Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Swarm Memory

Harry Burge

Version 1.0, 2020-November

Supervisor: Simon O'Keefe

Acknowledgements

Contents

List of Figures

List of Tables

Executive Summary

1 Introductory Material

1.1 Introduction

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. Reliable storage of data on an ever-changing network of devices is a hard task to complete, handling loss of connection between servers, enforced reliability of access to data or loss of services, whether from non-correlated or correlated failures [9].

Most elements to cloud storage policies at a high level of abstraction could work effectively within a swarm based environment. However as explained 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 effectively a highly dynamic network.

The objective of this dissertation is to merge three areas of study into an effective/suitable storage policy for swarm-like agents in a setting with high locality and dynamic connection behaviours. Then to perform multiple analyses on the created policy using a variety of simulations to gauge its capability compared to the desired abstract behaviour.

To complete this objective, we will programmatically break down the problem described in Section 1.2 into solvable tasks. Therefore the report will be structured as follows, firstly we'll define a clear and concise problem, of which incompaces all disgruntlements described above. Secondly, bring the reader up to date with relevant literature and explain key concepts that will be required for a complete understanding of where and how the proposed solution has been derived and why said solution might be a relevant stepping stone for future research. 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 solution's design, how it is supposed to act and react in different scenarios, and the reasons for why. This leads to analysing how effective the proposed policy has kept to standards derived in the methodology section, and whether it solves the problem defined in Section 1.2.

1.2 The Problem

The problem that this report will cover is to create a storage policy for agents of a swarm, to be able to store directional abstract data as a collective, without complete duplication.

The problem can be split into two separate sub-problems. One of them is handling data duplications throughout the swarm, as to be able to control for failure of an agent, and provide a mechanic for recovery from said failures.

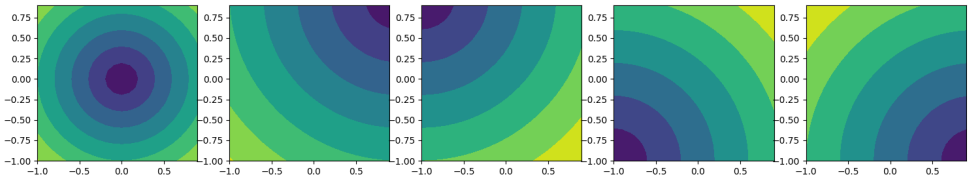

Figure 1.1: Data duplication density based off distance to datas desired location

The second is to control where data is duplicated, to give directional characteristics of the spread of said data. A visualization of this can be seen in Figure 1.1. Which is the density of duplications within an area based on distance from the desired position of that data. As the reader will see, the further we get from the desired point we should have less duplicates of that data.

A real life example of where this type of problem arises, is in mine field operation [2]. In our example, agents would need to map out where potential mines might be and record their location. Once an agent locates a mine it wants to spread the word of its 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 local mine locations.

In a practical solution, agents could fail individually for example running out of battery, or fail as a group from a mine going off. The swarm doesn't want to lose any collective data from these failures occurring.

With this increase in complexity, the solution needs to be able to withstand fluctuations of the swarm. With reliable redundancy to handle correlated (Mine detonation) and non-correlated (Power loss) failures.

1.3 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.

1.4 Motivation

In sections 1.1 and 2.2, we have painted a clear picture as to why the study of swarms is becoming an integral part for leading us to the future. With sequential computation reaching its limits [7], and the world becoming increasingly data rich, the need for distributed problem solving is heavily required. As technology gets more efficient, the ability to make swarms become exponentially viable, especially with recent advances in nano technology/biology.

Swarm robotics is limited by the technology of its time. The use of research is limited and far between in our day and age. We research this for the future where technology can support and utilize the fullest potential that swarm behaviour can offer.

The main uses nowadays are within surveillance [21, 18], delivery or 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].

It is for these reasons that I have decided to contribute my part to this extensive and breakthrough field, and will hopefully see it grow to its fullest potential.

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 and use of the cloud, varying from everyday people storing files to large businesses storing harvested data. This led to the need for much sophisticated storage solutions, which can handle the increasing file size and frequency of use. A component that increased this complexity was the Legal Services Act. 2007 [11]. This enforces that cloud storage suppliers must provide reliable and fast data collection for users. Not only that but to also provide near guaranteed longevity of stored data. In this section we will focus on cloud-based storage policies and some background terminology needed for this report.

Firstly the reader will need 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. An example of this within cloud based computing, a server node can shut down due to a software failure, therefore we lose connection and access to the node's data, typically this is completely independent of other servers in the rack. A correlated failure as the name suggests, is when a device fails with other devices with relation linking why they have failed. A typical example of this in cloud computing would be mass restarts because of a power surge from a storm. The power surge event is what links the failures together, therefore making it correlated. To be correlated doesn't mean they need to be geographically close together, however within our swarm solution will mean geographically close failures.

To control for these two types of failures there are many different solutions providing different features. Locally what is typically used is a RAID system for local failures within a node, and then a replication policy [9] for internode duplication. These in tandem provide stable node storage and redundancy for when a possible node failure acquires.

The most common forms of replication policies will take a piece of data, decide whether the data needs to be duplicated and if so will completely

copy the data onto another storage device. This provides a backup in case of failure on either one of those devices. A simplistic approach would be a random replication policy, where data is randomly chosen to be duplicated, typically duplicated within the same datacenter, so on a neighboring rack. This is an efficient design policy for handling non-correlated errors, however, lacks the robustness against correlated errors, and without a tracking of global duplications can lead to over used storage. We can mitigate for correlated errors by allowing for duplications to happen over data centers, however, this leads to downsides which will be explained below. 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.

