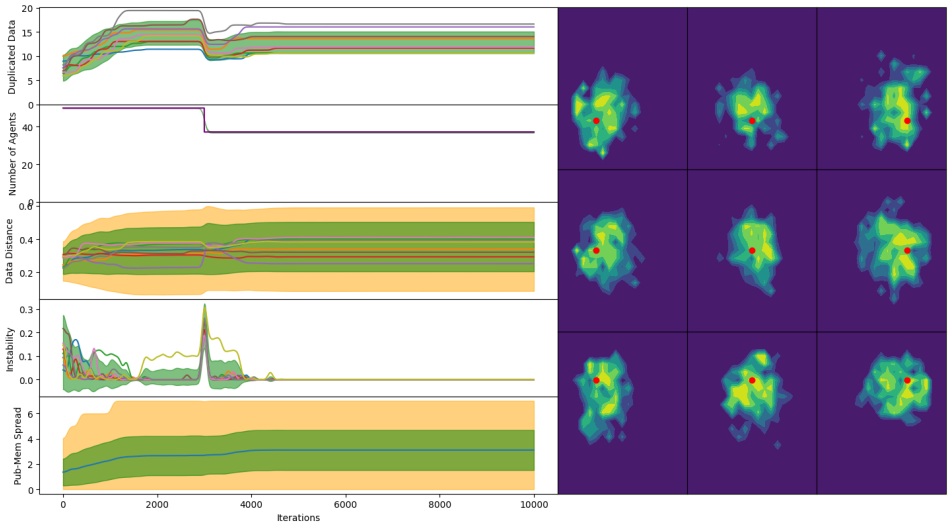

Figure B.2: Static Heuristic on semistatic movement swarm, with correlated failures

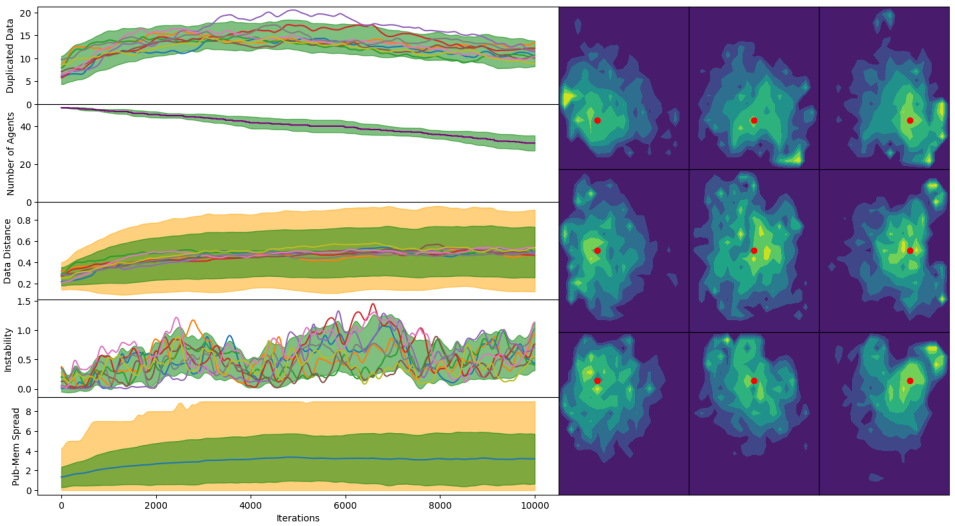

Figure B.3: Static Heuristic on circular movement swarm, with non-correlated failures

