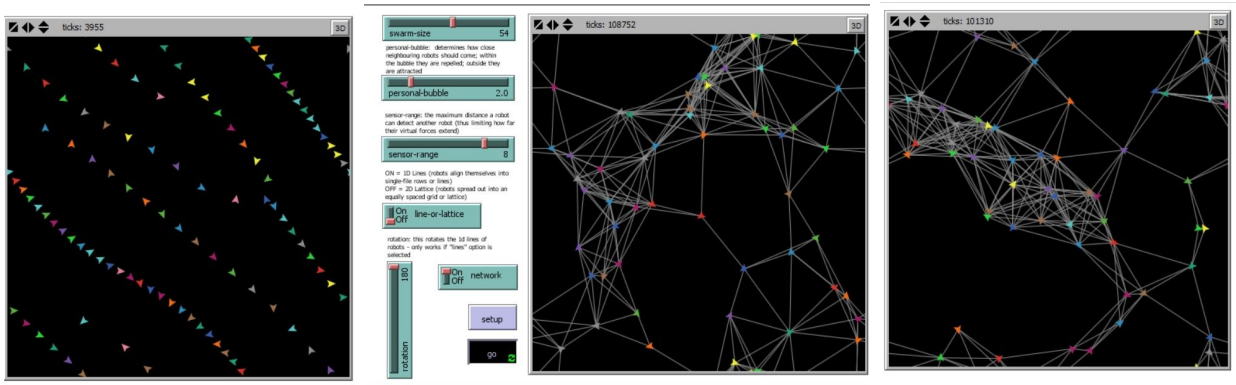
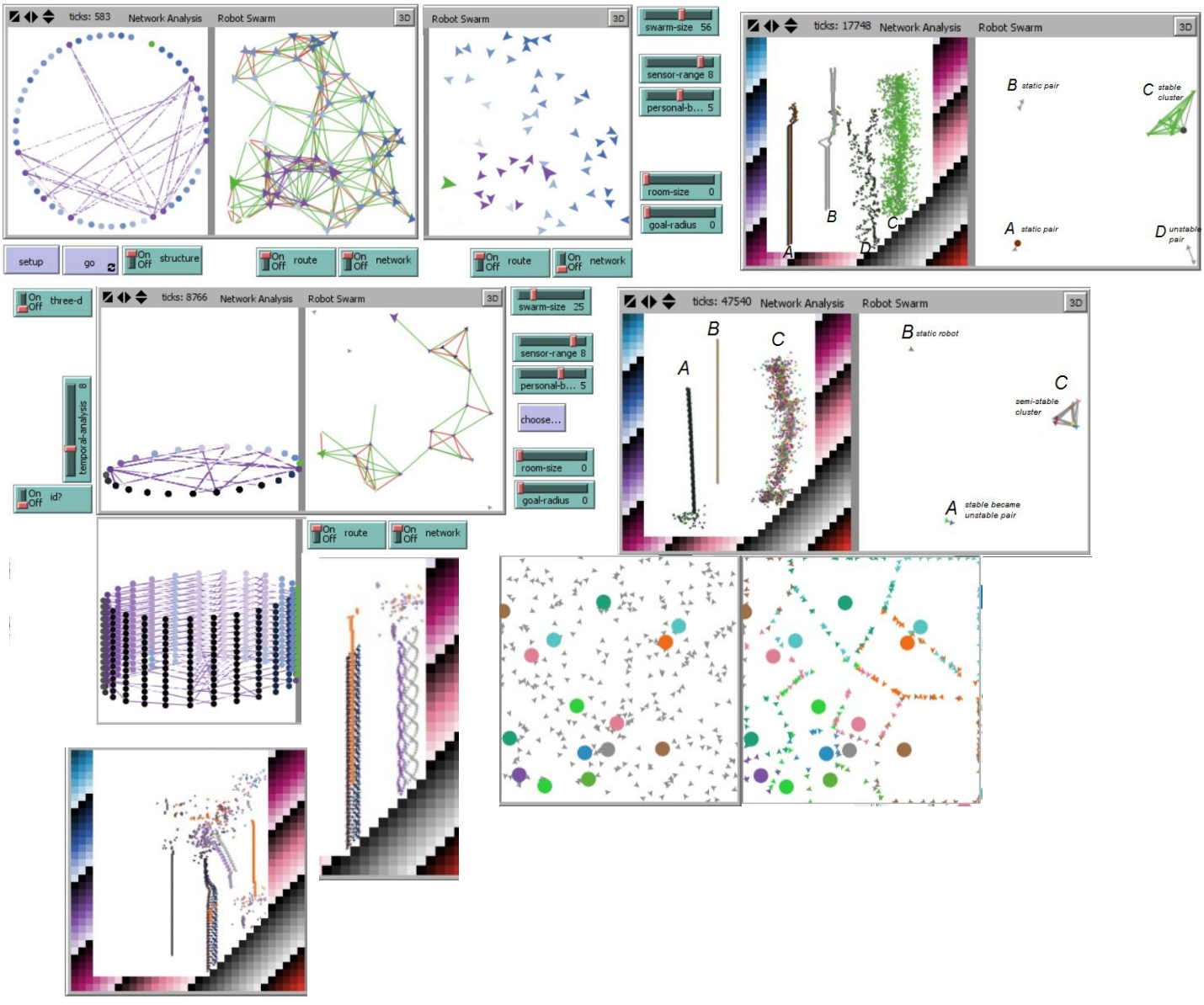


Spatio-Temporal Patterns act as Computational Mechanisms governing Emergent Behaviour in Robotic Swarms



Spatio-Temporal Patterns act as Computational Mechanisms governing Emergent Behaviour in Robotic Swarms

Mohammed Dery Alim
b. Terry b. Jack

MSc Artificial Intelligence and Robotics
De Montfort University,
Leicester, UK
2017

Abstract

Our goal is to control a robotic swarm (a decentralised group of mobile robots that are inexpensive and unintelligent as individuals - making the swarm fairly robust to faults and failures - yet through their interactions, a collective intelligence unwittingly emerges across the swarm which can solve monumentally complex tasks with great speed, flexibility and adaptability) without removing its 'swarm-like' nature. In other words, we aim to intrinsically control a robotic swarm's emergent behaviour (i.e. the self-organisation of a decentralised system from a disordered to an ordered state - e.g. scattered ants spontaneously forming a bridge).

Past attempts at governing robotic swarms or their self-coordinating emergent behaviour, has proven ineffective, largely due to the swarm's inherent randomness (making it difficult to predict) and utter simplicity (they lack a leader, any kind of centralised control, long-range communication, global knowledge, complex internal models and only operate on a couple of basic, reactive rules). The main problem is that 'emergent phenomena' itself is not fully understood, despite being at the forefront of current research. Research into 1D and 2D Cellular Automata has uncovered a hidden computational layer which bridges the 'micro-macro' gap (i.e. how individual behaviours at the micro-level influence the global behaviours on the macro-level). We hypothesise that there also lies embedded computational mechanisms at the heart of a robotic swarm's emergent behaviour. To test this theory, we proceeded to simulate robotic swarms (represented as both particles and dynamic networks) and then designed local rules to induce various different types of intelligent, emergent behaviours (as well as designing genetic algorithms to evolve robotic swarms with emergent behaviours). Finally we analysed these robotic swarms and successfully confirmed our hypothesis; analysing their developments and interactions over time revealed various forms of embedded spatio-temporal patterns which store, propagate and parallel process information across the swarm according to some internal, collision-based logic (solving the mystery of how simple robots are able to self-coordinate and allow global behaviours to emerge across the swarm).

In future, we hope to conduct a more in-depth study of the discovered spatio-temporal patterns (cataloguing each of their characteristics) and their computational dynamics (mapping their interactions and pattern changes) in order to accurately model the swarm's computational mechanics. Thereafter we hope to conduct an investigation into possible methods of manipulating these individual spatio-temporal patterns (i.e. using external stimuli, modifying the environment, manipulating key robots, their parameters, the initial configurations, noise, communication delay, etc). Using these manipulation methods with our predictive model, we could potentially control the robotic swarm's emergent behaviour via reprogramming its internal spatio-temporal computations.

Aknowledgements

First and Foremost, I would like to thank God; Nurturing-Lord of the (seen and unseen) Worlds. There's no mental or physical strength except with Almighty God; Most High. May Your veneration be upon our Grand-Master Muhammad and his family and his companions and peace.

Thereafter I'd like to thank my family; Elsa and Terry Weizemann (my beloved parents), Roohi and Fahtima Zahra (my darling wife and daughter), Jérui and Idris (my dearest brothers), Dr. Sahar, Imran and Faizan Khan (my loving in-laws), and all aunties, uncles, cousins, nephews and nieces for their continual love, support and encouragement.

To my respected Professors in Artificial Intelligence and Robotics at De Montfort University - Archie Singh and Kay Owa - a very special thankyou for your guidance, advice and enthusiasm throughout this project.

I'm also very grateful to my close friends, Ayman (Ian) Vicente and Sheikh Amienoellah Abderoef, for their invaluable companionship and stimulating discussions during my time in Abu Dhabi.

Last but not least, I'd like to thank Santa Fe Institute's 'Complexity Explorer' for their terrific online courses and groundbreaking research into Complexity Science, which sparked my interest in the fascinating field and eventually lead to the birth of this project.

Table of Contents

(I) Introduction

(1) Aims

(I.1.1) Main Aim & Challenges

The overall goal of this project is to intrinsically control the emergent behaviour of a robotic swarm.

(I.1.1.A) Challenge 1: Conventional methods fail to control Decentralised systems like swarms

At first, controlling a swarm's behaviour sounds straightforward enough, however, upon closer inspection it becomes apparent that this is a complex and nonlinear problem. Robotic swarms are entirely distributed systems [16, 46, 51, 37, 26] with a distributed group intelligence [45]. This means that, rather than being focused within any one robot, the intelligence and control of a robotic swarm is equally distributed across all individuals [54] (a large reason why it is so hard to control a swarm).

Furthermore, robots only have access to localised information gained through directly interacting with their neighbours (i.e. via contact or short-range communication) and occasionally via indirect interactions (i.e. stigmergic signals left in the environment - like pheromone trails left by ants). Therefore, a swarm's individuals may even interact in a haphazard, disorderly manner, giving rise to the turbulent or chaotic characteristics of a swarm. Viewed at an individualistic level, robots can be seen busying themselves with their own, individual jobs; without a future vision and unaware of any larger picture [45]; they remain oblivious to the positive contribution their work and interactions are having on the global behaviour of the swarm.

Decentralised systems (like swarms [46, 59]) coordinate in a manner completely alien to the norms of centralised control [39, 13, 45, 46, 48, 8]. This means that robotic swarms are void of common control structures (like hierarchies, chains of command [16, 46] or leaders - contrary to the misconception that the behaviour of a swarm is somehow governed by the telepathic powers of a queen [7] - etc). Unfortunately, centralised control strategies are highly dependent upon these infrastructures [8] and will fail when applied to decentralised systems which lack them. Ordinary control methods hold little influence over the governance of a robotic swarm's behaviour [16, 45, 46, 29, 59, 51, 37, 26].

(I.1.1.B) Challenge 2: Extrinsic vs Intrinsic control

There have been attempts at controlling robotic swarms by enhancing its individuals [45]. Under normal circumstances, a swarm's individuals are unintelligent and operate under very rudimentary conditions (i.e. have minimal processing power and are ignorant of global conditions pertaining to the swarm - such as their absolute position within the swarm [48]). If the robots were modified to have a more sophisticated processing power and enough memory to store an internal map or model of the environment, they

could be programmed with the intelligence to estimate their own pose and location within the swarm (i.e. via Simultaneous Localisation And Mapping - SLAM - techniques), and the swarm could essentially be controlled at the level of its individuals. This approach has been tested and produced accurate robotic behaviours (despite suffering minor difficulties when faced with very symmetrical environments such as corridors [48]), however, it is computationally expensive and ultimately requires the robots to be enhanced with more powerful, onboard processing [48].

To avoid increasing the robot's computational power, which in-turn increases their complexity, expense and energy consumption (removing some key advantages of a robotic swarm), individuals have been modified to use 'off-board' processing [48] (maps or models of the environment are stored externally and are accessed wirelessly by the robots during run-time). Therefore robots need only be equipped with enhanced, long-range communication (something regular robotic swarms lack), which enables robots to access centralised, global information (i.e. GPS data [48], knowledge of the swarm's overall goals or plans [34], the current state or behaviour of the entire swarm [48, 51, 8], etc) and thus forces a form of centralised control structure onto the robotic swarm. Unfortunately, this modification reduces the swarm's adaptability (removing yet another advantage of a robotic swarm) since the limitations imposed by wireless transmission times and feedback delays [48] severely slow down the robots' reaction times, making the swarm less adaptive in dynamic terrains with rapidly changing features [41].

Although a modified robotic swarm is able to perform feats such as splitting itself up into heterogenous teams of robots which can then simultaneously solve subtasks which contribute to a larger, externally set goal, the resulting swarm behaviour is very rigid (a piece-wise, step-by-step behaviour), compared to the turbulent, organic, subtly self-organising behaviour which emerges in unmodified robotic swarms (consisting of purely homogenous, unintelligent individuals [45]). By attempting to tame a wild swarm (i.e. by imposing restraints upon it to control its behaviour), it seems to cease behaving like a swarm. This suggests that enhancing a swarm's individuals in order to impose a centralised control structure on the swarm is not the secret to controlling its emergent behaviour [51]. Controlling a swarm demands an alternative approach; one which avoids imposing any external influences and artificial structures onto the swarm [22, 45]; the control must come naturally, from within!

(I.1.1.C) Challenge 3: Emergent Phenomena

Currently, designing emergent behaviours in robotic swarms is like working in the dark and relatively limited progress has be made. Swarm robotic designers have resorted to two main approaches which try to bypass the micro-macro gap problem (i.e. how localised, individual behaviour on the microscopic level lead to globally coordinated, intelligent behaviour emerging at the macroscopic level): a bottom-up behaviour-based approach (which is timely and unsystematic) and a top-down evolutionary-based approach (which is a black-box).

If we are to ever, truly control a robotic swarm's emergent behaviour, an investigation into its 'Emergent Phenomena' is inevitable. Researchers investigating Emergent Phenomena in Cellular Automata have discovered a

hidden, computational layer which seems to bridge the the mysterious micro-macro gap. There is a lot left to explore in order to understand how computation emerges in many natural systems [24], however, we hypothesise that emergent computation is the secret to understanding the emergent behaviour in robotic swarms (and likely all types of complex adaptive systems).

Therefore, controlling the behaviour of a robotic swarm (while retaining the high levels of independence demanded by the individuals within the swarm [51]) requires more insight into a robotic swarm's rich, spatio-temporal information [44]. Swarms can be viewed as a form of parallel processing system, complete with inputs, outputs and an asynchronous spatial logic [46, 5] which stores, propagates and modifies information across the swarm over time. The secret to unveiling (and eventually manipulating) the swarm's mysterious self-managing behaviour lies in better understanding this intrinsic logic and inherent parallelism [5, 8]. What are the mysterious mechanisms which link the micro-level components (which execute rules based on purely local information) to the subsequent structures and interactions that appear at the macro-level [22]? If the information processing mechanisms which drive emergence and self-organisation in robotic swarms can be uncovered, the understanding and insight gained would allow us to control artificial swarms [17] and perhaps even engineering new forms of parallel computing systems [13].

(I.1.2) Sub-Aims

There are several proposed tasks; each progressively more challenging than the last. Tasks are also accumulative; building upon prior tasks (hence the accomplishment of most tasks rest upon the completion of tasks preceding it);

1. Design a rule-set that produces intelligent, emergent behaviour in a (simulated) robotic swarm
2. Design a genetic algorithm that evolves a robotic swarm with emergent behaviour
3. Discover the (hypothesised) computational mechanisms which underlie a swarm's emergent behaviour
4. Understand the computational mechanics discovered by:
    a. identifying all spatio-temporal patterns
    b. modelling the characteristic behaviours of each spatio-temporal pattern
    c. mapping the interactions between each spatio-temporal pattern
5. Predict the swarm's emergent behaviour by simulating its underlying computational mechanics
6. Investigate methods to manipulate the underlying computational mechanisms, including; (i) Injecting, (ii) Removing, (iii) Reflecting, (iv) Attracting and (v) Repelling spatio-temporal patterns
7. Intrinsically control the swarm's emergent behaviour by reprogramming its underlying computational mechanics

(2) Motivation

(I.2.1) Advantages to Controlling a Swarm

Not only is it beautiful to witness large numbers of individuals - preoccupied with their own, self-serving objectives [51] - effortlessly cohere and self-organise [16, 45, 29, 59] into various emergent-behaviours [45, 59], without realising that their actions

and interactions are inadvertently contributing toward a greater, global behaviour [51,8], but robotic swarms also carry a number of advantages [13]. Complex problems are solved relatively easily with swarm-like strategies based on 'self-organisation' and emergent behaviour [51], while conventional methods struggle [46]. Distributed control is also cost-effective [47] because any burden (i.e. power consumption, sensing and processing requirements [48], physical strength, etc) is shared equally across every individual. This is an especially attractive concept for aerospace systems since they "must be limited in size, weight, and power consumption" [51]. Yet despite individuals being small, limited and exhibiting very simplistic behaviours (like insects), a large number of them can culminate their abilities to form highly complex, intelligent group behaviours that are able to accomplish a wide range of significant tasks (e.g. robotic swarms inspired by ants can cross ditches by connecting together to form a bridge [8]).

Most artificial systems are too rigid to operate under high levels of uncertainty and incomplete knowledge [45], commonplace conditions in natural environments. However, a robotic swarm's ability to spontaneously reorganise itself makes it extremely 'flexible' [16, 50, 29, 8] and 'adaptable' [38, 47, 50, 45, 51] to cope with a broad spectrum of situations, problems and tasks [28, 8]. Robotic swarms can even respond to unforeseen events [16, 45], even if the environment itself is dynamic (i.e. has continually changing features or conditions [50, 45]). Even though the individual behaviour of the robots are too basic to adapt or change, the resultant global behaviours which emerge across the swarm can flexibly adapt (e.g. a robotic swarm may spontaneously split into new group formations [45]). Adaptive systems also possess the potential to 'learn' [45, 29] (i.e. robot swarms may 'learn' to favour a particular response when faced with a specific set of environmental changes [46]).

A swarm's computational power is dynamic and can be increased (to solve more difficult problems) via the rapid deployment [47] of additional individuals injected into the swarm [5] (similar to increasing the number of neurons in a neural network to find better solutions to more complex problems). Likewise, for efficiency, only a fraction of the swarm need be used to solve simpler problems [5]. Since swarms are scalable to different group sizes [47, 50, 41, 59, 8], its global behaviour is almost completely unaffected by the number of individuals within the swarm [47]. Hence, even if the swarm size changes mid-task (via the introduction or removal of individuals) its overall behaviour does not drastically change [8] (other than a slight improvement toward larger set sizes [47]). Networks of distributed individuals [47] like robotic swarms have the advantage of being 'robust [38, 45, 28, 59, 8] to individual losses [8], failures [50, 41] and physical damage [16, 29, 5] and thus, even if individuals are added, removed or destroyed, the swarm is minimally affected [47]. Because communication in decentralised systems can occur between any neighbour [45] (rather than needing to communicating with a central entity and await new commands from them), the remaining individuals quickly re-adjust to compensate for any loss avoiding any significant effect on the final result [34] (which is why it is so difficult to exterminate social pests [8]). Robustness makes swarms very well suited to military applications. If space missions were conducted using swarms of miniature spacecraft [16] (as opposed to single spacecrafts), some members of the swarm could potentially sacrifice themselves for the 'greater good' [16]. This natural 'fault tolerance' [47, 5] is largely promoted by "redundancy" [8], "the absence of a leader" [8] (in most systems, "if the central coordinator is injured or lost, the entire system collapses" [13]) and the strange fact that swarms (and other distributed systems) are only weakly sensitive to any one, individual influence [23] (a reminder that controlling an individual is not the key to controlling the swarm).

Finally, swarms can handle multiple inputs [37, 26] and easily digest large amounts of information [16] by parallel processing it in an unsynchronised manner across its large collection of individuals. By dispersing the information across the entire system [16], as opposed to 'bottlenecking' the data (as is done during linear (centralised) information processing [38, 13]), higher information exchange rates [31] can be achieved at great speeds and efficiency.

(I.2.2) Applications for Controlled Swarms

It is a wonder how swarms ever synchronise their decentralised information to come to a common decision [8] or make group decisions and 'think' as a whole. Yet they can and do. Examples of such collective decision-making behaviours include: task allocation, consensus, collective counting [10], collective memory [10], etc.

Swarms can perform incredible feats, otherwise overburdensome for a single individual, due to their pooled resources. Collective transportation (wherein individuals cooperate by connecting together to increase their overall pulling power) is useful for carrying large, heavy objects [39, 8], such as gallons of water or chemicals to pollinate a field of crops [16].

A simple lattice formation [47] can be very useful to quickly establish a distributed computer grid (also known as a 'distributed sensing grid' [47]). Applications include:

i. a dynamic, emergency communication network [51] ideal for situations where traditional, stable network infrastructures (i.e. satellite communication) have broken down [16, 46] or become inaccessible due to extremely long-distances or barriers [51]. Emergency communication channels is a primary security requirement in disaster relief scenarios. "multiple tiny robots… quickly disperse into the open spaces. Upon detection of a survivor, a robot emits a …message signaling the discovery. This message is propagated locally between robots only …(and) makes its way back to the entrance where rescue team members can now follow (it)… to the survivor." [41]).
ii. Surveillance [16, 47, 41, 8] - an obvious application due to its wide use in reconnaissance [41], traffic monitoring [16], image processing [3, 10] and weather and climate mapping [16].
iii. There are various other safety, security and environmental applications [16, 46] including
    - intrusion tracking [41, 8],
    - hazardous environment exploration [16] (e.g. NASA is investigating swarm-based spacecraft to explore deep space without risking human lives [51]) or
    - hazard detection [41] and removal (e.g. removing old landmines [8]. "swarms of tiny (robots)… explore the ocean floor and clean up the marine bays [51]". Equally by detecting and cleaning up harmful chemicals, pollutants and oil spillages [16, 46, 8]) as well as its future maintenance.
iv. A distributed display (each robot serving as a pixel in the display) embedded in the environment and actively annotating it (e.g. terrain features may be highlighted or added, such as synchronised blinking lights to form a route toward a particular site or person, etc) [41].

Swarms are probably most known for their mystically self-organising computational geometries [10]. Randomly distributed individuals will automatically and spontaneously organise themselves and their surrounding objects [8] into orderly formations. Spatially-organising behaviours [47] that manipulate the environment

can be useful for assembling physical constructions [21, 28, 8] such as those built by bees, termites and other social insects [22] or alternatively for the assembly of complex, large-scale, virtual systems [45]. Spatially-organising behaviours that focus on manipulating the swarm itself can be used for morphogenesis [8] (self-assembly [5]) which includes self-management, self-optimisation, self-protection, self-healing or self-repair [47, 5, 51], self-configuration [51] and self-construction [45].

The most common spatially-organising behaviours include (a) aggregation [39, 22, 28] (often seen when social animals flock [8] or swarm [28] together), (b) dispersion or splitting up (a form of predator avoidance [28, 29] often utilised in schools of fish or motorcycle gangs being chased by the police), (c) segregation [21, 22] which can be used to sort, group and cluster together different classes of objects [22] into patches, bands (as with annular sorting) or any other geometrically-organised formations. For this reason, this is sometimes referred to as shape or pattern formation which can be extended to chain formations and bulk alignments [28], all of which have numerous applications for civil defence (i.e. automatic perimeter defences [47]) and offensive military operations (i.e. battlefront formations [21], convergent attacks on targets from multiple sides [51], etc).

A swarm's ability to self-organise can produce many beautifully unified swarm behaviours such as coordinated motion [39] (to increase stability when travelling through rough terrain [8]), and foraging [39, 29], collective exploration [16], path-planning and other navigation behaviours [3, 10, 8] which have inspired efficient search methods and optimisation algorithms [3, 10, 51] (e.g. ant colony optimisation, bird swarm algorithm, etc) due to their "rapid terrain coverage" across "expansive environments" (or virtual search-spaces) via their inherent "parallelisation and redundancy" [48]. Thus swarms would serve well in search and rescue operations required in the aftermath of a natural disaster [16, 8].

Eventually, as the field of nanotechnology advances, swarms of nanobots could be used in healthcare [16, 46] (like an artificial immune system - just as virtual swarms are already being utilised to defend and optimise computer networks, by automatically rerouting and repairing nodes, discovering and attacking malicious viruses, etc [45]) . As well as targeting natural ailments and foreign viruses, they could improve or enhance our own bodily functions. They could even potentially "discover and kill cancer tumours" [51].

(I.2.3) Dangers of Uncontrolled Swarms

Despite the ability swarms have to scale, if the number of individuals in the swarm become too little, the intelligent group behaviour of the swarm disappears and only the simple, unintelligent behaviours of the individuals remain (this is why it is impossible to understand a swarm's emergent behaviour by studying the individuals in isolation). Large numbers are required for intelligent swarm behaviours, such as self-organisation, to emerge [21].

There are two large disadvantages which discourage a more wide acceptance of swarm usage, both due to the unpredictable nature of a swarm.

1. Firstly, the time taken to converge to the desired, global behaviour varies greatly for each run; it can sometimes converge very quickly and sometimes very slowly. This is because the convergence speed is dependent on the combined feedback times between neighbouring individuals communicating within the swarm and any subsequent spatio-temporal pattern propagations transporting, converting and combining local behaviours across the swarm [54] (which can be significantly slow depending on the spatial medium - for

instance it took several days for slime mold in a petri dish to produce a voronoi diagram [5]).

2. Secondly, the flexible, adaptive nature of swarms mean that their global, collective behaviours are difficult to predict. They often vary with different environmental conditions, despite individual-level behaviours within the swarm remaining the same [7]. This unpredictable adaptability poses sophisticated management challenges to controlling a swarm [46] and it is widely accepted that controlling a swarm (i.e. changing its global goal [45], stopping it if it is behaving too dangerously [8], etc) once it has started operating is not yet possible [8]. Thus far, most control features of true swarms have been to design global behaviours and features into the swarm during its design phase, prior to its deployment and execution within the environment [45]. Nevertheless, it remains the ultimate goal of this project.

Until a method to properly control swarms during run-time has been developed, it is far too risky to deploy them among humans [8] due to the potential threats they pose over safety, security and confidentiality [46]. Furthermore, a swarm can be viewed as a mobile dynamic network, and thus faces the many risks associated with networks (e.g. Cyber attacks to the physical, software or network layer of the swarm could easily disrupt communication links between mobile-robot-nodes). The emergent behaviour of the swarm could potentially dissipate if its network were significantly disrupted.

(II) Background

1. Robotic Swarms

(II.1.1) Robotic Swarms Vs Robots

A single robot is an autonomous system in itself [45, 51, 8] requiring minimal manual intervention during run-time [8]. Robots are thus well suited for "remote exploration missions" in hazardous locations with "harsh conditions" (i.e. deep space [51]) or other jobs that are considerably risky and dangerous for human beings. A robotic swarm is a multi-robot system [16] and is thus comprised of multiple autonomous individuals [46] (often large numbers [46, 51, 8]; in the thousands [51]). Whereby a single robot can only carry out a linear sequence of tasks, a robotic swarm can break up the list of tasks and distribute it across the swarm to be completed in parallel [22], completing the job more quickly, easily and efficiently.

(II.1.2) Robotic Swarms Vs Collective Robotics

However, swarm robotics is not the same as other approaches to collective robotics [8]. The robots in a robotic swarm are extremely basic, simple and reflexive individuals [51, 8] that merely react to sensory stimuli [8] (they "do not direct their work, but are guided by it" [22]). They are extremely unintelligent robots [41]; they have no memory of past actions or previous state information [41, 49] nor any internal models to map their current environment or represent their present states [41, 48]; and since they have no memory or models to plan and predict future actions [8], they are incapable of sophisticated [29], goal-based behaviours; unable to proactively plan ahead, make complex decisions or solve problems individually [46]. Furthermore, individuals in a robotic swarms do not have long-range communication [48] (nor global information like global positioning, by extension) and are limited to communicating locally (i.e. with nearby neighbours rather than the whole swarm) via direct interactions [34, 6, 45, 41, 48, 51, 8] or via short-range

sensors primarily used to detect immediate surroundings (e.g. infrared [6], virtual pheromones [41], etc).

(II.1.3) Robotic Swarms & Emergent Phenomena

The major advantage of such simple individuals [45] is that each robot requires minimal on-board processing [41] and only a basic processing power which is advantageous for miniaturisation [51] (robots can potentially be the size of dust particles [41] if coupled with advances in nanotechnology). Whatsmore, a society of low-level individuals cooperating reactively behaves intelligently as a collective, and complex tasks are solved in ways superior to solutions planned in advance via conventional, proactive, high-level methods [16, 29]. This artificial swarm intelligence [29, 51] emerges "without any active push for it at the individual level" [51] nor via properties from any single individual [29] and gives a whole new meaning to the age-old cliche "the whole is more than the sum of its parts" [45]. The emergent behaviour [56, 19, 16, 46, 8] of swarm robotics destroys the assumption that individuals obeying simple rules can only ever produce simple behaviours [58]. Complex global behaviours need not result from complex rules [58] as they can also emerge from very simple rules [34, 19, 51, 8] as demonstrated by the many natural systems from which swarm robotics draws inspiration (e.g. birds do not plan or knowingly cooperate to collectively fly in a 'vee-shape', rather each bird focuses on simple rules like flying at a certain speed and proximity relative to adjacent birds [58]).

2. Emergent Phenomena

(II.2.1) What is Emergent Phenomena?

The ability for natural systems to create order from chaos has gained the attention of a large body of academic researchers and spawned a whole new cross-disciplinary science, called "complexity science" or "complex (adaptive) systems", devoted to understanding this phenomena. Yet emergent phenomena is still not fully understood, despite being at the forefront of current research. "It is not well understood how such apparent complex global coordination emerges from simple individual actions in natural systems or how such systems are produced by biological evolution" [13] and thus "understanding and harnessing the fundamental organising principles of emergence remains one of the grand challenges of science" [55].

What makes emergent behaviour so difficult to understand is that, unlike resultant behaviour, the system's global behaviour is counterintuitive since it "is not a property of any of the components of that system" [58] and thus shows no correlation to the individual behaviour of the individuals making up the system [58] (e.g. analysing the behaviour of an ant will reveal nothing about how ant swarms are suddenly able to self-organise into an ant bridge). When analysing the local behaviour of any one individual, the emergent phenomena disappears. If emergent phenomena is to be studied, it must be done by analysing the distributed individuals in parallel. This may be hard to comprehend because "people have a centralised and deterministic mindset - they expect their to be a centralised leader (a bird leading the flock, queen bee controlling the swarm, etc) and are uncomfortable believing that randomness can sometimes give rise to orderliness or patterns" [60].

(II.2.2) Examples of Emergent Phenomena in Nature

Emergent behaviour is a mysterious, natural phenomenon which allows a group of randomly distributed individuals lacking any global information, intelligence, or global communication (via a central controller) to spontaneously self-coordinate into

an organised collective group, capable of a coherent, intelligent behaviour (e.g. emergent phenomenon transforms a collection of simple neurons into a complex, intelligent brain that can produce abstract thoughts). It is believed that emergent phenomena, like group learning, artificial evolution [39], global organisation and self-coordination, are all side-effects of individuals communicating with one another, explicitly and implicitly, in a decentralised, swarm-like manner.

Self-organising, group behaviours emerge across physical [21, 23], biological [21, 51] (insect [7, 16, 21, 28, 46, 59] or animal [21, 46, 59, 8]) and sociological settings. Physical systems may include stable magnetic orientations and domains [23] or vortex problems in fluids [23], etc. Biological systems include single-celled organisms [28] which exhibit emergent behaviours when in large groups (such as the spontaneous aggregation of bacterial colonies [13, 8], or the subtle adjustment of tumbling rates due to the perceived chemical concentrations which allows bacteria to move toward regions rich in nutrients [34]). Larger living organisms (like humans) are but a collection of cells, self-assembling and interacting locally [44] to form tissues, [8], organelles [54], organs [8], organ systems and other necessary body systems (such as the immune system - which has inspired many network intrusion detection algorithms [45]). The brain (which has inspired the creation of Artificial Neural Networks) is nothing more than a large collection of specialised cells called neurons [13] that interact locally (by exchanging electrochemical signals [57]) to parallel process external sensory information [13] via emergent computations [43], resulting in our internal thoughts and emergent mental images [43].

Emergent phenomena is also rife in social systems like insect colonies, such as flies [12], fireflies (which are able to flash together, synchronously), spiders [45], cockroaches [8], termites [22, 41, 8] wasps [29], bees [34, 45, 29, 8] and ants [34, 9, 22, 45, 41, 29, 51, 8] (which are by far the most commonly studied social insects). In particular, the way ants forage (which has inspired network routing optimisations), build nests, sort their brood (eggs are sorted and grouped by developmental stage [22]), manage their dead (experiments involving the random distribution of dead ants will result in workers forming clusters within a few hours [9]), divide their labour (inspiring task allocation solutions), self-assemble (physically connecting to build bridges, rafts, walls and bivouacs [24, 8]), make collective decisions, come to a consensus (deciding between the shortest of two paths [8]) and implicitly cooperate (i.e. to carry heavy food). Similarly, emergent phenomenon readily occurs in animal societies, such as schooling fish [56, 34, 9, 28, 29, 8], flocking penguins [8], migrating birds [56, 9, 28, 51, 8], herding gazelles [56] and many other social animals [28]. Crowds [34, 28] and 'mob' mentalities [54] are some examples of emergent phenomena which often occur in human societies [45, 60].

According to Physicist David Bohm (in his theory of 'implicate and explicate order' [1] which elegantly resolves long unanswered questions in the field of Physics - such as how is quantum physics and general relativity unified? How does quantum entanglement allow for faster-than-light communication? etc), reality itself is nothing more than an emergent phenomenon! The explicate order (each temporal moment in our spatial reality) is a surface phenomena - an emergent projection - "that temporarily unfolds out of an underlying implicate order" [42]. This idea concurs with earlier revolutionary ideas supported by Physicists Stephen Wolfram [2] and Richard Feynman, which state that the entire Universe is a parallel processing information via emergent, spatio-temporal computational structures, like those found driving emergent phenomena in cellular automata.

(II.2.3) Important Conditions to establish Emergence

Sociology is not just applied psychology, just as "psychology is not applied biology, nor is biology applied chemistry", neither is chemistry applied physics, nor is physics merely applied quantum mechanics [58]; "The whole is greater than the sum of its parts' ...The extra bit is the consequence of how the parts interact" [58]. At the ground level, the action of one individual activates another individual, like a complex domino-effect or chaotic chain-reaction (which is why this process is sometimes referred to as "chaining"). In this manner, a system of individuals (who only require rudimentary reasoning [9] themselves; enough to reacting to external stimuli via very simple rules) is able to behave intelligently as a collective by executing sophisticated 'rule-chains' constructed via the complex, parallel 'chaining' of numerous such individuals.

However, chains of interactions won't necessarily produce emergent phenomena [58]. Or if it does, it may not necessarily be emergent behaviour which is intelligent and useful. Some systems only exhibit globally emerging patterns which seem to be no more than aesthetically intriguing (e.g. "zebra stripes" [45] or the natural ordering of stones by size as they wash upon the seashore[45]) rather than more advanced emergent behaviours like the type whereby individuals can self-organise to solve complex task, perhaps only producing patterns as a side-effect (e.g. ants foraging, termites constructing nests, etc).

If a system's behaviour can re-influence the original system (i.e. there is feedback [58, 7, 17, 45]), then the system's behaviour begins to modify itself dynamically, becoming nonlinear in the process [58]. Positive feedback reinforces and amplifies certain behaviours [7] (promoting the creation of structures via a "snowball" effect [17]) whereas negative feedback is like like a regulatory mechanism [17] which stabilises patterns and counterbalances positive feedback [7]. Nonlinear [58, 50, 23] interactions [58, 50] are a key element to establishing and thus intelligent, emergent, self-organising behaviours. This (coupled by an element of randomness [7]) is why emergent behaviour is near impossible to predict [45] at the level of the individual. Thus the secret to unveiling emergent phenomena does not lie within the swarm's individuals, but within their (spatio-temporal) interactions [58, 34, 23, 45, 8].

3. Embedded Parallel Computations

(II.3.1) Investigating Emergence using CAs

Similar lines of research into understanding, controlling and predicting Emergent behaviour have been conducted using a number of distributed systems other than robotic swarms. The most popular distributed systems used (due to its relatively simple nature) are Cellular Automata (a group of virtual 'cells' - with a discrete size, conserved number and fixed position. They can only really change their states, and the simplest CA's only have binary states; dead or alive; black or white; etc). The "majority of these studies is focused only in the one-dimensional (binary) CA" [15] which is just a 1D row of virtual cells (as opposed to a 2D lattice) since "it is important to study how this phenomenon emerges in even the most basic complex system" [27]. Using "a simplistic representation that maintains the most important features of the complex system being modeled" [26] makes the task of understanding a complex concept like emergent computation easier and clearer [12, 13, 49]. The general principles behind how complexity arises from simple rules and many of the secrets behind emergent computation have been revealed through studying 1D CAs [26, 45, 52]. A lot of progress was made when studying 1D binary CAs evolved via a Genetic Algorithm (GA) [13,14,35] to perform intelligent emergent computations such as calculating its own global density (the classification task); i.e. the CA must decide

which state are the majority of the cells at the start (the density of its initial configuration) and then slowly make the CAs final configuration (output) into that state globally (e.g. if the majority of cells in the initial configuration were “alive”, then the final configuration should transform all cells to become “alive”, and vice versa) [35]. Bare in mind that this is no trivial task for a 1D line of fixed cells which can only communicate its binary state to its nearest neighbours. Nevertheless, researchers have also extended their studies of emergent phenomena to 2D CAs [38] (an example of an emergent phenomena in 2D CAs is object boundary detection in images. A 2D CA can achieves this fairly easily using the “majority rule” - i.e. a cell adopts the same state as the majority of its neighbours [38]).

CAs update their cell’s states based on predefined update rules (i.e. a lookup table which defines which state a cell should change to based on its current state and the states of its neighbouring cells [57]). Some update rules have been noted to produce Emergent behaviour, while others do not (those which do have been studied deeply to better understand the beautiful, mysterious phenomena of emergence) Thus, different update rules have been classified into four distinct classes [2] based on the global CA behaviours they produce. Class I: “fixed-point-attractors” & class II: “periodic-attractors” both have short-lived transient times and converge too soon to produce dynamic behaviour; quickly collapsing into orderly homogeneous (class I) or heterogeneous (class II) states. Class IV: “strange-attractors” (a.k.a. ‘the edge of chaos’ [30]) has a long, indefinite transient time and eventually converge into complex states; although it is globally disordered, embedded sites of order emerge. Class III: “chaotic-attractors” have infinite transient times and never converge because it diverges into random, aperiodic states of chaos. Only classes III & IV are capable of creating the correct conditions for emergent behaviour because emergence can only occur in the fluid ‘transient time’ before the system solidifies into a converged state [25, 30, 40, 32, 36]).

(II.3.2) Collision-based computing

Collision-based computations [10] (a.k.a. computational mechanics) offer a convincing theoretical explanation to explain intelligent, self-organising, global behaviours (i.e. make decisions, remember, classify, categorise, generalise, recognise, problem-solve, correct-errors, etc [23]) in cellular automata [13]). The theory views complex systems (including robotic swarms) as decentralised networks of emergent information processors [23]; architecture-less computers (as opposed to conventional, von-neumann-type, central-processing computers). Individuals within the complex system indirectly (and unknowingly) communicate with one another via an embedded computational layer composed of dynamic, spatio-temporal ‘structures’ that serve to parallel process information across the entire complex system [38, 13]. This means that even in the absence of a central controller and access to global information, individuals within the distributed system can still communicate globally via this embedded, computational layer.

(II.3.3) Embedded Spatio-temporal Structures

The essence of ‘collision-based’ computations are “nonlinear logical operations” [23] performed by emergent information-processing elements; ‘spatio-temporal patterns’ which parallel process information [13]. Without these emergent spatio-temporal patterns, information processing could not occur, since the alternatives would either be too ordered and unchanging to transport or modify information [24], or too dynamic and changing to store information long enough to modify or transport it. So what exactly are these fundamental spatio-temporal patterns and from whence do they emerge?

During investigations into the emergent behaviour of 1D CA's, "solid, long and narrow structures" [52] (termed 'particles') were seen emerging from localised dynamic regions of chaos (chaotic regions seem to be favoured over static, orderly regions due to their random perturbations which act as "nucleation sites" [52] from which the embedded, particle 'structures' grow via smaller 'proto-particle' structures [52]). Similar discoveries were soon made in 2D CAs too (Conway's "Game of Life" is a famous example of a 2D CA with a rich variety of spatio-temporal patterns. More commonly known as "gliders" - the 2D CA's equivalent of a 1D CA's "particle").

These emergent patterns (also referred to by various names across the vast yet scattered body of research literature - "attractors / stable points" [23], "distributed embedded devices" [45], "vehicles" [52], "embedded structures" [10], "momentary wires" [3], "signals" [13, 52, 3, 44], "communication blocks" [44], "virtual particles" [47], "gliders" [57, 52, 10, 4], "gestalts" [23], "domain boundaries / walls" [13], "mobile self-localisations" [10], "wave fronts / fragments" [3, 10, 4], "travelling localisations" [4], "compact configurations of non-resting states" [4], "active zones" [5], "dynamic computational mechanisms" [22] etc) are in actual fact boundaries between adjacent, localised, homogenous regions [38]. The types of regions which border one another determine the pattern of the spatio-temporal structure produced. The space-time conditions of the system must be complex and chaotic enough to encourage multiple regions to exist (as is the case with Class III and IV CAs). If the entire system is too static and ordered (as with Class I and II CAs) there can be no bordering of localised regions of order, and thus no spatio-temporal structures or emergent behaviour. Over time, as the dynamics of the complex system change, so too do these local regions (changing shape, expanding or shrinking, merging with other local regions like two bubbles suddenly combining, etc) thus causing their boundaries to shift (appearing as if it the spatio-temporal structures are propagating, colliding, transforming, etc).

These spatio-temporal structures (regional boundaries) are not explicitly represented in the system [38] since they are embedded within the nonlinear interactions of individuals and are only revealed via analysing the spatial interactions over time. We could even say that these spatio-temporal patterns are the underlying dynamics of emergent behaviours [34, 13] and the fundamental processors [13] of emergent collision-based computations and the driving force behind global emergent phenomena (the interactions are merely the carriers or media in which they exist and the links themselves only existing as a result of the physical individuals interacting, making it a 3rd order entity, or 2nd order emergent phenomenon).