Two key concepts of availability and popularity are not taken into account when using a random replication policy. When cloud based storage transitioned from a semi-local backup system to a worldwide daily driver. Random replication can't withstand the variability of how users interact with files nowadays. An example of why these concepts are needed in today's age, are videos. 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.

We will be looking into two different algorithm concepts, of which try to handle the maximisation problem. Both withstand non-correlated failures well because of the nature of replication, so we will only be looking into the other effects. The algorithms have been abstracted from papers on 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 it uses its global knowledge to make decisions about whether and how to duplicate items [9, 12]. This doesn't have to just be data replication, but the same principles can be seen within schema changes [13]. Due to the nature of having a privileged user, the control of the policy is a lot easier to make specific behaviours be exhibited and to be understood. This means we have higher guarantees for correlated 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 coded onto the master node.

Having an approach that uses a master, 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 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 over other agents which will first of all reduce processing capability and power loss will be more significant to agents closer to the master, possible 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. We follow a distributed control approach where each node (In the case described below its each key-value pair) has its own controller. 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 is the move to a lower cost or more redundant servers. Suicide is the removal of itself, this is usally be because of to many duplicates. Replication is when the data decides that it needs to be duplicated and sent to another node and Nothing is as the name indicates.

Because of the highly distributed nature of said approach, when coming to suicide we need to handle the edge case where the only two nodes with duplicates make the decision to suicide at the same time, therefore leaving us with no replications. 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, therefore leading to the ideal case in this example of only one duplicate.

Within the domain of server storage networks an approach like "SKUTE" is less commonly used because it adds complexity which is not needed within a global 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 redundancy because of having no single point of failure.

An approach like this is highly adaptable towards a homogeneous swarm. However with heterogeneous swarms the previous algorithm may work a lot better. Differences between the swarm types will be explained in Section 2.2.

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 [**?**].

One of the key components of multiple RAID levels is the use of parities [8]. This is where a function (typically an XOR) is done on two or more sets of data to create one or more parities. The 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 failures go over that then all 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, because of cost and possible reads after failure of heads.

The methodology is that as long as the nodes individually are redundant enough and then there is control on a higher level with our replication schemes, that is a sufficient redundancy. RAID could be adapted to run over multiple different storage nodes, however the complexity compared to performance is heavily in the favour for the above methodology for storage networks. We will come back to the possible uses of RAID internally and externally in Section 5.1.

2.2 Swarm robotics

As explained in the introduction, the study of swarms are split into two subsections, mechanics and intelligence. Both explained broadly in the Section 1.1.

Continuing on the discussion about swarm intelligence. Typically swarm intelligence focuses on solving abstract problems, like the traveling salesman problem, in a local distributed manner compared to a global manner. From the papers read by the author, it seems that predominantly the topics undertaken are testing distributed solvers compared to already researched solutions to see how they measure up. An example of this within the TSP domain is a genetic algorithm versus AS-TSP [5]. These algorithms provide benefits and drawbacks compared to their counterparts.

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. A 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. This encompasses all of which swarm intelligence doesn't cover. Swarm mechanics focuses on problems that are less abstract and are usually in the domain of physical implementation [2, 4]. Swarm robotics can be seen as the same as swarm mechanics, however, it doesn't have a broad enough scope/name to fit everything that can be researched in this author's opinion.

Two arguments to justify the naming of swarm mechanics are as follows; 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 robotics we have three different methodologies, heterogeneous, homogeneous and hybrid [1]. These can be adopted in multiple ways, however, we will focus upon the adoption of these methodologies on decision making and physical attributes of agents. Firstly we will talk about the physical adoptions.

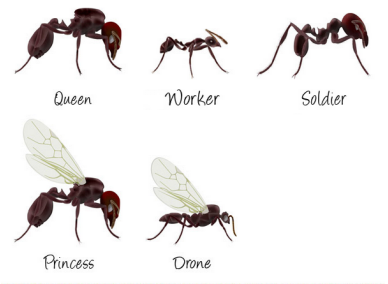

Figure 2.1: Example of a hetrogenus ant colony. https://www.pinterest.co.uk/pin/777363585651532845/

A heterogeneous swarm is defined by having differences between agents of the same swarm, as in Figure 2.1, whether physical or mental [1, 5]. These occur commonly in nature and are less studied [5], because differences in agents are a rarely needed property in research-based problems. For real-world solutions, heterogeneous swarms can be of great use, allowing other agents to pick up the slack of the swarm, or complete tasks that other swarm members cannot complete. A good example as 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.

The argument against heterogeneous swarms is that 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 to have a recharge point. Without the boat the swarm fails after some time. The decision to have 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 many agents of the same type, that's work is key to the survival of the swarm, is mitigated. This vulnerability can be controlled for in multiple ways. Common rules are found in nature's swarms and can be extracted from them. 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.

Ants typically fit into this category where ants have different types, as shown in Figure 2.1. 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]. This leads us more towards a hybrid approach which is explained later on in this section.

Because practical implementations of what humans can achieve currently in their robots, they don't have the adaptability that biology can provide, without making agents too complex.

Homogeneous swarms are defined by each agent being the same. This is found less often in nature, except for at the microbial level, and is commonly found in man-made agents. Because of 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, however with a sufficiently complex agent homogeneous swarms are the most optimal, but that is getting into speculative futuristic technologies of self replication, advanced intelligence and nano size.

Homogeneous swarms benefit from maximum redundancy, this is because if any agent goes down there are still an entire swarm's worth of agents to take its place. With this benefit of redundancy we acquire some possible losses in efficiency which could have been exploited with agents with specific hardware. The design for homogeneous agents is a complex one, either the agent is too simplistic therefore loses efficiency in their tasks. Or they are too over engineered 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. An example to show this dynamic is if we have agents that need to mine and farm, they

all have hands so can do the task at a suboptimal speed. We then as an improvement give them picks and hoes instead of arms. This gives us an increase in mining and planting speed, but why would the miners need a hoe.

