Evolutionary Swarm Robotics using Epigenetics Learning in Dynamic Environment

Faqihza Mukhlish

A thesis in fulfilment of the requirements for the degree of Doctor of Philosophy

School of Mechanical and Manufacturing Engineering

Faculty of Engineering

February 2021

1. Thesis Title and Abstract

Thesis Title

Evolutionary Swarm Robotics using Epigenetics Learning in Dynamic Environment

Thesis Abstract

Intelligent robots have been widely studied and investigated to replace, fulfilling a complex mission in a hazardous environment. Lately, swarm robotics, a group of collaborative robots, has become popular because it offers benefits over a single intelligent system. Many strategies have been developed to achieve collective and decentralised control applying evolutionary algorithms. However, since the evolutionary algorithm relies principally on an individual fitness function to explore the solution space, achieving swarm robotics' collaborative behaviour in a dynamic environment becomes a problem. This is due to the lack of adaptation in most of the evolutionary methods. In order to thrive in such environment, external stimuli and rewards from the environment should be utilised as "knowledge" to achieve the intelligent behaviour currently lacking in evolutionary swarm robotics. The aims of this research are: (1) to develop novel reward-based evolutionary swarm learning using mechanisms of epigenetic inheritance; and (2) to identify an efficient learning method for the epigenetic layer achieving a decision-making strategy in a dynamic environment.

This research's contributions are the development of reward-based co-learning algorithm and co-evolution using epigenetic-based knowledge backup. The reward-based co- learning algorithm enables the swarm to obtain knowledge of the dynamic environment and override the objective-based function to evaluate internal and external problems. An advantage of this is that the learning mechanism also enables the swarm to explore potentially better behaviour without the constraint of an ill-defined objective function. Simulated search-and-rescue missions using a swarm of UAVs shows that individual behaviour evolves differently although each member has the same physical characteristics and the same set of actions. As an addition to reward-based multi-agent learning mechanisms, epigenetics is introduced as a decision-making layer. The epigenetic layer has two functions: there are genetic regulators, as well as an epigenetic inheritance (the epigenetic mechanism). The first is the function of an epigenetic layer regulating how genetic information is expressed as agent's behaviour (the "phenotype"). Thus, utilising the regulatory function, the agent is able to switch genetic strategy or decision-making based on external stimulus from the aforementioned reward-based learning. The second function is that epigenetic inheritance enables sharing of genetic regulation and decision-making layer between agents.

In summary, this research extends the current literature on evolutionary swarm robotics and decentralised multi-agent learning mechanisms. The combination of both advances the decentralised mechanism in obtaining information and improve collective behaviour.

2. Originality, Copyright and Authenticity Statements

ORIGINALITY STATEMENT

☑ I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

COPYRIGHT STATEMENT

☑ I hereby grant the University of New South Wales or its agents a non-exclusive licence to archive and to make available (including to members of the public) my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known. I acknowledge that I retain all intellectual property rights which subsist in my thesis or dissertation, such as copyright and patent rights, subject to applicable law. I also retain the right to use all or part of my thesis or dissertation in future works (such as articles or books).

For any substantial portions of copyright material used in this thesis, written permission for use has been obtained, or the copyright material is removed from the final public version of the thesis.

AUTHENTICITY STATEMENT

☑ I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis.

3. Inclusion of Publications Statement

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in the candidate's thesis in lieu of a Chapter provided:

- The candidate contributed **greater than 50%** of the content in the publication and are the "primary author", i.e. they were responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has obtained approval to include the publication in their thesis in lieu of a Chapter from their Supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis.

☑ The candidate has declared that **their thesis has publications - either published or submitted for publication - incorporated into it in lieu of a Chapter/s. Details of these publications are provided below.**.

Publication Details #1

Full Title:	Evolutionary-Learning Framework: Improving Automatic Swarm Robotics Design
Authors:	Faqihza Mukhlish, John Page, Michael Bain
Journal or Book Name:	International Journal of Intelligent Unmanned Systems
Volume/Page Numbers:	Volume 6 / Issue no 4 / Page 197–215
Date Accepted/Published:	October 8, 2018
Status:	published
The Candidate's Contribution to the Work:	The candidate contributes more than 80% of the work, literature reading, writing, drafting and editing
Location of the work in the thesis and/or how the work is incorporated in the thesis:	Chapter 2. Background and Current Swarm Robotics Design

Publication Details #2

Full Title:	Reward-Based Epigenetic Learning Algorithm for a Decentralised Multi-Agent System
Authors:	Faqihza Mukhlish, John Page, Michael Bain
Journal or Book Name:	International Journal of Intelligent Unmanned Systems
Volume/Page Numbers:	Volume 8 / Issue no 3 / Page 201-224
Date Accepted/Published:	April 13, 2020
Status:	published
The Candidate's Contribution to the Work:	The candidate contributes 80% of the work, formulation, writing, drafting, editing, simulation.
Location of the work in the thesis and/or how the work is incorporated in the thesis:	Chapter 3. Reward-based Epigenetic Algorithm

Candidate's Declaration

I confirm that where I have used a publication in lieu of a chapter, the listed publication(s) above meet(s) the requirements to be included in the thesis. I also declare that I have complied with the Thesis Examination Procedure.