(II.3.4) Representing and Storing Information

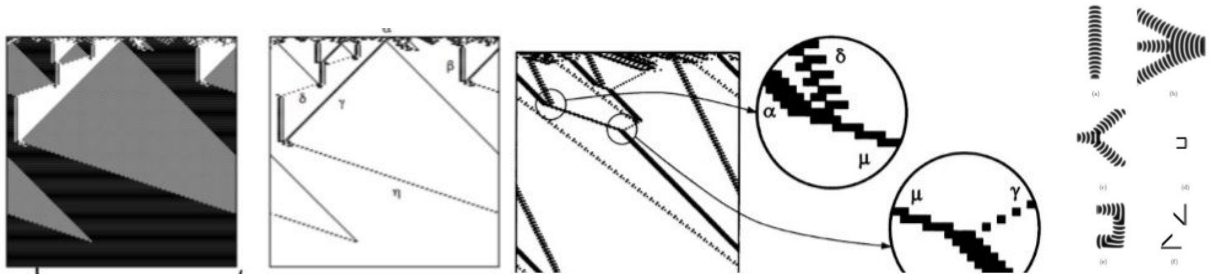
Spatio-temporal structures (e.g. "virtual particles" in 1D CAs, or "gliders" in 2D CAs) come in different, unique patterns, each representing specific pieces of information [13, 44]. The data is stored in the system so long as these structures persist (like a memory [12, 13, 44]). Researchers studying the computational mechanisms of 1D CAs managed to identify five unique "stable particles" (spatio-temporal patterns), which they labelled as "α, γ, δ, η, μ" [13], and curiously, one unstable "particle (β)" [13], which suggests some spatio-temporal structures spontaneously change (i.e. their patterns or velocities) without any external influences. However, the majority of spatio-temporal structures are "stable" and require an external event to drive a change.

(II.3.5) Transferring information

Information is transferred across the system via the spatio-temporal pattern propagating over time [38, 13, 44]. These spatio-temporal structures are essentially

emergent signals used to process, store and communicate information across the system, with its medium of travel being the system itself [44]. Therefore "almost any part of the medium's space can be used as a (momentary) wire - a trajectory of (a) travelling (spatio-temporal) pattern" [3, 10]. Each spatio-temporal structure will have a set velocity (speed and direction) at which it propagates through the system [38, 13, 44].

(II.3.6) Changing & Computing Information



Information is modified if the spatio-temporal pattern representing it becomes modified and so to changing a pattern is how to change data [13, 44]. Spatio-temporal patterns (as well as their velocities) are most commonly changed via collisions between two or more spatio-temporal structures [25, 24, 26]. Collisions change the structure's velocity and pattern according to an intrinsic logic specific to that system [25, 24, 26]). Interestingly, collisions always follow a deterministic logic (e.g. spatio-temporal patterns "α" and "δ" always produce spatio-temporal pattern "μ" upon collision). This means that spatio-temporal structures travelling and interacting (i.e. colliding) in space form the "basic logical operators of (dynamic, massively-parallel, architecture-less, collision-based) computation" [13, 52, 3, 10]. Logical collisions correspond to computations that transform the data. 'Logical operations' occur at the place where the spatio-temporal structures collide, annihilate, fuse, split or change direction (these sites correspond to the 'logical gates') [3] and various forms of logical gates are realisable [3, 10] (including xor gates and diodes [3]). The presence (or absence) of spatio-temporal structures represent the boolean truth values of logic gates [3, 10]. Collision-based logic-gates typically have inputs corresponding to the presence of the colliding spatio-temporal structures. Its outputs correspond to all the possible outcomes of their interaction, including and output resulting from non-collisions (i.e. the output will be the same as the input) [3]. "To generate dynamics representing basic logical operators (is) the foundation of computation" [52]

The research into emergent phenomena in CAs have inspired the theoretical foundations of our investigation into the underlying mechanics of emergence of robotic swarms. It is not unreasonable to hypothesise a parallel system of embedded spatio-temporal structures underlie the emergent behaviours of robotic swarms since there have been odd reports to suggest that such computational mechanisms exist for complex systems other than CAs. For example, "structures that propagate in a coherent direction and speed" [4] have been experimentally manifested in a chaotic 2D chemical media (BZ mediums [10]) and its behaviour and computational dynamics are comparable to those of 2D cellular automaton. As "wave-fronts" in the chemical media expand, their collisions produce new wave-fragments in a deterministic manner [4]. Thus even physical media are capable of collision-based computing, since the collision of spatio-temporal structures emerging in their "space-time evolution" are represented by interacting wave fragments "geometrically constrained" to the "chemical medium" [10]. Unlike simulated 2D CAs, however, the emergent structures in the chemical media disintegrate after some time. "Stable" spatio-temporal "entities" (more popularly referred to as "gestalts" [23]) have also been observed emerging in the "flow" of

(discrete or continuous) phase spaces in artificial neural networks. Gestalts store information in their locally stable structural configurations and act as a form of memory [23]. "Various classes of flow patterns are possible" [23] and likewise, if these local stable points can be induced (or manipulated) the neural network could be controlled and a specific memory could be assigned [23].

(III) Literature Review: Emergence in Robotic Swarms

(III.1) Engineering Emergence in Robotic Swarms

One of the earliest robotic swarms was comprised of virtual agents (called 'boids' [29]) and inspired by the emergent behaviour observed in flocks of birds. What began as an educational exercise to more realistically simulate the dynamics of birds and bats flocking turned into a study of emergent phenomena. This type of emergent behaviour is useful for predator evasion, coordinated motion, coherent steering toward common goals and avoiding obstacles [29]. Via a painstakingly slow process of simplification, refinement, trial and error, the designers developed a rule-set for the individual behaviour of each boid which would allow them to flock together without colliding into one another: (Rule 1) Repulsion: No other boid is allowed to come inside the boid's personal bubble (an imaginary 'zone of repulsion' immediately surrounding each boid). If they did, the boid must 'separate' itself by steering to avoid them. (Rule 2) Cohesion: Boids must steer toward the average centre position of all flock mates so that they remain a flock. (Rule 3) Orientation: boids must 'align' themselves with the flock by steering toward the average heading of nearby flock mates. Obeying these three simple rules, the boids behaved similarly to that of a real flock of birds.

Alpha-algorithm [53] further builds upon the idea of simulating emergent flocking behaviour but with a simpler set of rules than those used for boids (the product of even more hours of deep thought): (Rule 1) Move: To prevent the swarm from stagnating, each agent moves forward by default. (Rule 2) Cohere: To stay inside the swarm, each agent makes a 180 degree U-turn if the number of its local neighbours drops below a specified threshold (this reasonably assumes that a robot's neighbours will decrease if it leaves the swarm). (Rule 3) Disperse: To avoid collisions with neighbouring robots, each agent turns randomly if the number of its neighbours rises above a certain threshold. Larger threshold values tend to result in a more dense, highly compact and interconnected swarm [53].

An even more unique approach which brings about similar flocking behaviour in robotic swarms uses only a single rule! The approach has become the most popular and is known variously as 'virtual attraction-repulsion forces' [34, 47, 31, 45, 41, 48, 59, 8] (a.k.a. Artificial physics, artificial potential field, physicomimetics, etc) based on well established principles and laws of physics. Each robot in the swarm need only calculate imaginary attraction and repulsion vectors (virtual forces) using physics-based equations (rules). Any point in the environment can be set as a 'goal' that the swarm will gradually gravitate towards by associating it with an 'attractive' imaginary force. Also, the swarm can be kept together (within communicating range) by associating neighbouring robots with a force that becomes more 'attractive' the further they move away from one another. Similarly, to avoid obstacle collisions, other robots and environmental objects which are too close can be assigned an imaginary 'repulsive' force. It also prevents the swarm from collapsing in on itself due to 'attractive' forces. This 'repulsive' force may be very strong for closer objects, yet weaken with distance to gently encourage the swarm to spread out (while cohering as a swarm) to maximise the exploratory coverage of the environment.

To communicate, robots only require basic sensors (e.g. ir, sonar, etc) to estimate their relative distances from one another and of nearby obstacles, which can all exert an imaginary attractive or repulsive force on the robot (the object's imaginary attractive or repulsive force is a mathematical vector consisting of a magnitude and direction - i.e. a sizer and angle). The robots' single rule is a mathematical function which smoothly translates the distance (d) and direction (θ) of the object or robot sensed, into a force vector (f). After each virtual force is calculated, the robot finds the average net-force using simple vectorial operations. The resultant net force vector is used to determine the robot's new speed and direction. The final effect within the swarm is 'velocity matching' (robots attempt to match the average velocity of its neighbours).

A similar approach uses potential fields (rather than forces), which directly maps and integrates a robot's multiple sensor readings to form an overall imaginary field surrounding it (similar to an internal map). Although this approach is slightly more computationally heavy than calculating force vectors, it has the advantage of automatically summing all the imaginary forces together to calculate the average net force field. Virtual fields also work better than virtual forces in more condensed and cluttered environments since "it is often more effective to reason about the free space surrounding a robot" [41].

Aside from the simplicity of the rule when using the virtual forces or fields approach, the resulting emergent behaviour is far richer and easier to manipulate. Like the previous versions, the robots flock together in a common direction, however, unlike the previous versions whereby robots had to be explicitly programmed to flock in a common direction by matching their speeds and directions with neighbouring robots, in the virtual forces method, the common direction is simply a side-effect of robots moving toward a common goal, which can be specifically selected and changed during run-time, making it easier to navigate the swarm using the virtual forces method. As the swarm of robots are being attracted to their long-range goal, they are simultaneously being repelled from any close-range obstacles or neighbouring robots in order to avoid collisions. Perhaps one of the greatest advantages of the virtual forces method over the previous approaches is the ability to design patterned group formations using physics-based equations for phase-transitions (e.g. liquid to solid). Robot swarms flowing around objects in a dynamic, adaptive swarm without any particular formation (liquid state - excellent for collision avoidance) [47] can suddenly 'crystalise' into groups of rigid square or hexagonal bumble-bee-like lattices (solid state - excellent for sensing grids) [47, 8]. (This occurs because there are stable points where the attractive cohesive forces delicately balance or cancel out with the repulsive dispersive forces). For example, each robot has a "potential well" around itself, such that a neighbouring robot that is too close will be repelled and yet attracted if it goes too far away. This can settle into a stable hexagonal formation with a robot in the middle surrounded by neighbouring robots that are at exactly the right distance away (i.e. the pre-defined radius) [47]. "Larger forces result in deep potential wells, allowing (robots) to form very stable sensing grids" [47]. Unfortunately, there are only a limited subset of patterns which can be formed without the use of global knowledge like a common orientation (ie. acquired by a compass) [8]. More shapes can be formed if the swarm of robots is heterogenous [8] (e.g. there are two types of robots which have different attraction-repulsion thresholds). By making the "potential (force) wells" disappear, the swarm will phase-transition back into its liquid state [47].

Unknown terrains which have many obstacles can be hard to map using traditional methods, yet if a robotic swarm were deployed in its 'liquid' state, it would effortlessly flow around unforeseen obstacles while spreading out across the environment. Thereafter it would solidify into a static formation and act as a sensing grid embedded in the environment. This sensing grid (solidified swarm) acts as an external map of the terrain, each robot acting as a simple processor that can sense the piece of terrain in its locality and propagate information across the grid via its nearest neighbours. The sensing grid would not suffer the same communication loss as traditional long-range methods because it only needs to stay in contact with its closest neighbours close-range communications. The sensing grid could map out blocked or hazardous areas in this uncharted terrain and plan optimal routes using wavefront propagation methods such as Dijkistra's algorithm [41]. Certain robots could then light up to display the shortest path on the sensing grid like an external map.

A disadvantage of this technique over the previous rule-based approaches is that virtual forces require fine-tuning and even then, it may be that "determining the parameters for a desirable group behaviour is computationally infeasible" [48]. As the forces are modified, the robot's behaviour is also affected [45], and is it this intricate feedback that makes tuning the force parameters more complex. Some found that a 'neutral zone' needs to be factored in to avoid unwanted attraction-repulsion oscillations [41]. Nevertheless, the technique is commonly employed and many interesting variations and modifications have been designed on top of the standard model described above.

One interesting variation which was "inspired by the way in which bacteria react to a chemotactic gradient" [34] factors in the effect of sensorial time delays (δ) (the time required for the robot to calculate a response after sensing the object's distance [34]). Usually subtle factors such as sensorial delays are assumed to have negative impacts and are almost always ignored during the modelling phase. "Only few theoretical works have considered its possible constructive effects" [34]. One team noticed that robots spend more time in regions with low net forces or low intensity fields [34] and that adding a positive time delay pronounces the robots' tendency to do this, whereas a negative time delay (i.e. predictive) encourages robots to move toward regions of stronger forces [34]. The delay parameter can thus be used to control a swarm's dispersion and aggregation behaviours [34]. Adjusting the delay negatively reduces the sensitivity of robots and allows them to disperse slightly, expanding the swarm enough to explore the environment more effectively, while adjusting the delay positively can bring the robots back into a tight cluster so that the gathered information can be shared more easily. [34]

Another team exploits the idea of swarms with robots that can change their internal state (i.e. adjust their orientation using a random rotational effect [34]) to encourage more intelligent behaviours to emerge (i.e. reactively wall following the boundary of obstacles [31] to escape from local minimas in the landscape which the swarm has become trapped inside [31]). A major challenge that exists for all reactive path planning and search algorithms is known as the 'local minima problem' (e.g. "the whole swarm will be trapped at the barrier because the (robots) trapped inside the barrier will suffer two opposite forces; the first force is the repulsion from the barrier while the other.. (is) the attraction to the goal" [31]). Emergent vortexes are "a new technique to escape the local minimum position" [31]. Rather than the swarm fully solidifying into a rigid grid, it relaxes into semi-stable, semi-dynamic vortexes [31]. The vortexes manipulate the virtual forces or fields of the environment enough for the robots to escape from the local minima. The swarm's central velocity is linked to

the strength of the goal's attractive force so that as the swarm's velocity slows down (i.e. it is becoming trapped in a local minima) the strength of the goal's pull weakens and thus the local minima disappears. "At the same time when the goal effect decreases, a swarm vortex pattern emerges" [31] as the robots rotate around their group centre.

This emergent vortex behaviour was further exploited by manipulating another internal state of the robot. Individuals closer to a wall slow down while those further from it speed up. This causes a distortion in the vortex's rotational velocity which makes the part nearer the wall drag and thus the entire vortex begins to tumble. A rolling effect ensues that lets the swarm follow obstacle boundaries as a rolling vortex. Thus, the robotic swarm was directly controlled and an emergent vortex behaviour created by manipulating the internal states of the robots. This effect was made more intelligent by changing the relative velocity of the robots in the vortex to cause a unique wall following technique to emerge. Despite its ability to solve the local-minima problem, it has been noted that such applications are limited and the solution, although efficient, is not optimal since the swarm will take longer to reach the goal using wall-following behaviour of the tumbling-vortices [31].

Yet another modification to the virtual forces method, inspired by ant colonies, employs 'virtual pheromones' [41]. A robot will send a communication 'ping' to any nearby neighbouring robot (without specifying a specific recipient). Upon receipt of the message, the neighbouring robot's internal state will be modified. Messages are relayed from robot to robot to slowly propagate through the swarm, however, to reduce obsolete and irrelevant information being spread, each message begins with a known intensity and an inbuilt feature ensures it gradually decays with distance. Robots use the intensity of the relayed messages to estimate the original distance and direction of the first robot that sent it. "Pheromone diffusion gradients provide important navigational cues and also encode useful information about barriers in the environment that block pheromone propagation" [41]. Virtual pheromones increases the communicability of robots within the swarm by creating implicit (temporal) network links. The added bonus of using 'virtual' pheromones as opposed to real chemicals, is that additional information can be sent in the message 'pings' if so desired.

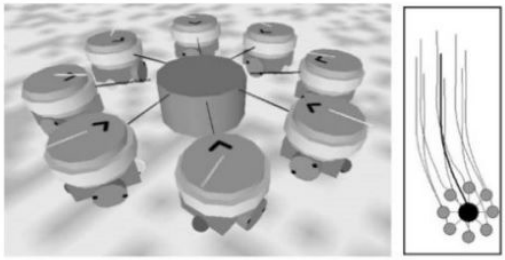
(III.2) Evolving Emergence in robotic swarms

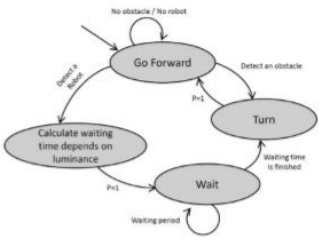
The emergent behaviours have thus far focused on coordinating the movement of swarming robots and were each hand-designed with inspiration drawn from nature (i.e. birds, ants, bacteria, particles and forces, etc). Such emergent behaviour is useful for numerous practical applications, such as search and rescue, collective transportation, task division and embedding mobile sensing grids in an unknown environment. The next few examples were designed automatically using artificial algorithms inspired by natural evolution. Unlike the previous techniques reviewed, which seek to imitate nature through a lengthy design and refinement process based on trial and error, these emergent behaviours were haphazardly evolved using modern genetic search algorithms.

The first artificially evolved robotic swarm we shall review, 'sbots' [17], makes use of robots with small artificial neural networks consisting of five neurons. Rather than being programmed with simple rules, each robot's 'brain' stores a specific value for its traction, plus a single bias value. These neurons are further connected to two motor neurons which directly control the movement of the robot's wheels and motor chassis (joint). The weights for each artificial neuron were all evolved and thus the value stored by the robots have been automatically fine-tuned to suit their

environments containing obstacles. The emergent behaviour of the sbots is unlike the previous swarm behaviours which were hand-designed to imitate a natural system. Individual s-bots have learnt (using their artificial neural networks) how to avoid obstacles using an unusually clever approach "When an s-bot hits an obstacle, the collision generates a force on the chassis in the direction opposite to the obstacle. This force is interpreted by the s-bot as a traction force (and) as a consequence, the s-bot tends to turn so as to cancel this (perceived) traction force" [17]. Even more amazingly, is that the s-bots as a swarm have evolved to collectively transport heavy objects (i.e. an injured body) in a coordinated fashion by physically connecting themselves to the object, or alternatively physically connecting themselves together around the object to push or pull it. They also evolved the behaviour of dispersing to avoid colliding with the obstacle; "the traction resulting from the collision is transmitted to the other s-bots through the physical links, forcing the whole group to reorganise and change direction ... avoiding the obstacle" [17].

In comparison, the bee-clust algorithm [6] uses state machines to produce similar emergent behaviour (as opposed to using artificial neural networks or evolution). However, the algorithm is far more complex than the evolved approach, and even though the logic of the final evolved rules can be difficult to understand, the approach is often unique and one that would not have been considered were it to be designed by hand.





Another popular method of evolving emergent behaviour in robotic swarms is by co-evolving two different breeds of robots side by side [56, 21]. The two breeds adopt a predator-prey relation and the competition drives each type to adapt and improve more quickly; the predatory robot becomes better at overcoming the prey and in-turn the prey robots are driven to evolve emergent behaviours that better avoid being overcome by the predator, which then forces the predator type to up its game, (similar to how viruses evolve in response to antiviruses). Therefore this is the indirect fitness function which drives the evolutionary process.

The environment in which the robot swarm was evolved was virtual (an unbound, two-dimensional plane). During the evolutionary process, each robot is assigned the following information; species (predator or prey), gender (male or female), location (x-y coordinates to know which robots are near to one another), age (robots constantly age until they reach an old age limit at which point they die and are removed from the evolutionary process), health / energy (this value is set to the average of the robots' parents' health and constantly decreases as the robot moves, until the robot replenishes itself by 'eating'. However, if the health reaches 0, they 'starve' to death and are removed from the evolutionary process [56]). Aside from these values, each robot contains a behavioural rule which is essentially a set of symbols, each representing a specific behaviour that the individual robot will do. The robot execute each behaviour in the rule in the order of the sequence the symbols are stored in the set. During the evolutionary process, robots can mate, at which

point a new robot is produced whose set of symbols is a combination of its parents' symbols (crossover). In this way, different combinations and sequences of robot behaviours can be explored. If the child's new combination of symbols produces effective behaviours, then it will stand a better chance of surviving and reproducing and continuing the line of search for that combination, however, if not, it will quickly die off and end the search there.

Although the individual robot behaviours are not subject to change (i.e. via mutation), their order and priority of execution is free to change during evolution. Some of the behaviours (and their symbols) included:

- Moving (R)andomly (a stochastic element can improve the exploration of a robot
- Seeking the (P)rey, & (F)leeing from the Predator (robots can only see other nearby robots within their visual sensing radius)
- (E)ating Prey (predator robots must eat prey robot to replenish their health. If a predator is "hungry" and a prey is adjacent, it kills it and a portion of the prey's health transfers into the predator [56]) & Eating (G)rass (Prey robot must eat grass to replenish their health. Surprisingly, robot prey often evolve to favour moving less, even though this increases the chances of being caught by predators, it reduces their own energy expenditure and increases the chances of finding "freely-floating" food [56])
- A(L)truism (the robot stays close to a neighbouring robot of the same species, perhaps to hunt cooperatively or huddle in a herd to increase safety [56]), & Moving (A)way (the opposite of altruism, the robot selfishly distances itself from neighbouring robots of the same species [21] - the robot "eschews (its) own kind in favour of moving toward prey" [56])
- Seeking a (M)ate (the robot looks for a robot of the opposite sex within its species [21]), & (B)reeding (the female robot creates a child robot if it is healthy and has a mate adjacent to it)

Emergent behaviours such as flocking were co-evolved in both predator and prey species. Also, "fascinating emergent macro-behaviours such as the defensive spirals" [21] were evolved in prey as defensive mechanisms "to disrupt the process of predation" [56]. Predators generally evolved to hunt cooperatively [56] and eventually a sophisticated, long-term strategy evolved whereby a swarm of predators form "a spiral battlefront … (and) slowly rotate around as (prey) are born, eat, die and move" [21]. "When prey (robots) are not disturbed by predators they stick more or less together and swim playfully in their home area" [56].

One advantage of evolving emergent behaviours instead of engineering them, is that often counterintuitive behaviours are evolved (behaviours which human designers would not have considered beneficial without hindsight). For example, 'altruism' would not have been a behaviour designed to ensure a breed of robot survives in a competitive environment with limited resources. However this is exactly the behaviour which ended up being evolved that best benefited multiple groups faced with limited resources. It seemed that, although selfish algorithms dominate when resources are plentiful, they do not perform well when resources are scarce [21]. Altruism, on the other hand, reduces the overall competition for resources [29], thus allowing it to regenerate and increase its supply to meet the larger demands [13].

From the range of work reviewed, it is clear that artificial evolutionary methods produce more sophisticated emergent behaviour than engineering emergent behaviour by hand, which requires a lot of thought and multiple revisions at the level of the rules before the desired behaviour is found and fine-tuned. Artificial evolution

is essentially an efficient search algorithm, however, unlike with emergent behaviours designed by hand, the evolved emergent behaviour can be a black box and the way it works is often peculiar, unlike how we would have designed it, and thus mysterious. This project aims to identify the underlying computational mechanisms within artificially evolved emergent behaviours in robot swarms so that the underlying computational mechanisms can be exposed and controlled at the design level and during run-time.

(IV) Application of Theoretical Framework

(1) Design Methodologies

(IV.1.1) Designing Rules for Global Emergence

Swarm Engineering is a relatively young research field [45] dedicated to the systematic and scientific design, verification and validation [8] of Artificial emergence [58] within robotic swarms. This includes suppressing the emergence of unwanted behaviours as well as encouraging desired emergent properties [45]. Emergent behaviour in swarm robotics can be modelled from two angles: the microscopic level (i.e. a bottom-up behaviour-based approach) and the macroscopic level (i.e. a top-down evolutionary-based approach) [8]. Which of these approaches is better at designing globally coherent, self-organising behaviours in robotic swarms [8], is the subject of study and research

(IV.1.2) The Behaviour-based Approach

Each robot runs on a simple set of rules which govern how the individual reacts to localised changes or events [6]; in other words, how to interact with nearby robots and the environment. “In swarm robotics, there are still no formal or precise ways to design individual level behaviours that produce the desired collective behaviour” [8]. Behaviour-based design methods tackle the problem using a bottom-up approach [8] (i.e. designing for global emergence at the level of the rulesets which govern how individual robots behave locally with regards to robot-robot and robot-environment interactions [8]). The designer will seek out that the simplest, nonlinear behaviour that produces the desired complex global behaviour [58] using their personal intuition [8], trial and error and continual tweaking. The bottom-up approach (a.k.a. exploratory-based method [62]) is somewhat similar to the scientific method [54]:

1. Aim: decide on the desired global emergent behaviour
2. Hypothesis: identify the local conditions which you think are the root causes that give rise to such global effects (the hypothesis stage will ultimately decide how successful this approach is)
3. Design: design a working prototype based on your hypothesis
4. Test: Evaluate how well this prototype reached the desired global effects
5. Analyse: identify any counterproductive conditions that are preventing the system from reaching the desired global effects
6. Redesign: modify and fine-tune the prototype
7. (repeat the testing, analysis and redesign stages until the goals are reached)

The skill of identifying the root causes that lead to desired global behaviours is still largely dependent on the designers intuition. However, there are some approaches that try to determine these root causes in a principled way [59]:

(IV.1.2.A) Imitation

The simplest approach is to take an example of the desired emergent behaviour from a natural system and imitate its local behaviours or what are

"perceived to be the critical components" of it in order to produce a ruleset. This approach is now considered inadequate and often "something is always missing" [58]. However, this approach has produced good results such as the virtual-field method, which imitates the physics-based rules of a natural system. One reason this approach may work so well is that these physics-based rules have been tried and perfected within its field by many great minds over a long stretch of time.

(IV.1.2.B) Decomposition

Decomposition is a slightly more sophisticated approach which attempts to deduce the ruleset from the global behaviour of the swarm, or its subparts. However, it can be very "difficult to predict which behaviour results from a given set of rules and which are the rules behind an observed behaviour" [17] due to the complexity of nonlinear spatial or temporal consequences resulting from local interactions. These "indirect relationships between the rules executed by the robots… and the overall group behaviour" [50] are what make it difficult to "analyse each subpart and then draw conclusions about how these subparts will produce a collective global behaviour" [54]

Part of the reason complex emergent properties are not easy to predict or deduce from the simple behaviours of individual robots [29, 59] is because emergence is a holistic phenomenon; one must consider the entire system in its spatial and temporal context, rather than isolating and divorcing these components for simpler analysis [17]. In fact, many biological or social systems which exhibit a rich variety of emergent phenomena "are not obvious from even a detailed knowledge of the workings of the individual component parts" [21] (e.g. years of analysing individual ant behaviours in isolation have shed no light on how ant colonies are able to spontaneously build bridges. It would be like analysing the optimal pricing of goods in an economy by studying the individuals who obey the rules of commerce [13]). "(Even) if you know the micro behaviour, it is still difficult to predict what the global behaviour and emergent patterns will look like" [17, 60].

Another problem for the deconstructivist approach is that emergence is scale dependent; at each level of complexity entirely new properties appear. "Properties describing one level of a complex system do not necessarily explain another level, despite how intrinsically connected the two may be (e.g. understanding the emergence of the structure of molecules does not necessarily allow one to predict the emergence of cellular biology)" [55].

(IV.1.2.C) Inference

Another approach aims at inferring the rules from the bigger picture [58]; analysing the entire swarm holistically across various examples of desired behaviour allows designers to take the commonalities as rules. This approach does require a great many examples, however, to observe how a robot behaves in a variety of different situations and environmental conditions to infer the correct rules because the "individual behaviour is the emergent result of the interaction between agent and environment" [17]

(IV.1.3) Limitations of the Behaviour-based approach

There are currently limitations to how precisely you can design emergent behaviour. Although general emergent trends and features can be designed for [54], it is widely accepted that designing emergent behaviour so exactly that each robot's position

and velocity can be mapped out in advance, is impossible (due to the inherent nonlinearity and unpredictability of emergence [45, 54]).

(IV.1.4) The Evolutionary-based approach

One of the main problems of designing local rules by hand is the process is somewhat subjective and ad hoc, often taking a long time to refine rules before yielding any successful results. Essentially, a rule is only formed from a specific combination of local spatial and temporal parameters which somehow give rise to global emergent behaviours. Therefore, there is a chance a successful ruleset can be formed by the random combination of some of these local parameters. The total number of possible combinations mean that there is an incredibly rich search space in which to draw potential solutions [17]. Automating this search to explore every single variation (a brute-force search) is a huge waste of time (it may even take longer than a lifetime to complete the search [21]) and a waste of resources since only a very small percentage of such combinations would produce rules that lead to any meaningful behaviour.

However, there are algorithms designed to efficiently and intelligently find near optimal solutions within very vast search spaces [21]. Many of these artificial intelligent algorithms are inspired by nature, (such as Genetic Algorithms (GAs) which is directly inspired by biological evolution [38]). GAs can be used to 'evolve' solutions (i.e. simple reactive rules which define individual robot behaviours that cause the swarm to converge toward a specific emergent behaviour) [21]. GAs were used to evolve the GKL rule which was capable of emergent computation [40] and GAs were used extensively by Mitchell and Crutchfield at Santa Fe Institute for their research into the computational mechanisms of emergent phenomena [25].

"Automatic design methods (like GAs) can be considered top-down methods because, in theory, the process is driven by the global goal" [8] - i.e. the desired holistic-characteristics of the entire swarm [8]. The perspective is shifted away from the individual to the higher-level of the collective [8]. Unlike Bottom-up approaches (i.e. 'Behaviour-based approaches) which focus on designing at the level of the local rule, iteratively refining it based on careful observations of the global effects they produce when thoroughly tested [8]. In direct contrast, top-down approaches (a.k.a. Phenomena-based approaches [62]) focus only on the big picture, designing at the global-level, some desired model of swarm behaviour, which is then used to guide and direct a quick, automatic search through a sea of potential rules until some are found which can produce the desired global properties.

(IV.1.5) Advantages of the Evolutionary-based approach

Like many intelligent search methods, GAs often find "surprisingly complex and interesting" [18] solutions. This is the advantage of using a method which is objective, unbiased toward any form of simplistic, deterministic logic. Rather, they search past the self-imposed limits of simplicity and logic; freely exploring in any direction to evolve solutions that are creative and counterintuitive. Furthermore, an evolutionary process would allow for adaptive rules which evolve even during run-time (especially beneficial for swarms in dynamic environments). "No matter how good a set of rules seems to be at time T, there is always a chance of unwanted behaviour emerging at time T + dt" [58].

(IV.1.6) Limitations of Evolutionary-based approach

Although it sounds appealing to evolve rules automatically using a genetic algorithm, the process is computationally intensive and, due to the nature of the design method, very unpredictable. "When the evolution mechanism throws its blind

operators, there is almost nothing we can tell in advance" [56]. It can be interesting to watch the development of rules as they evolve - the fitness of the evolved solutions often tend to "increase in rapid jumps...epochs...each corresponding to the discovery of a new, significantly improved strategy" [38]. However, this unpredictability also means there are no guarantees that a rule will ever be evolved (i.e. the search algorithm may never successfully converge onto an acceptable solution) [8].

Furthermore, the automated search process "cannot be applied blindly and effortlessly" [50]. Designing the artificially intelligent algorithm well and fine-tuning its parameters determines how quickly, efficiently and optimally the solutions are evolved. "It is not sufficient to start somewhere completely random and hope to evolve a solution somewhere in phase space" [21].

(IV.1.7) Design Parameters for the Evolutionary-based approach:

We shall now look in more detail at the various design parameters required for the Genetic Algorithm, which include: (a) Representational methods (Genotype and Phenotype), (b) Selection Pressures (Fitness functions), (c) Exploitation (Crossover) and Exploration (Mutation) of the search space, (d) Memory to allow the influence of past solutions (inheritance).

(IV.1.7.A) Representational methods (Genotype & Phenotype)

Before the automatic search can search for potential solutions (in our case, a solution refers to a simple rule-set which dictate how individual robots react to their surroundings; including other robots, obstacles, goals, etc), the solutions must be represented in a way that the genetic algorithm can easily decode and evaluate (i.e. test how well the solution fulfils the algorithm's search criteria in order to rate it with a fitness value).

A very common example of a representational method is the 'bit string' [38] or 'binary string' [49] (the candidate solution encoded in binary - a single line of 0s and 1s), the length of which may vary depending on the size of the solution being represented (e.g. 8 bits can represent a number in the range -10 to +10. 128 bits can represent a single transition rule for a 1D binary state CA of radius 3 [49]). Hexadecimal may be used instead of binary to allow for shorter string lengths [18]. Alternatively, the encoding can be completely specified by the designer, such that the genotype may consist of "different letters of the alphabet (e.g. A,B,C,D)" [18]. The template representation [49] is an example of a user-defined representation to suit the solution being encoded. A unique feature is the inclusion of a special character "#" to represent "any option" (i.e. if the string is binary, then # would mean 0 or 1). This is a makes the representational method far more expressive by cleverly allowing for generalisations (e.g. the string [0,#,#] could mean any of the following: [0,0,0], [0,0,1], [0,1,0] or [0,1,1]). The template representation was found to "produce superior performance to the bit string traditionally used for representing Cellular Automata [49].

The representational method used to encode the solution's meaning can negatively affect the evolutionary search (i.e. add a bias) just as it can positively affect it (i.e. make it more efficient). Thus careful consideration needs to be taken when deciding on the representational method in which to encode the local rules. For example, "a binary string representation is not able to exploit symmetries inherent in the" [49] system, which would aid the genetic search. This is because "the left and right-hand neighbourhoods

cannot be treated in the same way by the crossover operator" [49] when using the binary string representation, and thus the algorithm would search in favour of the left or right-hand side of the neighbourhoods, making it biased toward finding transitions of a particular state. The representational method is a key part of improving the efficiency of the search.

(IV.1.7.B) Selection pressures (Fitness Functions):

A 'chromosome' or 'genotype' refers to the encoded solution which the genetic algorithm can search and evolve [59]. The 'phenotype' refers to the coding method which dictates how the chosen representation (directly or indirectly) maps to the real solution (i.e. the rule set used by the robot) [18]. It is used by the genetic algorithm during its search to translate encoded solutions into their corresponding behaviours so that evolving solutions can be evaluated (i.e. to know if the solutions are getting closer to the desired emergent behaviour). Every candidate solution that has been evaluated will carry the results with it to easily compare and rank it against other candidates. This rank is better known as the 'fitness' of the solution, and thus the evaluation method to calculate this fitness is known as the 'fitness function' [38].

Along with selecting a good representation, selecting an appropriate fitness function greatly influences the success of the evolutionary process [18]. There are many different types of fitness functions:

- Functional (i.e. numerically calculated using a formula),
- Behavioural (i.e. judged on the pattern of behaviour),
- Internally judged (i.e. a fitness which can be judged at a local level by the individual robot and its sensors - such as its acceleration - i.e. to promote individuals travelling at a constant speed the fitness could be inversely related to the acceleration. Alternatively the change in acceleration, or 'jounce', could be used to indicate how 'shaky' an individual's behaviour is and thus promote smoother dynamics by making the fitness function inversely related to jounce. This type of fitness function is also good for heterogeneous swarms where each individual may have different roles and be running a different genome from one another)
- Externally judged (i.e. a fitness judged at a global level of the entire swarm which must therefore be judged by an external observer - i.e. an overhead camera [17] - rather than by the individuals. Usually these are global properties or behaviour [59] such as the system's final configuration [38, 49] or the absolute position of all robots [17]. If it is a simulation, the fitness function could be applied directly to the swarm, rather than the individual, to evaluate the fitness of the collective behaviour [8]. Providing the swarm is homogenous and individuals are all running the same genome, this swarm-level fitness would actually be the fitness for that genome [17]),
- Explicit (i.e. directly tested on an individual or swarm placed in a test environment [18])
- Implicit (i.e. evolving "through guidance rather than control" [50]. It can "relax the constraint that an explicit fitness function must return a numerical evaluation of each agent" [18] and "is much closer to the way fitness is assessed in natural settings" [18]. Implicit evaluations are useful if the desired goals are changing, or the landscape and other selection pressures are varying [21]. This is also often the case for Co-evolution - whereby multiple swarms are being evolved

together in a 'predator-prey' type fashion - the implicit fitness could simply be survival [56] - life or death [21]. The fitness may also be based on maximising its energy by killing other prey for their energy without moving too often to reduce energy loss [56]. Individuals may "acquire resources by interacting with others…through trade and combat" [18])

- Extrinsic selection pressures (i.e. the distance from an external goal, the setup of the environment, obstacle collisions, the accumulated number of rewards and penalties collected - if an action brings the robot closer to its objective it may be rewarded but if an action results in damage or increased risk of damage to the robot then it receives a penalty [19])
- Intrinsic selection pressures (e.g. task-independent evolution [50] such as sensor-motor coordination, or how well an individual's behaviours correlate with the dynamics of the swarm, perhaps to encourage coherent motion [52]. If such intrinsic pressures were to be applied at the embedded layer of the computational mechanics rather than the individuals' rule sets, the algorithm would encourage the evolution of spatio-temporal structures like 'gliders' in 2D CAs [52]).

(IV.1.7.C) Exploitation (Crossover) and Exploration (Mutation) of the search space:

To reduce the computational burden and increase efficiency, a genetic algorithm is not immediately presented with a complete solution set containing all possible variations that exists. Rather, candidate solutions are sampled from the search space. The initial generation consists of randomly sampled solutions, which the genetic algorithm uses to evolves the next generation of candidate solutions (i.e. modifies them using specific evolutionary search techniques). This allows the algorithm to intelligently navigate through the search space by exploiting the current best solution [38]. Two techniques are used by the Genetic Algorithm to modify a candidate at each generation: crossover and mutation [38, 18].

(IV.1.7.C.a) Crossover

Crossover involves a pair of (parent) candidate solutions exchanging genetic material (i.e. mating / sexually reproducing) [18] by stitching together pieces of their chromosomes to form a new, unique (offspring [56]) candidate solution. (They can swap every alternate gene, or simply swap one part of their chromosomes (split at a single-point randomly selected along the chromosomes) [49]). Occasionally they will combine the best genes of both parents [21] and produce fitter candidate solutions (thus driving the evolution).

Pairing off candidates for crossover need not require an even sample size, as individual candidates may mate with multiple partners (or multiple copies being made of fitter individuals), both of which have the same effect of promoting the next generation of offspring to have fitter features [18]. Alternatively, the problem can be overcome by allowing the last odd candidate to reproduce asexually [56].

"Once all individuals in the population have been evaluated (and ranked), their fitnesses are used as a basis for selection" [18]. By allowing fitter candidates to breed more often, the unfit candidates will die out thus slowly weeding out bad solutions.

(IV.1.7.C.b) Mutation

"Repeated reproduction with similar genetic information increases genetic homogeneity" [52] which can lead to the algorithm getting stuck at a local minima in the search space [21]. To avoid this and encourage exploration of the entire search space, a random element is introduced to add some variety back into the gene pool. Parts of the offspring's chromosome are randomly changed to a different value in a process known as mutation [38].

(IV.1.7.D) Memory to allow the influence of past solutions (Inheritance):

A new generation of candidates can occur all at once, or alternatively be a continually ongoing process (i.e whenever a candidate solution dies off, a new candidate replaces it [56]). Thus far, the new generation of candidate solutions contain only offspring of the prior generation [56]. However, there remains a small possibility that the offspring will not be as good as their parents since only the worst parts of both parents were combined. Thus, elitist inheritance allows the fittest candidates from the previous generation to survive on to (be copied into) the next generation along with all the offspring [38, 18, 49]. Inheritance can also be thought of as a way for the genetic algorithm to remember best candidate solutions searched in the past and thus acts like a memory to help improve the efficiency of the search.

Using small amounts of memory can also improve the efficiency of the search [49] (i.e. during the inheritance stage, maintaining some of the solutions already explored in past generations has been shown to aid the genetic search process more than exploring completely new directions at each generation and using no memory). Using an excessive memory, however, can actually inhibit the evolutionary process and so "selecting an appropriate amount of memory is thus important for effective problem solving" [49].

(IV.1.8) General Structure of the Genetic Algorithm

In summary, the general structure of a genetic algorithm will comprise of an initial set of candidate solutions encoded into genomes. At each generation, the genomes are decoded via the genetic algorithm one by one so that the fitness of the solutions they represent can be determined. Genomes will then be paired off to reproduce via crossover and mutation (the fitter candidates being given higher preference in mating to drive the evolutionary process toward the goal set by the fitness function). The offspring will comprise the next generation of candidate solutions along with a small percentage of the fittest parents from the previous generation to ensure that each subsequent generation of candidate solutions is either remaining the same fitness on average, or becoming fitter. This process is iterated until a generation evolves to meet an acceptable level of fitness. Although the genetic algorithm can find a desired solution fairly efficiently if a suitable representational method and fitness function is chosen, as well as properly tuning the various parameters (e.g. population size, probability of mutation, etc) and selection pressures, there is still no way to know for sure how long the evolutionary process will take nor is there any guarantee that it will ever successfully evolve an acceptable solution.

(2) Testing Methodologies

(IV.2.1) Real Robot Implementation

If we opt to experiment using real robotic swarm then there are no shortage of pre-built models of miniature mobile-robots to choose from (e.g. Colias, AMiR, Alice, Jasmine, E-puck, Kobot, Kilobot, R-one, SwarmBot, etc [19]). Testing using real robotic swarms is advantageous for considering factors such as the influence of noise on the system [34], which, in turn, serves to increase the accuracy of any spatio-temporal analyses. However, the added interference may do nothing more than obscure the underlying computational mechanisms which we are seeking. This is because, "when in a disordered state, a (swarm's) behaviour is deeply influenced by the result of random (individual-level) actions" [17]. On the other hand, if the swarm's global behaviour exhibits emergent behaviour, the same individual-level randomness no longer influences the swarm's global behaviour so significantly [61]. This is why those investigating computational mechanisms of emergent phenomena occurring in chemical reactions prefer to use a real, chemical medium (i.e. a thin-layer film or gel) [3, 10]. Though they too report that the computations occur at the "edge of instability" [3] wherein the medium is "highly sensitive to local perturbations and even a tiny local change… lead to drastic modification of the medium's dynamics" [3].

Nevertheless, if we desired to investigate the influence of noise on a swarm's behaviour, the easiest option is to use real robotic swarms. Attempting to simulate this noisy randomness is complex, especially as the number of robots increase [6], and thus the results will only ever be as accurate as the model representing the external environmental conditions (e.g. the model must take into account the effects of wind and frictional forces [47], uneven terrain surfaces [34] - all of which affect the motion of individual robots - as well as signal delays and fluctuations in robotic sensor readings - which also impacts on the way robots perceive and react to changes in the environment [34]). Furthermore, we would then need to consider and model the effects of internal noise (i.e. motor vibrations or active brownian motion effects which change the orientation of robots over time [34]).

(IV.2.2) Computer Simulations

The simplicity offered by computer simulated models can be advantageous; "realism is not necessarily needed or helpful" [62]. It is not necessary to depict every detail of the robots in the robotic swarm (i.e. by realistically rendering the swarm in High Definition, 3D), rather, a basic particle swarm representation is more than sufficient to models the swarm's useful information. Decluttering unnecessary information is something computer simulations can offer which real-robotic swarms cannot. The unique opportunity to model the robot swarm as simply as possible (without removing the important properties of the phenomena being modelled) allows us greater clarity. Simplification allows us to focus on features which truly concern us and can lead to greater insights. Thus, what computer simulations lack in realism and detail, they more than make up for in usefulness [62].