Within the practical implementation of swarms, everything gets a bit messy. Usually there is not a clear cut framework or design that a swarm is designed to be like. They are designed to be as efficient as we can make them to be in any type of problem faced, this is where research and engineering have a bit of a disconnect. This is where hybrid approaches come into their own.

A hybrid approach tries to exploit the benefits of both heterogeneous and homogeneous designs without the downsides. Carrying on from our example of farmer and miner swarms, a hybrid approach to the physicality of an agent would go something like as follows. Each agent would have 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 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 person creating the hybrid swarm. Usually it is easier to go for the simpler solution and deal with the possible loss of either efficiency or adaptability.

Moving on to the control/decision making of a swarm following these approaches. Homogeneous control follows the purest form of swarm robotics, where each agent has control of its own decision making based on what other agents in its locality are doing, sometimes labbelled distributed intelligence. This therefore creates an emergent/structured behaviour of the swarm even though each agent is acting of its own fruition and can usually only see locally. A simple example of this is [23].

Heterogeneous control is also very simple in nature, where certain agents control other agents' decision making. This can be handled in multiple 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 hivemind controller will say the swarm needs to move to the left and then the agents handle that the way it decides. In a hierarchical approach it follows the same style as hive mind however deals with scalability better. Where we somehow have a power structure of certain agents being sub leaders of leaders. This control structure leaves itself vulnerable in the same way as its physical implementation, however, also has the fear of bad actors and power loss distributions over the swarm.

These still can affect homogeneous swarms however is less of a threat than on a heterogeneous controlled swarm.

Hybrid approaches to control of a swarm are just heterogeneous control policies that can adapt to changes of leaders and are usually designed for homogenous/hybrid swarms. The choice of leader(s) is usually done through a consensus algorithm [15] rather than based on any form of physical or mental difference. This allows for homogenous style bots 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.

Humans themselves are a great example of a fully hybrid based swarm both in design and communication. Though humans have variations in characteristics they can be seen as pretty homogeneous in terms of the tasks that they can perform, obviously removing edge case actions like child birth. Tools and knowledge can be spread between humans to make the swarm more efficient and an agent can specialize in a certain area. However, if some agents are lost other agents can replace them by using the same tools and knowledge from the remaining agents of that specialism. Also, the natural power-based structure of humans fits a hybrid model in terms of electorship of some kind, and not of genetics (Except with royalty, however, this is more of a label rather than a genetic difference). The leaders aren't needed for every single action so fit into a usually hierarchical power structure, compared to something of a hivemind model.

3 Design

3.1 Simulation Infomation

This section describes the simulation used for collecting results. 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.

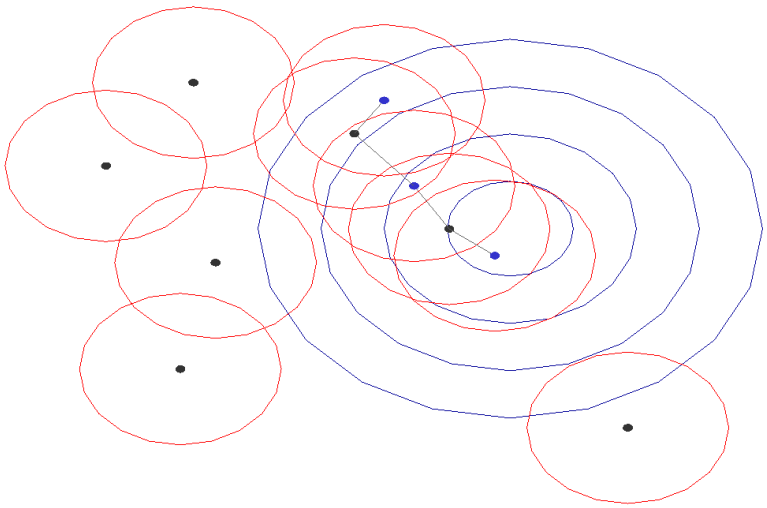

Figure 3.1: Example of simulation looks when running

Agents are homogenous, with a small connection radius around them. The world is width/height 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. The threading process happens in random order to provide as realistic as possible real world scenario, with available hardware to the author. Ideally running all agents at once in individual threads would be the most accurate, however, scheduling policies are likely to significantly suffocate some agents' processes, therefore giving less realistic results. Running on hardware that could support this many threads would be a good future improvement.

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.

We now move to lines 8-10 which are the factors described above. 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 **??**. 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 further improvements.

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 $1 dupes_ratio$.

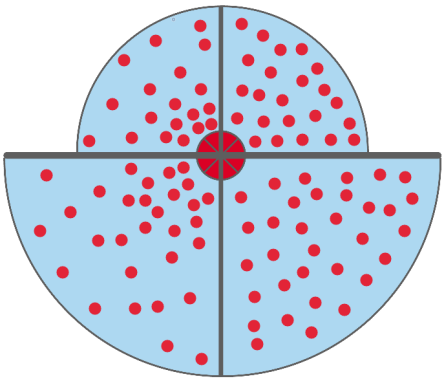

Figure 3.2: Example of changing of replication and suicide threshold on a uniform agent density

Changes to weightings and thresholds applies significant behavioural changes to the swarm. The increase of replication threshold means that 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 3.2. 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.

4 Analysis

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 1.2. Changing of replication and suicide threasholds allows insight into internal stability and behavourial control of the soloution.

All tests will be of 5 simulations, for 10,000 iterations. Data will also have a gaussian filter applied as to make the graphs readable, kernel of size 50.

"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.