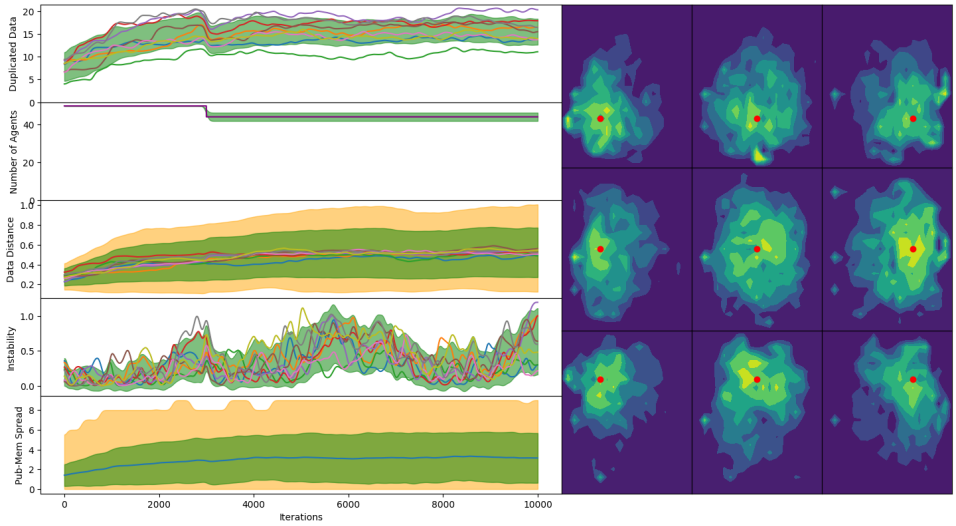

Figure B.4: Static Heuristic on circular movement swarm, with correlated failures

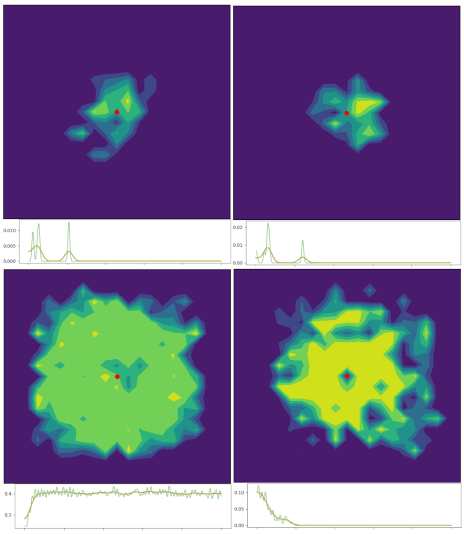

Figure B.5: Dynamic Heuristic on semistatic movement swarm, with threshold changes

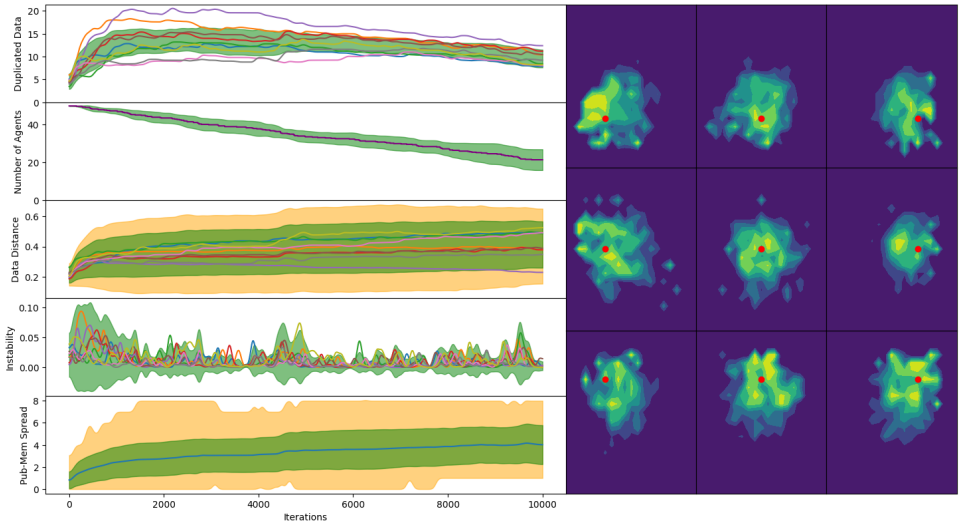

Figure B.6: Dynamic Heuristic on semistatic movement swarm, with non-correlated failures