Simulating a robotic swarm has many other key advantages over real robot implementation [60, 8]. The absence of hardware issues means the experimenter need not consider as many safety risks [8] nor waste time ensuring the maneuverability, endurance, flexibility, reliability, computational power and size of the robots (i.e. the smaller the robot the better, yet it needs to be large enough to carry the appropriate sensor equipment and power supplies, etc). The cost of scaling up the number of robots is also not a main concern for multi-robot simulators [8]

(increasing the number of robots is a key parameter to investigate during experimentation since a swarm can behave very differently at different sizes).

The stochastic element in real world experiments require that they be run multiple times at each setting to reveal any patterns, however, repeating the same experiment in simulations is not problematic since the temporal element can also be manipulated (i.e. paused, sped up, etc) to allow for a quick overview a swarm's developmental patterns, or the more careful observation of key points during experimentation [21, 60]. Since noise is virtually non-existent (unless programmed in) a simulated swarm's internal parameters and external environmental conditions can be set-up, controlled, adjusted and fine-tuned very precisely [45] (i.e. the exact same experimental conditions can be recreated to repeat an interesting phenomenon observed and recorded - impossible with real world experiments). The model can also be changed ever so slightly (at the stroke of a few keys) to explore alternative versions [60]. Also the swarm's emergent behaviour eventually needs to be analysed for spatio-temporal structures and such analyses will naturally favour computer simulations [45] (since a real robotic swarm will need to be recorded via an overhead camera to be processed via a computational analysis).

(3) Analysis Methodologies

(IV.6.1) Spatio-Temporal Analysis

In this section we will look at methods to analyse the simulated swarm across a given space and time to try to detect any stable, consistent spatio-temporal structures. The analytical software will have a split-view to represent both the surface-level behaviour of the actual swarm beside a real-time analysis of the swarm's embedded spatio-temporal behaviour (i.e. the development and spatial movement of stable computational structures over time). Thus different methodologies to represent this will be discussed next.

(IV.6.2) Particle swarm Representation

The most straight-forward way of modelling a robot swarm in a computer simulation is to represent each robot as a single point (a particle) which can move around on a 2D landscape. This allows us to clearly see the movements of each robot and focus on the development of the swarm's local individual behaviour as well as the swarm's collective global behaviour (which is useful if we wish to observe the development of any emergent features of the swarm). Furthermore, by representing the robots as simple point particles, we minimise the computational burden [60] to allow resources to be directed to more important areas of the simulation (i.e. the execution speed, etc) [21]. Although the particle swarm representation is useful for observing the development of the robotic movements of the swarm, it is not the best representation to use to highlight the development of internal robotic communications or interactions.

(IV.6.3) Dynamic Network Representation

The great number of interacting individuals in a robotic swarm can be viewed as a dynamic communication network [48, 8], (each simple robot acts as a mobile node that become spatially and/or temporally interconnected via short-range communications) and thus it is not uncommon to refer to robotic swarms as distributed, robotic sensor networks [46, 48] or mobile, (parallel) computer networks wherein robots act as embedded sensing and processing elements in the environment [41, 46, 48]. A swarm is a 'dynamic' network, as opposed to a 'static' one, because the individual robots (mobile nodes) have no fixed position (absolute or relative to one another) and thus their interactions continually change along with

the neighbours encountered (i.e. network links are continually formed and broken) [39]. Just as the state of a network node may change, so too may the current 'state' of a robot change (e.g. individuals in a robotic swarm inspired by the emergent clustering behaviour of ants can be in a state of 'depositing' or 'taking' [9], whereas individuals in another robotic swarm designed for exploration can be in the states 'beacon' or 'explorer' [48], etc). Unlike most networks, however, a network formed by a robotic swarm will continually change shape and size (i.e. expand) due to the relative changes in position of each mobile node (robot). Nodal links within a network formed by a robotic swarm can be created explicit (spatial) and implicit (temporal) interactions [46, 51, 8].

Explicit interactions refer to direct actions or interactions between individuals [29, 51, 8] (i.e. robot-robot interactions [29, 59]) including direct communication via close-range sensors within a local neighbourhood [41] (e.g. shining light [34], coloured LED displays, infrared, audio and acoustic signalling, coil induction, radio-frequency broadcasted messages, body-language, sign-language, coloured patterns on robots [46], robot recognition [6], and other indirect clues such as the perceived density of the robot population or net force of robots on an object [8], etc). Implicit interactions, also known as 'robot-environment' interactions [29, 59, 8] or 'stigmergic' communications [9, 46, 59], refer to indirect links formed after a short time delay from the robot's initial signal. Since the external environment can also act as a stimulus to affect the behaviour of individuals [52], the environment can be manipulated by structurally modifying it [8] (i.e. changing its shape, temperature gradient [8], etc) removing or adding material [22] (i.e. ant-inspired chemical pheromone trails [58, 34, 45, 41, 8], etc), anything to leave a trace of an individual's event for communicating with other individuals at a later time (an indirect interaction). For instance, termites create terrain configurations that stimulate other termites who encounter it to add more building material [22]. By environmentally encoding events in this way, communication signals can be temporally frozen at that location to eventually influence other robots. (Hence stigmergy can not occur in environments which cannot be changed by individuals [22] because it is too stiff or too empty, like space, nor can it occur in environments which are too dynamic, like air and water, because the terrain needs to remain modified long enough for the stored event to affect another individual's behaviour [22]). In other words, the environmentally embedded event creates an open-ended link that attaches to other mobile nodes occupying that same space at different times (highlighting the advantage of 'dynamic' networks over conventional, static networks unable to change their size, layout or nodal positions and thus unable to create stigmergic network links). Essentially, the environment is being exploited by simple individuals (with no internal memories) as a shared, external memory [22, 46] that allows the robots (mobile nodes) to extend their sensory capabilities [49] and indirectly increase the range of their nodal links to beyond the local neighbourhood. Stigmergy cleverly converts temporal data into spatial information (as the time delay extends the spatial range of the link connecting mobile nodes) and therefore also has the potential to compress spatial information into temporal data [49].

It is worth noting that implicit interactions (indirect links) are not the reason complex systems like swarms (dynamic networks) can communicate large distances and self-coordinate to allow globally intelligent behaviour to emerge [38]. This is proven by the fact that systems without stigmergic links (i.e. cells that only interact explicitly with their immediate neighbours - direct nodal links) still exhibit emergent behaviour. However, implicit interactions (indirect nodal links) will allow for a more complex network structure, giving the swarm (dynamic network) greater flexibility and greater potential to compute more sophisticated emergent behaviours [59].

(IV.6.4) Static Network Representation

Thus, instead of modelling the swarm as a dynamic network (whereby each node moves around to represent the robot positions in the environment) we model it as a static network (whereby each network node still represents an individual robot, but remains fixed in place while the network links still change formation according to the swarm). This representation allows us to focus solely on the inter-robot interactions (direct and indirect communications) by ignoring any irrelevant surface-layer behaviour (i.e. changes in robot positions). The clearest position for the fixed network nodes to be places is a circle, so that every node can be linked with one another via the middle of the circle which makes it clearer to see each link.

(IV.6.5) Converting 2D to 1D

It is also possible to represent the swarm as a 1D row of static nodes, similar to the layout of a 1D cellular automaton. This would make the task of tracking and representing the swarm's temporal developments far easier. The new row of nodes (representing the updated swarm) would be stacked above the old row. The problem with simplifying a swarm this far is that it may flatten out important information related to the emergent phenomenon we are investigating. Features such as "mobility, social interactions (negotiation, competition and collaboration) and a high number of entities interacting in a networked environment" make the swarm too complex to be modelled as a 1D Cellular Automata or alternative numerical simulations and mathematical models which are too simple (and only serve to describe average trends and patterns across the swarm - which may not accurately describe the swarm's emergent behaviour, especially if it is multimodal) [45, 60, 49]. Furthermore, 1D Cellular automata assume all cells to be homogenous, whereas a swarm can be heterogenous (different robots performing different local behaviours) [60].

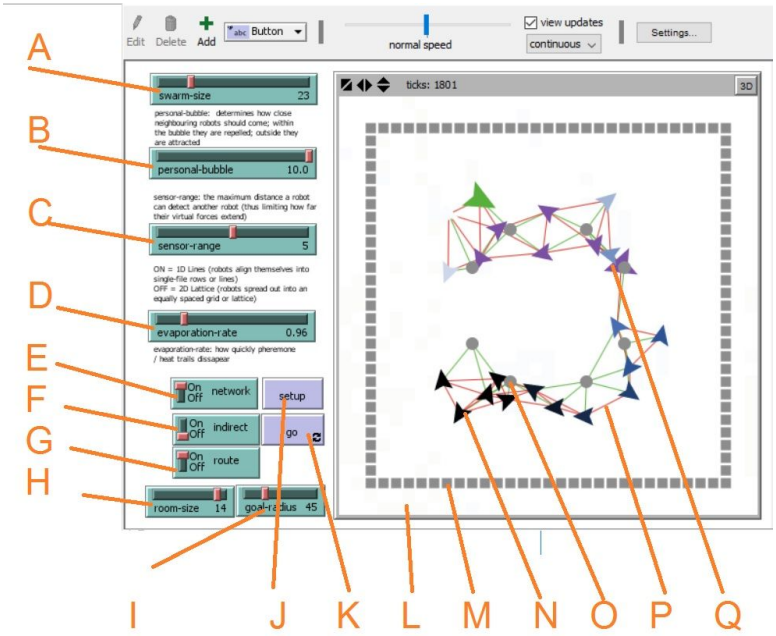
(IV.6.6) Converting 2D to 3D

Rather than updating the previous swarm's behaviour at each tick, the developments need to somehow be stacked so that the changes can be compared and any consistent spatio-temporal structures can be identified. When the temporal element is added to the spatial analysis (i.e. analysing the development of the swarm's behaviour over time) an extra dimension is required to represent this change over time. "In the case of 1D CAs, the space-time diagram is a 2D surface. However, in the case of 2D CAs, the space-time diagram is a 3D surface" [38]. Since a particle swarm representation or a network representation are both 2D representations, the added dimension required for the temporal element of the spatio-temporal analysis will have to be represented via a 3D representation. This is possible in a computer simulation by cleverly manipulating the perspectives.

(4) Final Design

(IV.4.1) Virtual-Forces Swarm

The final interface designed to simulate the robotic swarm (operating on the virtual forces model) is shown below:



"A" shows the "swarm-size" variable which defines how many robots the simulated swarm consists of, and the user can tune this value to any integer between 1 and 100.

"B" shows the "personal-bubble" variable. If a neighbouring robot is closer than the "personal-bubble" radius, then it is repelled by the virtual forces which both robots emit. If the neighbouring robot is further than the "personal-bubble" radius, then the virtual forces are attractive instead. Essentially this value determines how close neighbouring robots can come to one another. This variable can be tuned by the user to any integer between 1 and 10.

"C" shows the "sensor-range" variable which determines the maximum distance a robot can detect another robot, thus defining each robot's neighbourhood of influence (i.e. the reach of their their virtual forces). This variable can be tuned by the user to any integer between 1 and 10.

"D" shows the "evaporation-rate" variable which determines how long the robots' stigmergic (i.e. chemical pheromone or heat) trails remain in the environment. This variable can be tuned by the user to any number between 0 and 5.

"E" shows the boolean variable "network" which allows the user to choose between viewing the simulation as an actual robotic swarm ("network = off") or as a dynamic network ("network = on") with the network links revealed. "F" shows the boolean variable "indirect" which allows the user to let robots leave an indirect stigmergic (i.e. chemical pheromone / heat) trail in the environment ("indirect = on"). "G" shows the boolean variable "route" which tells the robotic swarm to find the shortest route between two given points and then display this emergent route using specific colours ("route = on").

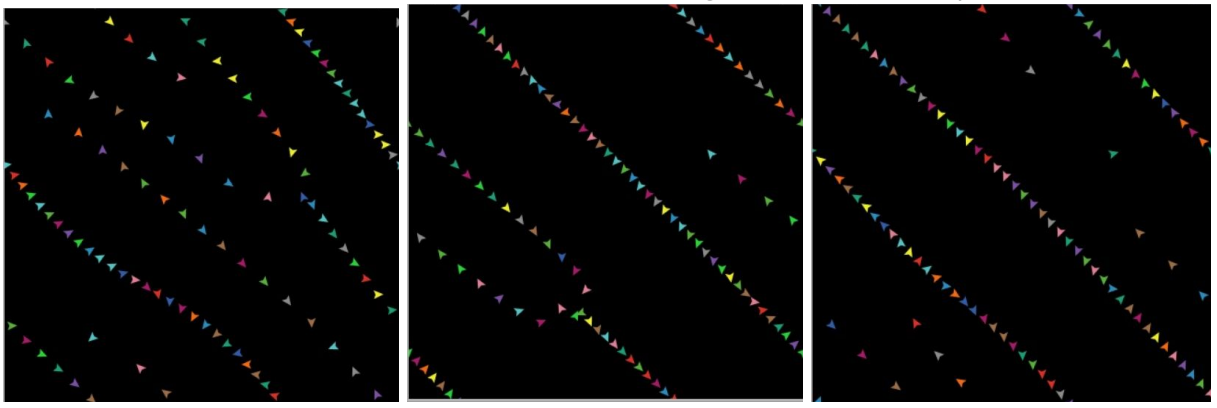
"H" shows the "room-size" variable which determines the width of the walls (i.e. the obstacles placed in the environment in a square formation). The user can tune this variable to any integer between 0 (indicating there should be no obstacles) and 16 (the maximum width of the environment). "I" shows the variable "goal-radius" which determines the radius of the goals (which are placed in a circle formation from the centre of the environment). This variable can be tuned by the user to any integer between 0 (indicating there should be no goals) and 150.

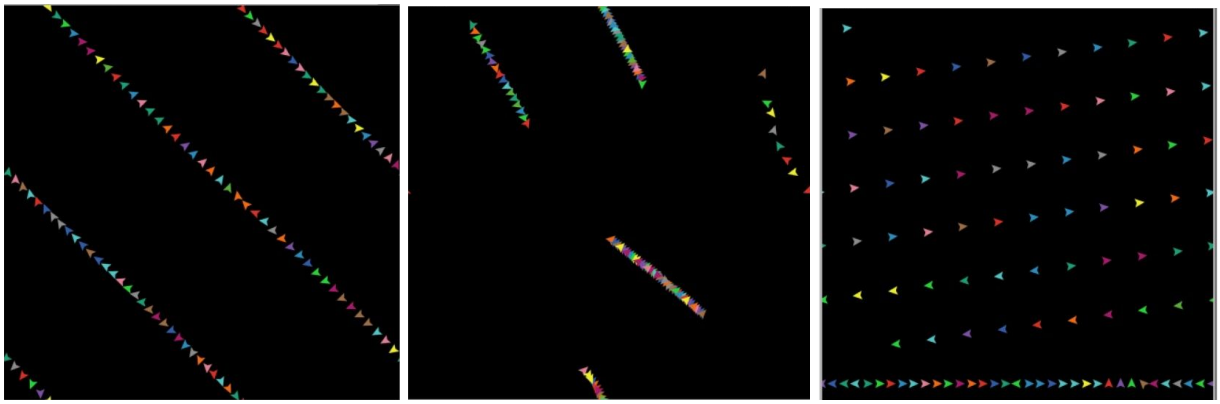
"J" shows the function "setup" which resets all the variables and initialises the world, agents and their internal models. This function must be run before selecting the "go" function for the first time. "K" shows the function "go" which continually executes the program's main algorithm (i.e. the control loop). The program can be paused at any time by unselecting "go". The control loop continually repeats the following three steps; 1) simultaneously allowing each robots to observe and measure their local surroundings (i.e. read the distances obtained from their sensors), 2) update their internal model based on these observations and measurements (i.e. calculate the virtual-forces from those distances), 3) and then react based on their internal models (i.e. update their motion to coincide with the direction and magnitude of the net virtual force experienced).

"L" shows the graphical display window of the program which displays the simulated robotic swarm. "M" shows an obstacle which indicate locations which robots must avoid. "N" shows a robot (a random colour initially, but will change to purple to display the best route) . "O" shows a goal which indicate locations that robots must move toward. "P" shows a network link (red links represents repulsive virtual forces while green links represent attractive virtual forces). "Q" shows the robot which has been selected to be the start or end point of the best route (it is indistinguishable from other robots but has been enlarged on the simulation for clarity).

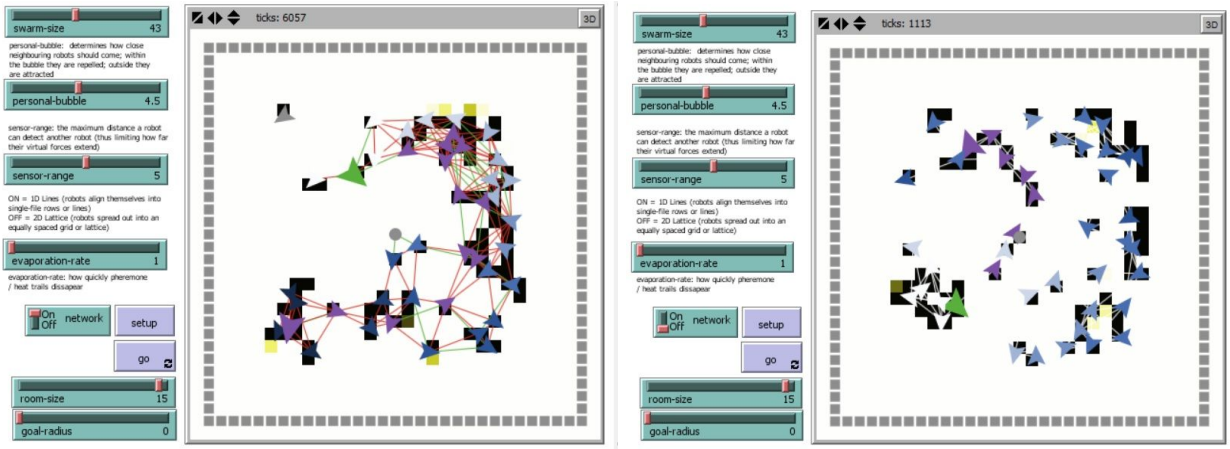
The swarm is capable of two types of emergent behaviours; one external and one internal.

The external emergent behaviour involves its motion. When first deployed, the robots move chaotically and react to the virtual forces they experience locally. The swarm is very dynamic at this stage and able to easily flow around obstacles (its state is analogous to a fluid). Eventually, in a static environment, the swarm comes to a rest in certain formations in equilibrium wherein all the virtual forces on each robot are equally balanced and the net force is near zero, hence the robots no longer move. The stable formations are either straight lines (this state is analogous to a solid polymer) or regular lattices (this state is analogous to a solid crystal). The time taken for the robots to come to rest and solidify into a stable formation is dependent on the number of robots in the swarm ("swarm-size") and the maximum observable radius of the robots' sensors ("sensor-range") as well as their "personal-bubble".



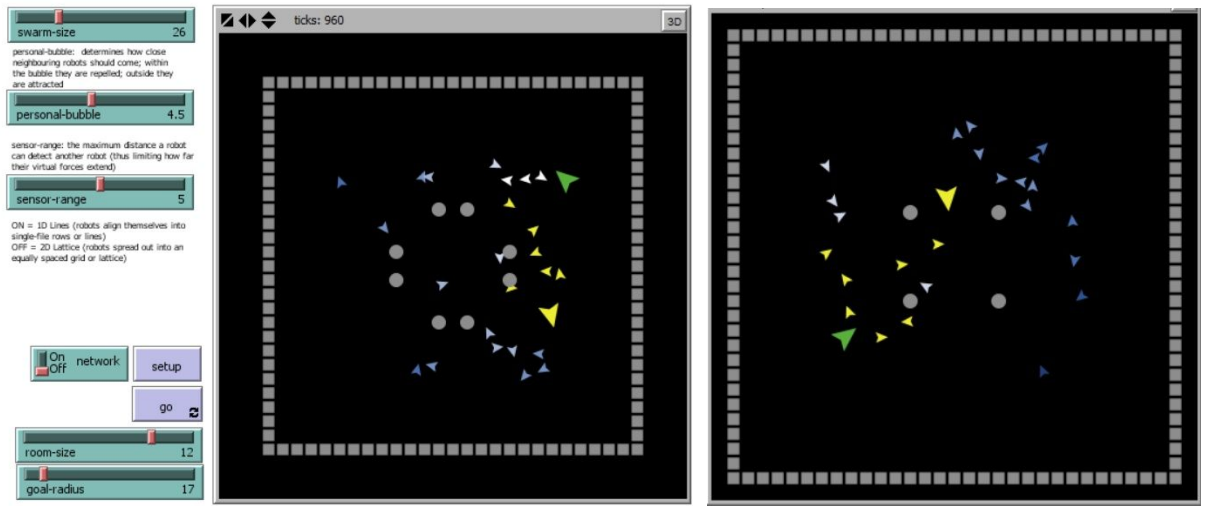


Robots also have the option to leave a chemical pheromone or heat trail in the environment which allows them to act upon other robots stigmergically over longer periods of time (via environmental traces) and thus indirectly extend the reach of their interactions to other robots in the swarm (encouraging a more interconnected swarm).

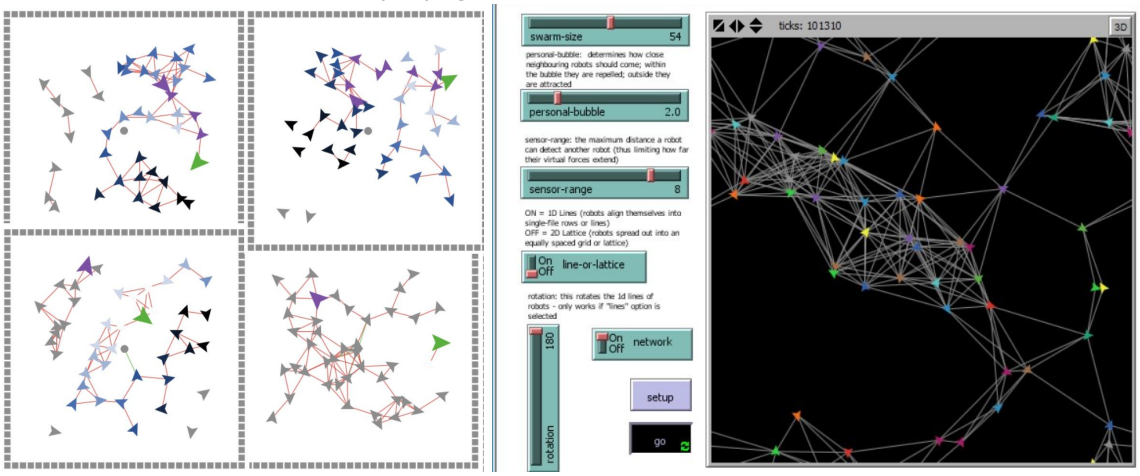


It was noted that when this option is selected, the swarm resists solidifying as quickly and remains in a semi-fluid state. It also induced a liquid state even after the swarm has solidified into a stable formation, and thus can potentially be used to liquify the swarm and get it moving again.

The second, 'internal' emergent behaviour that the swarm is capable of is not related to the swarm's motion, rather, to finding a 'best route' between any two points. Once the swarm has solidified into a stable lattice spread across the environment, it can act as an external map of the terrain and display routes across the terrain. Two robots on certain points of the terrain are selected and the best route between them emerges via a simple propagation among the robots via a manipulation of their internal states. Once the best route between the two points has emerged, the robots along that route will light up (i.e. yellow) to display that best route.

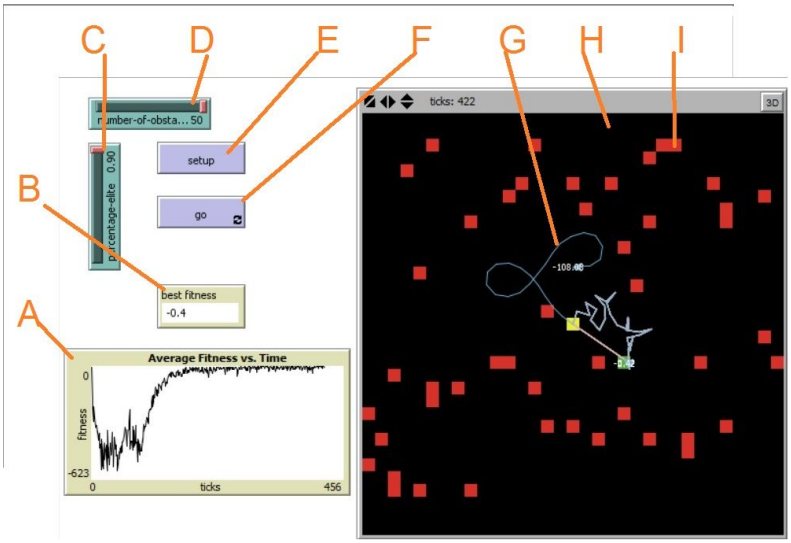


To better understand how interconnected a swarm is, it can be viewed as a dynamic network instead, wherein each mobile node represents a robot and each network link represents a robot interaction. Using the dynamic network view, it is clear which robots are not actively connected to the swarm and it is also easier to see the channels which robots propagate information in order to find the best route.



(IV.4.2) Genetic Algorithm

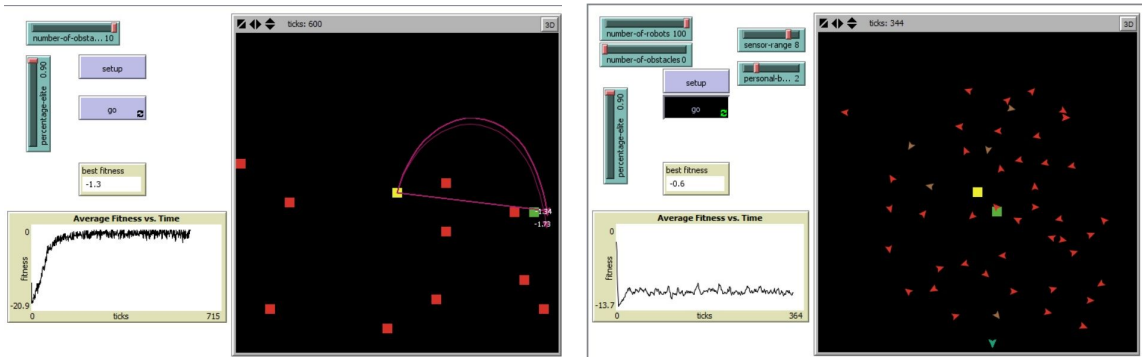
The final interface for the genetic algorithm (used to evolve emergent behaviour) is shown:



"A" shows the average fitness of the candidate solutions being evolved over time and gives the user an indication as to the genetic algorithm's current

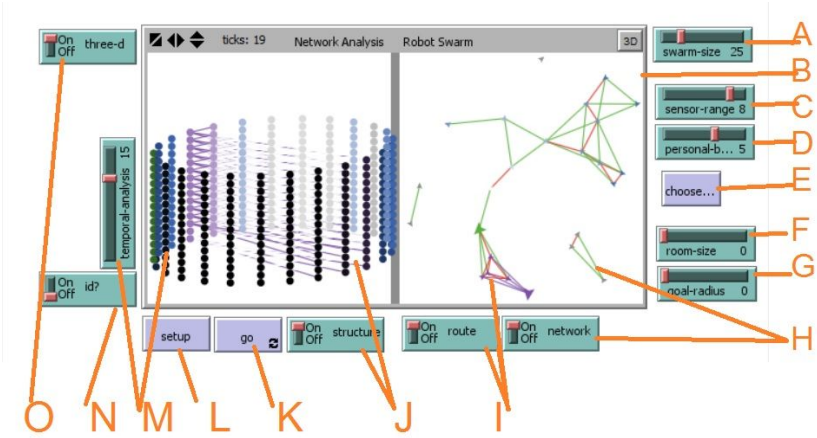
level of progress. "B" displays the fitness value of the current best candidate solution. "C" shows the variable "percentage-elite" which determines what bottom percentage of the current candidate solutions (in terms of fitness) will be discarded to leave the remaining candidates as potential parents in an 'elite' gene pool that will be selected from for reproduction (via crossover and mutation) until a the desired amount of candidate solutions for the next generation has been reached. This value can be tuned by the user to any number between 0.1 (10%) and 0.9 (90%). "D" shows the variable "number-of-obstacles" and can be tuned by the user to any integer between 0 and 50. The goal is kept as 1 (this is using a fitness function based on the robot's final distance from a randomly placed goal). "E" shows the "setup" function (which initiates the world, agents and internal models) and "F" shows the "go" function (the program's main control loop). G shows the route of one of the evolved candidate solutions. "H" shows the graphical display used to play out the evolving candidate solutions. "I" shows an obstacle placed randomly in the environment.

A number of swarms were evolved by the genetic algorithm using various fitness functions. 'Explicit' fitness functions (i.e. those based on how close the robots came to a specified location / goal) evolved robots which were able to reach specified locations. When combined with 'implicit' fitness functions (i.e. instantly killing off robots that are involved in a collision), the algorithm evolved robots which could reach a goal while simultaneously avoiding other robots and obstacles along the way (see image on left). However, when a purely 'implicit' function was used, robots would avoid colliding and moved together as a group in a coherent manner (ie. the swarm exhibited an emergent behaviour) although the swarm as a whole would not move in any specified direction or toward a specific location (see right screenshot below).



(IV.4.3) Analysis software Version 1: 3D static network analysis of virtual forces model

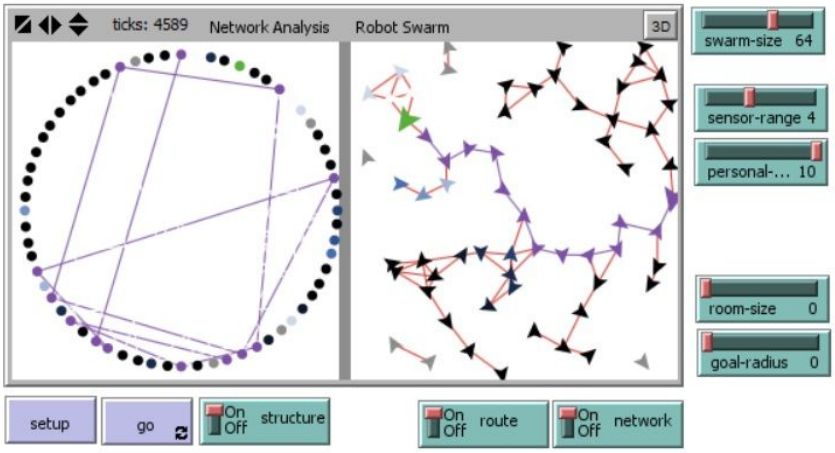
The first version to the interface for the spatio-temporal analysis:



Some of the features are the same as those described in the interface for the virtual-forces model above (section IV.5.1): e.g. “A” shows the user-defined variable “swarm-size”. “C” shows the user-defined variable “sensor-range”. “D” shows the user-defined variable “personal-bubble”. “F” shows the user-defined variable “room-size”. “G” shows the user-defined variable “goal-radius”. “H” shows the boolean variable “network”. “I” shows the boolean variable “route”. “K” shows the function “go”. “L” shows the function “setup”.

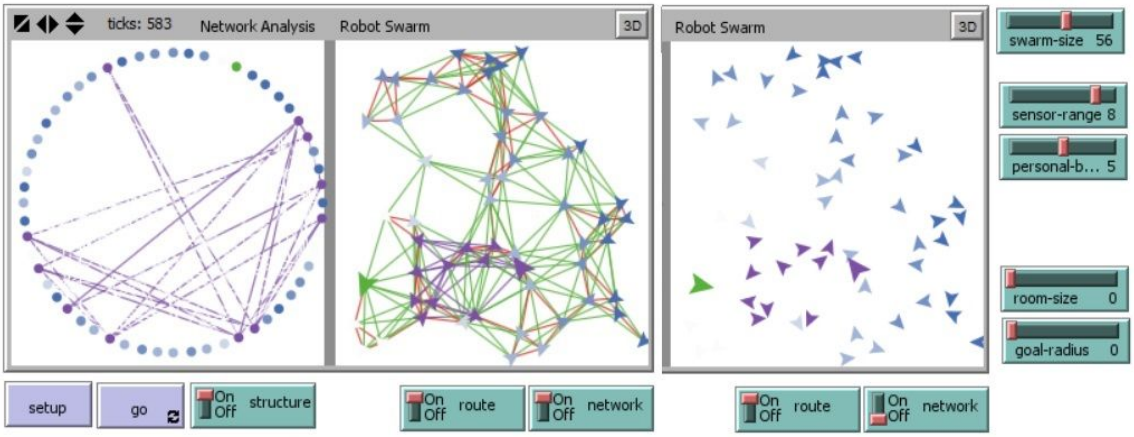
Other features are unique to the analysis software, such as: “B” which shows the graphical display window which has been modified to look like a ‘split-screen’. The robotic swarm is displayed on the right and the 3d static network representing the robotic swarm (used for spatio-temporal analysis) is displayed on the left. “E” shows the function “choose-bots” which, whenever pressed, randomly selects two new robots in the swarm to be the start and end point for the emergent best route. “J” shows the boolean variable “structure” which allows the user to see all the network links displayed on the 3d static network (“structure = off”) or have the majority of the network links hidden and only display those related to the emergent best route (“structure = on”). “M” shows the variable “temporal-analysis” which determines how many past versions of the 3d static network to keep for temporal analysis. This can be tuned by the user to any integer between 1 and 20. “N” shows the boolean variable “id?” which was used for testing to reveal the turtle numbers of each network node for debugging (“id? = on”). “O” shows the boolean variable “three-d” which lets the user switch the static network representation of the swarm from a flat 2d circle (“three-d = off”) to a 3d circle, so that past versions of the network can be stacked on top of one another to more easily identify any changes in the network over time.

In many ways, this software is the most important of the three since it directly addresses the original aim of the project; to discover hypothetical spatio-temporal computational mechanisms embedded in the emergent behaviour of robot swarms. Whereas the other two softwares are a means to facilitate this software, serving as test swarms with which to analyse using this final software. There are far too many parameters to analyse the behavioural developments of the robotic swarm directly from its simulated swarm representation. Hence the software incorporates a second window (split-screen) to view a simplified static network representation of the swarm which will be analysed. A network representation has the benefit of clearly revealing the individual robot interactions of the swarm, which is where the spatio-temporal structures are believed to be found (particularly for the internal emergent behaviour of how the swarm finds the best route between two points).

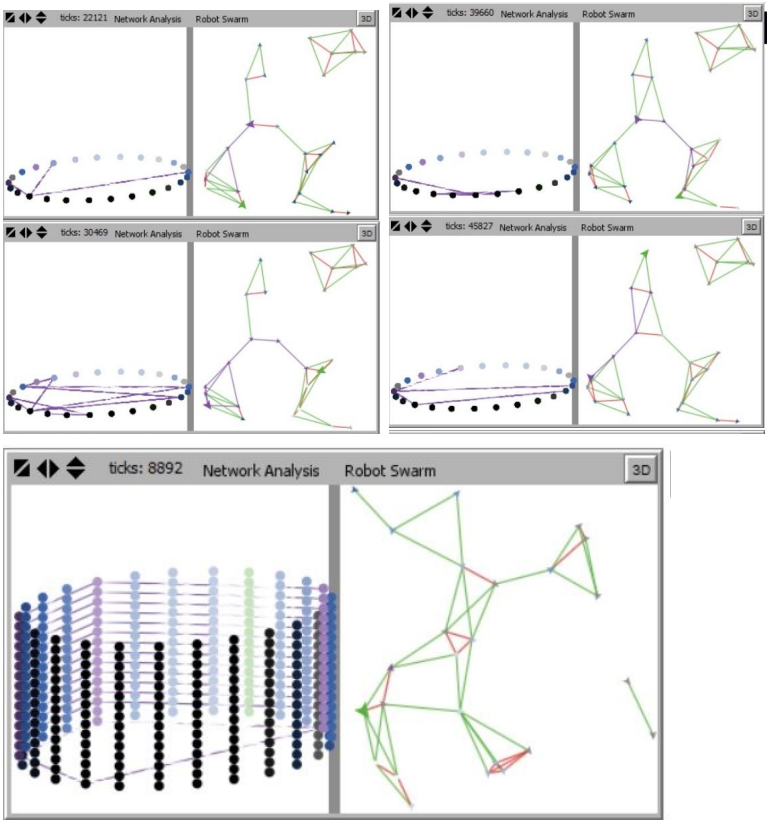


Although the software incorporates a dynamic network representation of the swarm, it is not the one used for analyses since a static network representation has the added benefit of ignoring the individual motion of the robots (thus simplifying the parameters further).

Below shows some of the representational methods used to depict the robotic swarm: the swarm represented as a particle swarm (right); the swarm represented as a dynamic network wherein the network links explicitly represent inter-robotic interactions (centre); the swarm represented as a static network wherein the motion of the robots are ignored (left)

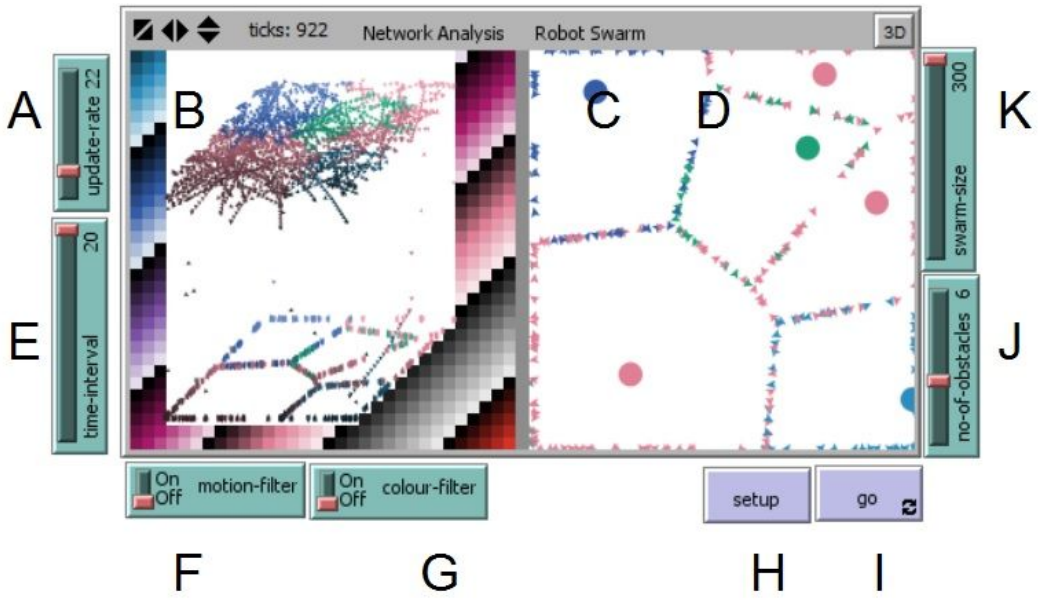


The 3d effect was then introduced to allow multiple instances of the static network at different times to be shown (stacked on top of one another via the z-axis), thus permitting us to view the spatio-temporal developments of the static network.



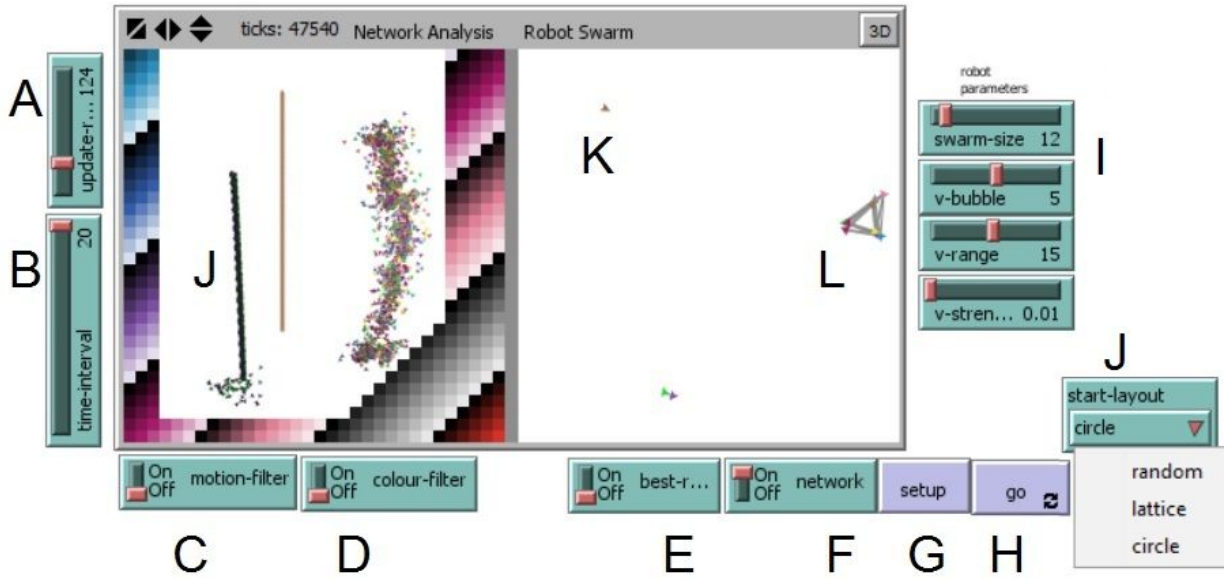
(IV.4.4) Analysis Software Version 2: 3D dynamic swarm analysis of voronoi model

This version analyses the emergent behaviour of the voronoi model (instead of the virtual forces model).



"A" shows the variable "update-rate" which determines how often the analysis window updates itself. The number indicates how many iterations pass before the screen will update. This variable can be tuned by the user to any number between 1 and 100. The lower the value, the shorter the number of iterations between updates, and the more often the analysis window will update, and thus more detail can be seen. However, due to the limitations of the screen size, only a relatively small span of time can be viewed in such detail at any one time. Hence, a larger value for the "update-rate" will provide a broader overview of the swarm's development over a long time. "B" shows the development of the swarm over time. The swarm is viewed in three dimensions to allow the temporal development to be shown. This way the motion of each robot can be traced throughout space and time. "C" represents an obstacle which the robots indirectly interact with to position themselves around the environment. "D" shows the robots which take the colour of their nearest obstacle. "E" shows the variable "time-interval" which controls how many past versions of the swarm are displayed (via the z-axis). The variable can be tuned by the user to any number between 1 and 20. The larger the number, the more past versions of the swarm can be viewed at once to see the spatio-temporal developments across the swarm over time. However, a smaller number allows for a simpler, less cluttered 3d view of the swarm nearer to the real-time updates. "F" shows the variable "motion-filter" and "G" shows the variable "colour-filter". Viewing the swarm in 3d over time can become very cluttered and complex. To simplify the view, filters can be used to ignore any robots whose position or colour has not changed since the last update. "H" and "I" show the functions "setup" and "go" respectively. "J" shows the variable "no-of-obstacles" which controls how many obstacles are placed randomly around the environment. The variable can be tuned by the user to any integer between 0 and 5% of the number of robots in the swarm (e.g. if there are 20 robots in the swarm, there can be a maximum of 1 obstacle. If there are 40 robots, there can be 2 obstacles, etc). K shows the variable "swarm-size" and controls the number of robots in the swarm. This value can be tuned by the user to any integer between 20 and 300.