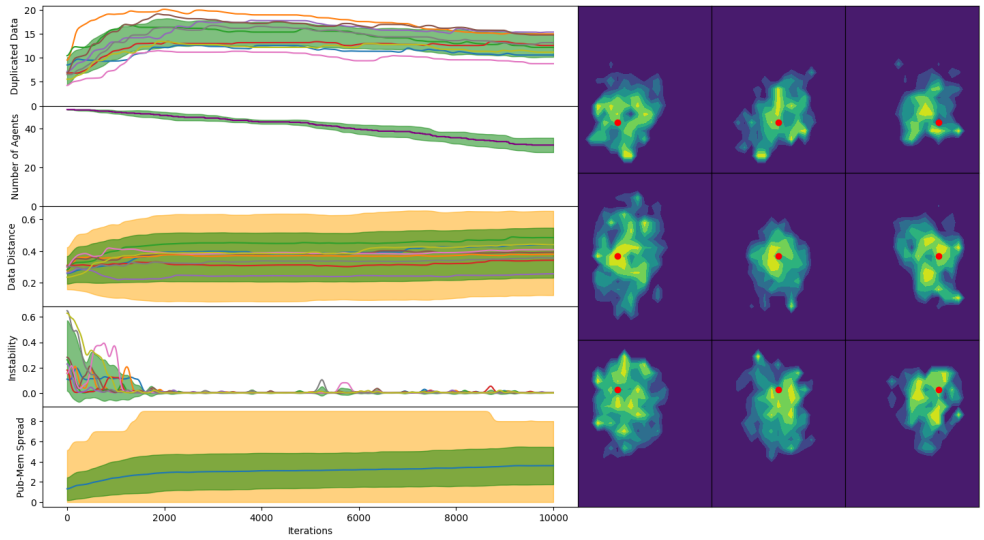

Figure 4.1: Static Heuristic on semistatic movement swarm, with non_correlated failures

"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.

4.1 Static Heursitic

Firstly, we check whether the solution in Section 3.2 works as intended with different threshold values, as in Figure 3.2. 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 /refsec:Simple2a 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 tern 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

Within this section we review the results found of the algorithm described in Section 3.4. We did not compute threshold changes graph, this is because nothing would have changed since the last computation unless there was more than one public data to be learned. This is because the algorithm in our case needs at least three data points to be even activated.

The aim of this algorithm was to decrease the spread of duplications per agent. We can see in Figure **??**, that this is the case, however at the cost of instability in the system. We can also see that it also affects the spread in Firgure **??**, however by less of a degree but with roughlt the same increase to instability.

In the duplication ratios sections on all dyanmic heurisitic with migration we can see the spread has been increased in size, espically in the semi-staic movement data. This could be an affective way of creating an artifically larger area with also keeping the duplications down.

When comparing this algorithm to the prevouis algorithm not much has changed, the affect of memory spread to the instability is a cost which will need to be investiagted for your persoanl needs of the project.

// Need to redo test data and ensure that we aren't losing more duplicates because of migrating to a agent already with said migrated data

5 Conclusion

5.1 Futher Improvements

There are some major improvements that could eb amde to this algorithm to make it much more effective, in performance and also computationally. First lets focus on computationally. Currently at the start of each agent cycle it will message each other agent for the infomation its needs to gain perspective of its local scope. This is inefficent because for every agent in the vicinity each agent will have to send a packet to it every time it cycles. If we flipped the packet sending the other way around we would take a hit on the memory required to run the agent however would be computationally faster. Each agent at the start of its cycle or end will tell other agents its infomation which will be updated in tables on each indurvdual agent. We can then use the internal tables to calculate what needs to be done. This would offer overall less transmissions over the network, and would speed up the frequnecy of which we can run the agents control loop.

For performance of the algorithm there are many possible changes that could be improved, ranging from better parameter adjustments either from machine learning techniques or hardcoded adjustments. This would give diffrent desirable affects based off what is currently happening in the network, an example of this is for stability. Currently we have it restrict replication however there might be more effective ways to control stability changing multiple parameters at varying scales.

With our ability to solve for corrilated failures, we lack range of which a corrilated failure can happen. To be able to handle corrilated failures that could be much larger, larger than the average distance to the point. Soloutions to this problem could be after a certain range we make sure that duplications have no other duplications in their vissinity, and lock the data from suiciding. This wold mean that data further away would survive with minimising density. To get the data out their currently relys on movement as can be seen in the diffrence in dulication density when comapring static movement to circular movement. To solve for this we could have a directionaliy component to migration, or after a certain distance we push data away from the data point.

// Possibly talk about the effectivness of how much data the network can store compared to the amount of data storage.

5.2 Conclusion

The algorithms proposed, can be seen to atleast solve the problem provided in Section 1.2. How effective they are compared to other theoretical solouions is a diffrent topic. With the first algorithm proposed in Section 3.2, we saw that it worked effectively for a simulation based of this problem. We had good results when coiming to spread, less so in a semi-static swarm, and also duplications were handled correctly. However when abstracting this algorithm to a real world soloution, floors became apprant in the algorithm which relyed to much on perfect communication, well picked threshold values and smaller sized corrilated failures.

Due to the thought expriment of abstracting this idea to real world soloution, some things within the heurstic needed to be changed. This was the instability of the system in certain scenrios, for example a more contrived situation was when the suicide threshold is to low. This would mean that agents would suicide to often and then other agents would replicate again, theerefore leading to useless transistions of data which was not needed.