Abstract

Intelligent robots have been widely studied and investigated to replace, fulfilling a complex mission in a hazardous environment. Lately, swarm robotics, a group of collaborative robots, has become popular because it offers benefits over a single intelligent system. Many strategies have been developed to achieve collective and decentralised control applying evolutionary algorithms. However, since the evolutionary algorithm relies principally on an individual fitness function to explore the solution space, achieving swarm robotics' collaborative behaviour in a dynamic environment becomes a problem. This is due to the lack of adaptation in most of the evolutionary methods. In order to thrive in such environment, external stimuli and rewards from the environment should be utilised as "knowledge" to achieve the intelligent behaviour currently lacking in evolutionary swarm robotics. The aims of this research are: (1) to develop novel reward-based evolutionary swarm learning using mechanisms of epigenetic inheritance; and (2) to identify an efficient learning method for the epigenetic layer achieving a decision-making strategy in a dynamic environment.

This research's contributions are the development of reward-based co-learning algorithm and co-evolution using epigenetic-based knowledge backup. The reward-based co-learning algorithm enables the swarm to obtain knowledge of the dynamic environment and override the objective-based function to evaluate internal and external problems. An advantage of this is that the learning mechanism also enables the swarm to explore potentially better behaviour without the constraint of an ill-defined objective function. Simulated search-and-rescue missions using a swarm of UAVs shows that individual behaviour evolves differently although each member has the same physical characteristics and the same set of actions. As an addition to reward-based multi-agent learning mechanisms, epigenetics is introduced as a decision-making layer. The epigenetic layer has two functions: there are genetic regulators, as well as an epigenetic inheritance (the epigenetic mechanism). The first is the function of an epigenetic layer regulating how genetic information is expressed as agent's behaviour (the "phenotype"). Thus, utilising the regulatory function, the agent is able to switch genetic strategy or decision-making based on external stimulus from the aforementioned reward-based learning. The second function is that epigenetic inheritance enables sharing of genetic regulation and decision-making layer between agents.

In summary, this research extends the current literature on evolutionary swarm robotics and decentralised multi-agent learning mechanisms. The combination of both advances the decentralised mechanism in obtaining information and improve collective behaviour.

Acknowledgments

I wish to express my sincerest thanks to my supervisors, Mr John Page and Dr Michael Bain, whose support, counsel and encouragement have guided me through these years. I would also like to thank Prof. Chun-Hui Wang for taking care of me as a formal supervisor for the past couple of years and giving full support to make sure my study is on track. I would especially like to thank my Mother for her patient in giving me the courage and strength that takes to complete this program. Lastly, I would like to thanks all my best friends and colleagues at The University of New South Wales who have always been supportive over the years.

List of Publications

The following is a list of publications produced throughout the duration of the thesis research

Journal Publications

- Mukhlish, Faqihza, John Page, and Michael Bain. "Evolutionary-Learning Framework: Improving Automatic Swarm Robotics Design." International Journal of Intelligent Unmanned Systems 6, no. 4 (October 8, 2018): 197–215.

- Mukhlish, Faqihza, John Page, and Michael Bain. "Reward-Based Epigenetic Learning Algorithm for a Decentralised Multi-Agent System." International Journal of Intelligent Unmanned Systems 8, no. 3 (April 13, 2020): 201–24.

Conference Publications

- Page, John, and Faqihza Mukhlish. "Simulation the Only Way to Investigate Self-Organising Swarms." In Australasian Simulation Congress 2017. Sydney: Simulation Australia, 2017.

- Mukhlish, Faqihza, John Page, and Michael Bain. "Evolutionary-Learning Framework for Swarm Robotics Using Epigenetics Layer," 14:P86. Jeju, South Korea: International Society of Intelligent Unmanned System, 2018.

- Mukhlish, Faqihza, John Page, and Michael Bain. "Evolving Swarm for Search and Rescue Mission: Potential, Development and Risk." Beijing, 2019.

- Mukhlish, Faqihza, John Page, and Michael Bain. "Reward-Based Evolutionary Swarm UAVs on Search and Rescue Mission." In AEROSPACE - 18th Australian International Aerospace Congress, 348–53. Melbourne, Australia: Engineers Australia, Royal Aeronautical Society, 2019.

- Mukhlish, Faqihza, John Page, and Michael Bain. "From Reward to Histone: Combining Temporal-Difference Learning and Epigenetic Inheritance for Swarm's Coevolving Decision Making." In 2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), 1–6. Valparaiso: IEEE, 2020.