(IV.4.5) Analysis Software Version 3: 3d dynamic swarm analysis of Virtual-Forces Model



This is the final version of the analysis software which analyses the emergent behaviour of the virtual-forces model, however this time using a 3d representation of the particle swarm representation to display the spatial developments of the swarm over time (rather than using a 3d static network to perform the analysis). This is the version in which embedded spatio-temporal computational patterns were eventually discovered (Displayed are an example of three spatio-temporal patterns; "J" shows a stable pair, "K" shows a static robot and "L" shows a semi-stable cluster of robots)

"A" and "B" show the variables "update-rate" and "time-interval" which allow the user to control the time span and rate of the analysis window on the left. "C" and "D" show the switches which allow the user to filter out static movements (i.e. robots which do not change position over time) or unchanging internal states (i.e. robots which do not change colour over time). "E" is a switch which allows the user to activate the swarm's internal emergent behaviour for analysis; finding the best route using the robots' internal states. "F" allows the user to view the swarm (in the right window) as a dynamic network, so that the network links can be seen and the robots which are influencing one another can be clearly identified. "G" and "H" are the setup and go functions. The above features have already been explained above and so are only briefly mentioned again here. "I" shows the variables which the user can adjust to control the behaviour of the robots to observe how it affects the emergent behaviour of the swarm. These variables are the "swarm-size" (the number of robots in the swarm. It is limited to 100, after which point the analysis becomes overburdened and slow), the "v-bubble" (the radius of the virtual repulsive bubble around the robot. Robot virtual forces are usually attractive to encourage swarm cohesion, however within this radius the virtual forces become repulsive to avoid collisions), the "v-range" (the maximum reach of the virtual forces on their surroundings. This value determines the robot's neighbourhood of influence. It is limited by the maximum range of the robotic sensors - i.e. 30) and "v-strength" (the strength of the robots' virtual forces). "J" is a drop down menu allowing the user to determine how the swarm is initially setup. The initial configuration of the swarm corresponds to its computational inputs and so it may have a significant impact on the swarm's behavioural development.

(5) Software Development

(IV.5.1) Software used

The robotic swarms used in this project were all evolved, modelled, simulated and analysed using the multi-agent programming language called "Netlogo". Netlogo is the language of choice for modelling complex adaptive systems and emergent phenomena [60] because it was designed for the efficient computation and representation of thousands of heterogenous individuals running in parallel [60] and interacting in a complex and dynamical manner (that can be computationally burdensome if modelled in most, conventional, programming languages - i.e. C, Java, Python, etc).

(IV.5.2) Developing the Virtual Forces Model

(IV.5.2.A) Main Algorithm

The program's main control loop ("go" function) to simulate the robotic swarm operating on the virtual-forces model is very simple:

```
to go  ...
        ask-concurrent robots
        [  update-vfmodel
           update-motion ]
        tick
end
```

On each iteration (tick), robots update their individual views of the world ("update-vfmodel" function) followed by the resulting motion ("update-motion" function). To update a given robot's view, the robot must calculate the strength of each push and pull it experiences from virtual forces attributed to nearby robots, obstacles and goals (within a specified radius). Robots use inbuilt physics-based equations to calculate virtual forces as a function of distance. Based on the overall (net) virtual force that the robot experiences, its new heading (direction) and speed can be calculated so that its resultant motion is updated accordingly. Since the new position of each robot subtly influences the dynamics and virtual forces of other robots, all robot must be updated simultaneously, in parallel (using the "ask-concurrent" command), rather than consecutively, one after the other.

(IV.5.2.B) Representing the Robots

Inbuilt agents (referred to as "turtles") were used to represent the robots of the swarm (the default triangular shape was kept). Before the program is run, during the preparatory stage ("setup" function), the robotic swarm is populated with robots which are randomly positioned ("random-xcor..") in the environment (an amount specified by "swarm-size" - a user-defined variable).

```
to setup ...
        create-robots swarm-size [set size 2 setxy random-xcor random-ycor ] ...
end
```

When the program is run ("go" function), robots continually update their position ("update-motion" function) at each iteration (tick), so that they move around the environment.

```
to update-motion
        facexy (xcor + Fx) (ycor + Fy)
        setxy (xcor + Fx) (ycor + Fy)     ...
end
```

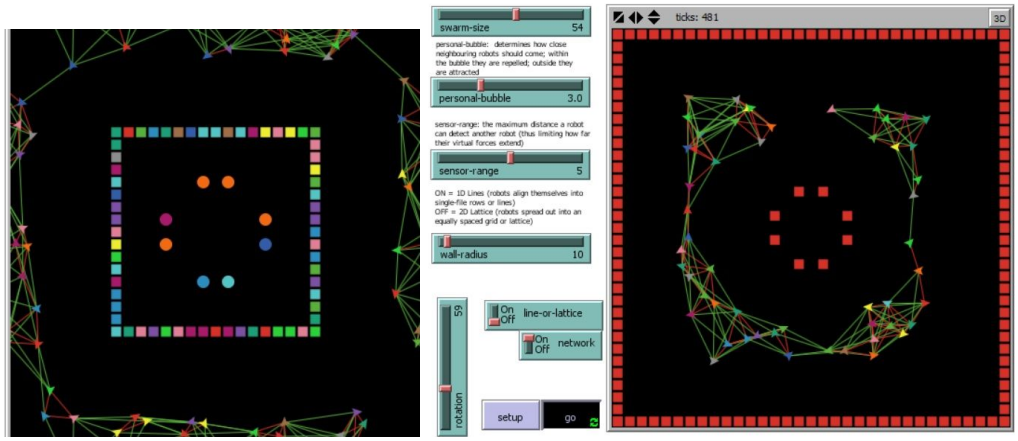
The speed and direction of the robot directly depend on the net virtual force it experiences (i.e. the stronger this force, the further the robot will be moved). Thus the robot's heading ("facexy...") and position ("setxy...") are

based on the x-y coordinates of the net virtual force which the robot is currently experiencing.

(IV.5.2.C) Representing the Environment, Obstacles and Goals

The robots operate within an unbound 2D world which wraps around vertically and horizontally with obstacles and goals (“crt-obstacles” function). The obstacles and goals are also modelled using turtles, however, to differentiate them from robots, they are given a different breed (“breed [obstacles…]”) and, unlike robots, remain static (although by modelling them as turtles, there is potential in future testing for the obstacles to move around; creating a dynamic environment).

breed [robots robot] breed [obstacles obstacle] breed [goals goal]



Obstacle breeds are given a square shape and laid out in a rectangular arrangement, with a variable size (set by a user-defined variable called “room-size”), whereas goal breeds are laid out in a circular arrangement with a variable radius (set by a user-defined variable called “goal-radius”).

```
to crt-obstacles
        set-default-shape obstacles "square"
        set-default-shape goals "circle"
        ask patches with [ max ( list abs(pxcor) abs(pycor) ) = room-size ]
                [ sprout-obstacles 1]
        ask patches with [ pxcor * pxcor + pycor * pycor = goal-radius ]
                [ sprout-goals 1 ]
end
```

(IV.5.2.D) Calculating Virtual Forces

The internal model of each robot allows them to calculate virtual forces from nearby robots, obstacles, goals and chemical-heat trails. The relative strength of each force is defined during setup using global variables (“goal-strength obst-strength rbt-strength env-strength” respectively) and fine-tuned to delicately balance the various attractive and repulsive forces. The pull of chemical-heat trails is strongest, followed by the repulsive force of obstacles. The attractive force of the goal is the weakest.

```
globals [ goal-strength obst-strength rbt-strength env-strength …]
to setup…
        set goal-strength 0.001
        set obst-strength 1.0
        set rbt-strength 0.01
         set env-strength 1
end
```

On each iteration, robots calculate all the surrounding virtual forces (“update-vfmodel” function), first coming from neighbouring robots (“robot-forces” function), then the repulsive virtual forces from walls and other obstacles (“obstacle-forces” function), then attractive virtual forces

from goals (“goal-forces” function) and chemical-heat trails (“environmental-forces” function).

```
to update-vfmodel ...
        robot-forces
        obstacle-forces
        goal-forces
        environmental-forces
end
```

To calculate the virtual forces coming from other robots (“robot-forces” function), the robot will query all other robots within a certain radius from itself (“ask other robots in-radius...”) known as the “sensor-range” (a user-defined variable). This limits a robots local communication to the maximum range of its sensors. The neighbouring robots’ distances and directions from the robot will be measured (stored in the temporary variables “D” and “Th” respectively).

```
to robot-forces
        ask other robots in-radius sensor-range [
                ...
```

The neighbouring robot’s virtual force will be made positive (attractive) by default (“set delta 1.0”), to encourage the swarm to cohere and stick together. However, if the neighbouring robot’s distance (“D”) is less than a certain radius known as the robot’s “personal-bubble” (a user-defined variable), the virtual force it exerts will be made negative (repulsive) because it is too close and needs to move away to avoid collisions (“set delta -1.0”).

```
...
set delta 1.0
if ( D < personal-bubble) [ set delta -1.0 ]
...
```

The magnitude (strength) of the neighbouring robot’s virtual force (“F”) is calculated as a function of its distance (“D”) from the robot. The equation F = 1 / D^2 is used to ensure the virtual force increases non-linearly; so that the closer the robot comes, the stronger the force gets, whereas the force is negligible at farther distances, similar to gravitational forces. This was found to provide more stable swarm behaviour than simpler, linear inverse relationships (i.e. F = 1 / D).

```
...
let F (rbt-strength / (D * D))
...
```

Rather than summing all the virtual forces at the end, which would require the robots to store more variables in their limited memories, each virtual force is added as an accumulated total into three numbers in the robot’s memory which represent the net force; the direction of the force vector (“delta”) and the x-y components of the force vector (“Fx, Fy”). Basic vector equations are used to calculate the x and y components. E.g. Finding the x-component of the force vector, the cosine of the angle “Th” (representing the direction of the neighbouring robot) plus “heading” (representing the sensor reading for the robot’s own direction) is multiplied by the magnitude of the Force (“F”). A similar operation involving “sine” is used to find the y-component of the force vector.

```
Turtles-own [  Fx Fy delta ]
...
ask myself
[           set Fx (Fx + (cos(Th + delta * heading) * F))
            set Fy (Fy + (sin(Th + delta * heading) * F))  ]
...
End
```

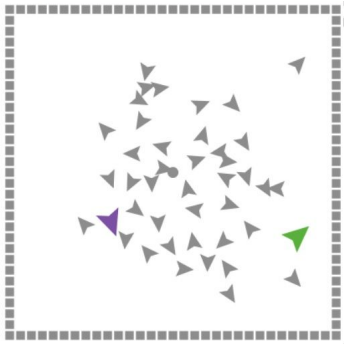
This process is then repeated for the obstacles (“obstacle-forces” function) and then the goals (“goal-forces” function), with the exception that,

regardless of their distance from the robot, the virtual forces are always negative (repulsive) for obstacles and always positive (attractive) for goals.

(IV.5.2.E) Displaying an emergent Route

As well as using virtual forces to design emergent behaviour in the motion of the robot swarm, each robot is given a state (i.e. a colour it displays) and this state (colour) can change. Similar to the states which cellular automata have, the state (colour) of the robots allow the swarm to display things like routes between two points. The way the routes appear across a swarm lacking a central controller or global information is also an emergent phenomena. Thus the swarm will simultaneously display two emergent behaviours (motion and route). In order to allow a route to emerge, two robots of the swarm are randomly selected to be the starting point ("startbot" - coloured green) and end ("endbot" - coloured purple) of the route.

```
globals [ ...N startbot endbot]
to setup ...
        set startbot one-of robots
        set endbot one-of robots ...
end
```



In order to find the best route emergently, each robot is also given a single variable to remember ("chain-weight"). This represents the accumulated total count of robots linking this robot to the start of the route ("startbot"). This value is an indicator of how far away the current robot is from the start.

```
turtles-own [ ... Chain-Weight ]
```

Initially this "chain-weight" of each robot is set to some high value (i.e. 100). However, the "startbot" is given a "chain-weight" equal to 0. To find the best route we begin with the "startbot" ("best-route(startbot)" function). Once the "endbot" is reached, the best-route is shown ("show-route" function).

```
to update-internal-states
        ask robots [set color gray set size 2 set Chain-Weight 100]
        set N 0
        ask startbot [set color green set size 3 set Chain-Weight 0]
         best-route (startbot)
        ask endbot [set size 3 show-route ]
End
```

The startbot queries its neighbouring robots, but only those neighbours with an unset "chain-weight" (i.e. their "chain-weight" is still set to the default start weight of 100 - using the command "ask link-neighbors with [Chain-Weight = 100]"). Robots avoid requerying robots with a set "chain-weight" because the "chain-weight" should only be set once and should not be overridden.

```
to best-route [ rob ]
        let my-weight [Chain-weight] of rob
        ask rob [
                ask link-neighbors with [Chain-Weight = 100]  [
                        ...
```

The neighbouring robot's "Chain-Weight" is set to the querying robot's own "chain-weight" value + 1. The robot's colour is shaded to represent the value of the robot's "Chain-Weight" ("set color scale-color…").

```
                                        ...
                                        set Chain-Weight (my-weight + 1)
                                        set color scale-color blue Chain-Weight 10 1
                                        ...
```

Then the neighbouring robot repeats this process recursively ("best-route" function) with its own neighbouring robots, until the "endbot" is reached. The idea is to create a route from the lightest weighted robot (i.e. the startbot) to the next heaviest robot, to the next, until the endbot (which will be the heaviest weighted robot).

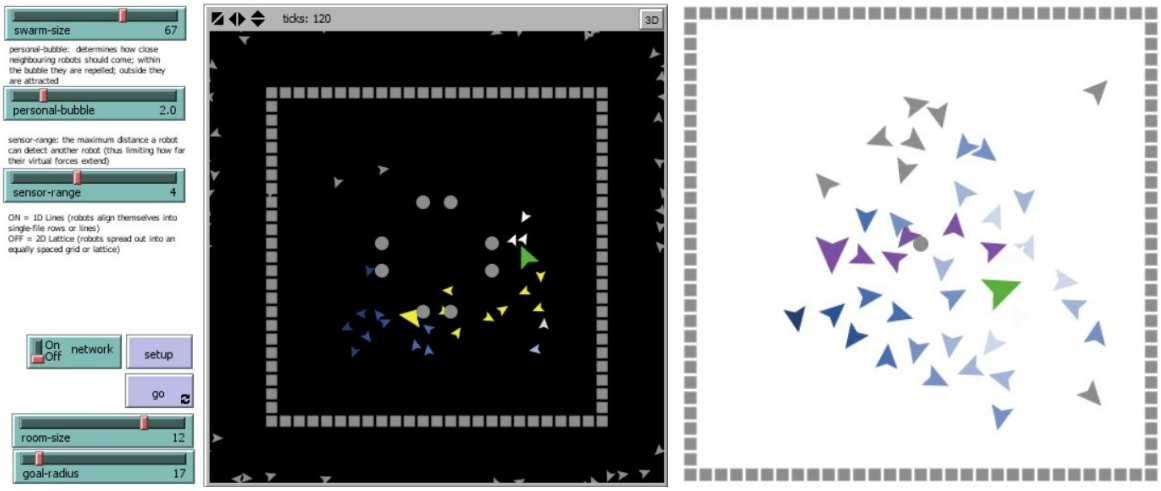
```
                                        ...
                                        best-route (self)
                                ]
                        ]
                end
```

The best route is then shown using another recursive function ("show-route" function). Each robot is checked for a "Chain-Weight" of 0 (which would indicate the robot is the startbot. In that case, the base condition is met and the recursion ends.

```
        to show-route
                ifelse Chain-Weight = 0 [ ] [
                                ...
```

Otherwise, the robots on the way to the "startbot" (Starting from the "endbot"), are coloured a distinct colour, such as yellow or violet ("set color…") to indicate it is part of the best route. Then the same process ("show-route" function) will repeat with the robot's neighbour that has the next lightest "Chain-Weight" ("ask link-neighbors with [Chain-Weight = W] [show-route]"). It ignores robots with equal or heavier weights since they do not lead closer to the "startbot".

```
                                        ...
                                        set color violet
                                        let W (Chain-Weight - 1)
                                        ask link-neighbors with [Chain-Weight = W] [ show-route ]
                        ]
                end
```



(IV.4.2.F) Stigmergy: Chemical-Heat trails

The final modification added to the program was to investigate the effects of indirect communication via manipulation of the environment to temporarily

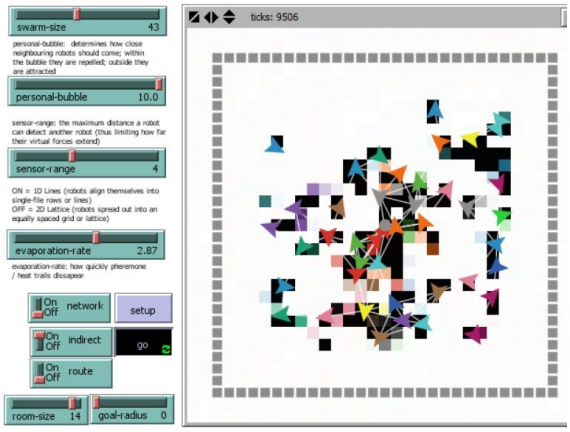
store past events (stigmergy). Each (square) patch (of the environment) has a variable associated with it (called "heat-chemical") which indicates the temperature or potency of the chemical pheromone left at that location.

patches-own [heat-chemical]

If there is a robot at that location, the value of its "heat-chemical" will be set to the maximum value. The value then slowly decreases to represent the heat slowly dissipating or the pheromone evaporating. The speed the value decreases is controlled by a user-controlled variable ("evaporation-rate").

```
to indirect-links
        ask patches [
                set heat-chemical heat-chemical * (100 - evaporation-rate) / 100
                ...]
end
```

The shade of the patch is set to reflect the current "heat-chemical" value



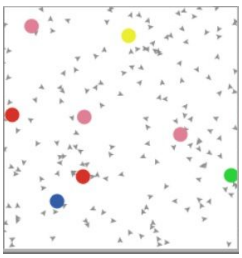
(IV.5.3) Developing the Voronoi Model

The voronoi model is an alternative swarm designed for emergent behaviour based on a very simple rule. This model is based off robots interacting with obstacles in the environment, rather than robot-robot interactions as with the virtual-forces model. The resultant emergent behaviour is a pattern resembling a voronoi map, wherein the swarm creates equally spaced boundaries around the obstacles. A set number of obstacles (determined by the user-defined variable "no-of-obstacles") are randomly distributed around the world ("setup-obstacles" function). Obstacles are displayed as circles ("set shape 'circle' ").

```
to setup-obstacles
        create-obstacles no-of-obstacles [
                set shape "circle"
                set size 2
                setxy abs(random-xcor) random-ycor
]
```

Then the robots are also distributed randomly around the world as before.

```
to setup-swarm
        create-robots swarm-size [ set color grey  setxy abs(random-xcor) (random-ycor)]
End
```



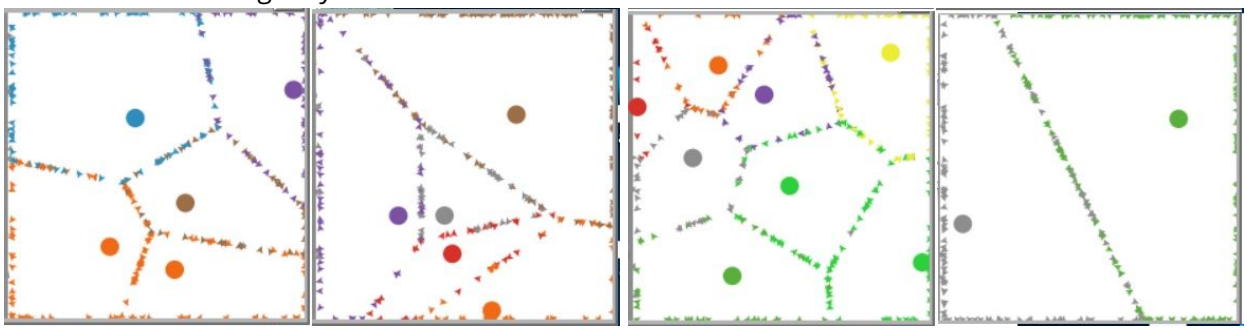
Each robot (“ask-concurrent robots”) runs the same simple rule (“swarm-rule” function). The rule consists of the robot measuring the distance to its closest obstacle (“nearby-obstacles” function).

```
to-report nearby-obstacles
        report obstacles with-min [ precision (distance myself) 1 ]
end
```

The robot then counts how many obstacles are at that distance from it. There is usually only one obstacle with that distance (i.e. the closest obstacle). In that case the robot will take the colour of that obstacle (“set color [color] of one-of nearby-obstacles”) and move forward slightly (“fd 0.04”) after which it will turn slightly at a random angle (“rt 0.5 - random-float 1”).

```
to swarm-rule
        ask-concurrent robots [
                if count nearby-obstacles = 1 [
                        if( xcor > 1 ) [fd 0.04 set color [color] of one-of nearby-obstacles ] ]
                rt 0.5 - random-float 1
        ]
end
```

Robots will continue to execute this simple rule again and again until they end up in a location where the condition is no longer true (“count nearby-obstacles = 1”). In other words, there are more than one ‘closest’ obstacles, or other obstacles which have the same distance to the robot. This will only occur at the midpoint boundaries between obstacles. The robot no longer do anything and remain in its position and emergently form obstacle boundaries.



(IV.5.4) Developing the Genetic Algorithm

As well as the hand-written algorithms used to generate emergent behaviour, we designed a genetic algorithm which evolved robotic swarms with emergent behaviours.

(IV.5.4.A) Phenotype & Genotype

The variable called “strategy” contained the encoded candidate solutions (a potential robot ruleset). Rather than making this a global variable which could be run by identical robots in a homogeneous swarm, the variable is attached to individual turtles, allowing multiple different rules to co-evolve in a heterogeneous swarm.

```
turtles-own [ strategy ...]
```

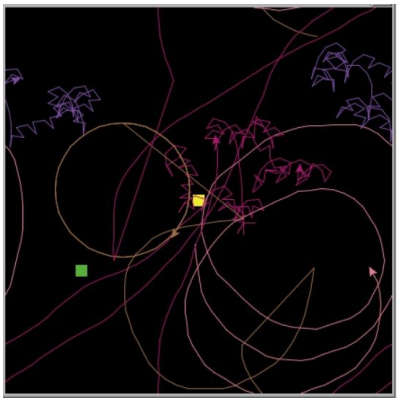
The “strategy” is an “L” length list (i.e. a variable length set by the user) containing “commands” (a global list of predefined movements which can command the robot’s motion; such as turning right by a random angle (“rt rnd”), moving forward by a variable distance (“fd d”), staying still (“fd 0”), etc).

```
globals [... commands d rnd...]
to setup
        set commands ["rt rnd" "lt rnd" "fd d" "bk d" "rt d" "lt d" "fd 1" "bk 1" "rt 90" "lt
90" "fd 0"] ...
end
```

The first generation of candidate solutions have randomly picked "commands" ("one-of commands").

```
to-report random-strategy
        let strat n-values L [ one-of commands]
        report strat
end
```

Each iteration the "strategies" would be decoded by the algorithm ("run-strategies" function) in order to observe the behaviour produced by the evolved "strategy" and calculate the fitness from the results of this behaviour.



```
to go ...
        run-strategies
end
```

Firstly the variables "rnd" is set to a new random value between 10 and -10. The variable "d" is set to the distance between the robot and its closest neighbour within the range of its sensors ("ask one-of other... set d distance myself"). Then each "command" in the "strategy" is executed ("run ?") one-by-one ("foreach strategy...").

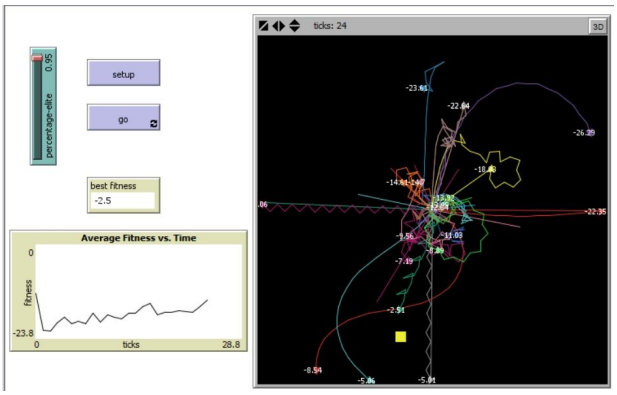
```
to run-strategies
        ask turtles [
                set rnd (random 20) - 10
                set d d
                if any? other turtles in-radius sensor-range [
                        ask one-of other turtles in-radius sensor-range [
                                set d distance myself ] ]
                        foreach strategy [run ?] ]
end
```

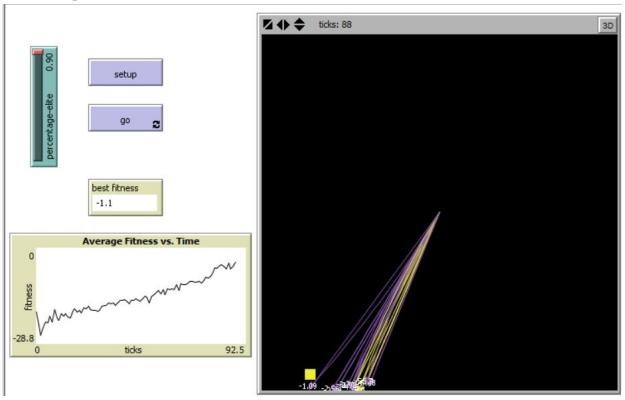
(IV.5.4.B) Fitness function

Different fitness functions were tried and tested. The first calculated the fitness as a function the variable "distance goal" (i.e. how close a robot rule strategy would make a robot came to a specified location would determined the strategy's fitness).

```
to-report fitness
        report distance goal * -1
end
```

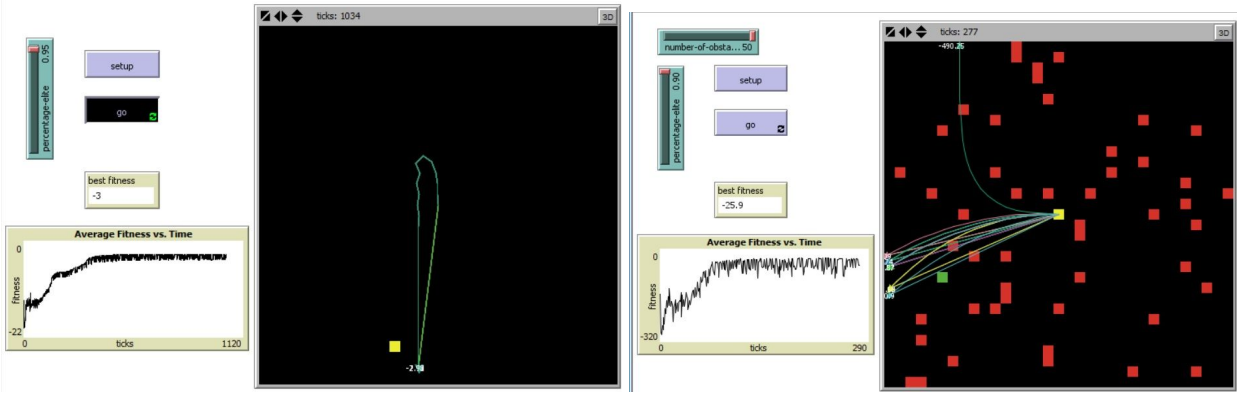
Each robots began with a random "strategy" (see left screenshot below) and quickly evolved into a handful of the fittest "strategies" which all lead the robot nearby to the "goal" (see right screenshot below).





Eventually, two alternative "strategies" were evolved and the fitness would not improve beyond any further (see left screenshot below). The evolved strategies could reach a goal from a given start point. However, these

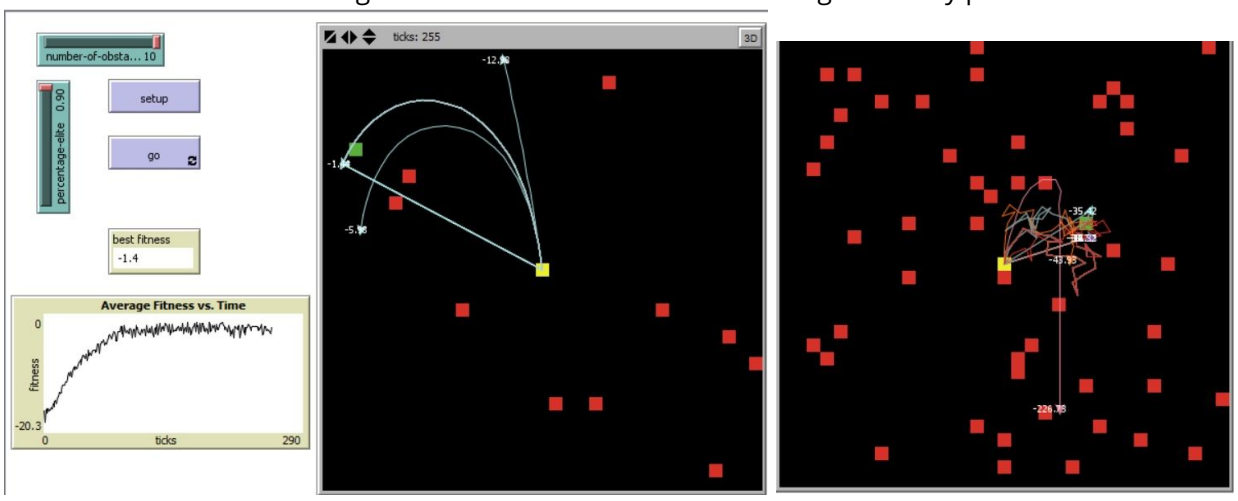
"strategies" had not evolved to consider "obstacles" and so they would simply run through them when they were introduced (see right screenshot below).



Therefore the fitness function was modified so that robot's were penalised for any collisions with "obstacles" on the way to the "goal". The results differed depending on the relative strengths of the collision penalties to the goal rewards. Eventually, an indirect penalty was decided upon. The fitness remained a function of the distance to the "goal", yet robot's on the same patch as one another (i.e. a collision) would immediately be killed off ("die") and replaced ("respawn" function).

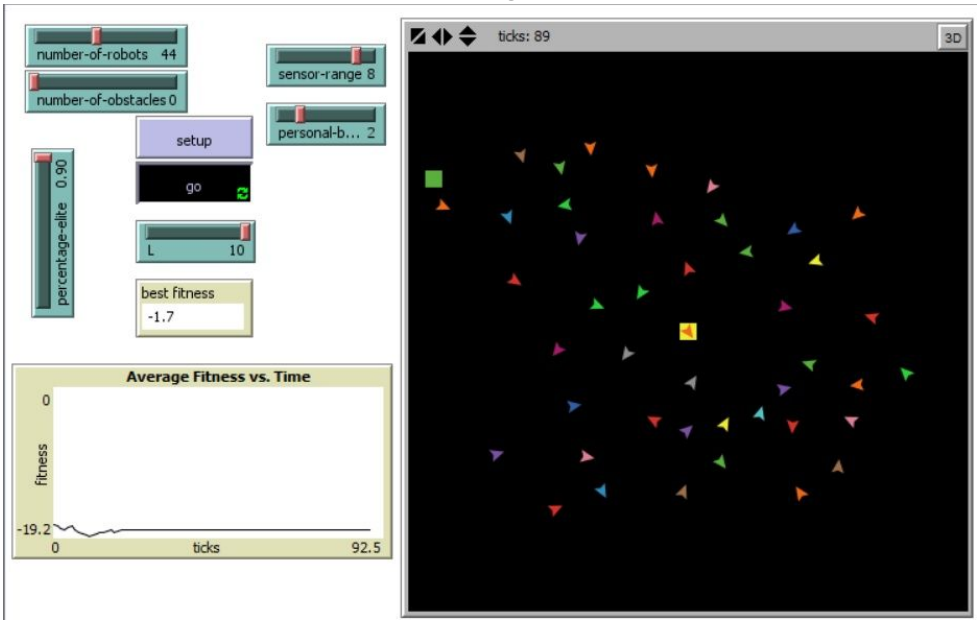
```
to replacement-inheritance
        ask turtles [
                ask other turtles-on patch-here [ respawn ]
        ]
End

to respawn
        let strategy_dad strategy
        hatch 1 [setxy random-xcor random-ycor set heading random 360 set strategy
strategy_dad set th heading set v 0 set a 0 set fitness 1000]
        die
end
```

The fitness function was successful at evolving robots which could reach a fixed goal from a set location while avoiding randomly placed obstacles.



However, the robot's didn't necessarily work together as a self-organised swarm nor did the swarm display emergent behaviour. The extrinsic element of the "goal" was removed and an "intrinsic" fitness function was employed. A purely life and death fitness function wherein robot's would not have their fitnesses calculated, but if they survived they were assumed fitter. Robots

would be allowed to survive so long as they didn't collide with an obstacle or one another. If they collided, they would immediately be killed off and replaced (using the method described above). This occasionally evolved swarms with coherent motion and emergent behaviours.



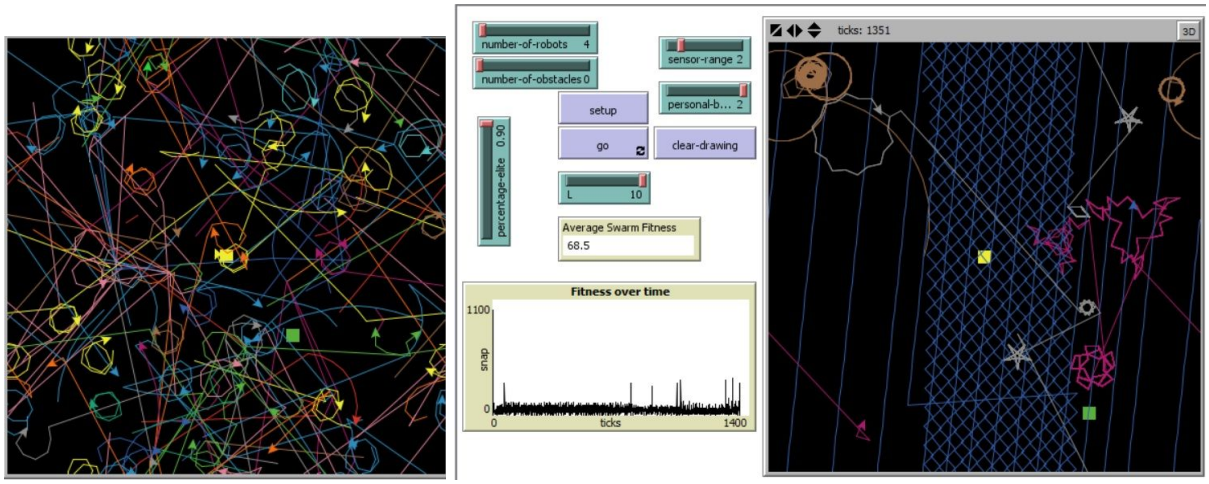
Still, another fitness strategy was explored, based on snap. Snap is the 4th order derivative of position. In other words, the rate of change in acceleration. It is often used to measure how much an object is shaking. The higher the snap, the higher the shakiness.

```
to go
        calc_avg_snap
        ask turtles [calc_fitness (snap_avg)]...
End
```

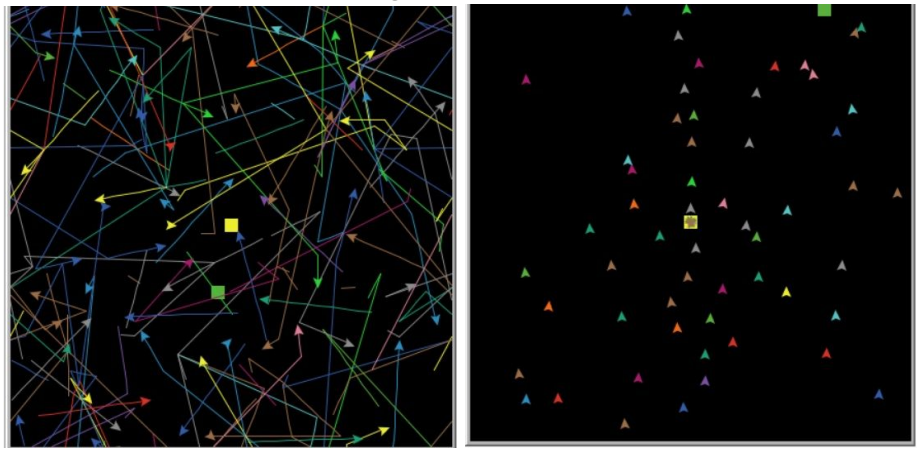
The robot needs to remember its prior velocity to calculate its current acceleration (by finding the difference between the current velocity and prior velocity - i.e. the change in velocity). The prior acceleration is also recorded by the robot in order to find its current snap (the change in acceleration).

```
turtles-own [ ... th v a fitness]
globals [goal ... th_avg v_avg a_avg snap_avg]

to calc_avg_snap
        let th_new mean([heading] of turtles)
        let v_new abs(th_new - th_avg)
        set th_avg th_new
        let a_new abs(v_new - v_avg)
        set v_avg v_new
        set snap_avg abs(a_new - a_avg)
        set a_avg a_new
end
```

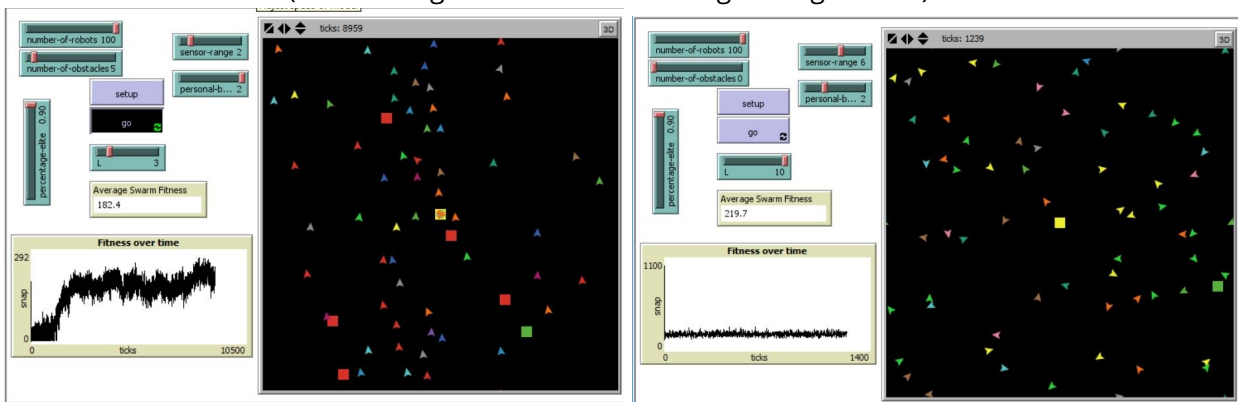
Therefore, if robots have a higher fitness with higher snaps, the swarm will be encouraged to evolve violent behaviour with sudden changes.



If, however, lower snaps produce higher fitness values, then the evolution of smoother behaviour is encouraged.



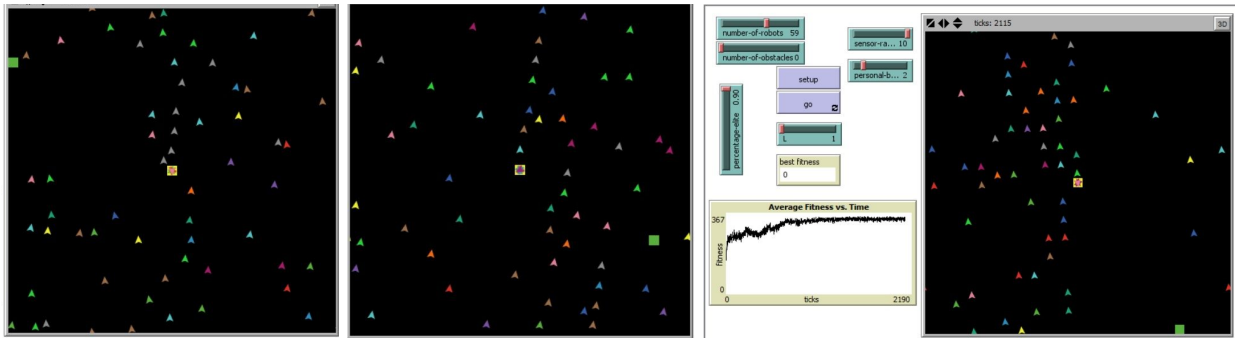
When the robots operated on an identical strategy being evolved (homogeneous swarm - see left screenshot below) they worked better as a swarm than when each robot was allowed to evolve independent strategies (in a heterogeneous swarm - see right image below).



The robot's evolved a variety of behaviours with smooth behaviour, yet were incoherent as a swarm. Finally, the fitness function was modified to take into account a measure of the level of order and coherence in the swarm. This was done by measuring how much the (individual) robot deviated from the average (group). The lower the deviation, the higher the fitness, thus encouraging the evolution of similar behaviours. At first, the individual

heading was compared to the average group heading. This resulted in swarms evolving with robots all heading the same direction. Then the change of heading was compared and that resulted in better results. The evolved behaviour was basic yet coherent.

```
to calc_fitness [avg_fitness]
        let v_new abs(heading - th)
         set th heading
         let a_new abs(v_new - v)
         set v v_new
         let snap abs(a_new - a)
         set a a_new
         set fitness (snap * snap)
         let dev (fitness - avg_fitness)
end
```



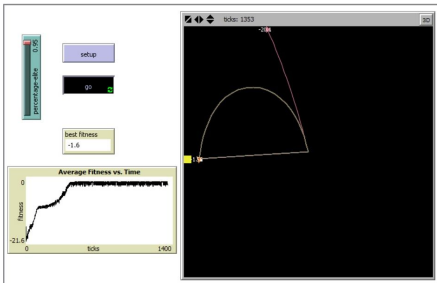
(IV.5.4.C) Mutation

As a “strategy” is reproduced, it is always mutated. One of the “commands” in the genome is picked at random and replaced (“replace-item(random…”) with a random, new “command” (“one-of command”).

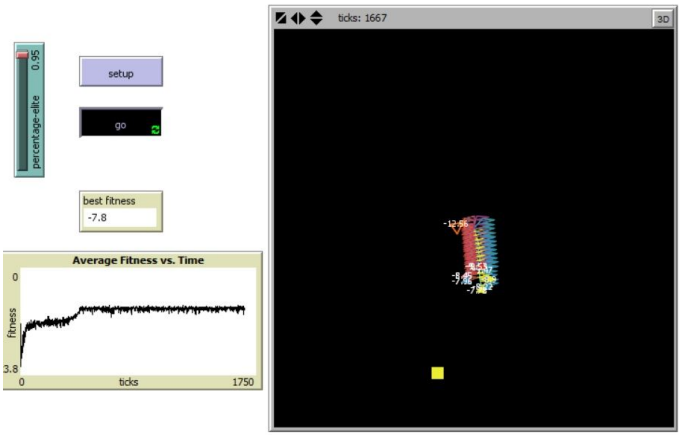
```
to respawn-and-mutate
        let strategy_dad strategy
        hatch 1 [setxy random-xcor random-ycor
                set heading random 360
                set strategy mutate strategy_dad
                set th heading set v 0 set a 0 set fitness 1000]
        die
end

to-report mutate [strat]
         set strat replace-item (random (L - 1)) strat one-of commands
         report strat
end
```

(IV.5.4.D) Crossover

At first, the reproduction phase included only mutation without crossover. Child genomes were cloned from a single parent genome before it died (“die”), and then mutated this strategy slightly. Mutation alone seemed to suffice the evolutionary process and successful strategies were found.