To cmbat this we create some dynamic threshold changes so that we can slow down the instability using diffrent parameters. We directly put a time based slow on suicides and for replication we slowed it by the amount of local instability. This gave us the best of both worlds with fast growth of duplication and a slower release of the data. Overall this significantly improved the algortihms perfomance, espically when coming to a circular movement swarm. We can see that even though we slowed down the stability issues we didn't end up actually solving the issues, this could be seen with the fact that stability graphs had roughly the same shape but at lower magnitudes. This was touched upon in Section 5.1, on ways that this could be improved.

The third algorithm proposed in Section 3.4 is the most controversial of the bunch. This is because it sets out to do what it was propsed to do however we gaina natural boost to the instability due to movement of the data duplications from, agent to agent. We could see that this algorithm was affective for larger distance corrilated failures, however amount of duplications seemed to be affected more than prevouis algorithms, and in a circular movement swarm the affects on the memory spread weren't even that pronounced as in a semi static movement swarm.

Overall Section 3.3's and Section 3.4's algorithm are the best and should be the only to be used, unless there are extreme memory and computation constraints on your agents. Section 3.4's algorithm should only be used in cases were large corrilated failres could happen and instability is less of a deal. However this would still not be as affective as a soloution propsed in Section 5.1.

The algorithms propsed assume that data is highly important and that all agents in that area need to know that data as much as possible. This therefore makes them not as relevent if you were trying to adopt them to a cloud based soloution. When coming to a swarm soloution this is definitly no the most affect soloution that could be possibly created. If we ruled out hybrid based models, which would be much more affect as a storage soloution, due to knwoledge of where all duplications are and we could specifically give data priorityes on how many duplications to spread out and how many redundancys you want further away in the swarm. We talk about possible more effective ways to store the data in Section 5.1.

This project achieved what it set out to do, and has highlighted downsides and possible improvements that could be researched into further to improve this algorithms performance and adaptability.

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

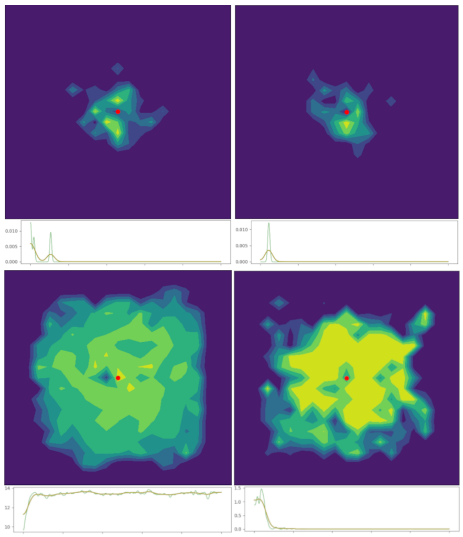

Figure B.1: Static Heuristic on semistatic movement swarm, with threshold changes

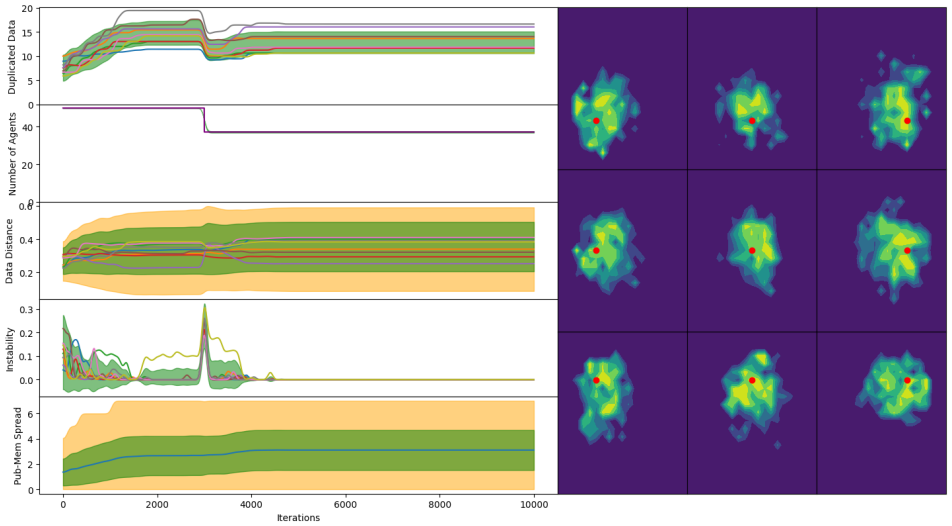

Figure B.2: Static Heuristic on semistatic movement swarm, with correlated failures

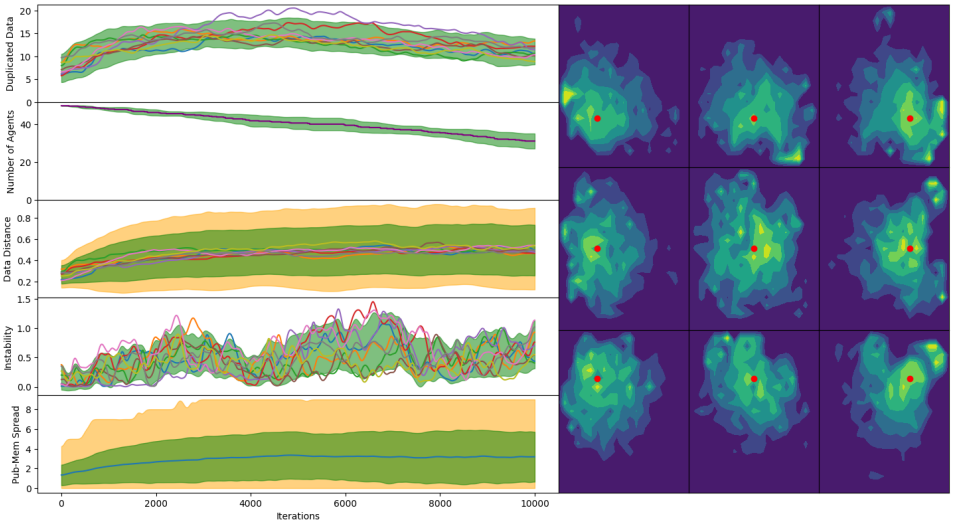

Figure B.3: Static Heuristic on circular movement swarm, with non_correlated failures