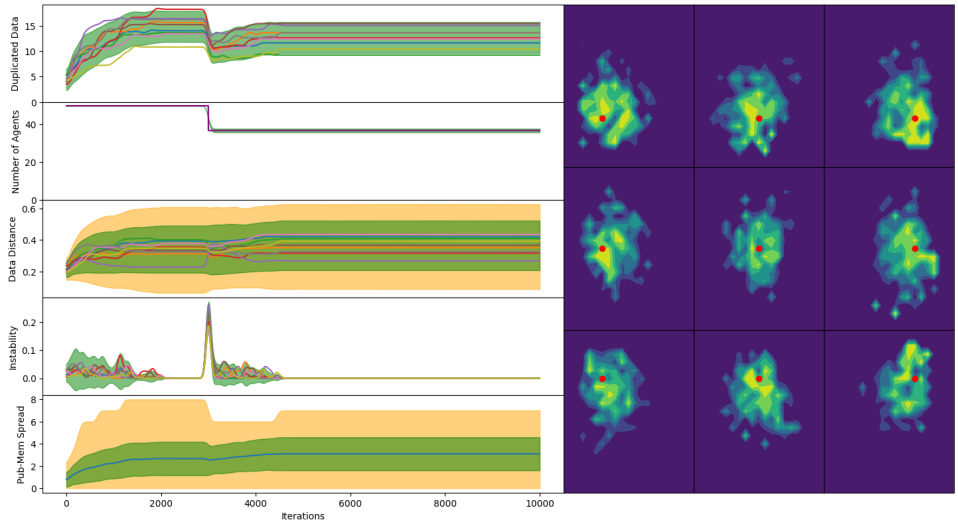

Figure B.7: Dynamic Heuristic on semistatic movement swarm, with correlated failures

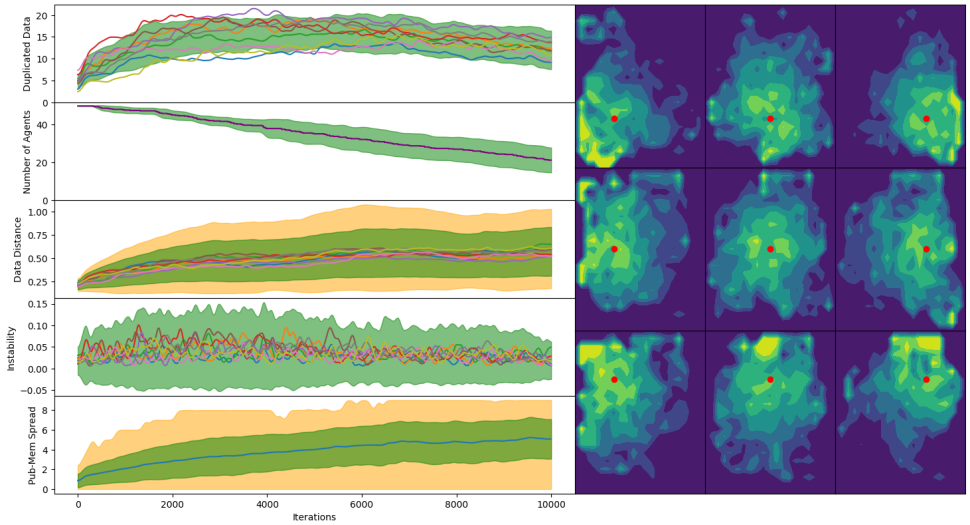

Figure B.8: Dynamic Heuristic on circular movement swarm, with non-correlated failures

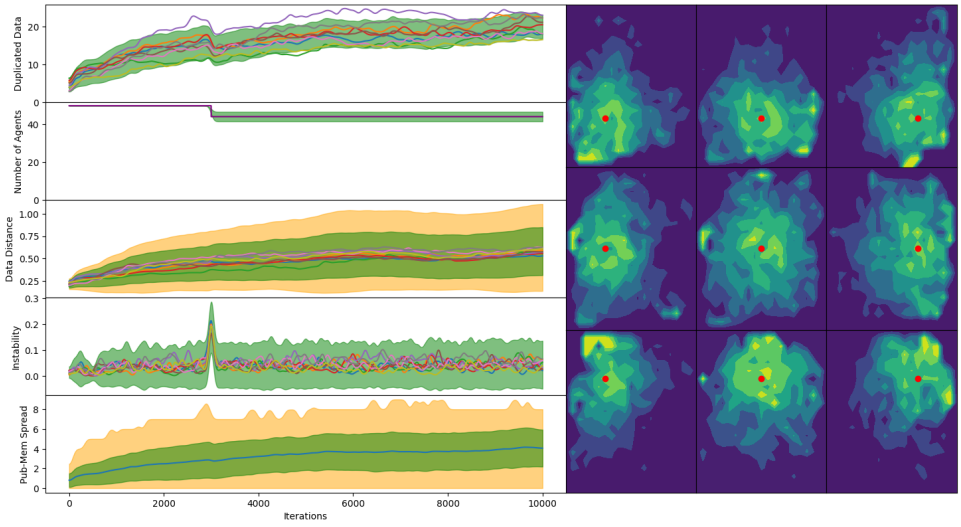

Figure B.9: Dynamic Heuristic on circular movement swarm, with correlated failures