However, it was often noted that using mutation only reproduction could get easily stuck at local maxima and needed “crossover” as a way of jumping out of such wells.



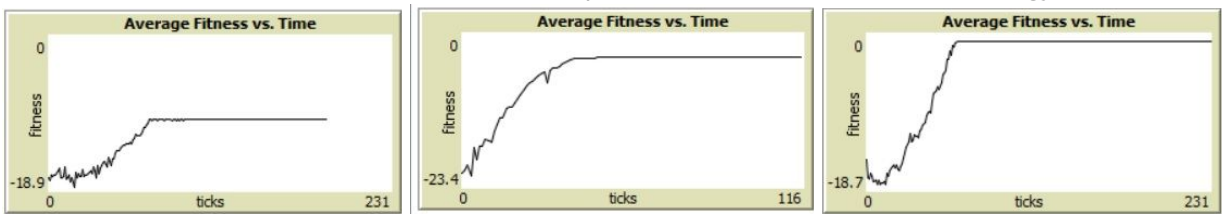
Later, crossover was introduced and child genomes were the combination of two parent genomes. A random number between 1 and “L” (the length of the genome) determines the location to split each parent genome (“split-point”).

```
let split-point random L
```

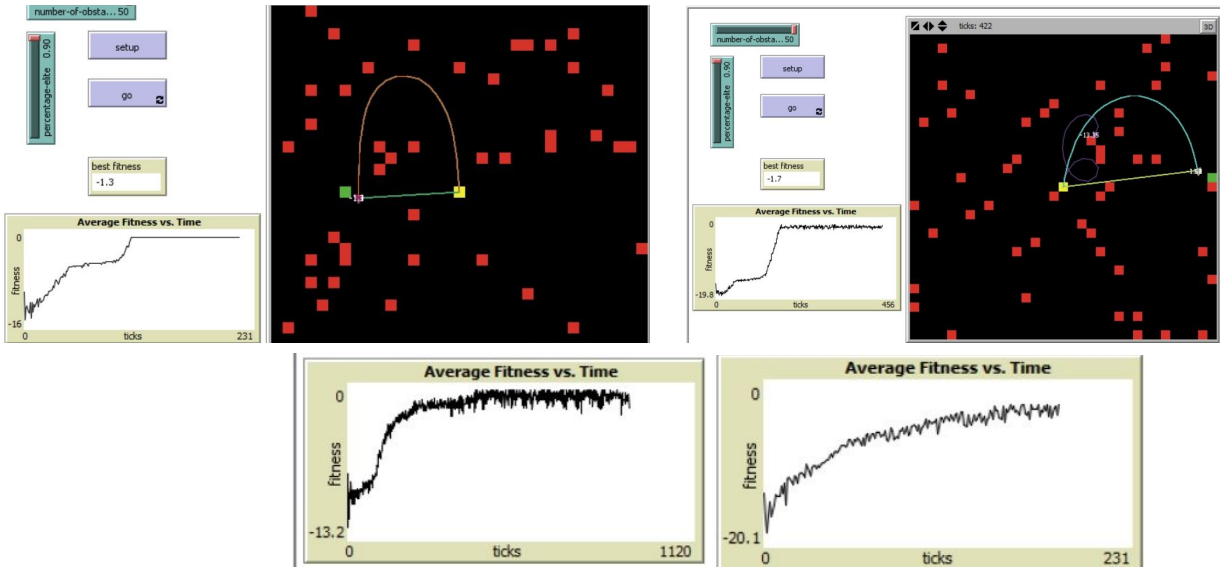
Then the genome of the father (“strategy_dad”) and mother (“strategy_mum”) is split at that single “split-point” (single-point crossover) and the first part from the father (“set strat_1 sublist…”) along with the last part from the mother (“set strat_2 sublist…”) are crossed-over and combined (“let strat_mix sentence strat_1 strat_2”) to produce a completely new strategy (“report strat_mix”). The remaining last part of the father’s genome is also crossed-over and combined with the remaining first part of the mother’s genome, to produce another unique combination and child. For every two parents who reproduce, both possible combinations are explored, thus two children are produced (e.g. A-A & B-B → A-B & B-A).

```
to new-child [split-point strategy_dad strategy_mum]
        hatch 1 [ set strategy crossover split-point strategy_dad strategy_mum
                set th heading set v 0 set a 0 set fitness 1000
                set color one-of base-colors  ]
end

to-report crossover [split-point strat_1 strat_2]
        set strat_1 sublist strat_1 0 split-point
        set strat_2 sublist strat_2 split-point (length strat_2)
        let strat_mix sentence strat_1 strat_2
        report strat_mix
end
```

Crossover was found to provide a quicker and wider exploration of the search-space, which avoided prematurely converged solutions. It also allowed for a steadier rate of improvement. However, without mutation, there would be no minute changes in the strategy to encourage local searches of the search space around the current best strategy found.



Then crossover was used in combination with mutation, to evolve slightly fitter solutions



Finally, the mutation-only reproductions were maintained alongside the crossover-mutation reproductions. Therefore, each pair of parents would produce the two crossover offspring ("new-child split-point..."), as well as a mutated clone of the father's "strategy" ("hatch 1 [...set strategy mutate strategy_dad...]") and a mutated clone of the mother's "strategy" ("hatch 1 [...set strategy mutate strategy_mum...]"). In addition to these four offspring, two more offspring are added which are exact clones of the father and mother ("hatch 1 [...set strategy strategy_mum...]"). This was done to allow a copy of the parent strategies into the next generation as a way of remembering the best solution from the previous generation, to ensure that following generations are always fitter than their parent generation (or at least equally fit, but never less fit).

```
repeat number-to-replace / 6 [
    ask one-of best-turtles [
        let strategy_dad strategy
        hatch 1 [ ... set strategy strategy_dad ...]
        hatch 1 [...set strategy mutate strategy_dad ...]
        ask one-of best-turtles [
            let strategy_mum strategy
            hatch 1 [...set strategy strategy_mum]
            hatch 1 [...set strategy mutate strategy_mum ...]
            let split-point random L
            new-child split-point strategy_dad strategy_mum
            new-child split-point strategy_mum strategy_dad  ] ] ]
```

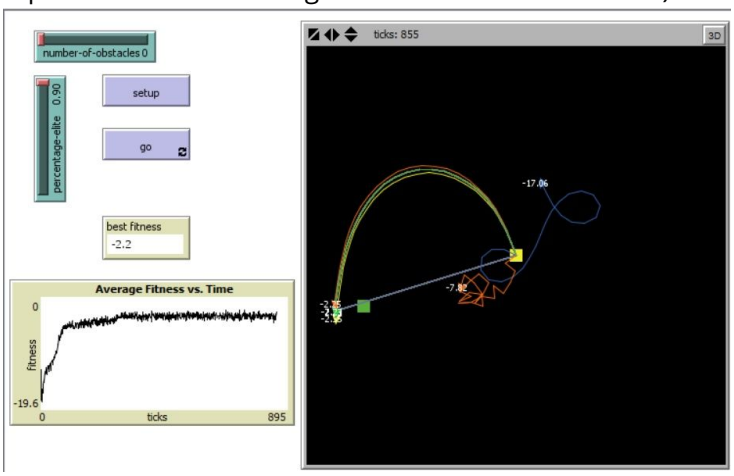
(IV.5.4.E) Selection & Inheritance

Not all the strategies will be used to reproduce. The user defines a percentage ("percentage-elite") of fittest strategies (the elites) that will survive to generate offspring for the next generation and the remaining unfit candidates will be discarded. The number of strategies to be discarded is first calculated using the percentage and the total "count" of the strategies ("let number-to-replace (1.0 - percentage-elite) * count turtles"). The set of strategies ("turtles") is then queried ("ask...") and the amount calculated ("number-to-replace") is systematically killed off ("die"), starting with the least fit strategies ("min-n ... [-1 * fitness]...").

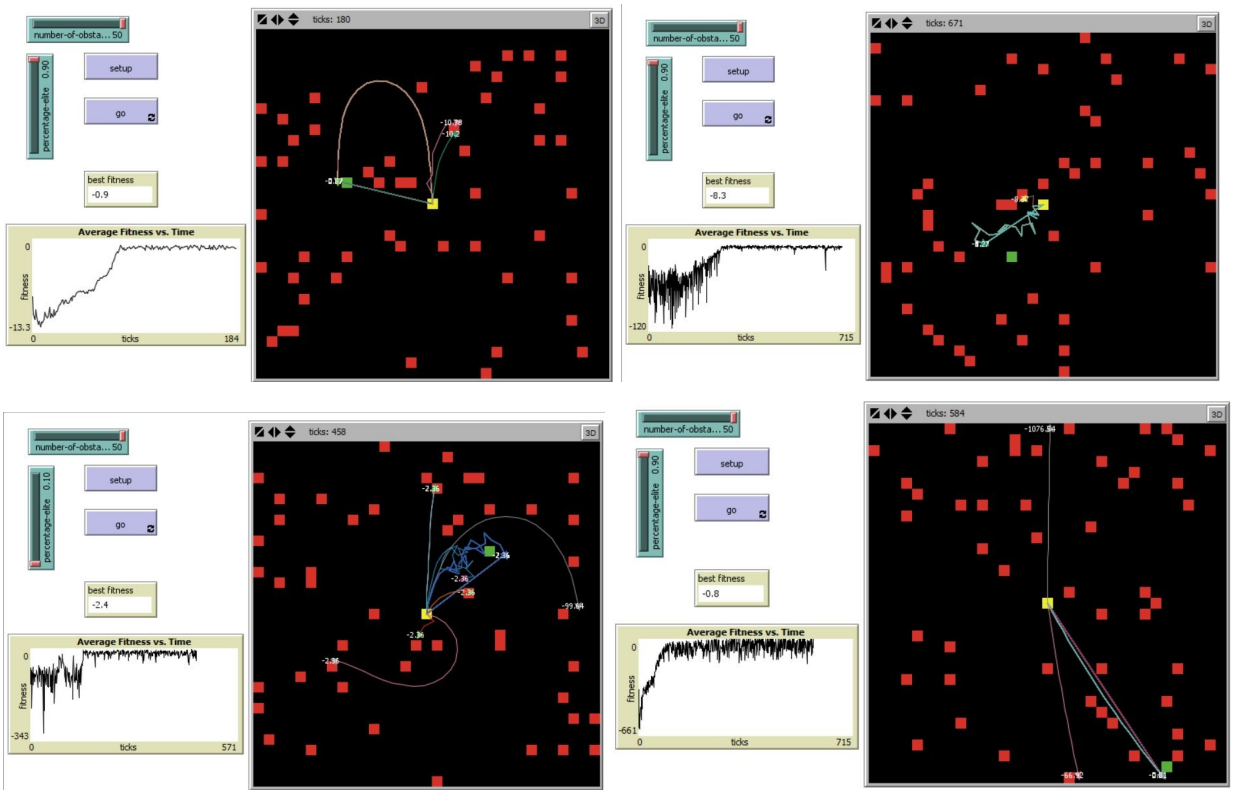
```
to elite-replace-inheritance
    let number-to-replace round ( ( 1.0 - percentage-elite )* count turtles)
```

```
        ask min-n-of number-to-replace turtles [ -1 * fitness ] [ die ]
        let best-turtles turtle-set turtles ...
    end
```

What remain at the end are the fittest strategies - the elites ("best-turtle-set") that will pass on their genes to the next generation, either by cloning or crossover-reproduction. All the elite strategies are sampled from randomly during the mating selection process, however, the probability of being selected is directly proportional to the robot's fitness (since there exists more copies of the fitter strategies and fewer of the weaker).



This elite inheritance happens every iteration all together, however, slightly quicker evolutionary results were obtained when using this alongside the immediate replacement of certain individuals that "die" from collisions (although notice the severe troughs during the evolution, indicating the times where "strategies" would cause robots to collide and thus be replaced).



(IV.5.5) Analysis software

The aim of this next piece of software was to confirm our hypothesis and discover if the emergent behaviour in the robotic swarm was being governed by hidden computational mechanism embedded in the swarm's behaviour. Therefore, the swarm's behavioural development is analysed over time to try and identify the continuous (spatio-temporal) patterns (akin to "gliders" in 2d cellular automata, or "gestalts" in artificial neural networks, etc).

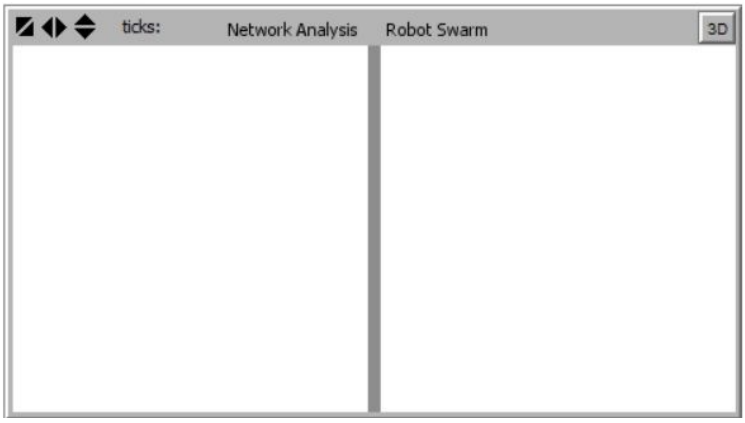
(IV.5.5.A) Split-Screen View

The software needed to display the development of the simulated swarm alongside its spatio-temporal analysis. Since Netlogo is limited to one screen, some clever programming was needed to manipulate our software so that it appeared to have a split-screen.

```
to startup
        split-screen...
end
```

To achieve this effect, the single screen offered by Netlog is modified as it is loading up ("startup" function). A single line is drawn down the middle of the screen (splitting it in half) and the line is coloured grey ("set pcolor grey") to match the colour of the screen edges.

```
to split-screen...
        ask patches with [pxcor = 0][set pcolor grey]
end
```



To complete the illusion, the robot swarm had to be confined to the right side of the partition, and the network analysis to the left (if they cross the middle partition, the illusion is broken). The middle of the screen (where the grey partition sits) is at the 0 x-coordinate. Therefore, to remain on the right of the partition, the swarm can only ever have positive x-coordinate values ("abs(random-xcor)"), whereas the network analysis must only have negative x-coordinate values to remain on the left.

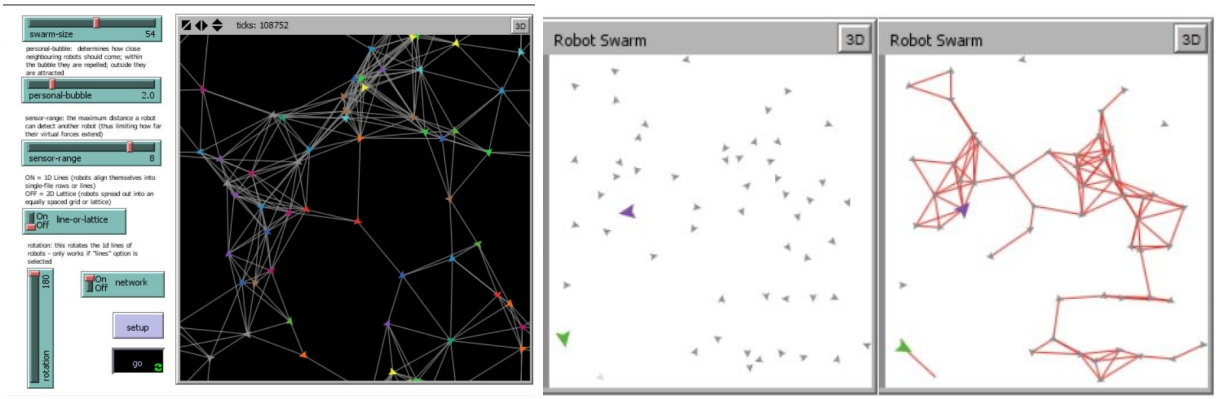
```
to setup-swarm
        create-robots swarm-size [ setxy abs(random-xcor) (random-ycor)] ...
end
```

(IV.5.5.B) Swarm representation

The swarm representation was identical to the methods described above (the swarm operating on the virtual-forces and routing via internal-states model and the genetic algorithm's evolved swarm) with a small exception; the swarm's behaviour needed to be limited so that its motion was contained on the right hand side of the split-screen (i.e. robots could only have positive x-coordinate values) .

(IV.5.5.C) Dynamic network representation

If the user wishes to see the swarm's local connections (i.e. which robots it is currently communicating with via virtual forces), the user can select the "network" option and the robots' virtual forces will be rendered as "links". In this view, the robot swarm more clearly resembles a dynamic network, wherein the links represent virtual forces and mobile nodes represent robots.

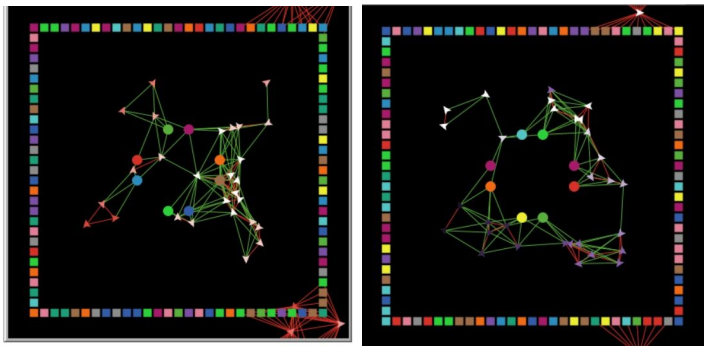


As each robot calculates the virtual forces acting on it, a physical "link" is also created ("create-link-with") to be able to display the force on screen as a line. These "links" are updated on each iteration so that when robots are no longer in communication with one another, the link is destroyed and as a result, the network continually changes (a dynamic network).

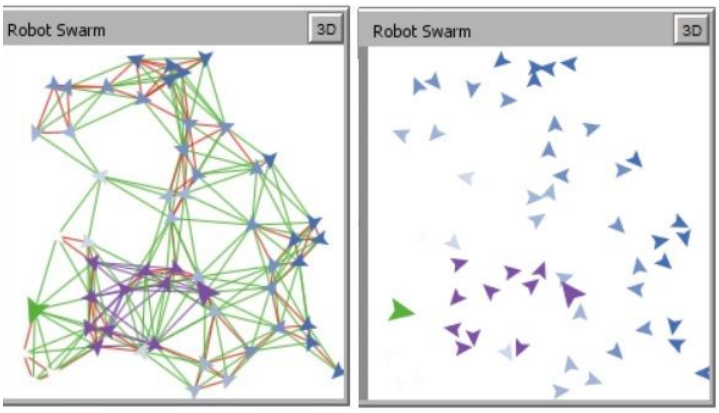
```
to robot-forces
        ask other robots in-radius sensor-range [...
                create-link-with myself [
                        ...
```

To show if the link represents a positive (attractive) virtual force, it is coloured green ("set color green"), while negative (repulsive) virtual forces are coloured red ("set color red").

```
                        ...
                        ifelse network [
                                ifelse ((personal-bubble - D) < 0)
                                        [set color green]
                                        [set color red]   ] ...
end
```

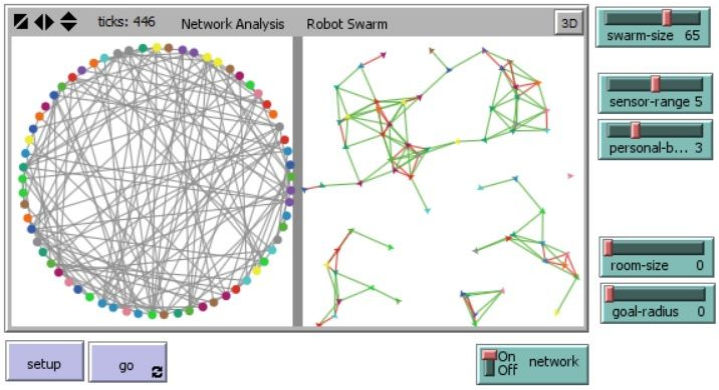


Furthermore, if a route between two points is found via emergent behaviour, the robots will change colour (i.e. to purple) to show this route. The links connecting those particular robots are coloured the same colour (i.e. purple).



(IV.5.5.D) Static Circle network representation

It can be difficult to analyse a swarm or dynamic network due to the continual movements of its robots (i.e. network nodes). It is also possible, that such physical movements have no effect on the movements of the spatio-temporal computational mechanisms we are looking for which should be embedded within the interactions of the swarm (i.e. the dynamic network links). Hence, it makes sense to also have a static network to represent the swarm, which updates its network links much the same way the dynamic network did, however, it disregards the individual movements of the robots and keeps the nodes fixed in one place. Unlike a grid or lattice layout, arranging the nodes (representing robots) in a circle would allow for an unobscured view of every single network link (i.e. robot-robot communication via virtual forces) possible in the network.



The nodes of the static network are represented by a distinct breed of turtles ("breed [nodes…"). As the network is setup ("setup-network" function), the number of nodes is determined by the number of robots in the swarm (user-defined variable "swarm-size") and rather than scattering the nodes randomly like with the swarm, they are laid out neatly in a circle ("set shape 'circle'").

```
breed [ nodes node ]
to setup-network
        create-nodes swarm-size [ … set shape "circle" ]
        layout-circle nodes (world-width / 4 - 1 ) …
end
```

The static network is updated on each iteration ("update-network" function) and this doesn't consist of updating the position of the nodes (since it is a static network), however, it does consists of updating the network links ("update-links" function).

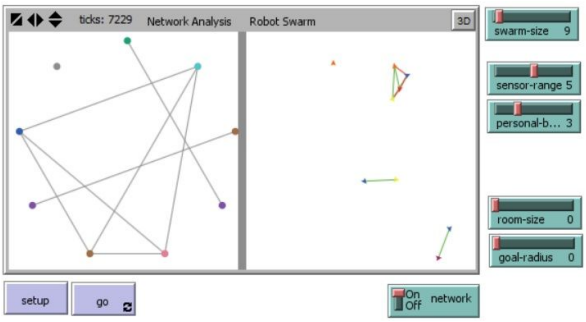
```
to update-network...
        update-links
end
```

Each robot in the swarm is queried ("ask robots") and each has a turtle ID number associated with them ("who"). The ID of the turtle representing the robot currently being queried is temporarily stored ("let Awho..."). Any other robots linked to this turtle are also queried ("ask link-neighbors") and their turtle ID is temporarily stored ("let Bwho...").

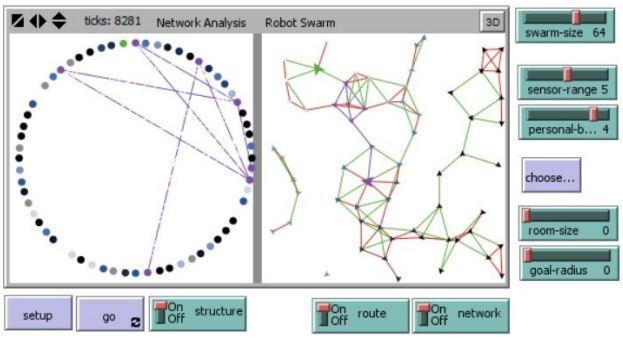
```
to update-links ...
        ask robots [
                let Awho who ...
                ask link-neighbors [
                        let Bwho who
                        ...
```

The total number of turtles (without including obstacles and goal breeds) should be at least twice the number of robots in the swarm (user-defined variable "swarm-size"), accounting for the turtles representing nodes in the static network (e.g. if the swarm has a size of 30 robots, then there are actually 2x30 turtles in the entire system, 30 additional turtles which represent the nodes of the static network). So for a robot to access the corresponding node which represents it in the static network, it should add on "network-size" to its own turtle id number (e.g. assuming the swarm size is 30 robots, the 3rd robot has the turtle ID 3, and the correspond 3rd node in the static network will have a turtle ID of 3+30 = 33). The corresponding turtle for the network node is queried ("ask turtle (Awho + swarm-size)") and a link is created ("create-link-with...") the corresponding network node for the neighbouring robot ("turtle (Bwho + swarm-size)").

```
                        ...
                        ask turtle (Awho + (swarm-size)) [
                                create-link-with turtle (Bwho + (swarm-size))[...] ] ] ]
end
```



The colours of each link are also copied over, and the user has an option to switch on the variable "route" which hides all the links in the static network except for those showing the emergent best route (displayed in purple). This simplifies the display and allows the route links to be shown more clearly.



The reason this method could not be done using the inbuilt "ask" functions is because they call a set of turtles in a random order, which makes it difficult to ensure the same nodes on the static network represent the same robots in the swarm on each iteration. When the "ask" function were used to map the robots to the network nodes, the nodes would not consistently represent the same robot making it impossible to identify any temporal patterns in the swarm. The above solution was a way of calling each robot in a sequential order and mapping it to each node in the 3d network in the same order on each iteration.

Eventually, a simpler method was discovered for calling a set of turtles in order ("update-network" function).

```
to update-network
  ...
```

The robots in the swarm were ordered into a list ("swarm") using the inbuilt function "sort" which uses the turtles' internal turtle ID numbers ("who" numbers) to order them.

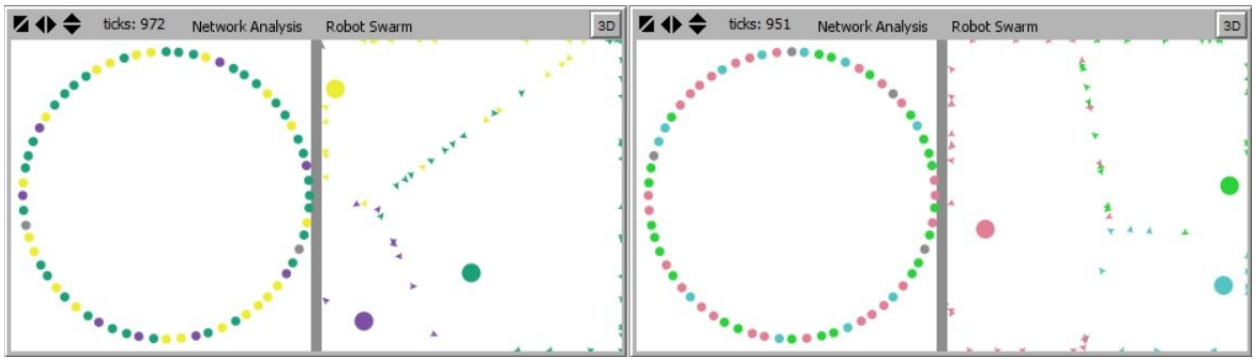
```
  ...
  let swarm sort robots
  ...
```

An empty list ("swarm_colors") is then created which will store each robot's internal state (i.e. their colour) according to their sequential order. Then each ("foreach") turtle in the list "swarm" is asked in turn ("ask ? [..") to add their colour ("lput color...") into the list of colours ("...swarm_colors").

```
  let swarm_colors []
  foreach swarm [ ask ? [ set swarm_colors lput color swarm_colors] ]
```

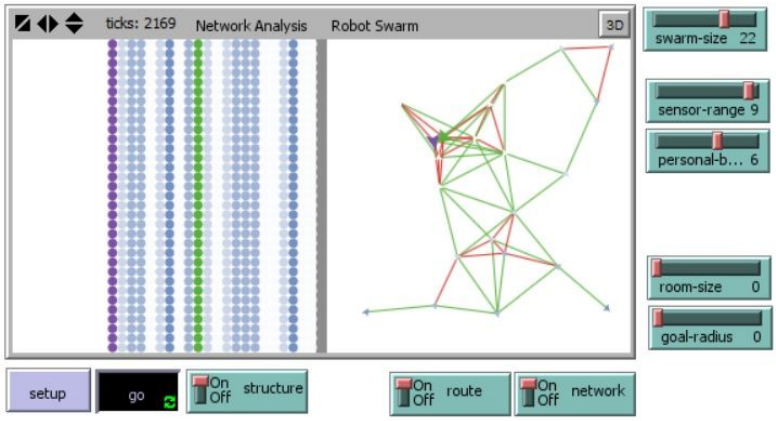
Then another ordered list is created ("network") of all the nodes in the 3D network. Each of the nodes ("foreach") in the list ("network") is asked in turn ("ask ? [..") to change their colour to that of their equivalent robot, using the ordered list of colours created earlier ("swarm_colors"). The variable "N" is used to keep track of which node or robot is currently being asked.

```
  ...
  let network sort nodes
  let N 0
  foreach network [ ask ? [ set color (item N swarm_colors)  ] set N (N + 1)] ...
```



(IV.5.5.E) 1D representation

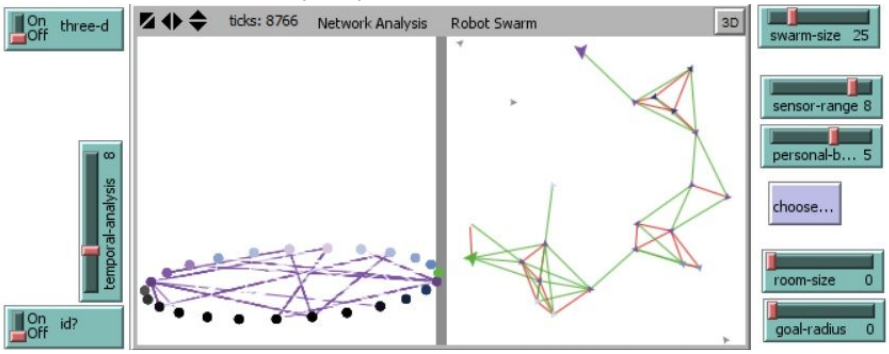
The next question to address was how to display the development of the swarm (as a static network) over time? The temporal component of the analysis is easy enough for a 1D system (like a 1D cellular automata), since you can simply stack the 1D system (a single line of network nodes) on top of one another. Then they would scroll off screen to continually update the swarm's temporal development.



This idea was tried, but was inadequate to represent the 2D swarm. This oversimplification (collapsing the 2D swarm into a 1D row) lost invaluable features, such as the network links (robot-robot interactions or virtual forces) which is where the computational mechanisms are believed to be embedded.

(IV.5.5.F) 3D Static Network representation

If the same idea was applied to the 2D static circular network, the changing network links over time could be displayed by stacking on top of the network in a 3rd dimension. Therefore, some sneaky graphical manipulation was required to give the illusion of an added dimension in order to display the static network from a 3D perspective.



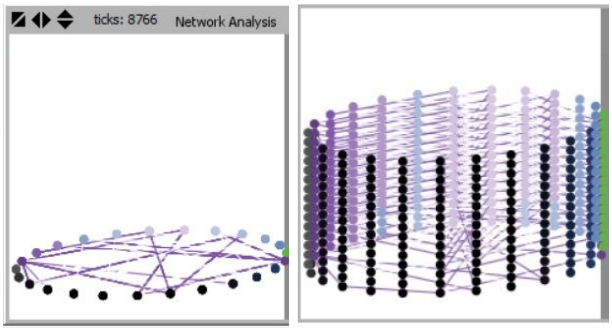
To convert the position of the nodes from 2D to 3D, the x and y coordinates of each node was recorded as "x3" and "y3" with an imaginary third coordinate "z3" set to 0.

```
to setup-network ...
    ask nodes [...
        set x3 xcor   set y3 ycor   set z3 0... ]
```

Then an equation was used to map the 2D xy coordinates into a 3D xyz coordinate system (the converted network was also shifted down to the bottom of the screen by moving it -10 in the y-direction).

```
setxy (x3 + (y3 / 4))  (z3 + (y3 / 4) - 10)
```

To represent the temporal element, the prior network is pushed upward and stacked on top of the current network to show the change from the current to the past time intervals.



All the nodes in the static network are asked (“ask nodes”) for their colour (“color”), x-y coordinates (“xcor” “ycor”) and other details.

```
ask nodes [
        let col color
        let xc xcor
        let yc ycor
        ...
```

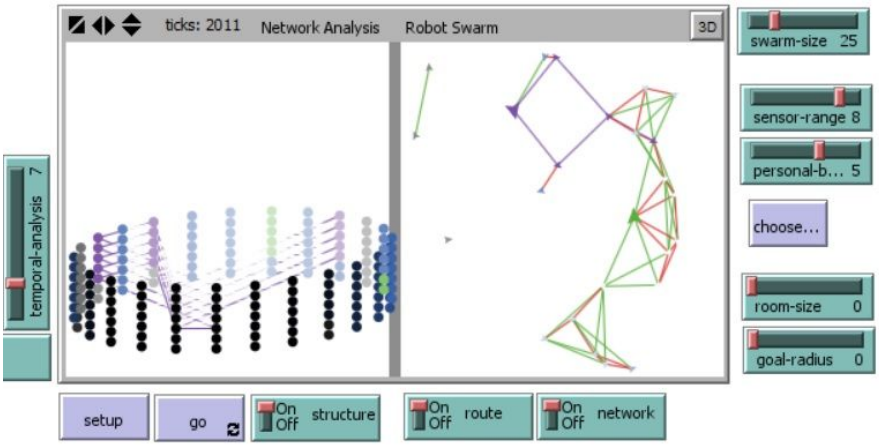
A new node is then cloned on top of each node (“sprout-oldnodes”) and all the information is copied across to the cloned node.

```
        ask patch-here  [
                sprout-oldnodes 1 [
                        set z3 0
                        setxy xc yc
                        set color col
                        set shape   "circle"
                ]
        ] ...
```

Then each of the cloned nodes (from this iteration and the previously cloned nodes) are asked to move up the screen slightly (“ask-concurrent oldnodes…”) or to be destroyed (“die”) if they have reached the limit of the screen.

```
...
ask-concurrent oldnodes [
        ifelse z3 <  time-interval
        [setxy xcor (ycor + 0.01) set z3 (z3 + 0.01)]
        [die]
]
```

Only the nodes at the bottom are updated on each iteration to keep up to date in reflecting any changes occurring in the robotic swarm. The advantage of keeping all the past nodes unchanged (except for moving their position upward) is that their links can remain unaltered and less processing is required.



(IV.5.5.G) 3D Dynamic Swarm representation

The simplest way of representing a dynamic network or the actual swarm in three dimensions (for temporal analysis) is to make a duplicate copy ("hatch-nodes") of each robot in the swarm.

```
to position-node
        ask robots [
                hatch-nodes 1 [ ...
```

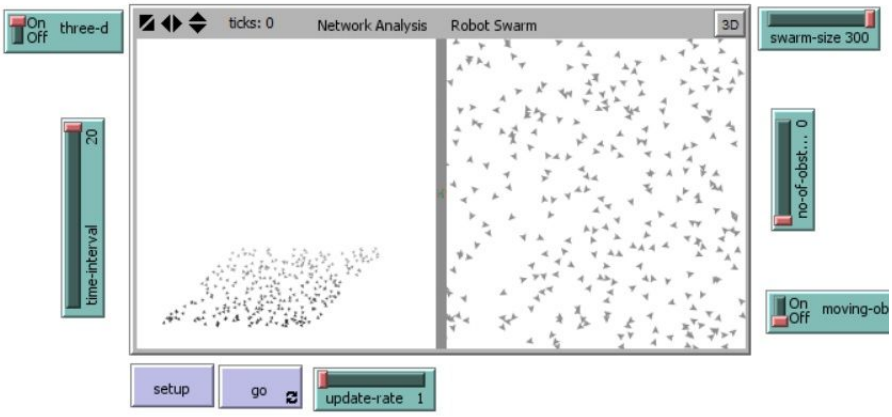
Then to shift the position of the copied robot to the left of the split-screen ("setxy ((xcor - 52) / 2) ..."). This will create a duplicate of the swarm in the left 'analysis' window. Thereafter, each of the cloned robots will be scaled to half their size to allow more space for the three-dimensional transformation to take place.

```
...setxy ((xcor - 52) / 2)  (ycor / 2)
set size 0.5 ...
```

Finally, each has their new coordinates calculated to give the 3d effect. The colours are shaded to enhance the effect and the 3d swarm's position is moved down ("-10") the bottom of the window.

```
...set x3 xcor
set y3 ycor
set z3 0
set color scale-color color y3 -12 20
setxy (x3 + ( y3 / 2) ) ((z3 + ( y3 / 2) - 10))  ] ]
```

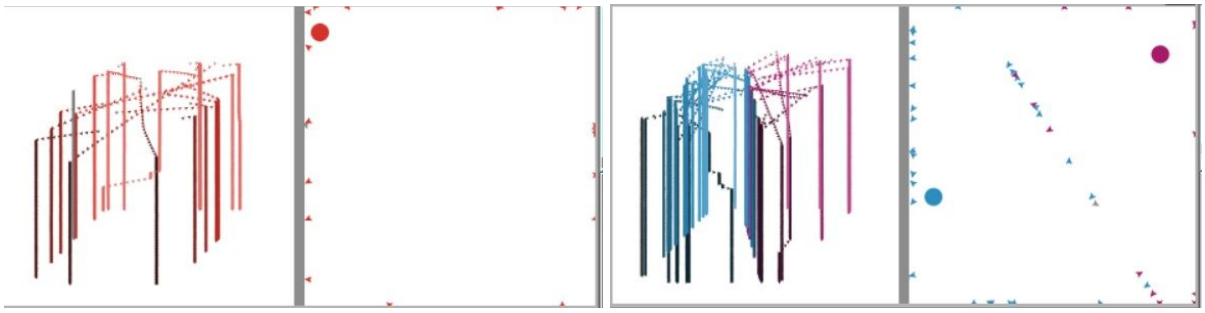
```
End
```



The simplest way to update the 3d swarm to show the changes over time is to move the entire swarm upward slowly on each iteration ("setxy xcor (ycor + 0.1)") and destroy the robots which go off screen ("[die]").

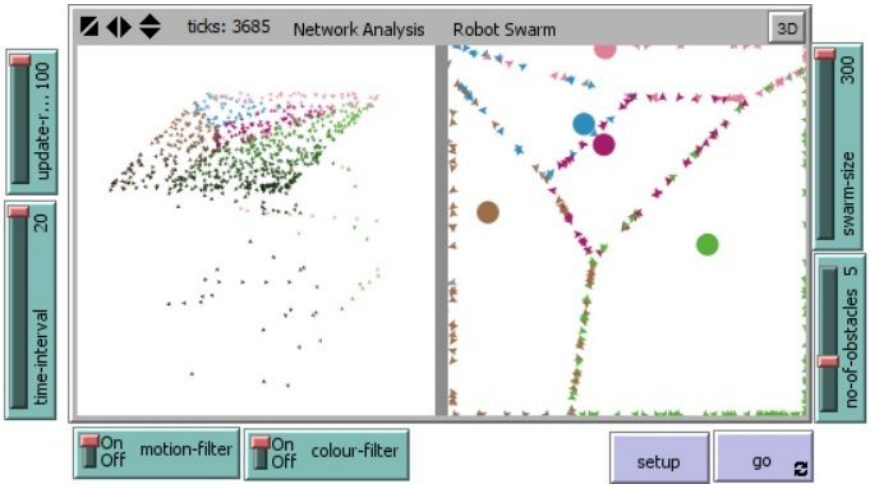
```
to temporal-analysis
        ask nodes [
                ifelse z3 <  time-interval
                [setxy xcor (ycor + 0.1) set z3 (z3 + 0.1)]
                [die]
        ]
        position-node
end
```

Then the 3d swarm shown at the bottom of the window is continually re-cloned ("position-node" function explained above) at the robot's new position to reflect the changes occurring in the real swarm (on the right of the split-screen in the robot-swarm window). This saves the trouble of keeping lists to map individual robots to specific nodes.



The user also has the option to filter the 3d swarm so that only the robots which change position are viewed (“motion-filter” function) or only the robots which change internal state / colour are viewed (“colour-filter” function)

```
ifelse motion-filter or colour-filter[
    if motion-filter [ask robots with [not (xcor = x3 and ycor = y3)]
        [position-nodes ] ]
    if colour-filter [ask robots with [not (color = c)]
        [position-nodes ] ]
]
```



(6) Software Appraisal

(IV.6.1) Virtual Forces Model

The software successfully simulated a robotic swarm possessing two distinct emergent behaviours. The first was external and related to the swam’s motion. The motion of the swarm, when first deployed, would flow smoothly around unforeseen obstacles as it spreads across the environment, resembling a ‘liquid’ state. Eventually it would crystalise into a ‘solid’ state whereafter the robots would remain more or less fixed in a lattice or grid formation. The swarm was even developed to allow robots to leave chemical / heat trails in the terrain to investigate stigmergic effects on the emergent behaviour of the motion. One effect noticed was that the trails was able to re-liquify a solidified swarm (useful to delay the robotic swarm from crystallising or to change the position of the swarm after it has solidified into a fixed position). The second emergent behaviour made use of the internal, binary states of robots to find the shortest route between any two given points on the swarm. Robots would light up to display the shortest path. After the swarm has spread out across the terrain and solidified into a sensing grid, this second emergent behaviour

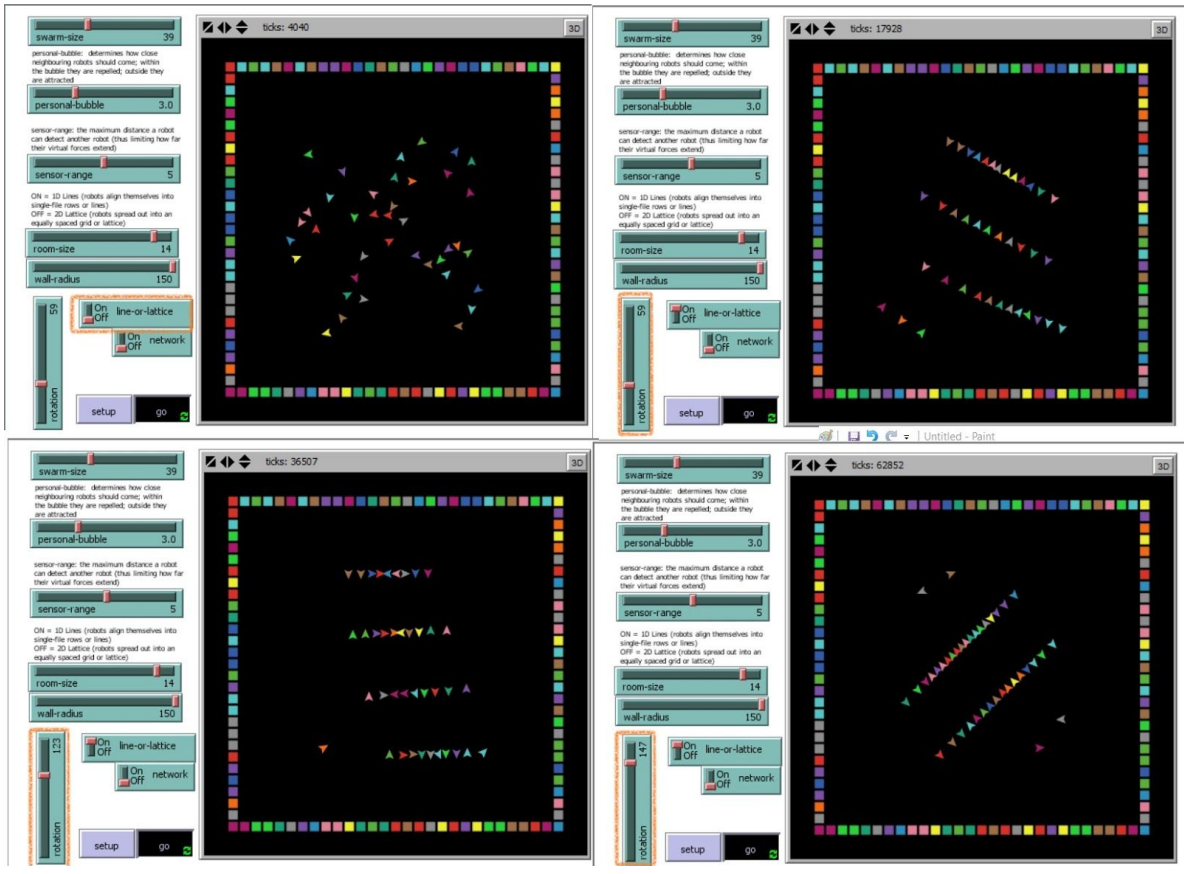
could be used to show the shortest route across two points on the terrain, like a map.