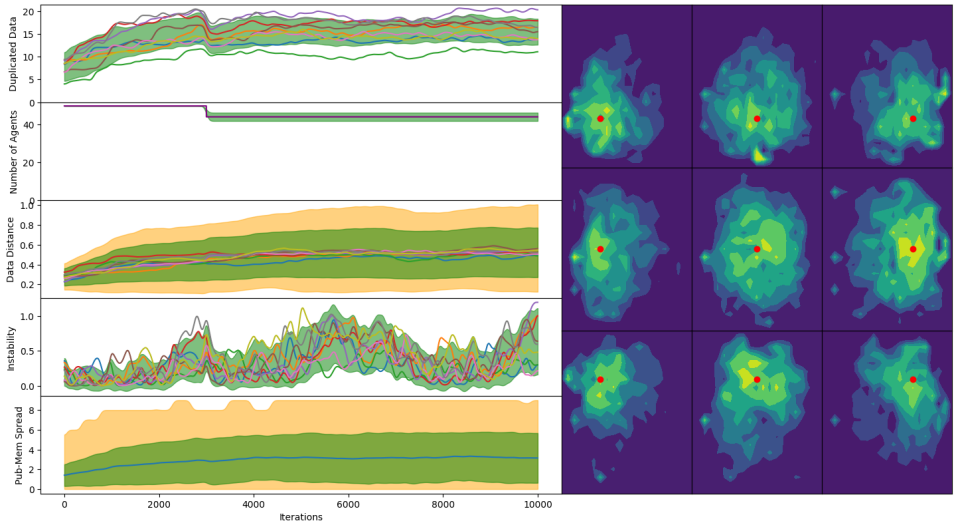

Figure B.4: Static Heuristic on circular movement swarm, with correlated failures

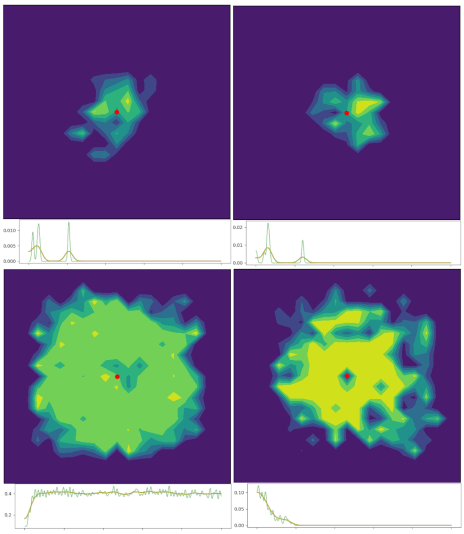

Figure B.5: Dynamic Heuristic on semistatic movement swarm, with threshold changes

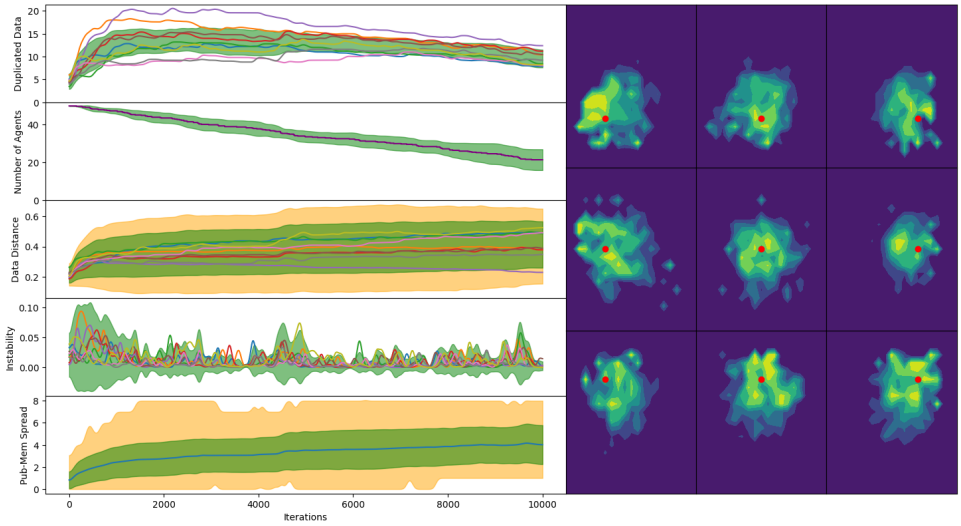

Figure B.6: Dynamic Heuristic on semistatic movement swarm, with non_correlated failures

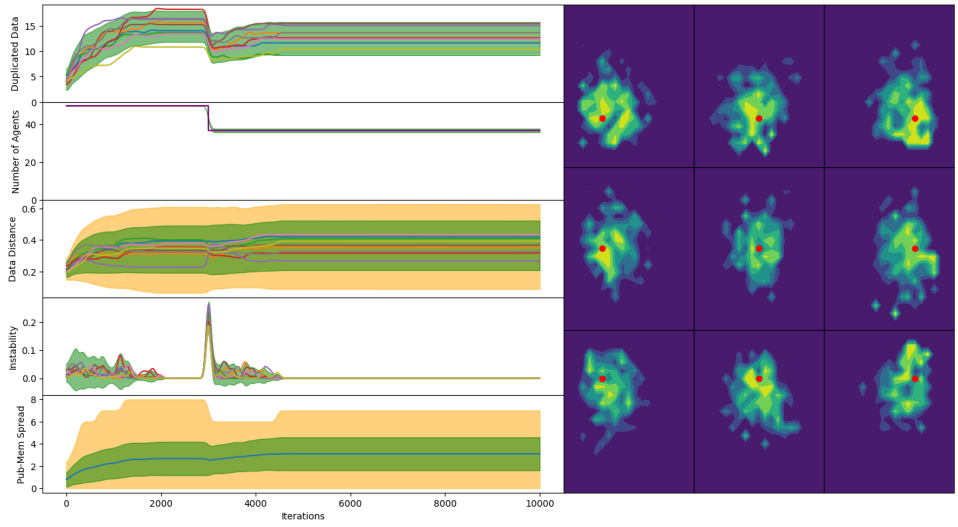

Figure B.7: Dynamic Heuristic on semistatic movement swarm, with correlated failures