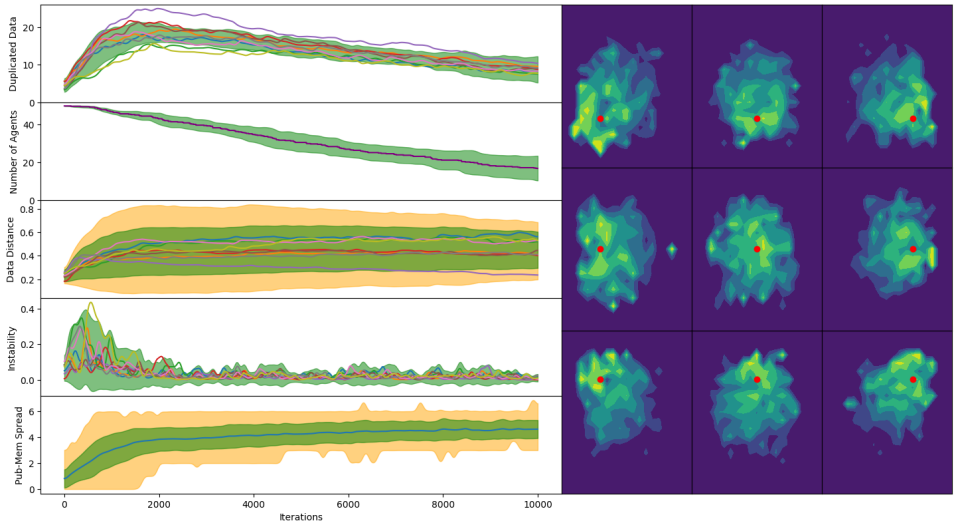

Figure B.10: Dynamic Heuristic with Migration on semistatic movement swarm, with noncorrelated failures

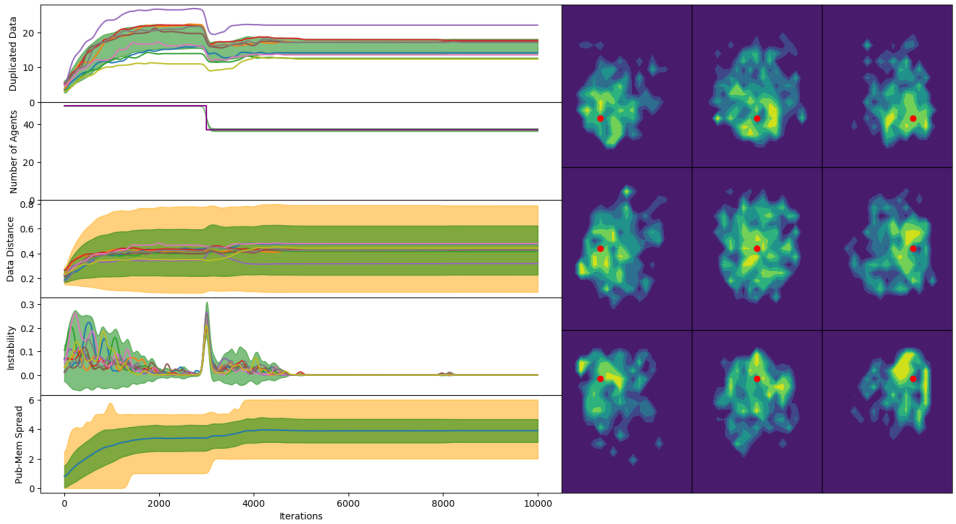

Figure B.11: Dynamic Heuristic with Migration on semistatic movement swarm, with correlated failures