When robots would calculate the virtual force of a nearby object (i.e. a neighbouring robot, goal or obstacle), the force was initially calculated to be directly proportional to the inverse of the object's distance (i.e. the closer the object and the smaller its distance, the stronger its virtual force would be). However, the resultant robotic behaviours were not stable and robots would seldom settle into a fixed formation, constantly remaining in a 'liquid' state. When the force calculations were adjusted to be the inverse of the square of the distance (similar to gravitational forces), the behaviour became more stable and smooth. Initially there was a problem with the calculation which translated the net virtual force to robotic motion. The virtual force contains both a magnitude and direction which can be directly translated into a direction and velocity, however, the robots would continually line up into lines instead of uniform lattices as expected. The source of the problem was eventually identified in the code; the angle between the robot and its neighbouring object ("Th") was incorrectly being used to determine the resultant direction of motion.

ask other robots in-radius sensor-range [...let Th (towards myself) ...

The robot must also consider its current direction when adjusting to the new direction. Although this bug was easily fixed (*...let Th (towards myself + heading)...*), the mistake had unwillingly lead us to the discovery of a new emergent behaviour: the spontaneous formation of rows. Thus an option was added to the program ("line-or-lattice") to incorporate this accidental feature and allow the user a choice whether they wish the swarm to form rows or a lattice / grid across the environment (Slightly modified so that an option to rotate the orientation of the rows was given via the user-defined variable "rotation").

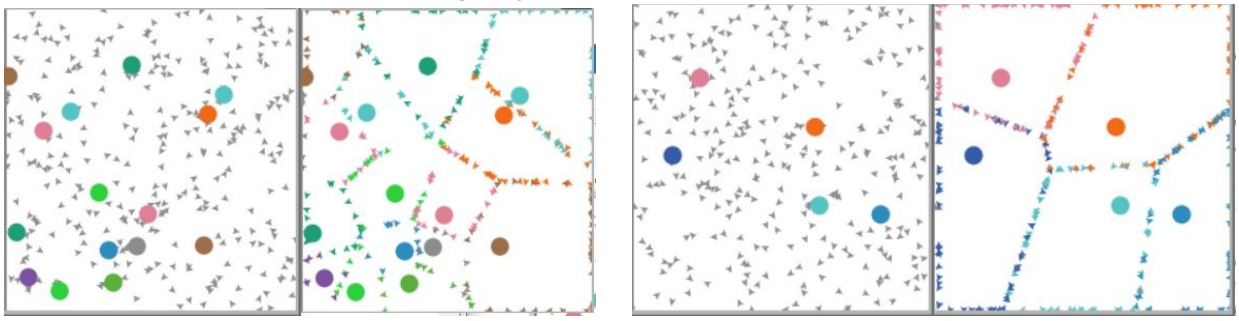
...if line-or-lattice [set Th (towards myself + rotation)]...



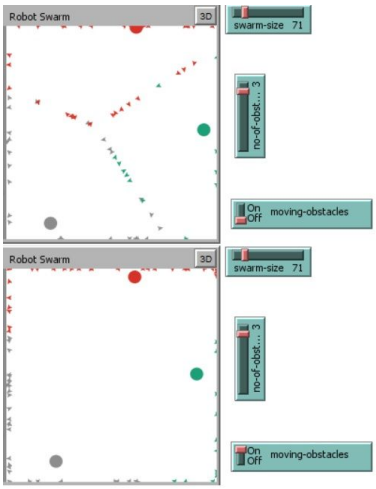
The advantages of the virtual-force method over the evolutionary approach is that robots only run on a single equation (which allows them to interact with other robots, avoid obstacles and find goals) as opposed to a set of rules; a rule for each situation. However, the equations do require more processing power such that the robots require an internal model and extensive testing to fine-tune the virtual-force parameter so that the different attractive or repulsive forces balance out to produce the desired emergent behaviours.

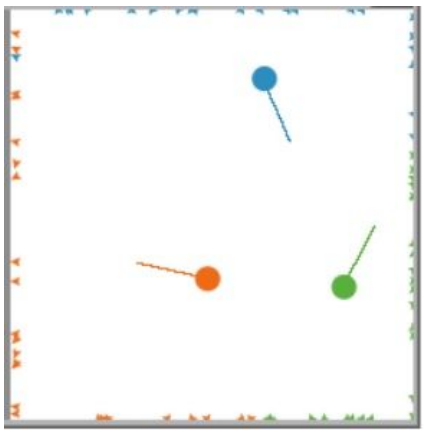
(IV 6.2) Voronoi Model

The next robotic swarm's emergent behaviour is immediately evident and the swarm very quickly converges into a voronoi pattern (due to the extreme simplicity of the voronoi model). The homogenous robotic swarm consists of purely reactive robots (unlike the virtual-forces model, robots are not required to do calculations or use a memory to store any internal models) all running on a single rule which doesn't need to consider any neighbouring robots (the robots only consider the number of obstacles nearby to decide their behaviour). Robots begin by looking at their closest obstacle(s) and quickly count how many other obstacles are at this distance (normally there is only one 'closest' obstacle since all other obstacles are necessarily further away than the 'closest' obstacle). If there is only one 'closest' obstacle, the robot continues to move around the environment l randomly. If it comes to a location that is directly between two or more obstacles wherein the robot detects many 'closest' obstacle (i.e. another obstacle is equally close as the closest), the robot will do nothing and remain in its place at the midpoint between obstacles. Eventually, all the robots will come to rest at the midpoints between obstacles and thus the robots emergently form their boundaries.



Despite the simplicity of the rule, the swarm behaviour which emerges is intelligent. The swarm can be used to equally divide up each obstacle by forming clear boundaries between each obstacle using the robots (boundaries form patterns resembling a voronoi diagram). Since there are a number of ways to divide up the obstacles, the exact position of the boundaries may differ each time. Thus, if obstacles are suddenly added or removed, robots immediately react to reform new boundaries around them. The disadvantage of the





voronoi model is that it doesn't work well with moving obstacles or dynamic worlds: when introducing obstacles that continually move (i.e. are not perfectly still), the boundaries become unstable and the robots disperse toward the edges of the world.

(IV 6.3) Genetic Algorithm

The genetic algorithm successfully evolved several strategies (using elite inheritance, single-point crossover and mutation), some of which lead to swarms with emergent behaviour. The final genetic algorithm evolves strategies with a more simple, direct approach which contained nothing but various robot commands (e.g. move forward 10, turn right 90, etc). A prior version used a more complex strategy which contained a mixture of "values" and "operators" (e.g. multiply, add, minus, etc). The final strategy that was being evolved only contained a single robot rule of varying length (the length could be set by the user-defined variable "L"), however, the initial plan was for the strategy to contain a ruleset of three rules; each to be used in a different scenarios (i.e. rule 1: if the robot has no other robots nearby, rule 2: if there are other robots nearby, rule 3: if there are robots too close).

The various fitness functions tested had different strengths; the goal-based fitness was excellent at evolving individual robots to seek goals while avoiding obstacles, however, not good for evolving swarms that could do that via emergent behaviour. The intrinsic fitness function based purely on survival (i.e. robots would die if they collided and so those that did not collide were favoured for evolution) managed to evolve swarms which either moved coherently or were static grids - which is a very basic form of emergent behaviour, especially when compared against the virtual forces method which could do the same while also seeking goals and avoiding obstacles. Fitness functions based on snap produced swarms with either jerky or smooth behaviour (depending if high or low snap was favoured for evolution) however, the swarm's behaviour was not necessarily coherent nor emergent. However, when measuring the individual robot's snap as a deviation from the average swarm's snap - the behaviour became more coherent - although still basic compared to the emergent behaviour produced by the virtual forces method. Furthermore, the snap method required the robot to remember more variables (i.e. past velocity and acceleration) to calculate the snap. Lastly, the evolved emergent behaviour was limited to the motion of the swarm, and not their internal states to produce the shortest route. This could be evolved too, however, it would need a separate fitness function to evolve the swarm to meet such emergent conditions.

Reflecting upon lessons learnt, one realises that automatic search algorithms (like genetic algorithms) may seem attractive because they seem to be the easier option. However, in actuality, they require far more effort to be fine-tuned and set up (i.e. selecting the appropriate fitness functions, inheritance method and genome representation) before a strategy can be successfully evolved. Even then, the evolved rulesets may be more basic than those obtained using the hand-designed methods. Automatic search techniques (i.e. genetic algorithms) become more useful when the search space is very large and complex or alternative solutions are unable to be designed by hand.

(IV 6.4) Analysis Software - version 1

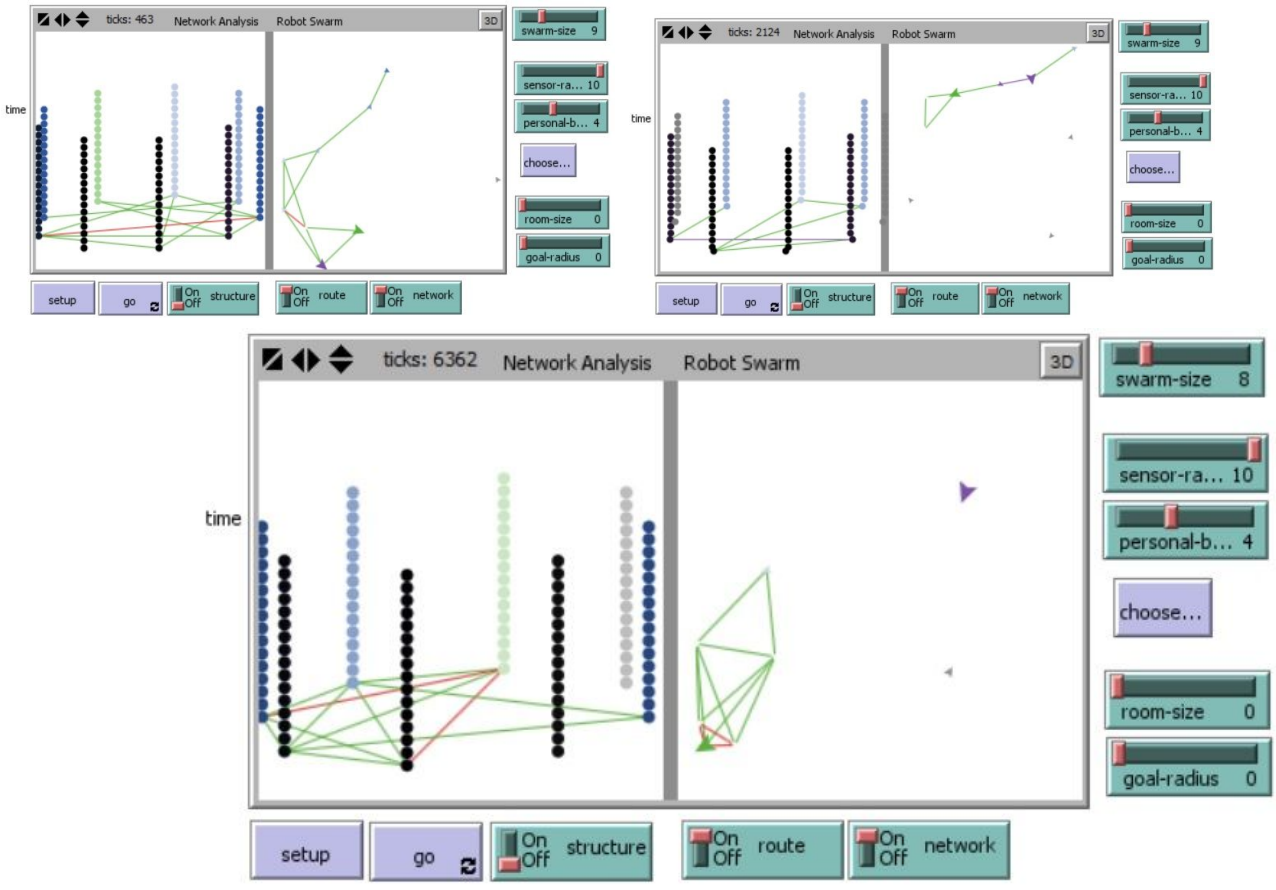
The spatio-temporal analysis software modified netlogo display into a split-screen:

- On the right-hand side of the screen, a robotic swarm is displayed with two distinct emergent features; 1) a fluid and coherent motion which reactively avoiding obstacles or clustering around specific terrain features ("goals") while simultaneously spreading out evenly across the environment into a solid formation to allow the swarm to act as a sensing grid or external map

of the terrain it is spread across. 2) the ability to find and light up to display the shortest route between any two given points on the terrain. (All this is done emergently, without the use of global information, central coordination or prior planning). There is an option to switch the representation of the robotic swarm into a dynamic network (i.e. the robots become mobile network nodes and their wireless local interactions are displayed as network links).

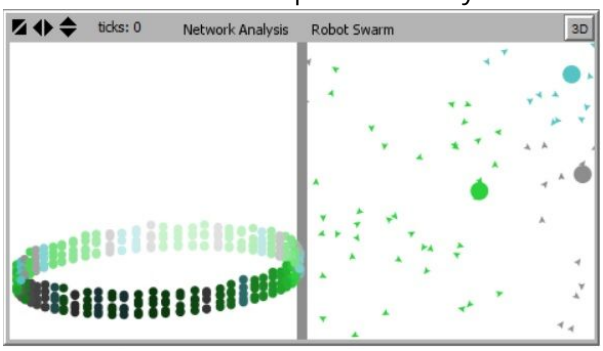
- On the left-hand side of the screen, a 3d static network representation of the robotic swarm is displayed (which allows the change and development of robot-robot interactions to be viewed over a span of time while discarding robot movements).

When trying to display the network links on the 3d static network, they had to be copied directly from the links on the dynamic network (which display the robot-robot interactions of the robotic swarm). This was done easily enough, however, the task became fairly complex when the network links needed to be stored and displayed over a given span of time to provide a large enough window for the user to clearly see the temporal development of the network and how the links changed. This was believed to be key in identifying potential spatio-temporal structures in the robotic swarm. At first the 3d static network successfully displayed past nodes, but not any past network links!

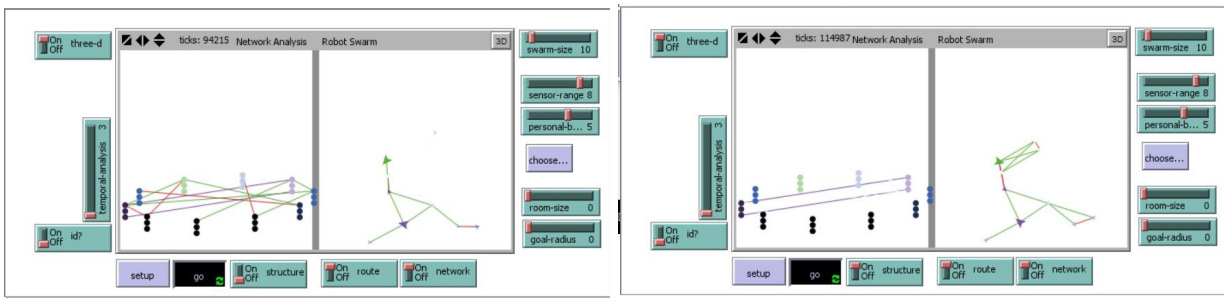


The problem was resolved by adopting an alternative, simpler method to update the network nodes and links: The network at the base is the only one that updated with the swarm. On every time interval, a clone of this network was made. All the cloned

networks were also moved up slightly on each iteration. This method was simple and worked well if the nodes were moved upwards in very small increments.

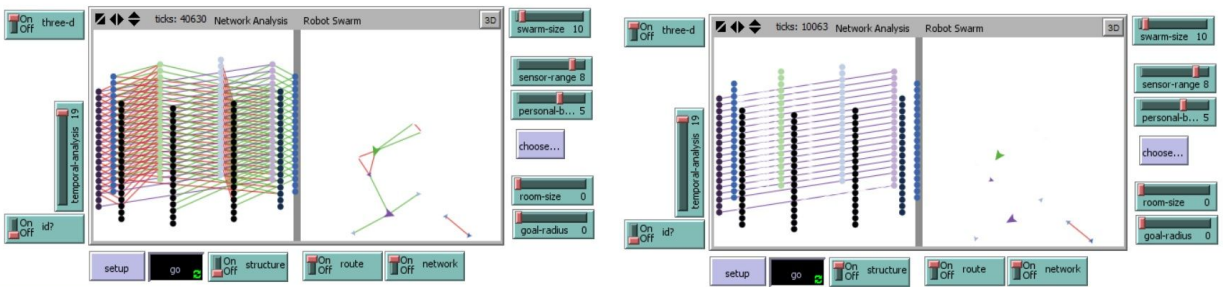


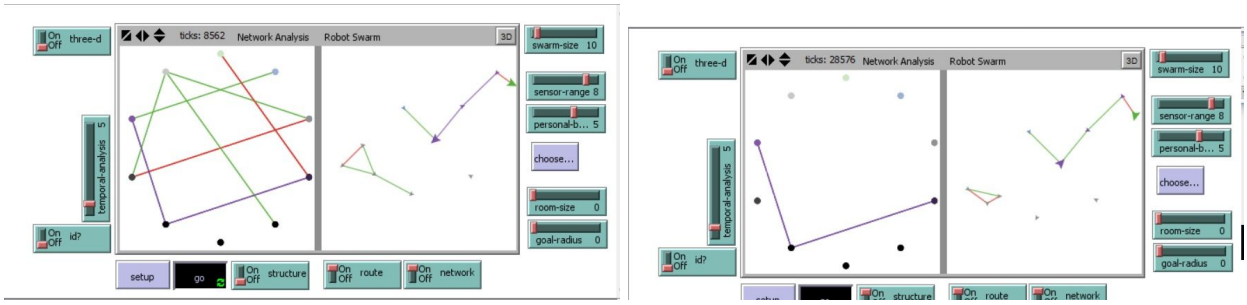
One strength of this analysis method is that the circular layout of the static network allows all possible links (i.e. robot-robot interactions) to be seen clearly, (which was not the case when viewing the swarm as a dynamic network because some links were obscured due to robot overcrowding). The third dimension also allows network links' temporal developments to be clearly seen. However, the 3d effect means that the display is angled, making the links near the rear more difficult to view. This display method also assumes that the development of the network links (robot-robot interactions) are where the embedded spatio-temporal computational mechanism are to be found. By converting the dynamic swarm into a static network, it assumes that the motion of the robotic swarm does not contribute to these computational mechanisms.



Furthermore, the display can become overly complex as the temporal element is increased (i.e. if the development of the network is viewed over too large a time period) or the number of nodes and possible node links is too high (i.e. the robotic swarm is too large).

To combat this problem, an option introduced which allows the user to view the network links related to the swarm's emergent route only (assuming this is the emergent behaviour we are analysing). However, by ignoring other network links, we may be discarding valuable information which can help to identify the spatio-temporal structures.

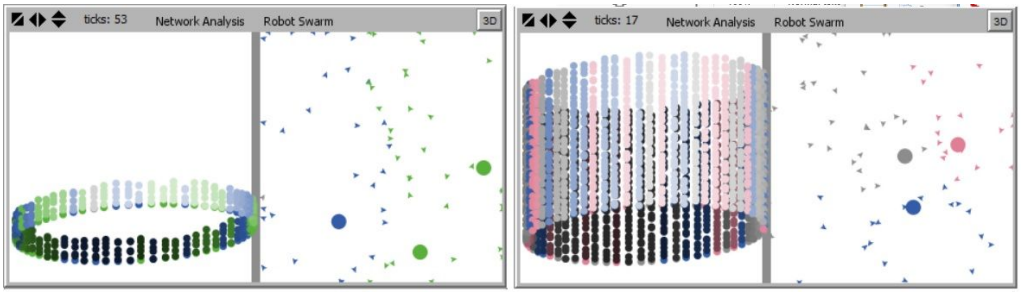




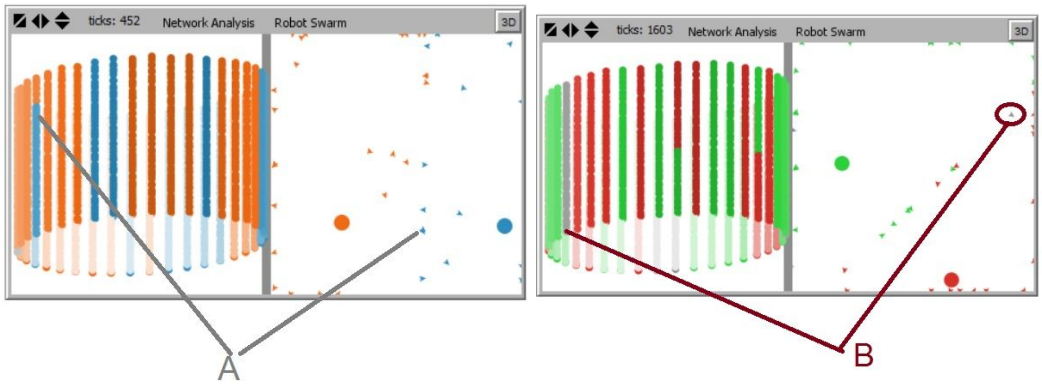
This initial version of the analysis software was unsuccessful at identifying the embedded spatio-temporal structures related to the hidden computational mechanisms hypothesised to underlie the emergent behaviour of the swarm.

(IV 6.5) Analysis Software - version 2

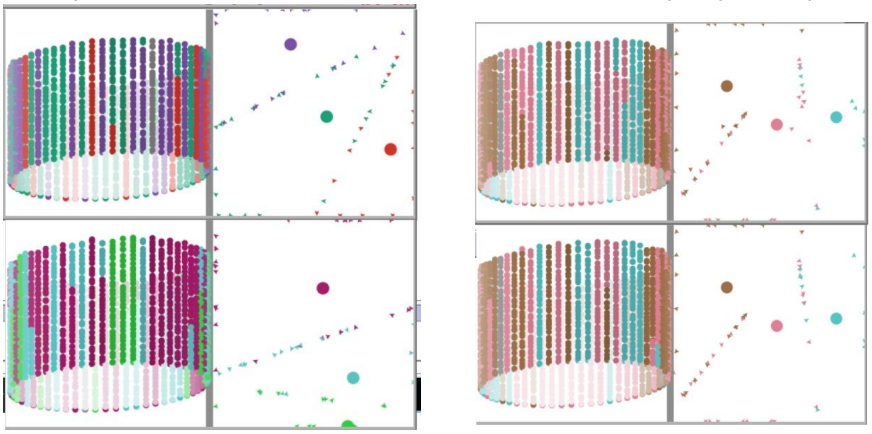
Upon analysing the voronoi model, we did not require any static network links (since its emergent behaviour does not dependent on robot-robot interactions). Its emergent behaviour arises via indirect interactions with obstacles as opposed to other robots. To indicate the indirect robot-obstacle interactions, robots (and their corresponding nodes) adopt the colour of their nearest obstacle. Analysing the colour changes of the robots over time is actually analysing the development of robot-obstacle interactions over time (without the need to analyse the nodal movements or links). It is possible that spatio-temporal patterns can be detected here.



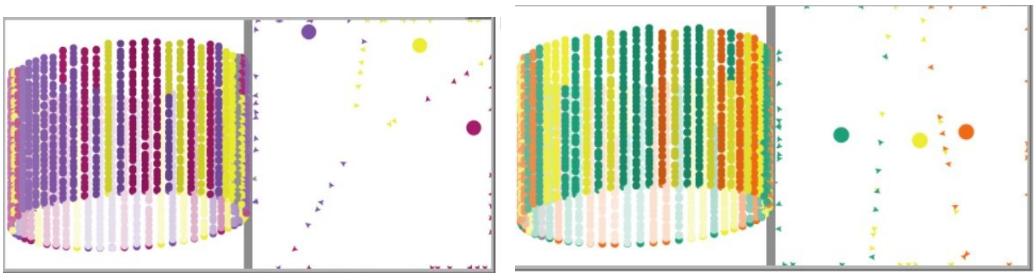
The problem was that the temporal window was too small to identify any significant developments occurring in the swarm. The solution was to reduce the rate at which the 3d network was updated. Rather than updating it on every iteration, the analysis software updates it every N iterations, as if it is taking snapshots at time intervals to allow a broader overview of the behavioural developments spanning over a wider time frame.



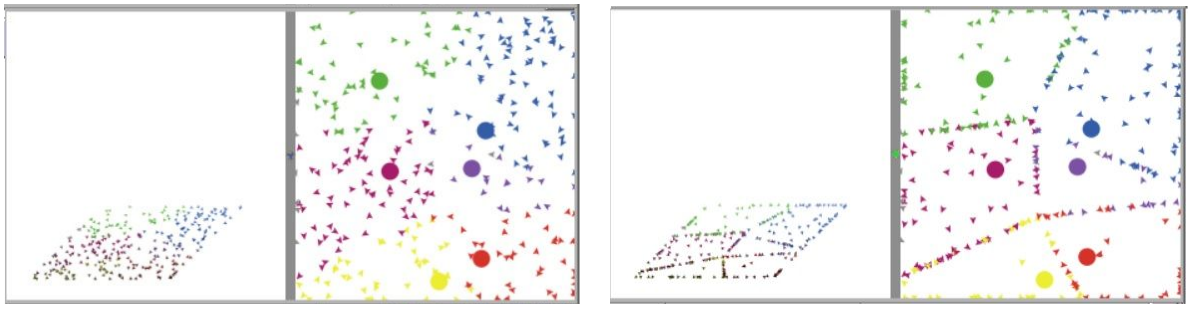
Now the developmental changes were becoming more evident in the 3d static network (A and B show nodes changing colour, corresponding to specific robots in the swarm changing internal state) increasing the chance of identifying spatio-temporal patterns. Spatio-temporal patterns did seem to be evident, however the representational method wasn't suitable to identify any clear patterns.



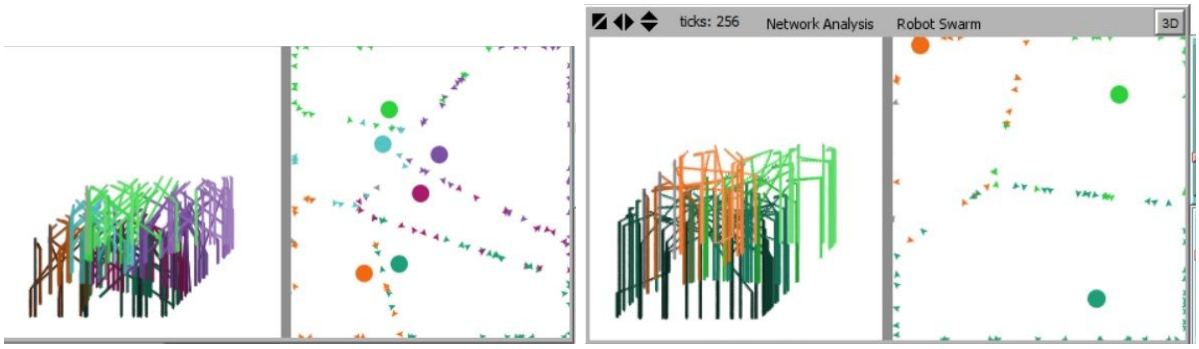
Perhaps the circular shape of the static network was actually distorting the shape of the spatio-temporal patterns.



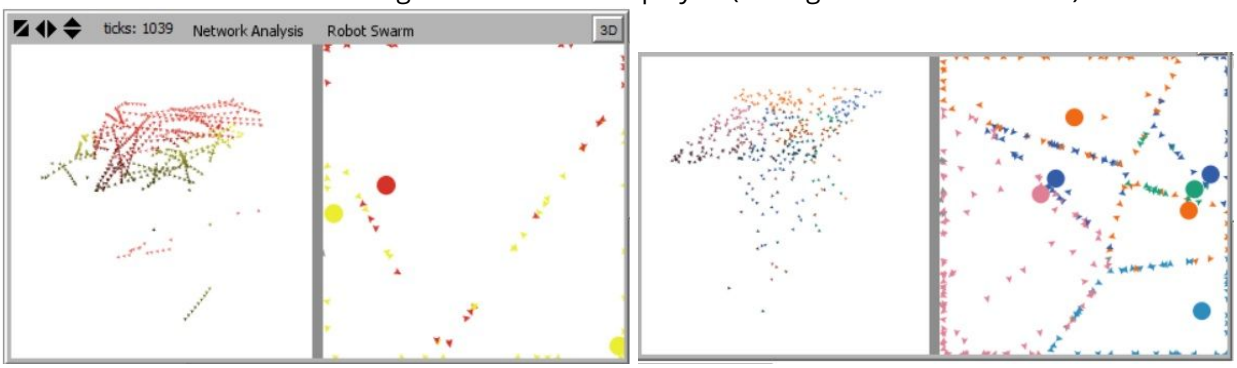
Since there was no need to analyse the links of the network anymore, the circular shape of the static network had now become redundant. Furthermore, the movements of the individual robots may be an important component to analyse and so the static network need not be used to represent the robotic swarm. Instead, a dynamic network or the swarm itself could be used directly in the analysis. Unfortunately, the 2D view of the robotic swarm can only ever show the swarm in real-time and if the development of the swarm is to be displayed over time, the swarm would need to be represented in 3d (to allow the Z-axis to be used for the temporal element - i.e. to show the changes in the swarm over time). Therefore, the right window of the split-screen displayed the 2d robotic swarm in real-time, whereas the left window displayed the 3d view of the robot swarm (as opposed to the 3d static network used in prior versions).



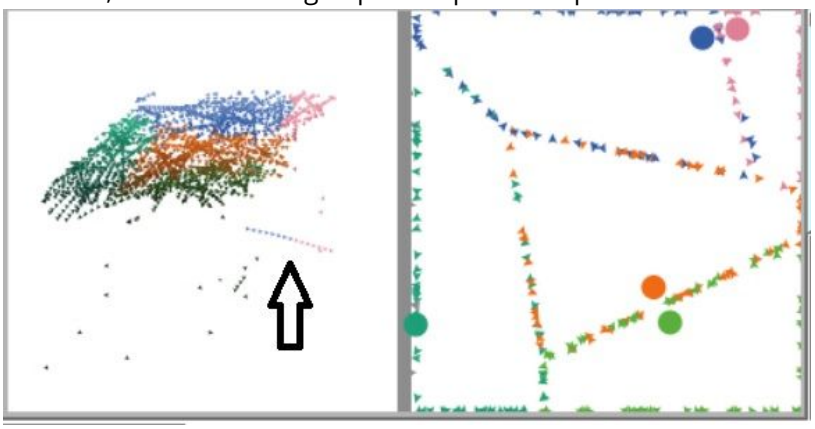
This worked well for the first iteration or two, however, after so many time lapses the analysis quickly becomes cluttered and complex, making it difficult to understand.



In an attempt to reduce the clutter, filters were added. The first filtered out static robots so that only robots which changed position over time would be shown in the analysis window (see left screenshot below). The second filter only allows robots which have changed colour to be displayed (see right screenshot below).



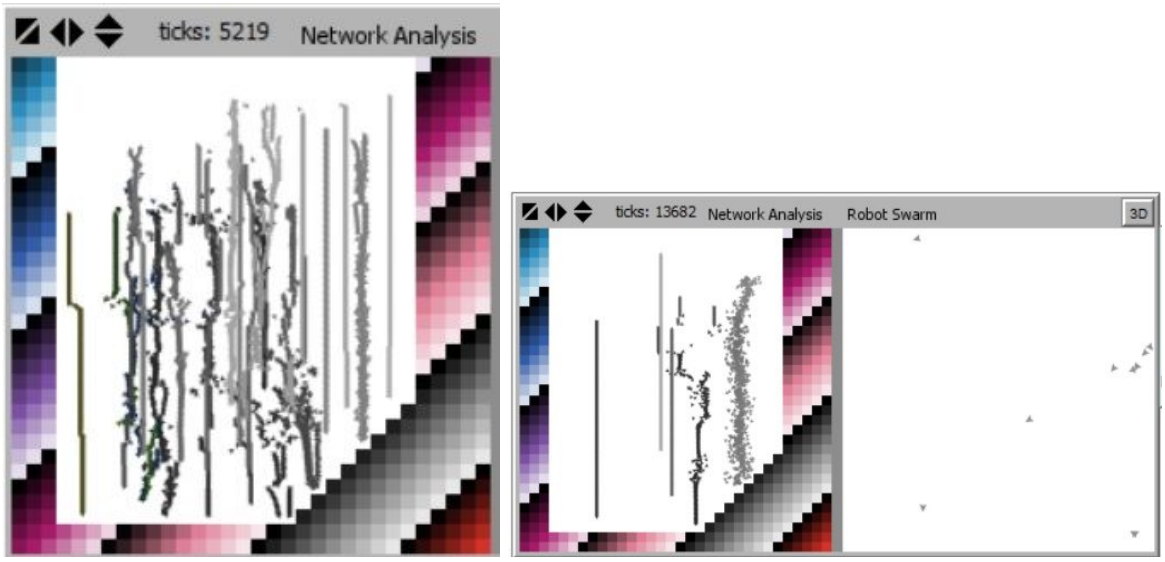
Despite the clearer analysis, no spatio-temporal patterns were revealed. There were, however, the occasional glimpse of spatio-temporal mechanics beginning to form:



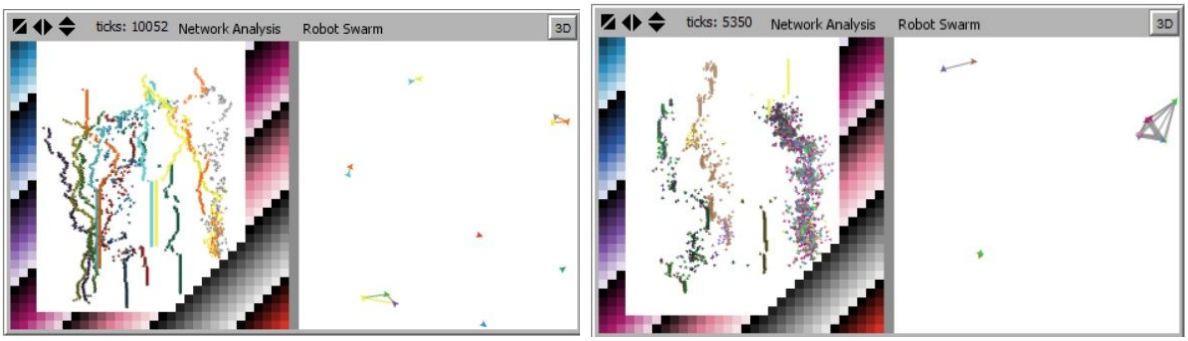
The lack of spatio-temporal patterns may have been difficult to see due to the quick convergence times of the voronoi model.

(IV 6.6) Analysis Software - version 3

When using the 3d swarm representation to re-analyse the virtual-forces model, an abundance of spatio-temporal patterns were discovered! Success at last.



A wide range of different types of spatio-temporal patterns were identified as well as some of their collision-based computational behaviour (i.e. merging, splitting, etc).



(V) Critical Analysis

Spatio-temporal patterns were successfully discovered upon analysing the emergent behaviour of robotic swarms, confirming our initial hypothesis that there exist embedded computational mechanics (i.e. the spatio-temporal patterns) which govern the emergent behaviour of robotic swarms. This section categorically presents examples of the various types of spatio-temporal patterns discovered, as well as some of their computational mechanics, with some commentaries to explain how each feature is believed to contribute toward the robotic swarm's emergent behaviour. The various findings are then compared against past research (conducted in Cellular Automata - wherein the theory of spatio-temporal structures and embedded computation was initially developed).

The following examples of spatio-temporal patterns were taken from the analysis of a robotic swarm operating on the 'virtual forces' model. On the right side of the analysis software's split-screen, a dynamic network representation of the robotic swarm was used to identify which robots were exerting an influence on one another (shown as dynamic network links). On the left side of the analysis software's split-screen, a 3D particle swarm representation of the robotic swarm was used to track the swarm's spatial movements (shown on the x-y axes) as it develops over time (shown via the z-axis).

(V.1) Types of Spatio-Temporal Patterns

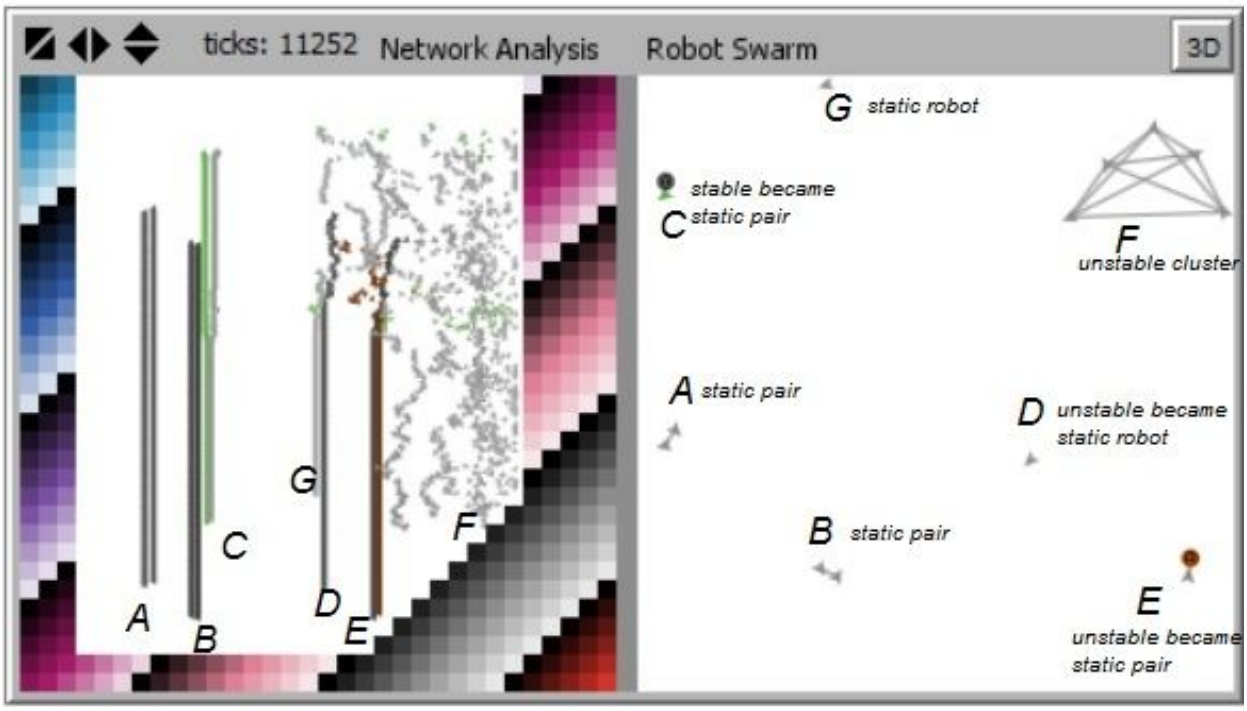
There were four distinct types of spatio-temporal patterns discovered during our analysis (which seem to correspond with the four classes of Cellular Automata):

1. Type I: Static Patterns (corresponding to Class I CAs: Fixed Point Attractors)
2. Type II: Stable Patterns (corresponding to Class II CAs: Periodic Attractors)
3. Type III: Non-Patterns (corresponding to Class III CAs: Chaotic Attractors)
4. Type IV: Cluster Patterns (corresponding to Class IV CAs: Strange Attractors)

(V.1.1) Type I: Static Patterns (Class I: Fixed-point attractors)

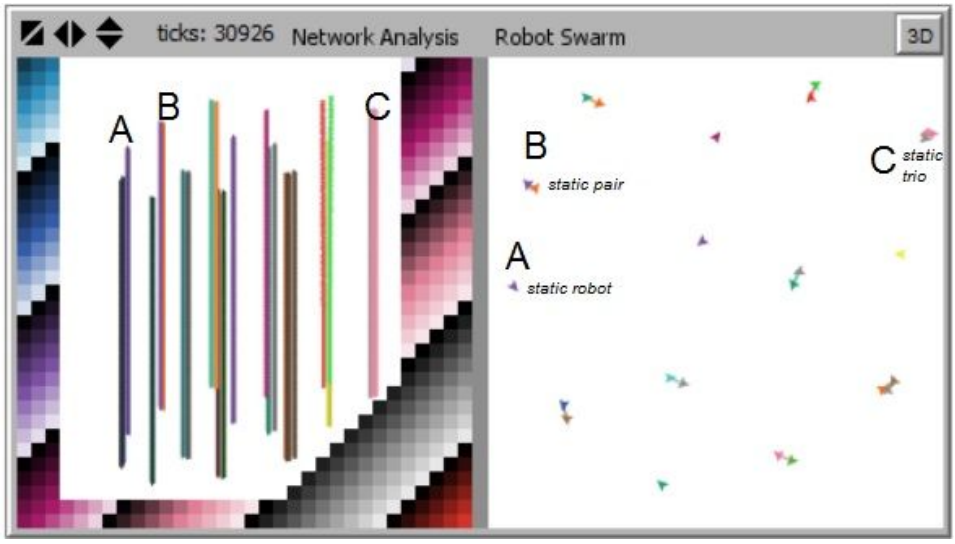
The first type of spatio-temporal pattern is static by nature; it remains unchanged over time (often shown as a straight line on the 3d swarm representation - see G below). This is commonly formed when a single robot becomes disconnected from the swarm and, having no external influences to react to nor virtual forces acting on it, it remains in the same position over time.

There were also examples of static patterns being formed via pairs of robots (depicted as two parallel straight lines on the 3d swarm - see A and B below). This may occur if two robots affecting one another mutually and their influences become balanced and cancel out. This could happen as the result of two robots wandering around until disconnecting from the swarm (see D below) or if a pair of robots settle into a balanced position after an initial chaotic dance (see C and E below).