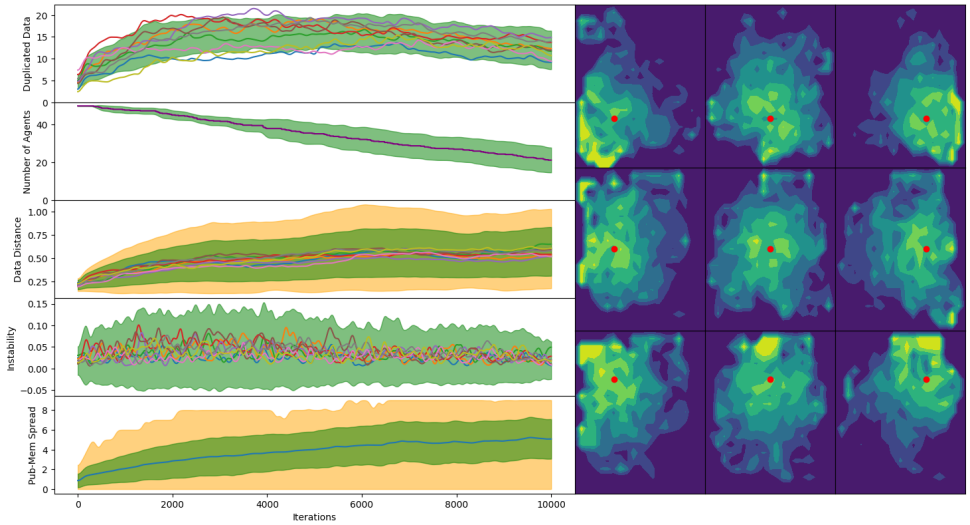

Figure B.8: Dynamic Heuristic on circular movement swarm, with non_correlated failures

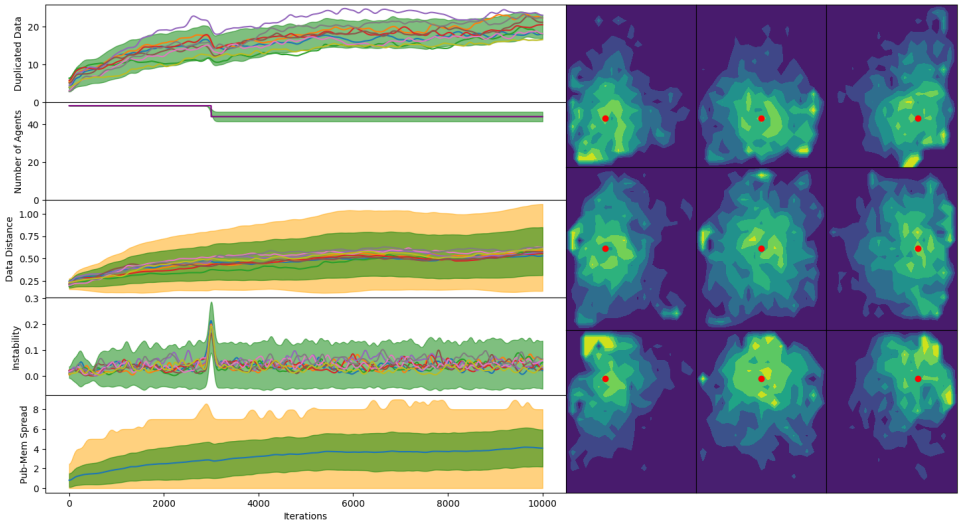

Figure B.9: Dynamic Heuristic on circular movement swarm, with correlated failures

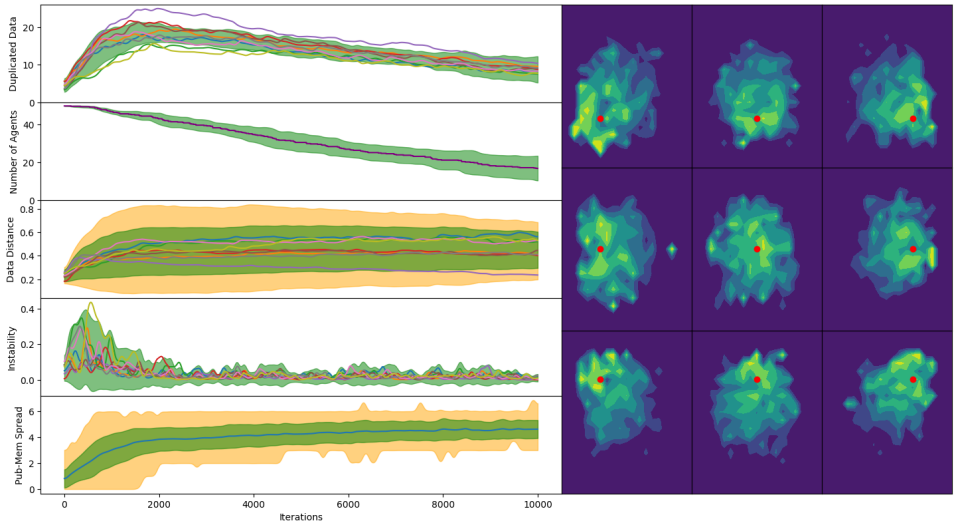

Figure B.10: Dynamic Heuristic with Migration on semistatic movement swarm, with non_correlated failures