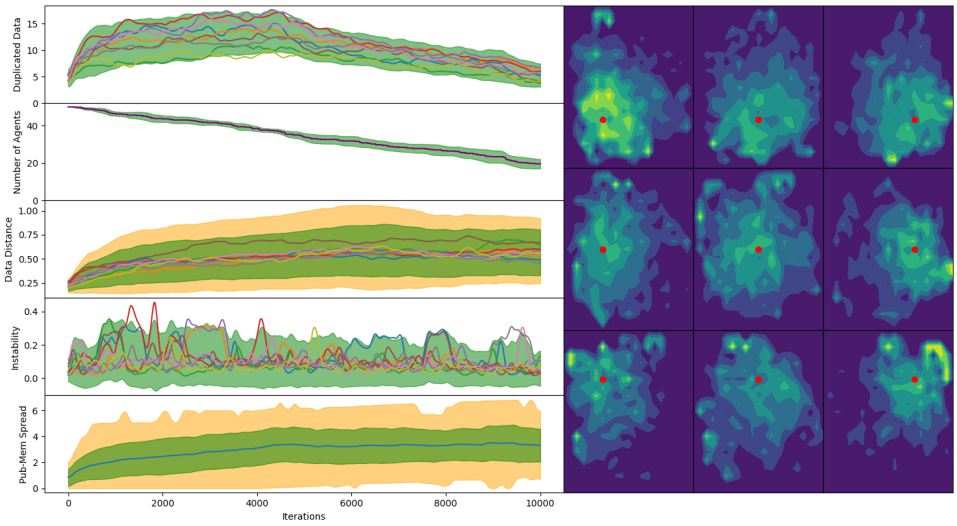

Figure B.12: Dynamic Heuristic with Migration on circular movement swarm, with noncorrelated failures

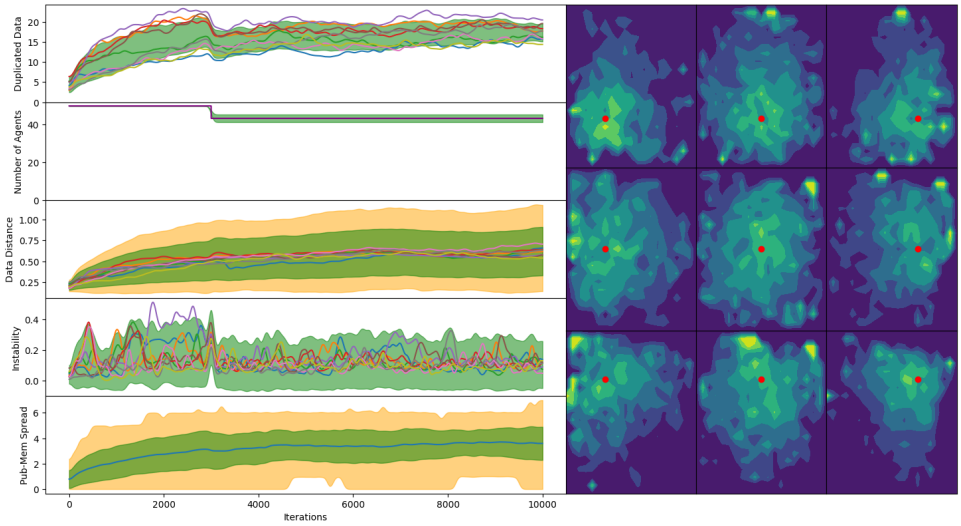

Figure B.13: Dynamic Heuristic with Migration on circular movement swarm, with correlated failures

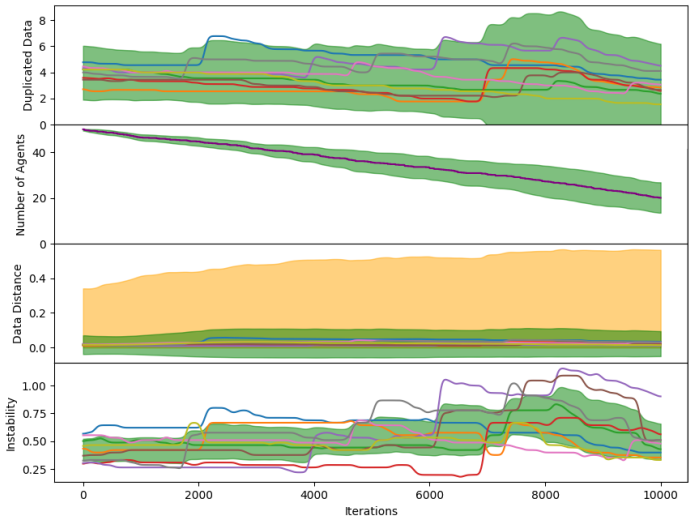

Figure B.14: Discouraging results from Neural Network Heuristic, semi-static movement and noncorrelated failures

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.