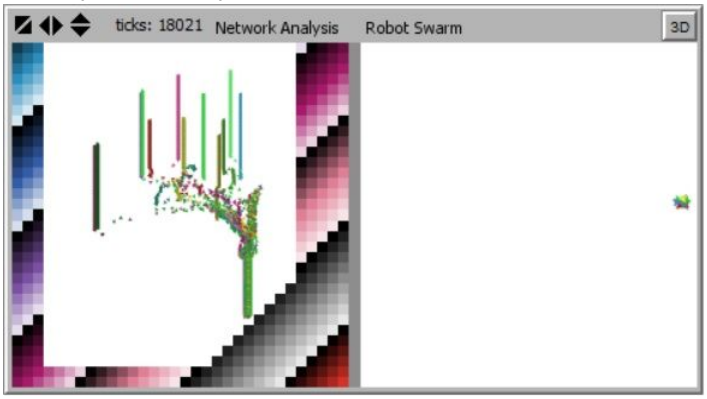


Static spatio-temporal patterns have sometimes been observed to form via robot trios (see C below), although it seems that static spatio-temporal patterns become exponentially rarer to find as the number of robots composing it increases. This may be because the positions for robots that robots settle into require a perfect symmetry to balance each other's

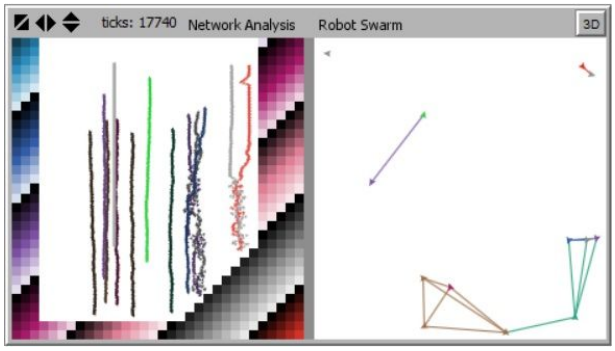
influence (even a slight perturbation is enough to shift a robot out of sync and thus cause the others to become unbalanced).

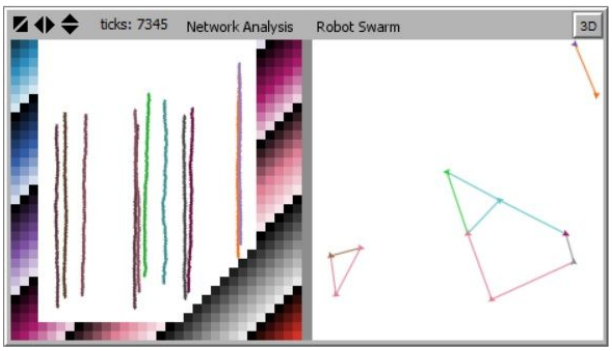


However, despite being rare, 'static clusters' (i.e. static spatio-temporal patterns formed of three or more robots) are theoretically possible. Technically, all the robots in the swarm could suddenly collapse into a single, static spatio-temporal pattern (see screenshot below: the entire swarm was forced to collapse onto a point in the environment to form a static cluster).

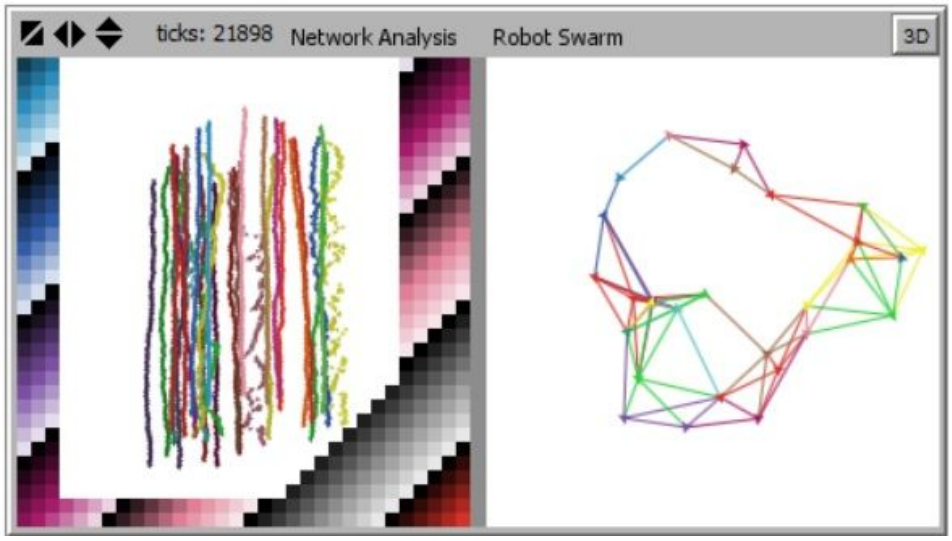


Although the above example was unnaturally induced (via the manipulation of robot parameters during runtime), there are examples of naturally-occurring static clusters (see two examples in the screenshots below. Left - a brown kite-shaped static cluster formed of four robots. Right - three separate groups of robots have solidified into static spatio-temporal patterns consisting of two, three and six robots respectively).



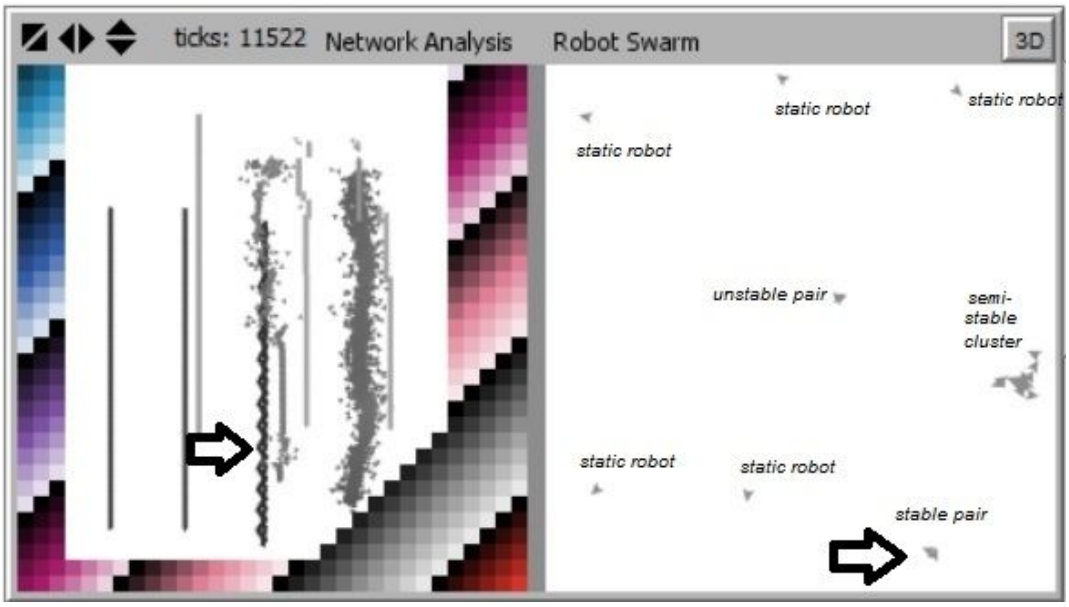


Sometimes a 'fluid' ('chaotic') swarm changes state into a 'solid' (forming shapes like lines, a voronoi pattern or a fixed lattice) whereby the entire swarm forms a super-massive static cluster. In the example below, the entire lower left side of the swarm has solidified into a static cluster retaining the same position over time (the upper right side remains slightly fluid due to the instability of the brown, blue and yellow robots - thus the shape of their formation in this part of the swarm changes over time).

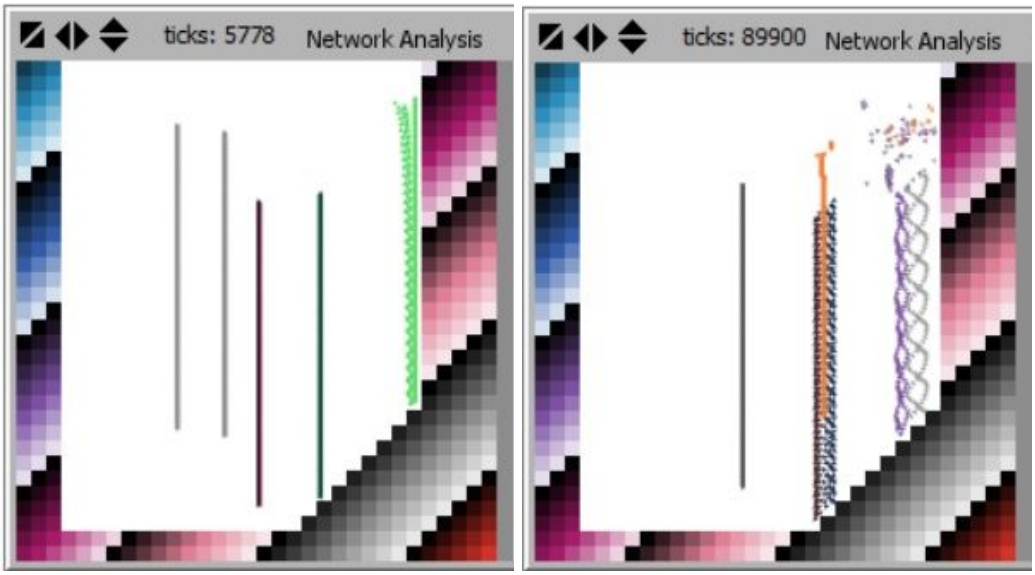


(V.1.2) Type II: Stable Patterns (Class II: Periodic Attractors)

The second type of spatio-temporal structure is a 'stable' (or cyclical) pattern (repeating the same, stable movements over time as in the example below) which corresponds to periodic attractors in CAs. These patterns cannot be formed by a single robot, since a single robot tends to remain still if nothing else influences it. Therefore, these patterns are most commonly formed from robot pairs caught in a stable dance around one another, reminiscent of binary stars moving back and forth around one another.

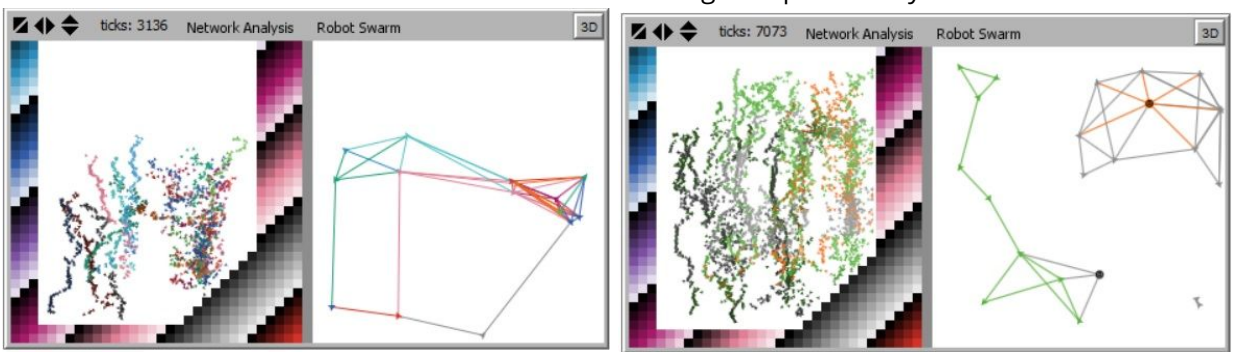


The exact shape of the pattern produced may vary (below are a few more examples to illustrate this point)



(V.1.3) Type III: Non-Patterns (Class III: Chaotic Attractors)

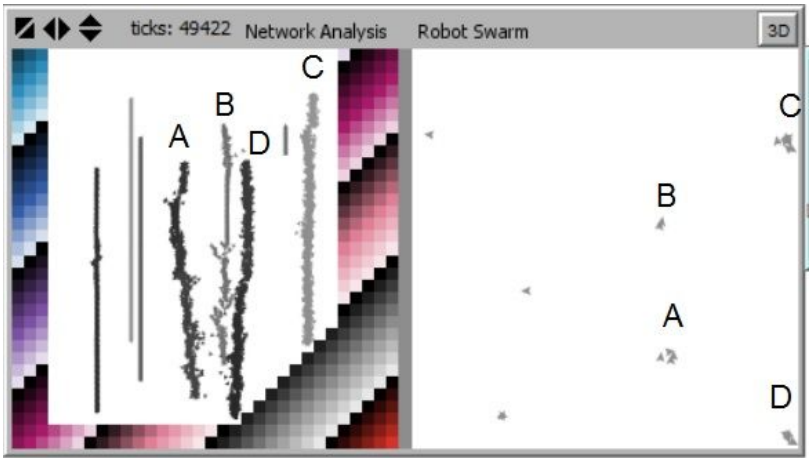
The third class of spatio-temporal pattern is when the swarm does not form any pattern at all; the robotic movements and interactions are unstable and extremely random such that no movements are exactly repeated over time. It is in a 'chaotic' state which changes unpredictably.



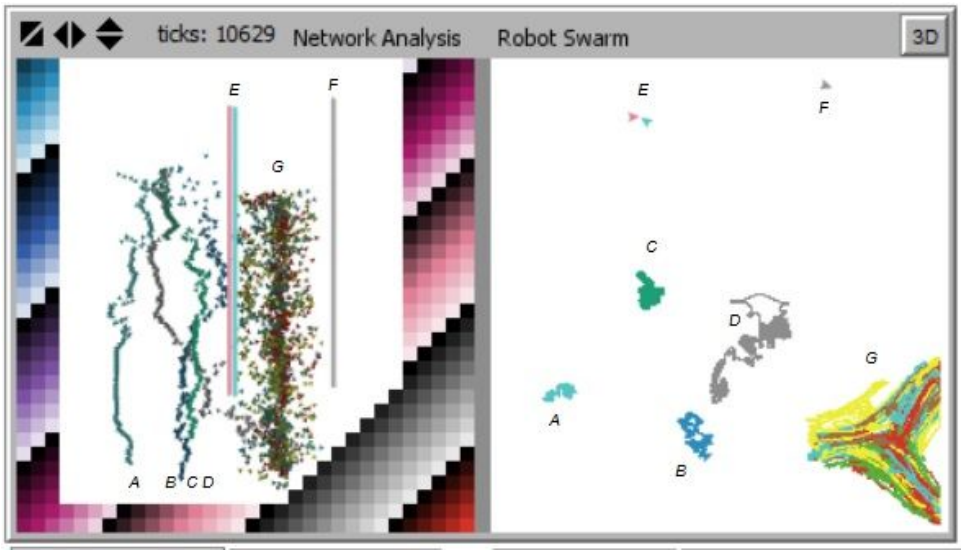
When the swarm is in this state its dynamics resemble a 'fluid', which is in direct contrast with the solid-like stillness of the first two types of spatio-temporal patterns (static and stable patterns).

(V.1.4) Type IV: Semi-stable Patterns (Class IV: Strange Attractors)

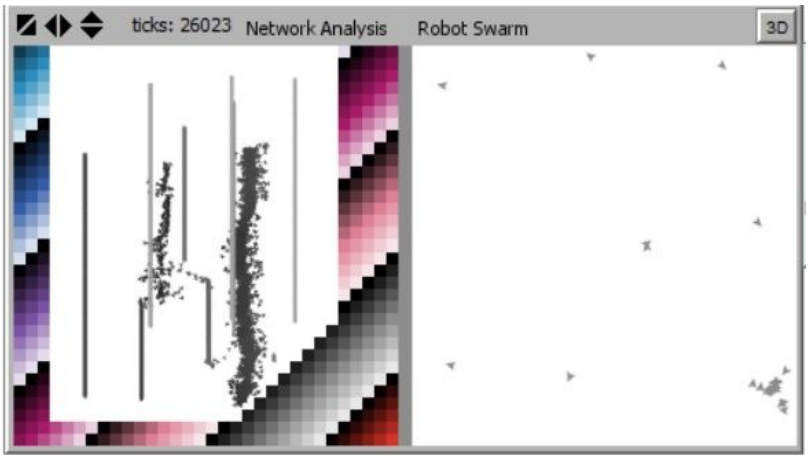
The fourth and final type of spatio-temporal pattern discovered is the semi-stable pattern (see A, B, C and D) which corresponds to the strange attractor in cellular automata. They are not completely random (like type III) nor completely orderly and repetitive (like types I and II). They balance delicately on the border of stability; between stable and unstable; between order and chaos (on 'the edge of chaos'). The pattern is complex; somewhat collected together, yet never repeating; resembling a localised pocket of chaos. To give an analogy in line with the solid and fluid descriptions given for the first three types of patterns, type IV patterns would most closely resemble the 'turbulent' vortex structures that sometimes appear within fluids (and then just as quickly disappear again).

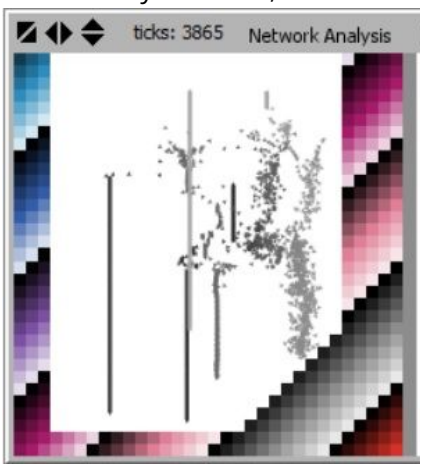


A semi-stable pattern (formed of a large cluster of robots) can be seen below (G) moving fluidly in an inexact tri-star shape. This is nicely contrasted against static (F) and stable (E) patterns, which remain refined to the same x-y coordinates over time.



The exact shape and size of type IV patterns can range from just one robot (as with patterns A, B, C, and D above) or large clusters of robots. The patterns can seem nearly-stable, mildly-exotic or wildly-unstable, etc:



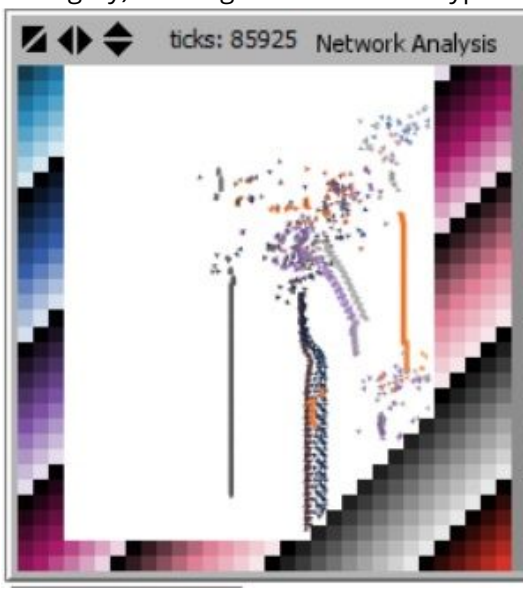


(V.2) The Behaviour of Spatio-Temporal Patterns

(V.2.1) Stationary Patterns

Solid, stationary spatio-temporal patterns (which resemble a vertical line or pattern) represent robots that remain at or around a particular x-y coordinate. Such patterns give the swarm stability and fix it into a specific formation or shape (which can serve as a memory - assuming information has been encoded into the specific patterns and formation of the swarm). It has been found that pieces of information are represented through the pattern of the spatio-temporal structure (and thus when the pattern changes, so does the information it is representing). Therefore, information can only be stored in the system if it remains unchanged as a type I (fixed-point static) pattern or a type II (periodic stable pattern).

Non-patterns (type III) cannot possibly represent information (since there have no recognisable patterns to represent any specific piece of information), yet they still serve an important computational purpose. Firstly their fluidity gives them the freedom to move across space and time and influence other parts of the swarm, changing nearby spatio-temporal patterns (and the information they represent) in their paths. Secondly, they act as random seeds from which new information can be created or modified (i.e. out of the chaos, spatio-temporal patterns may suddenly form or transition from one form to another). Below is an example of two static 'type I' patterns (black line and orange line) as well as two stationary type II patterns (purple and grey) forming out of a chaotic type III non-pattern.

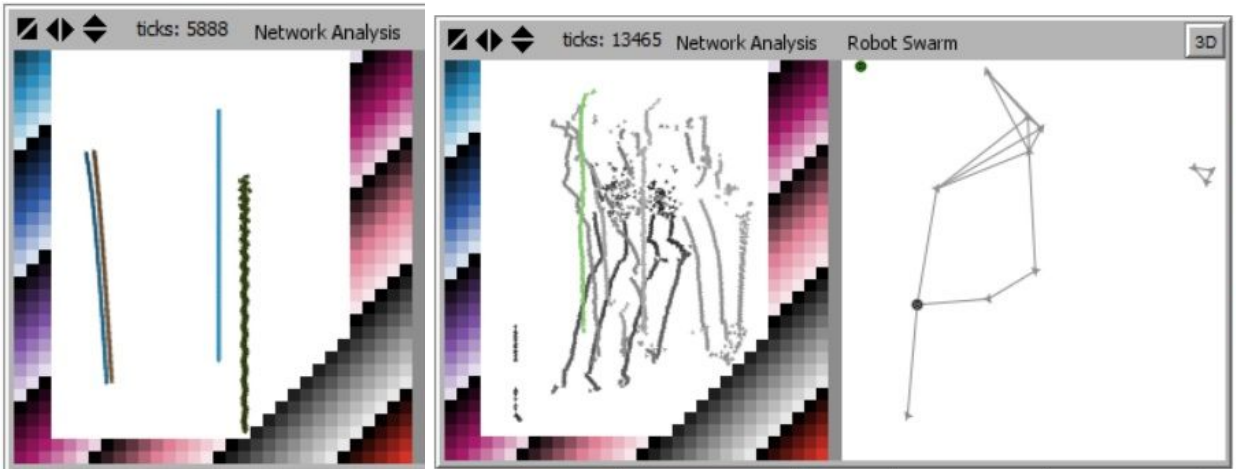


(V.2.2) Dynamic Patterns

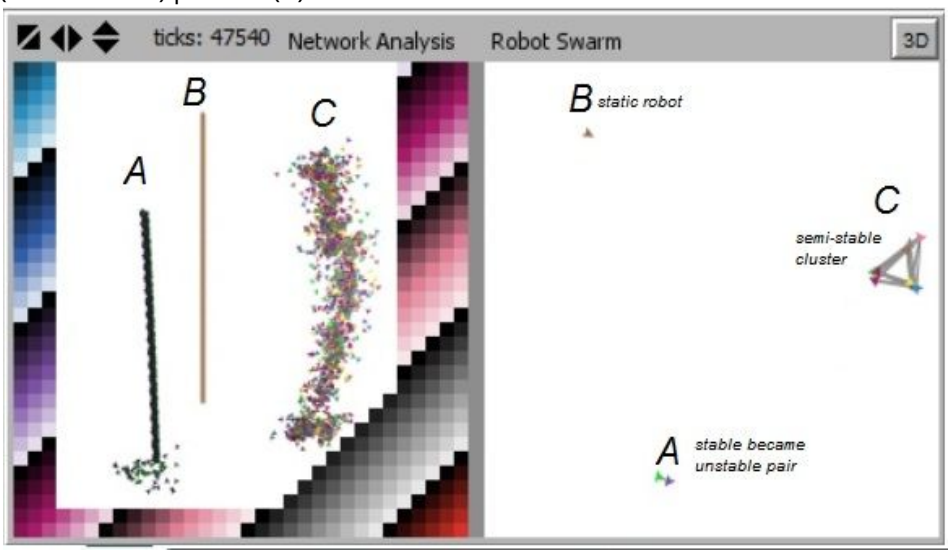
Diagonal, moving spatio-temporal patterns represent robots slowly moving around the environment while maintaining a shared formation relative to one another. Dynamic patterns serve to transfer information (stored in spatio-temporal patterns and formations) across the swarm, which may then influence a change in other patterns (and their information) through various collision-based computations (i.e. merging, etc) which shall be looked at shortly.

Below (left) is an example of a dynamic type I ('static') spatio-temporal pattern formed from a robot pair (a blue and black robot) moving toward a

stationary type I ('static') pattern formed of a single robot (blue) and a stationary type II ('stable') pattern formed of a single robot (black).

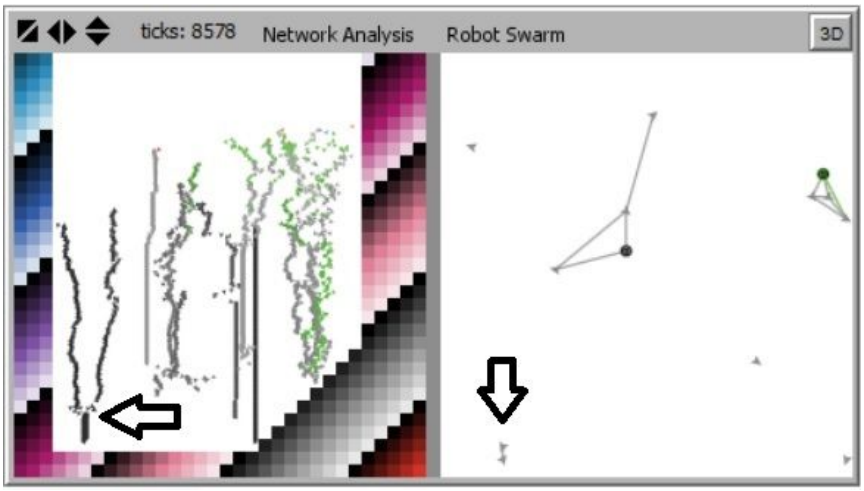


Above (right) shows a dynamic type I ('static') pattern formed of four robots (all black) connected together in the formation of a regular-polygon. Below shows a dynamic type II (stable) pattern (A) alongside a dynamic type IV ('semi-stable') pattern (C).

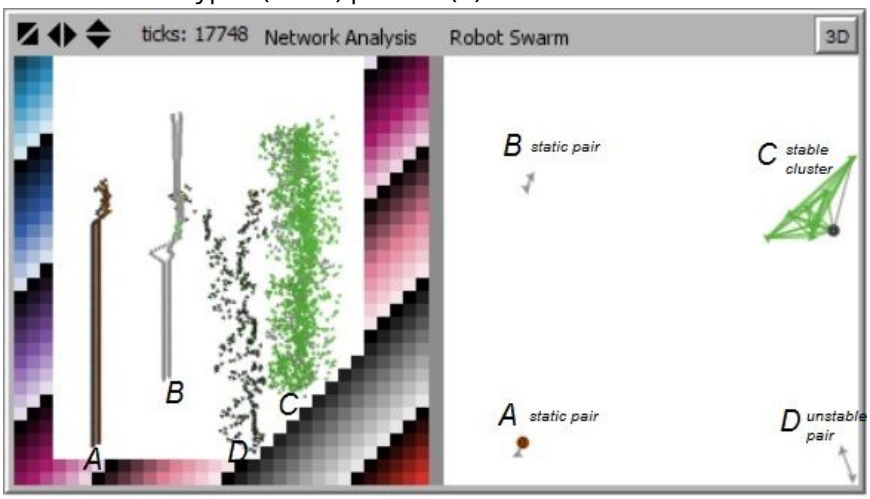


(V.2.3) Merging Patterns

Occasionally, two or more spatio-temporal patterns will collide and merge together to form a new, single pattern. This can be viewed as a collision-based computation (i.e. a computational mechanism which allowed information to be modified in a specific way). Below is an example showing two dynamic type IV (semi-stable) patterns (each formed from a single robot) merging together into a stationary type II (stable) pattern (formed of a robot pair).

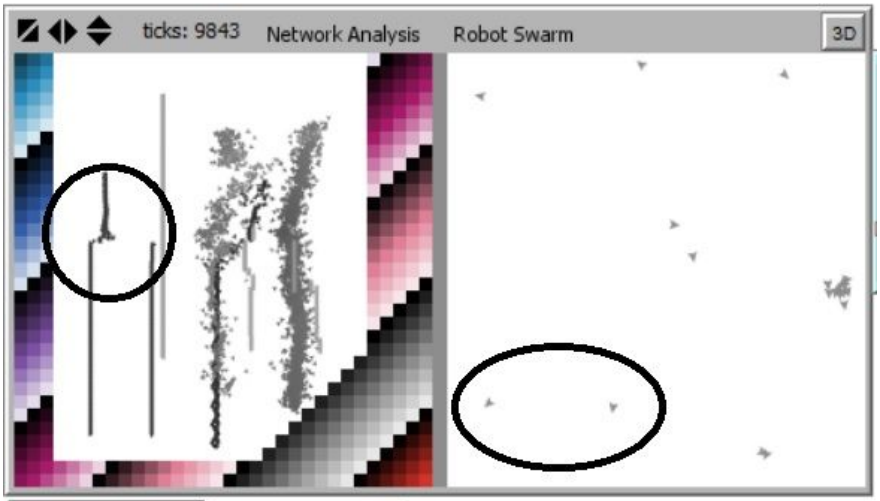


Below is another example of two type IV (semi-stable) patterns colliding to become a new type I (static) pattern (B).

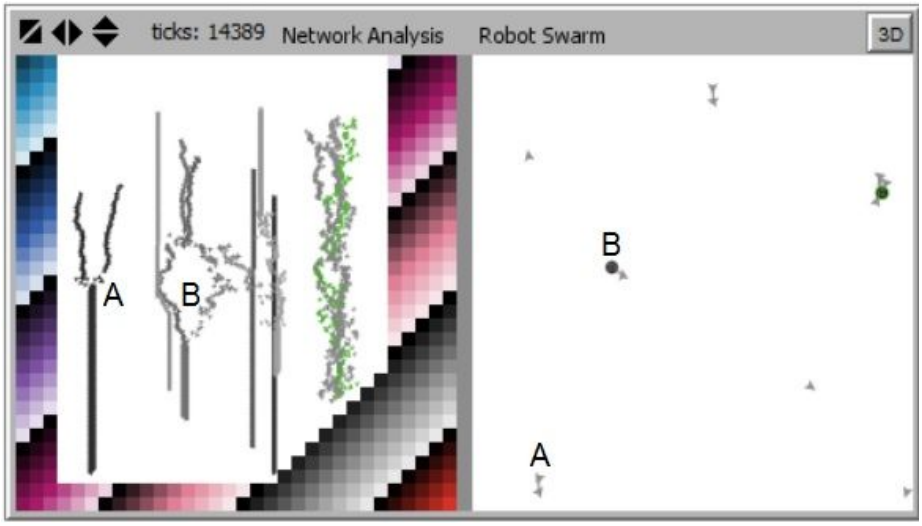


(V.2.4) Splitting Patterns

Information stored as a single spatio-temporal pattern may also split apart into two or more spatio-temporal patterns (each representing new information as in the example below).

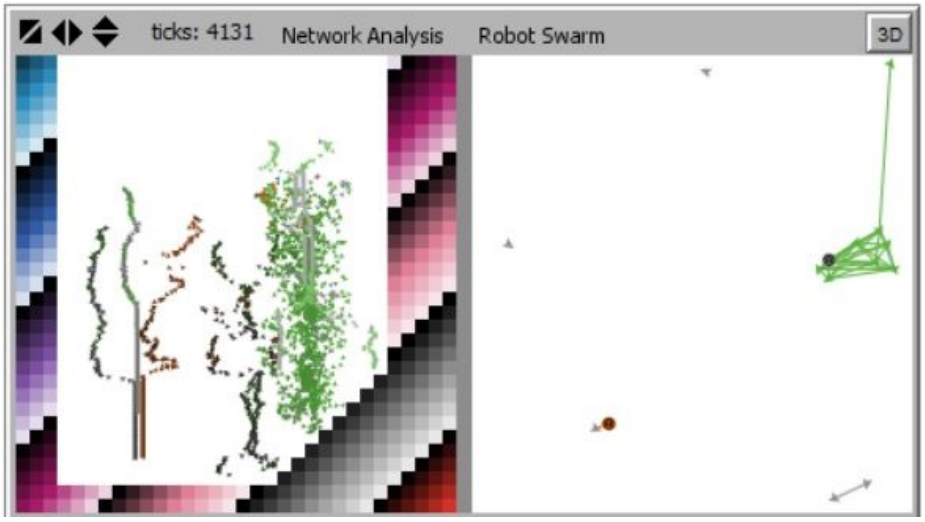


Below (B) is an interesting example of a spatio-temporal pattern splitting and then merging to form a new pattern; a type IV (semi-stable) pattern (formed of a robot pair) splits to form two new type IV patterns (formed of a single robot each) and then merges to form a type II (stable) pattern (formed of a robot pair).

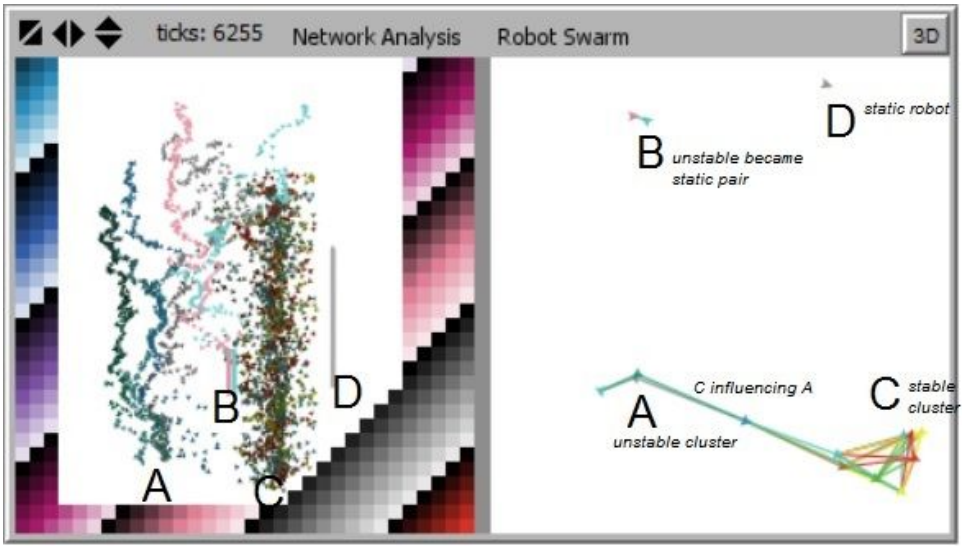


(V.2.5) Influencing Patterns at a distance

Sometimes a spatio-temporal pattern influences a nearby pattern; changing its pattern type while remaining unchanged itself. This type of change can occur if two or more patterns come into direct contact or within close proximity (in range of the robots' virtual forces). The patterns seems to have a form of momentum since the larger of the two patterns (i.e. the one consisting of more robots) tends to remain unchanged while the smaller pattern is more easily influenced (i.e. changed). Below is an example of a type IV (semi-stable) pattern (formed of a large green cluster of robots) destabilising a type I (static) pattern (consisting of a single robot).



Below (C) is another example of a type IV (semi-stable) cluster (consisting of eight robots) remaining unchanged while influencing (A) a nearby cluster (of three robots), turning into a type III non-pattern.

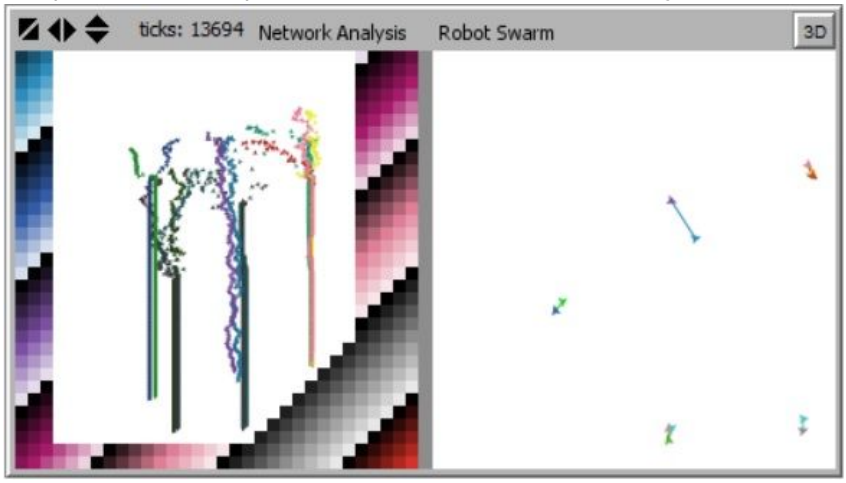


(V.2.6) Decaying Patterns

The final type of change occurs when a type III or IV pattern decays into another pattern type without any external influences, either gradually or all of a sudden. This may occur because the type III (chaotic) and type IV (semi-stable) patterns can easily fluctuate due to their innate randomness, unlike more orderly patterns (type I or II). This is also why a type I (static) or type II (stable) pattern never decays into other patterns without an external influence. Decaying is akin to a turbulent vortex structure (with a definite shape) but constantly changes in size and shape over time (sometimes even disappearing or reappearing momentarily). The exact time it takes for a pattern to decay varies, although smaller cluster sizes tend to have shorter lifespans than larger clusters, reminiscent of radioactive half-life decay times.

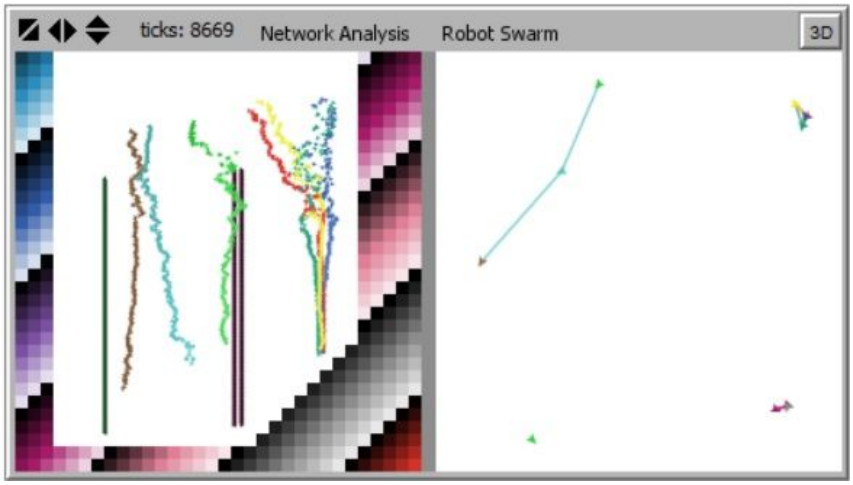
(V.2.3.A) Class III: Chaotic Attractors Decaying

The following examples all show class III: chaotic attractors / type III: non-patterns decaying. Below is an example of a single type III (non-pattern) decaying into five separate type I (static) patterns; each composed of robot pairs and each with different decay times.



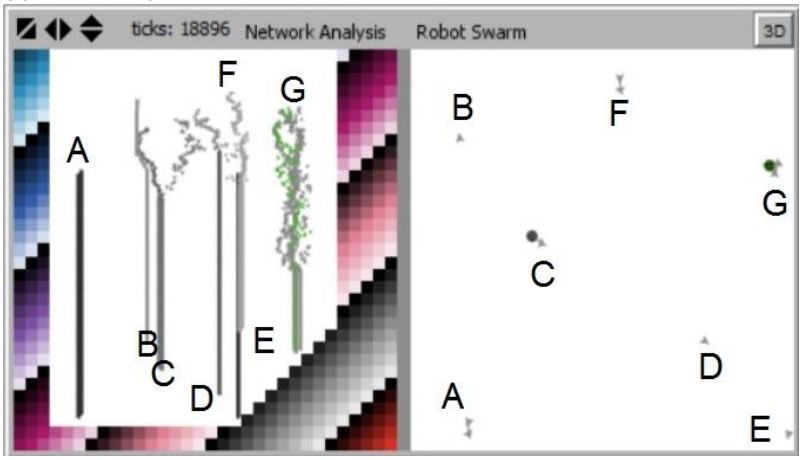
Below is another example showing a number of spatio-temporal patterns, but the pattern at the top-right which shows four robots (red, green, yellow and blue) was particularly interesting as they

chaotically formed a type III (non-pattern) but eventually decayed (or collapsed) into a type II (stable) pattern.

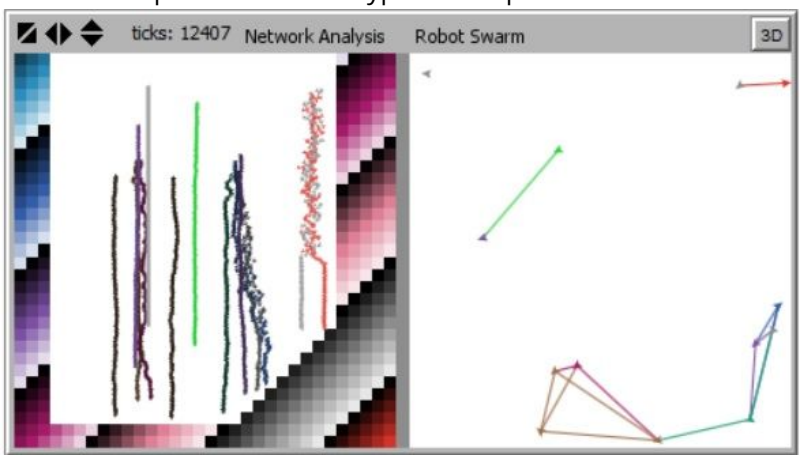


(V.2.3.B) Class IV: Strange Attractors Decaying

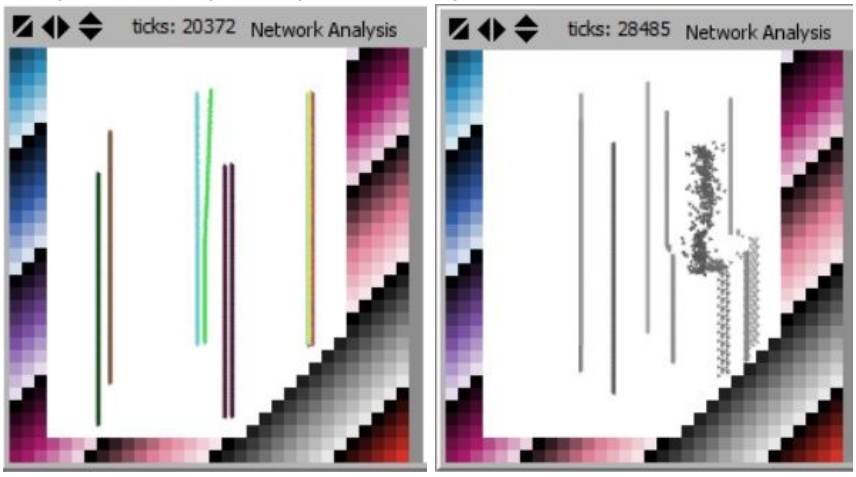
The following examples all show class IV: strange attractors / type IV: semi-stable patterns decaying. Below are two examples, Pattern G showing robotic pairs decaying into a class I: fixed-point attractor / type I static pattern.



The pink and grey robots below are a robotic pair decaying into a class I: fixed-point attractor / type I static pattern.

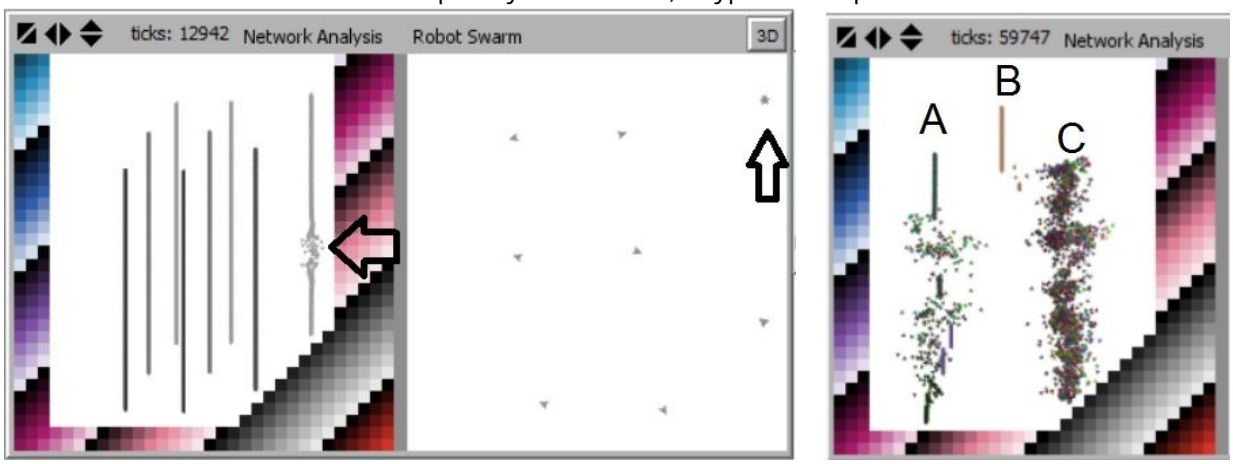


Below (left: blue and green) shows a similar decay, except this example decayed very gradually (unlike the examples above which decayed suddenly and spontaneously).