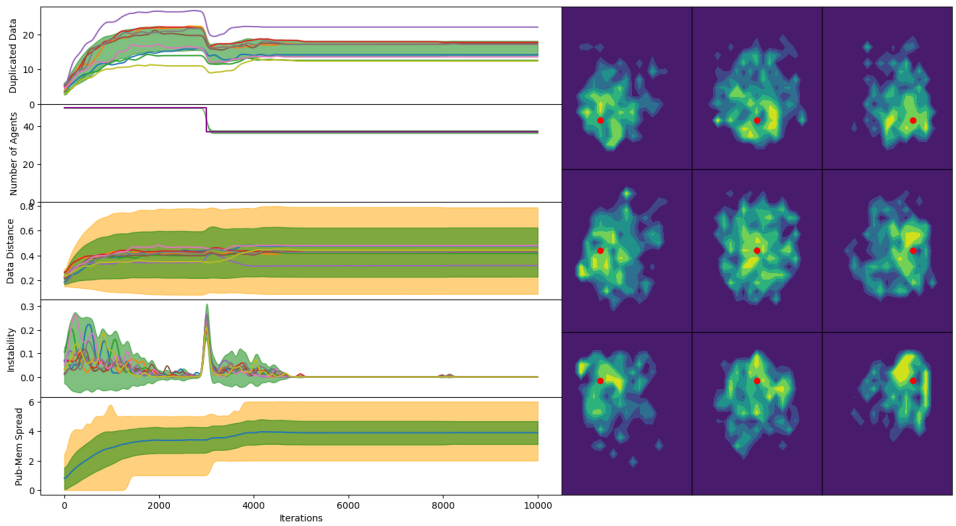

Figure B.11: Dynamic Heuristic with Migration on semistatic movement swarm, with correlated failures

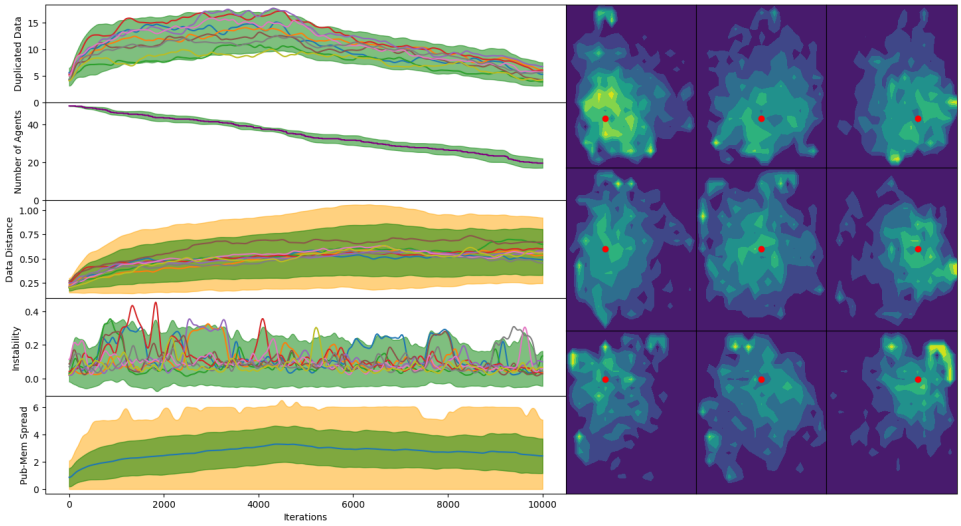

Figure B.12: Dynamic Heuristic with Migration on circular movement swarm, with non_correlated failures

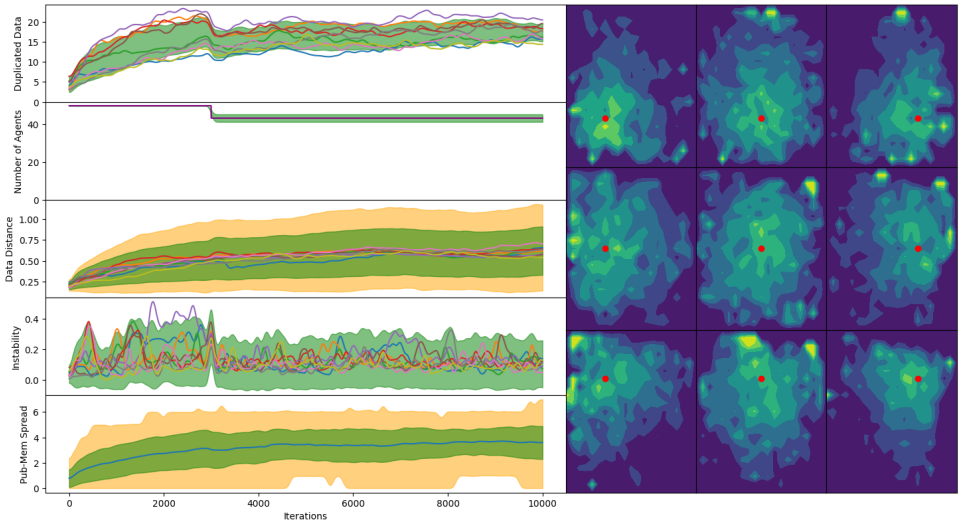

Figure B.13: Dynamic Heuristic with Migration on circular movement swarm, with correlated failures

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.