Above (right) also shows a type IV semi-stable pattern decaying into a class II: periodic attractor / type II: stable pattern. An interesting point to note here is that just before this pattern decayed, it influenced the type I (static) pattern on its right and forced it to change into a type II (stable) pattern along with it.

Below are two examples of type IV (semi-stable) patterns decaying into type III (non-patterns). They also show type III (non-patterns) decaying back into type IV (semi-stable) patterns. The first example (left) shows a spatio-temporal pattern that appears to be type I (static) in nature (class I: fixed-point attractor) but is actually a type IV (semi-stable) pattern (class IV: strange attractor) composed of a robot pair that are very close together. It then spontaneously decays into a type III: non-pattern (class III chaotic attractor) and almost immediately collapses back into a type IV semi-stable pattern. The proof that this change did not occur due to the influence of neighbouring robots is that every other pattern shown is type I static composed of single robots (i.e. they are stationary robots). The second example (right: A) is also decaying back and forth between pattern types IV and III. C is a type IV semi-stable pattern which has completely absorbed B, a type I static pattern.



(VI) Discussion

(1) Results Assessment

(VI.1.1) Do the results confirm or contradict previous research?

Upon analysing the spatial interactions of robotic swarms over time, an intricate subsystem of spatio-temporal patterns were revealed (emergent signals used to process, store and communicate information across the decentralised system) embedded within the medium of the system itself. The discovered patterns lend proof to the theory that computational mechanisms govern the intelligent, self-coordinating, emergent behaviours in swarms of simple, reactive robots (similar to how they govern the emergent behaviour in Cellular Automata). Although these findings are limited to only one type of robotic swarm (running the virtual-forces model) we are fairly confident that such patterns extend to other types of robotic swarms, as well as other complex adaptive systems (e.g. neural networks, etc), due to similar findings of spatio-temporal structures (e.g. gestalts) reported across various research literature.

The majority of our findings were in line with previous research findings related to spatio-temporal patterns (in Cellular Automata), including:

- There are **many different** spatio-temporal **patterns** (i.e. different spatial configurations perpetuating over time) [13, 44]
- Each **pattern represent** a separate piece of **information** encoded into the system [13, 44]
- The perpetuation of spatio-temporal **patterns allow** for pieces of **information to be stored**, like a memory. Data is stored so long as the patterns persist while remaining unchanged [13, 44, 12]
- **Moving** spatio-temporal **patterns** (i.e. not those remaining stationary but those which change spatial coordinates over time) **transfer** pieces of **information** across the system [38, 13, 44]
- **Changing a** spatio-temporal **pattern corresponds to changing** the associated piece of **information** [13, 44]
- A number of **Stable** types of Spatio-Temporal **Patterns were identified** [13]
- Some unstable (**semi-stable**) spatio-temporal **patterns** also **exist** [13]
- **Stable** Spatio-Temporal **Patterns can change** if influenced by an external event (i.e. **collisions with other** spatio-temporal **patterns**) [13, 25, 24, 26]
- Unstable (**Semi-stable**) Spatio-Temporal **Patterns can spontaneously change** pattern or velocity **without any external influences** [13]

However, some findings did differ, and even contradict, the findings of prior research (conducted into Spatio-Temporal Patterns governing emergent behaviour in Cellular Automata) which have provided us with new insights and additional details about the computational mechanisms governing the emergent behaviour of robot swarms. These include:

- Rather than the two pattern types identified (i.e. ‘stable’ and ‘unstable’), our research identified four distinct pattern types; static (fixed-point attractors), stable (periodic-attractors), non-patterns (chaotic attractors) and semi-stable (strange attractors).
- On top of discovering unstable (semi-stable) patterns that could spontaneously change without any external influence (decay), our research suggested that the average decay time (‘half-life’) of semi-stable patterns is

proportional to the number of robots involved in forming the semi-stable spatio-temporal pattern (i.e. the smaller the pattern, the shorter the half-life).

- Our research found that there were far more stable patterns types than unstable (semi-stable) types when the swarm was in the relatively rare 'solid' state. However, when the robot swarm is in its more common 'fluid' state, the chaotic and unstable (semi-stable) pattern types dominate the scene, in direct contrast to the research findings involved with Cellular Automata.
- As well as spatio-temporal patterns being modified via collisions with other spatio-temporal patterns [13, 44], our research showed that spatio-temporal patterns did not require direct collisions to change. Rather, patterns could influence one another at a distance proportional to the range of the robot's virtual forces.
- Our research identified at least seven distinct pattern changes (i.e. 'logic gates'), including; merging (two patterns combine to form a single, new pattern), splitting (one pattern becomes two new patterns), absorbing (one pattern disappears into another larger pattern which remains almost completely unchanged), annihilating (patterns dissolve into the chaos of a type III non-pattern), forming (patterns form out of the chaos of a type III non-pattern), overwhelming (the smaller of two patterns change while the larger remains unchanged), decaying (a pattern changes randomly without any external influence).
- Our research also seems to suggest that even though chaotic type III non-patterns are unable to represent, store or transfer specific pieces of information, they do contribute to the computational mechanics of the emergent behaviour in other ways; namely by keeping the swarm fluid and thus able to change, having an external influence on nearby patterns, and providing a chaotic environment for spatio-temporal patterns to form from or annihilate into. While our research supports the finding that emergent behaviour is not possible if the swarm is fixed in a solid state (i.e. composed of purely type I static or type II stable patterns), it doesn't limit emergent computation to type IV semi-stable patterns only. Rather, emergent computation is possible in any dynamic, fluid swarm state (i.e. at least partially composed of type III non-patterns or type IV semi-stable patterns), which includes chaotic (type III) spatio-temporal non-patterns, which have previously been considered incapable of emergent computation.

(VI.1.2) Accomplished Aims, Objectives & Deliverables

The project's deliverables were all completed as planned, including:

1. An engineered (hand-designed) robotic swarm exhibiting emergent behaviour (i.e. the virtual-forces model, the voronoi model, etc)
2. A genetic algorithm that could evolve a robotic swarm exhibiting emergent behaviour (a number of versions were designed to explore the effects of different fitness functions)
3. A software which analysed the swarm's emergent behaviour to detect spatio-temporal computational mechanisms

Furthermore, the first three sub-aims were successfully accomplished, wherein robotic swarms were designed by hand, evolved by a genetic algorithm and analysed to uncover (for the first time) the hypothesised computational mechanisms (spatio-temporal patterns) believed to underlie the emergent behaviour of robotic swarms. In doing so, we have made significant steps toward our ultimate goal of intrinsically controlling the emergent behaviour of robotic swarms.

.

(VI.1.2.A) 1st Aim

"1 - Design a rule-set that produces intelligent, emergent behaviour in a (simulated) robotic swarm"

The first aim involved designing a simple ruleset that would allow a swarm of robots to react to sensory data in a simple, unintelligent manner while emergently producing some form of intelligent, self-coordinating, group behaviour. This was achieved using: (1) the virtual-forces ruleset (which produced a fluid, coherent swarm behaviour that would solidify into a sensing grid and allow the best route to emerge between any two points on the landscape) and (2) the voronoi ruleset (which caused the swarm to form boundaries between obstacles placed randomly around the environment, which ended up resembling the shape of a voronoi map)

(VI.1.2.B) 2nd Aim

"2 - Design a genetic algorithm that evolves a robotic swarm with emergent behaviour"

The second aim involved designing a genetic algorithm which could automatically evolve rule-sets able to produce emergent behaviour in robotic swarms. A number of genetic algorithms were experimented with using various fitness functions, yet there is room for improvement as the evolved emergent behaviour was less intelligent than the virtual-forces ruleset and less elegant than the voronoi ruleset.

(VI.1.2.C) 3rd Aim

"3 - Discover the (hypothesised) computational mechanisms which underlie a swarm's emergent behaviour"

The third aim involved designing a software that could analyse the emergent behaviour of a robotic swarm. The analysis was first attempted using a 3d static network representation of the robot swarm but was later changed to a 3d particle swarm representation of the robotic swarm (so that the swarm's spatial developments could be viewed on the x-y axes while the swarm's temporal developments could be viewed via the z-axis). The software used a split-screen to display the spatio-temporal analysis in real-time beside the simulated robotic swarm (as either a particle swarm or a dynamic network). The third version of the software successfully confirmed the existence of the hypothesised spatio-temporal computational mechanisms governing the emergent behaviour of robot swarms.

(VI.1.3) Unaccomplished Objectives

(VI.1.3.A) 4th Aim

"4 - Understand the computational mechanics discovered by: a - identifying all spatio-temporal patterns, b - modelling the characteristic behaviours of each spatio-temporal pattern, c - mapping the interactions between each spatio-temporal pattern"

The fourth aim involves studying the spatio-temporal patterns and gathering enough information about them to accurately model the underlying computational mechanics of the swarm's emergent behaviour. This includes carefully cataloguing every spatio-temporal pattern and their characteristic properties (i.e. class / type, shape, velocity, etc) and mapping out each type of

change it undergoes upon interacting with other spatio-temporal patterns (the collision-based logic).

The critical analysis of this project has already shown sufficient evidence that differing types of spatio-temporal patterns exist. It has also given examples of some typical behaviours and interactions noted. However, these findings are too general and qualitative to answer the fourth aim. For this aim, a far deeper analysis is required in order to obtain the details required to model the intrinsic logic of the swarm's underlying spatio-temporal patterns. Aside from being a more systematic and in-depth analysis of the spatio-temporal patterns themselves (the fundamental information-processing elements), it would include numerous observations being made to map out their computational dynamics [13], mechanics and nonlinear logical operations (spatio-temporal pattern collisions and interactions cause the pattern and thus the information it represents, to change according to a specific, intrinsic logic [13, 23]). Therefore, just like an "artificial particle physicist" [24], we would carefully observe and catalogue each type of pattern and interaction (or collision) in a "lookup table" that can later be used to support "sophisticated particle-based information processing" [13]. This approach is known as "computational mechanics" (or alternatively "collision-based" computations) and was first developed as the result of the research conducted into "intrinsic computations embedded in CA space-time configurations" [11, 20]. Spatio-temporal "dynamics, representing basic logical operators, (is) the foundation of (collision-based) computation" [52]

The findings obtained from the fourth aim are also significant in confirming or contradicting previous research findings (including but not limited to):

- Basic Logical Operators (i.e. Spatio-Temporal Patterns)
    - The total number of static, stable and semi-stable spatio-temporal patterns (the basic logical operators) which exist (including their names e,g, " α , β , γ , δ , η , μ" etc) [13, 52, 3, 10].
    - The associated velocity (speed and direction) of each spatio-temporal pattern which determines how information is propagated throughout the system [38, 13, 44]
- Collision-based Logic Gates (i.e. Interactions / Pattern Changes)
    - Collisions change the spatio-temporal pattern and velocity according to an intrinsic logic specific to that system (e.g. spatio-temporal patterns "α" and "δ" always change into spatio-temporal pattern "γ" upon collision) [13, 25, 24, 26].
    - The inputs of the collision-based logic gates are represented by the spatio-temporal patterns present before the interaction [52, 3]
    - The outputs of the collision-based logic gates are represented by the spatio-temporal patterns present after the interaction. [52, 3]
    - The boolean truth values of collision-based logic gates are represented by the presence and absence of the spatio-temporal patterns [3, 10]
    - The different types of collision-based logic gates which can be realised via the interaction and change of spatio-temporal patterns (i.e. xor gates, diodes, etc) [3]
    - The total number of collision-based logic gates which occur at the sites where spatio-temporal patterns change via various

methods (i.e. Merging, Splitting, Absorption, Annihilation, Formation, Overwhelming, Decay, etc) [13, 3].

(VI.1.3.B) 5th Aim

"5 - Predict the swarm's emergent behaviour by simulating its underlying computational mechanics"

The fifth aim involves simulating the swarm's emergent behaviour and predicting future developments [19] using a model of its underlying computational mechanics (i.e. the spatio-temporal patterns, their dynamics and intrinsic logic) constructed using the mappings, categorisations and details gathered during the fourth aim (e.g. spatio-temporal pattern types, velocities, collision-based computational logic, etc) [13]. The predictive accuracy obtained when modelling the computational mechanics of Cellular Automata in this way was as high as 98.5% [25].

The model's accuracy can be evaluated by comparing its predictions against the swarm's actual developments. The task here is to try to get as low an error as reasonably possible, since "even small errors in the particle (spatio-temporal pattern) velocities or interactions… are compounded over time" [13].

(VI.1.3.C) 6th Aim

"6 - Investigate methods to manipulate the underlying computational mechanisms, including; (i) Injecting, (ii) Removing, (iii) Reflecting, (iv) Attracting and (v) Repelling spatio-temporal patterns"

The sixth aim involves investigating methods to manipulate the underlying computational mechanisms. This may be achieved through manipulating certain properties related to key robots [31] (i.e. their positions, velocities, virtual forces, etc) or those related to the robot swarm as a whole (i.e. configurations, etc), or the environment [46], or some external stimuli (i.e. light) or other, less explicit factors (i.e. noise, wireless communication delay times, etc - which can directly influence aggregation or dispersion in robotic swarms [34]).

One way in which the direction of computational mechanisms (spatio-temporal structures) can be influenced is via the addition of attractants or repellents in the computing space (i.e. the swarm's external environment) to directly manipulate the system's medium, which would subsequently influence the embedded spatio-temporal structures [46]. For instance, chemical pheromones can be used as 'attractors' to spatio-temporal structures [5], 'impurities' can act as 'barriers' or 'reflectors' to them [10] and the light intensity illuminating the medium (i.e. protoplasmic networks [3], swarms of photo-avoidant individuals [5], etc) can be used as a 'repellent' to them. "A propagating plasmodium wave hits an obstacle (of light - i.e. a suitably shaped domain of illumination) that is small enough to divert [the emergent wavefront]" [5]. Repellents have even been shown to divert propagating spatio-temporal structures by phase-shifting them [3] or splitting them into two independent signals (spatio-temporal structures) [3, 5]. "The signal-wave (spatio-temporal structure) will split then into two signals, these daughter waves shrink slightly down to stable size and then travel with constant shape… and the auxiliary wave will annihilate" [3]. By carefully timing and positioning these external stimuli (i.e. chemicals, light,

impurities, etc), “we can precisely control wave’s (spatio-temporal structures) trajectory - e.g. realise U-turn of a signal (spatio-temporal structure) [3] - and thus direct the evolution of the system’s emergent behaviour [10].

(VI.1.3.D) 7th Aim

“7 - Intrinsically control the swarm’s emergent behaviour by reprogramming its underlying computational mechanics”

The seventh and final aim involves putting all the pieces together; using the predictive model (developed in the fifth aim) along with the manipulative methods (investigated in the sixth aim), to parallel program the swarm’s collision-based computations by manipulating it at the level of its spatio-temporal patterns (i.e. injecting, removing, reflecting, attracting or repelling spatio-temporal patterns in order to manipulate the collision-based logical operations). For example, manipulating the velocity of key spatio-temporal structures can allow for prior planning of desired collisions and avoidance of unwanted collisions, thus manipulating when and how data is changed, and ultimately controlling, or programming, collision-based computations. Rerouting a spatio-temporal structure (by changing its direction and/or velocity) can be used to delay and better coordinate distributed pieces of information [3].

Trying to control the swarm’s global behaviour any other way (i.e. at the level of the local robot rules), without first understanding its underlying computational mechanisms (i.e. spatio-temporal patterns) and collision-based logic, is limited and unpredictable. For example, the robot rules react only to localised spatial or temporal factors [8] (i.e. inter-robot distances) are known to have a significant impact on the global outcome of the entire swarm (i.e. how the rule reacts to inter-robot distances can directly impact the swarm’s aggregation [28]). Many of these factors, if fine-tuned, can completely alter the global behaviour of the swarm by being modified ever so slightly and so such values can be used as leverage points (tipping points) to directly control swarm behaviour (i.e. cause a phase transitions) [60]. However, such control techniques are few and rudimentary. Furthermore, the specifics of how the global behaviour will be affected are very general, and thus trial and error is often required to to recognise and fine-tune such influences.

This aim (which is also our ultimate goal) assumes that spatio-temporal patterns are the fundamental processors of emergent collision-based computations and ultimately, the driving force behind global emergent phenomena in robot swarms.

(2) Improving Results

In this section we look at some aspects of the system to evaluate how they could be improved or developed further in future.

(VI.2.1) Genetic Algorithm

The genetic algorithm did not produce highly impressive results; despite the evolved robotic swarms displaying emergent behaviour, they were fairly basic in comparison with the engineered emergent behaviours produced via hand-designed rules (i.e. the virtual-forces and voronoi methods). Hence the evolved robotic swarms were not used in the spatio-temporal analysis stage of this project. However, the genetic

algorithm used to evolve emergent behaviour in previous research with cellular automata "evolved significantly more sophisticated methods of emergent computation" [13] when compared against the two human-designed cellular automata, such as the block-expanding rules [13].

The two main features of a genetic algorithm's design that affect the quality of the results evolved are; (a) the fitness function and (b) the parameters encoded in the candidate solutions.

A more effective fitness function will better guide the evolution of the emergent behaviour would be an automated method of detecting the swarm's spatio-temporal patterns. This would lead the genetic algorithm to evolve emergent swarm behaviours using the presence of spatio-temporal patterns to determine how emergent the swarm's behaviour is.

More effective parameters to encode into candidate solutions may include: the direction which robots would move, the swarm size (i.e. the number of robots in the swarm), the radius of the robots' "personal bubbles" (which determines the proximity another robot may be attracted before their virtual force becomes repulsive and pushes them away), the equation of the virtual force, as well as the strength and range of the virtual force, etc.

(VI.2.2) Analysis Software & Methodology

The final method used in the analysis software was successful at detecting spatio-temporal patterns was the 3d particle swarm representation. However, since the screen is only a 2d plane, showing the 3d swarm required some graphical manipulation to give the illusion of a 3d swarm. The viewing angle does make it difficult to see certain areas of the 3d swarm (particularly those near the rear) and so a rotation feature would be a beneficial modification to include in future versions of the software (i.e. allow the user to rotate the 3d network to any angle they desire).

"Various tools in the field of complex networks" could have been used instead to analyse the network representation in a "macroscopic manner" [39]. Prior methods we used to analyse the emergent behaviour of the robotic swarm which were unsuccessful included an examination of the temporal development of dynamic and static network representations of the robotic swarm. Although no spatio-temporal patterns were immediately evident using these methods, it may have been the case that they were simply not visible to the naked eye. Furthermore, there was no exact method for detecting emergent behaviour or spatio-temporal patterns (and this is the main weakness limiting the analysis software). It would be far more efficient and reliable to create an automated detection feature which is able to identify (and perhaps classify) spatio-temporal patterns without requiring human intervention. Some researchers "developed ways to automatically detect space-time patterns in Cellular Automata" [52] which allowed for a greater detection rate, especially when there were complex computational dynamics involving multiple differing spatio-temporal patterns occurring at the same time [13].

(3) Future Research

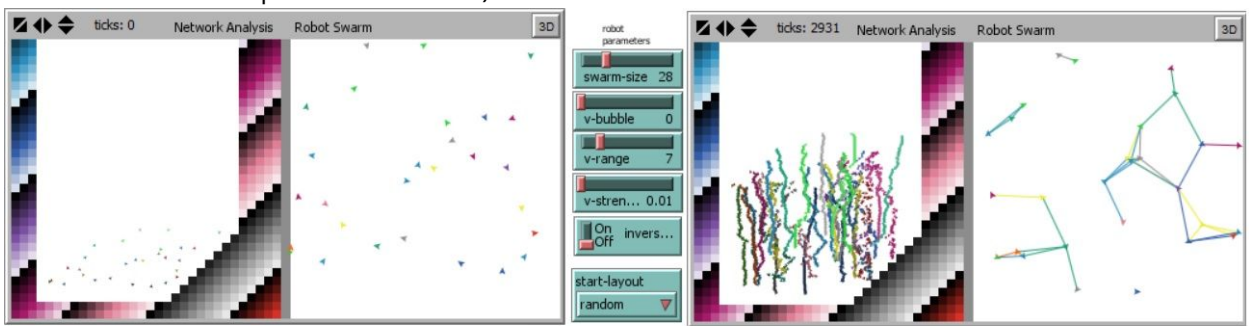
This section touches on new questions to be researched further in future, stemming from answers found during this project.

(VI.3.1) Effects of Initial Configurations & other parameters on the development of the Swarm's Emergent Behaviour

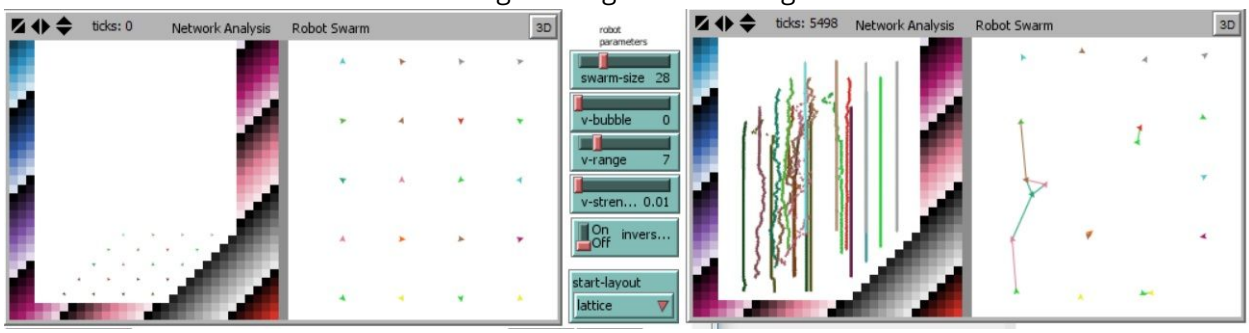
It would be interesting to explore how different parameters shape the outcome of the swarm's emergent characteristics (and later explain such impacts from the

underlying computational mechanics level by analysing the interactions and logical operations of the spatio-temporal patterns). Parameters which may have an effect on the emergent behaviour of the swarm include; range and strength of the virtual force, number of robots and their initial layout, etc. "Some research suggests the initial distribution and configuration of the robots plays a significant role in shaping the resulting parallel computations" [7] perhaps because "the initial and final configuration of these spatio-temporal structures correspond to the system's input and output data respectively" [5, 26].

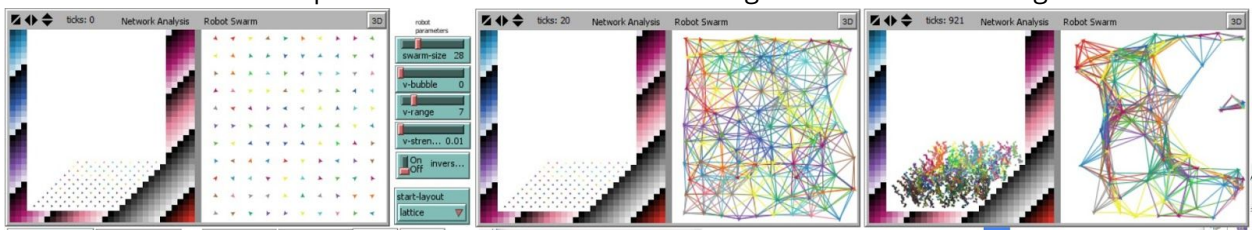
The initial configuration for the robotic swarms tested in this project were always random (i.e. robots were scattered randomly across the environment as in the examples shown below).



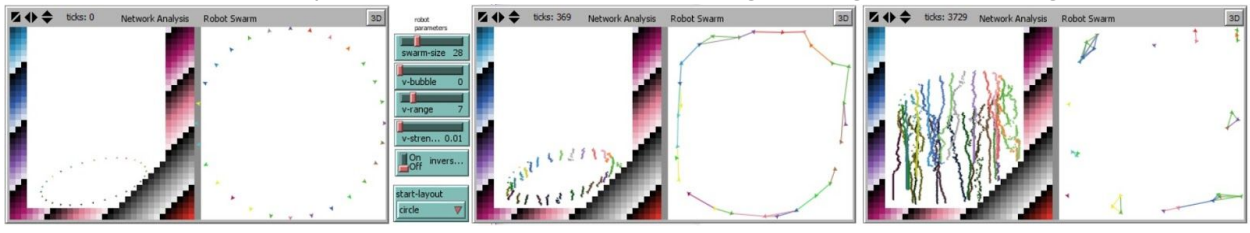
However, they could just as easily start from an initial lattice configuration; experimenting with different spacings between the robots. The example below shows the swarm starting in a large lattice configuration:



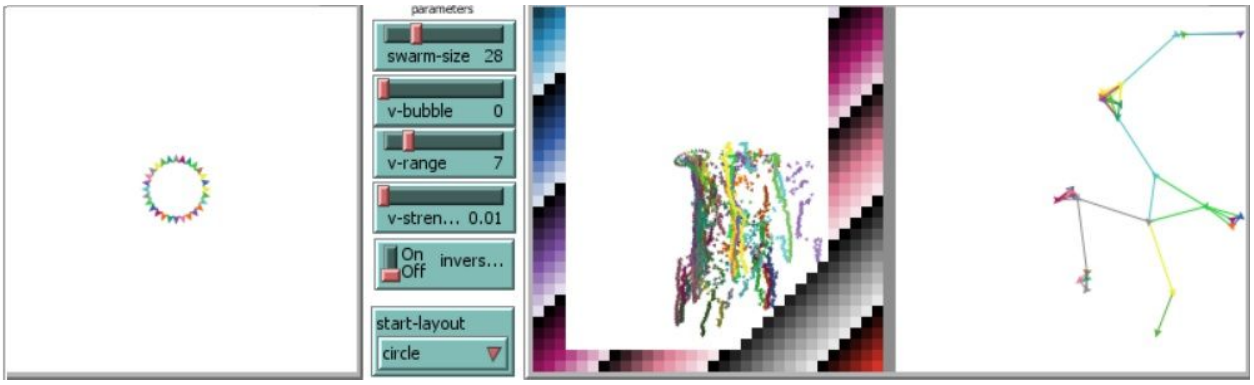
The example below shows the swarm starting in a small lattice configuration:



The example below shows the swarm starting in a large circular configuration:



The example below shows the swarm starting in a small circular configuration:



(VI.3.2) Effects of Noise on the Computational Mechanics of Spatio-Temporal Patterns

Unlike noiseless simulations, real robot swarms are noisy and subject to minor random influences from external and internal fluctuations (i.e. sensor inaccuracies, temperature change, friction, etc). Thus, studying the effects of noise may be one of the few perks of using a real-robot implementation over theoretical studies and computer simulations [8].

It would be interesting to understand exactly how noise impacts the underlying computational mechanisms of the swarm's emergent behaviour. Prior research does suggest that "in the absence of noise, the emergent computational mechanisms propagate indefinitely" [3], yet, when noise is introduced, there is a negative impact on the computation process (i.e. the spontaneous splitting of spatio-temporal structures or an unwanted dissipation over time [3, 10] - "travelling wave fragments keep their shape for around 4-10 time units, then decrease in size and vanish" [3]).

It is still unclear, however, whether such negative effects are large enough to affect the collision-based logic ("as long as the stochastic effects are small, the essence of local stable points - spatio-temporal structures - remains" [23]). Furthermore, solutions have been suggested to counteract such effects, such as 'amplifiers' which seek to extend the lifespan of spatio-temporal structures [3].

It may even be possible to exploit the effects of noise as an additional control mechanism that can influence the spatio-temporal patterns and thus the development of the swarm's emergent behaviour (i.e. the delay caused by noise has been used to cause a swarm to aggregate or segregate [34]).

(VI.3.3) Investigating Spatio-Temporal Patterns in other Robotic Swarms and Complex Systems

The findings in this project confirm that there exist embedded computational mechanisms (spatio-temporal patterns) at the heart of a robotic swarm's emergent behaviour. Similar computational mechanism have been studied in detail in 1D and 2D cellular automata and briefly noted in other complex adaptive systems (i.e. artificial neural networks, etc) but labelled under various different names [47, 13, 22, 23, 45, 52, 3, 10, 4, 5, 44, 57]. This suggests that the computational mechanisms govern the emergent behaviour of all complex adaptive systems and may, infact, be the key to understanding the mysterious nature of emergent phenomena.

(VII) Conclusions

This project has successfully uncovered the computational mechanisms (embedded spatio-temporal patterns) hypothesised to govern the intelligent, self-coordinating group-behaviours emerging across swarms of simple, reactive robots without the aid of any central controller or leader nor global communication or global knowledge. It provides proof that the theory of computational mechanics (developed to explain the emergent behaviour in cellular automata) explains the emergent behaviour in robotic swarms. It also suggests that this theory may potentially explain emergent behaviours in all forms of complex adaptive system (i.e. biological neural networks, ant colony behaviour, bee hives, etc) and may be the key that demystifies emergent phenomena itself.

Most of our findings confirmed prior research into spatio-temporal computations conducted in Cellular Automata and supported their findings (i.e. pieces of information are encoded in the spatio-temporal pattern and transferred across the system when the moves over time pattern. Changing a pattern corresponds to changing the information it represents. Stable spatio-temporal patterns can be changed via collisions while semi-stable patterns can change spontaneously without any external influences). Some findings however, did differ. These findings provided us with additional, unique insights (i.e. the average decay time of semi-stable patterns is proportional to the number of robots it consists of, spatio-temporal patterns need not collide directly and can sometimes influence one another from a distance, pattern changes form the basis of the collision-based logic gates and can occur via patterns merging, splitting, absorbing, annihilating, forming, overwhelming or decaying. Chaotic non-patterns contribute to the computational process by influencing nearby patterns, etc).

Following our discovery, we believe that the key to understand a robotic swarm's emergent behaviour is by understanding its underlying computational mechanics (spatio-temporal patterns) and collision-based logic (pattern changes and interactions). Therefore, the next step would be to conduct a careful study into the intricate dynamics of the swarm's spatio-temporal patterns; analysing each individual pattern and its characteristics in detail, as well as mapping out various interactions and pattern changes, so as to create an accurate model which is able to predict future emergent behaviours before they occur in the real robotic swarm. Further investigation into methods of manipulating (i.e. injecting, removing, reflecting, attracting, repelling, etc) spatio-temporal patterns is also needed, and this may include researching the effects of noise and initial configuration on the development of the swarm's computational mechanisms. Thereafter, we could reach our ultimate goal of controlling a robotic swarm's emergent behaviour intrinsically by predicting and influencing its underlying computational mechanics.

The deliverables for the project include a simulated robotic swarm running on different rule-sets (i.e. the virtual-forces model, the voronoi model, etc), each specifically designed to produce a different type of intelligent, global, emergent behaviour from very simple, reactive, individual behaviours and local robot interactions (without any central controller or leader, global communication or knowledge, etc). Also a genetic algorithm designed to evolve emergent behaviours in simulated robotic swarms (a far more effective technique to evolve robotic swarms with emergent behaviours is to use a fitness function that is able to directly detect

the embedded computational mechanisms). Thirdly, an analysis software created to detect spatio-temporally embedded computational mechanisms, hypothesised to be the responsible for the self-coordination of a robotic swarm's emergent behaviour (an automated detection mechanism for spatio-temporal patterns would also improve the detection rate and quality of the analysis software).

(VIII) References

(VIII.1) Books

[1] Bohm, D., 2002. *Wholeness and the implicate order* (Vol. 10). Psychology Press.

[2] Wolfram, S. (2002). A New Kind of Science. 1st ed. Wolfram Media, Inc. p1-1197

(VIII.2) Journals

[3] Adamatzky, A., 2004. Collision-based computing in Belousov–Zhabotinsky medium. *Chaos, Solitons & Fractals*, *21*(5), pp.1259-1264.

[4] Adamatzky, A., 2013. On oscillators in phyllosilicate excitable automata. *International Journal of Modern Physics C*, *24*(06), p.1350034.

[5] Adamatzky, A., 2013. Slimeware: engineering devices with slime mold. *Artificial life*, *19*(3_4), pp.317-330.

[6] Arvin, F., Murray, J., Zhang, C. and Yue, S., 2014. Colias: An autonomous micro robot for swarm robotic applications. *International Journal of Advanced Robotic Systems*, *11*(7), p.113.

[7] Bonabeau, E., Theraulaz, G., Deneubourg, J.L., Aron, S. and Camazine, S., 1997. Self-organization in social insects. *Trends in Ecology & Evolution*, *12*(5), pp.188-193.

[8] Brambilla, M., Ferrante, E., Birattari, M. and Dorigo, M., 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), pp.1-41.

[9] Chatty, A., Kallel, I., Gaussier, P. and Alimi, A.M., 2011, April. Emergent complex behaviors for swarm robotic systems by local rules. In *Robotic Intelligence In Informationally Structured Space (RiiSS), 2011 IEEE Workshop on* (pp. 69-76). IEEE.

[10] Costello, B.D.L. and Adamatzky, A., 2005. Experimental implementation of collision-based gates in Belousov–Zhabotinsky medium. *Chaos, Solitons & Fractals*, *25*(3), pp.535-544.

[11] Crutchfield, J.P. and Hanson, J.E., 1993. Turbulent pattern bases for cellular automata. *Physica D: Nonlinear Phenomena*, *69*(3-4), pp.279-301.

[12] Crutcheld, J. (1993). Evolving Cellular Automata Group (EvCA) Project. Available: http://csc.ucdavis.edu/evca/index.html. Last accessed 1st Jan 2015.

[13] Crutchfield, J.P. and Mitchell, M., 1995. The evolution of emergent computation. *Proceedings of the National Academy of Sciences*, *92*(23), pp.10742-10746.

[14] Das, R., Mitchell, M. and Crutchfield, J.P., 1994, October. A genetic algorithm discovers particle-based computation in cellular automata. In *International Conference on Parallel Problem Solving from Nature* (pp. 344-353). Springer, Berlin, Heidelberg.

[15] de Oliveira, G.M.B. and Siqueira, S.R.C., 2006. Parameter characterization of two-dimensional cellular automata rule space. *Physica D: Nonlinear Phenomena*, *217*(1), pp.1-6.

[16] Devi, N.R., Uma, G., Praveena, T.S. and Jyothi, M.N., Emerging And Challenging Applications In Swarm Robotics.

[17] Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T.H., Baldassarre, G., Nolfi, S., Deneubourg, J.L., Mondada, F., Floreano, D. and Gambardella, L.M., 2004. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, *17*(2), pp.223-245.

[18] Forrest, S., 1993. Genetic algorithms- Principles of natural selection applied to computation. *Science*, *261*(5123), pp.872-878.

[19] Givigi Jr, S.N. and Schwartz, H.M., 2006. A game theoretic approach to swarm robotics. *Applied Bionics and Biomechanics*, *3*(3), pp.131-142.

[20] Hanson, J.E. and Crutchfield, J.P., 1992. The attractor—Basin portrait of a cellular automaton. *Journal of Statistical Physics*, *66*(5), pp.1415-1462.

[21] Hawick, K.A. and Scogings, C.J., 2010. Complex emergent behaviour from evolutionary spatial animat agents. *Agent-Based Evolutionary Search*, pp.139-159.

[22] Holland, O. and Melhuish, C., 1999. Stigmergy, self-organization, and sorting in collective robotics. *Artificial life*, *5*(2), pp.173-202.

[23] Hopfield, J.J., 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, *79*(8), pp.2554-2558.

[24] Hordjik, W. (Summer 2001). Particles That. SFI Bulletin. 16 (.), p17-20.

[25] Hordjik, W. (2013). EvCA Project: A Brief History. Available: wim@WorldWideWanderings.net. Last accessed 1st Jan 2015.

[26] Hordjik, W. (1996). An Overview of Computation in Cellular Automata. PhysComp96. . (.), p.1-10.

[27] Hordjik, W. (December 1999). Dynamics, Emergent Computation, and Evolution in Cellular Automata. Dissertation. . (.), p1-241.
[28] Kelley, D.H. and Ouellette, N.T., 2013. Emergent dynamics of laboratory insect swarms. *Scientific reports*, *3*.
[29] Kramper, W., Wanker, R. and Zimmermann, K.H., 2012. Analysis of swarm behavior using compound eye and neural network control. *Open Computer Science*, *2*(1), pp.16-32.
[30] Langton C.G.. (1990). Computation at the edge of chaos: Phase transitions and emergent computation. Physica D. . (42), p12-37. [23] Packard, N.H.. (1988). Adaptation toward the edge of chaos. In: Kelso J. A. S., Mandell A. J., Shlesinger M.F. Dynamic Patterns in Complex Systems. .: World Scientic. p293301.
[31] Mabrouk, M.H. and McInnes, C.R., 2008. An Emergent Wall Following Behaviour to Escape Local Minima for Swarms of Agents. *IAENG International Journal of Computer Science*, *35*(4).
[32] Martinez G.J., Seck-Tuoh-Mora J.C., Zenil H.. (2013). Computation and Universality: Class IV versus Class III Cellular Automata. Journal of Cellular Automata. . (7(5-6)), p393-430.
[33] Mellinger, D. and Kumar, V., 2011, May. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (pp. 2520-2525). IEEE.
[34] Mijalkov, M., McDaniel, A., Wehr, J. and Volpe, G., 2016. Engineering sensorial delay to control phototaxis and emergent collective behaviors. *Physical Review X*, *6*(1), p.011008.
[35] Mitchell, M., Crutchfield, J.P. and Hraber, P.T., 1994. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, *75*(1-3), pp.361-391.
[36] Mitchell M., Crutcheld J. P., Hordjik, W.. (1996). Embedded particle computation in evolved cellular automata. Proceedings of the Conference on Physics and Computation. New England Complex Systems Institute (.), p.153158.
[37] Mitchell, M. (Oct 2002). Is the Universe a Universal Computer? Science: Sciences Compass. 298(1), p65-68
[38] Morales, F.J., Crutchfield, J.P. and Mitchell, M., 2001. Evolving two-dimensional cellular automata to perform density classification: A report on work in progress. *Parallel Computing*, *27*(5), pp.571-585.
[39] Ohkura, K., Yasuda, T. and Matsumura, Y., 2011, September. Analyzing macroscopic behavior in a swarm robotic system based on clustering. In *SICE Annual Conference (SICE), 2011 Proceedings of* (pp. 356-361). IEEE.
[40] Packard, N.H.. (1988). Adaptation toward the edge of chaos. In: Kelso J. A. S., Mandell A. J., Shlesinger M.F. Dynamic Patterns in Complex Systems. .: World Scientic. p293301.
[41] Payton, D., Estkowski, R. and Howard, M., 2003. Compound behaviors in pheromone robotics. *Robotics and Autonomous Systems*, *44*(3), pp.229-240.
[42] Peat, F.D., 1992. Non-locality in Nature and Cognition. In *Nature, Cognition and System II* (pp. 297-311). Springer Netherlands
[43] Piccinini, G. (March 2007). Computational Modelling vs Computational Explanation: Is everything a Turing Machine, and does it matter to the philosophy of mind?. Australasian Journal of Philosophy. 85(1), p93-115.
[44] Reynaga, R. and Amthauer, E., 2003. Two-dimensional cellular automata of radius one for density classification task $\rho = 12$. *Pattern recognition letters*, *24*(15), pp.2849-2856.
[45] Serugendo, G.D.M., 2006. Autonomous systems with emergent behaviour. *chapter in Handbook of Research on Nature Inspired Computing for Economy and Management, Jean-Philippe Rennard (ed.), Idea Group, Hershey, PA*, pp.429-443.
[46] Sharma, Y.K. and Bagla, A., 2009. Security challenges for swarm robotics. *Secur Chall*, *2*(1), pp.45-48.
[47] Spears, W.M., Spears, D.F., Hamann, J.C. and Heil, R., 2004. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, *17*(2), pp.137-162.
[48] Stirling, T., Roberts, J., Zufferey, J.C. and Floreano, D., 2012, May. Indoor navigation with a swarm of flying robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on* (pp. 4641-4647). IEEE.
[49] Stone, C. and Bull, L., 2009. Evolution of cellular automata with memory: The density classification task. *BioSystems*, *97*(2), pp.108-116.
[50] Trianni, V. and Nolfi, S., 2011. Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artificial life*, *17*(3), pp.183-202.
[51] Vassev, E., Sterritt, R., Rouff, C. and Hinchey, M., 2012. Swarm technology at NASA: building resilient systems. *IT Professional*, *14*(2), pp.36-42.
[52] Ventrella, J.J., 2006. A particle swarm selects for evolution of gliders in non-uniform 2d cellular automata. In *Artificial Life X: proceedings of the 10th international conference on the simulation and synthesis of living systems* (pp. 386-392).
[53] Winfield, A.F., Sa, J., Gago, M.C. and Fisher, M., 2005. Using temporal logic to specify emergent behaviours in swarm robotic systems. *Towards Autonomous Robotic Systems*.

(VIII.3) Websites

[54] Anders Broberg, Niklas Andersson, Agneta Bränberg, Kenneth Holmlund, Lars-Erik Janlert, Erik Jonsson, Jonny Pettersson. (.). *designing for emergence.* Available: http://www8.cs.umu.se/~bopspe/publications/MOMUC03/. Last accessed 23rd Aug 2017.
[55] Andy Martin and Kristian Helmerson . (October 1, 2014). *Emergence: the remarkable simplicity of complexity* . Available: http://theconversation.com/emergence-the-remarkable-simplicity-of-complexity-30973. Last accessed 23rd Aug 2017.

[56] J. Groff. (.). *GOOeFloys (Gooey-floweez: Greater Object Oriented evolving Floys).* Available: http://www.neocoretechs.com/alife/. Last accessed 23rd Aug 2017.
[57] Johnson, G. (23 March 1999). Mindless Creatures Acting 'Mindfully'. Available: http://www.nytimes.com/library/national/science/032399sci-cellular-automata.html. Last accessed 1st Jan 2015.
[58] Various. (2013). *Emergent Behavior.* Available: http://wiki.c2.com/?EmergentBehavior. Last accessed 23rd Aug 2017.

(VIII.4) Lectures

[59] Alan FT Winfield . (19 April 2010). *The Emergence Engineers' Dilemma: it seems we can evolve emergence, or prove emergent properties, but not both* . Available: https://www.cs.york.ac.uk/nature/emergeNET4/winfield.pdf. Last accessed 23rd Aug 2017
[60] Instructor: William Rand. (Summer 2016). *Introduction to Agent-Based Modeling: 1st Lecture* . Available: https://www.complexityexplorer.org/courses. Last accessed 23rd Aug 2017.
[61] Instructor: William Rand. (Summer 2016). *Introduction to Agent-Based Modeling: 3rd Lecture* . Available: https://www.complexityexplorer.org/courses. Last accessed 23rd Aug 2017.
[62] Instructor: William Rand. (Summer 2016). *Introduction to Agent-Based Modeling: 4th Lecture.* Available: https://www.complexityexplorer.org/courses. Last accessed 23rd Aug 2017.