List of Symbols

ε small probability
g gene
$\mathcal{V}$ genevalue
$\mathcal{G}$ genotype
$\boldsymbol{\mathcal{G}}$ set of genotypes
$\mathfrak{G}$ gene-family
$\mathfrak{G}$ gene-pool
C chromosome
$\mathcal{C}$ chromosome-set
$\mathfrak{C}$ chromosome-pool
h gene's histone
H chromosome's histone, behaviour value
η methylation rate
R reward
$\tilde{(\cdot)}$ approximated value
$\varepsilon_{\textbf{gene}}$ gene selection rate
$\varepsilon_{\textbf{chromosome}}$ chromosome selection rate
$\varepsilon_{\textbf{imprinting}}$ imprinting rate
μ mutation rate
$\varepsilon_{\textbf{regeneration}}$ regeneration rate
$\varepsilon_{\textbf{silencing}}$ silencing rate

$\hat{(\cdot)}$ evolutionary product
$\acute{(\cdot)}$ mutation product
$\mathcal{S}, S, s$ environment's state
$\mathcal{A}, A, a$ agent's action
$\mathcal{R}, R, r$ reward
$\mathcal{B}, B, b$ agent's behaviour
G total return
V, v state value
Q, q action value
π_a, π_b policies
γ discount rate
$\mathbf{r}$ path's origin
$\mathbf{q}$ path's end
$\mathbf{p}$ UAV's position
$\mathbf{s}$ UAV's projection
χ_d desired altitude
h_d desired heading
χ_q course line heading
χ_R resultant desired heading
h_R resultant desired altitude
$\mathbf{W}$ wind velocity

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Research Background

Fulfiling a challenging and hazardous mission by applying intelligent robots to help or replace humans entirely has been widely investigated. However, investigation on using a group of simple robots has been less explored. Lately, swarm robotics, a group of collaborative robots, has become popular because it offers benefits over a single intelligent system (Şahin 2005). Firstly, it is more flexible because of the decentralised setting. Secondly, because of the multi-agent setting, it is more scalable in size. Lastly, it is more robust to failures because of collaborative manners. A decentralised strategy is the foundational building block for a collaborative swarm. Generally, there are two main approaches in designing a swarm strategy: behaviour-based design and automatic design (Brambilla et al. 2013). However, building a swarm strategy is not without a challenge because control decentralisation and cooperative behaviour are challenging to achieve, especially to thrive in a dynamic environment caused by agents in the group and the mission's complexity.

Decentralised control can be achieved by utilising an evolutionary swarm (EA), an algorithm in which artificial genetics represents a set of behaviours. Selection, recombination, and mutation are operated on the genetics information to maximise each behaviour's fitness value. EA can be used in swarm robotics to attain collective behaviour of a self-organising swarm (Francesca et al. 2012). However, the random mutation often yields unpredictable behaviours when facing a dynamic environment (Yi et al. 2017). This shortcoming raises because the ability to receive information from the environment is lacking in EA. Thus, external stimulus from the environment should be used as "knowledge" to develop a better strategy accordingly to counter this problem. Utilising external stimulus to obtain a better action is known as learning mechanisms. Thus, to improve evolutionary swarm capability, a novel learning strategy for evolutionary swarms is needed to accomplish complex tasks and survive in a dynamic environment.

Embedding learning mechanism into evolutionary algorithms will advance adaptability to a dynamic environment. Moreover, there is evidence that genetics is adapting external stimulus from the environment using epigenetic layers (Holliday 1987). The Epigenetic layer is analogous to a decision-making layer for a gene to behave according to environmental stimulus. The regulatory function of epigenetics is developed through learning based on its interaction with the environment (Wang, Liu, and Sun 2017). To establish the dynamics based on the interaction can be achieved through Reinforcement Learning (Sutton and Barto 1998). Hence, investigating a novel collective strategy utilising the Epigenetic layer is necessary to overcome the dynamic environment and advance the capability of swarm robotics.

1.2 Aims

This thesis aims to incorporate epigenetic concept to evolutionary swarm and identify efficient learning mechanisms in response to a dynamic environment. Thus, the objectives of this research are:

1. To develop a novel evolutionary swarm learning incorporating epigenetic inheritance,
2. an to identify an efficient learning mechanism as a decision-making strategy for an epigenetic layer response to a dynamic environment

1.3 Thesis Outline

Chapter 1 – Introduction - A brief introduction to the thesis.

Chapter 2 – Background and Current Swarm Robotics Design - This chapter presents a review of the current practices in designing swarm robotics behaviour and the emerging topic of epigenetic inheritance in evolutionary computation.

Chapter 3 – Reward-based Epigenetic Algorithm - In this chapter, reward-based evolutionary computation is discussed. The formulation takes the direction of epigenetic-inspired computation and focuses on accumulating external stimulus and making use of it as knowledge to obtain better behaviours in the future.

Chapter 4 – Epigenetic Learning Framework - This chapter focuses on tackling the dynamic swarm problem by proposing a novel computational process tailored explicitly for a decentralised multi-agent system to comply with the complex problem, especially in a dynamic environment setting.

Chapter 5 – Epigenetic Learning Swarm for Search and Rescue Mission - This chapter aims to investigate the implementation of epigenetic swarm learning framework in a search and rescue mission.

Chapter 6 – A Bayesian Epigenetic Learning Swarm - This chapter investigates the possibility of incorporating Bayesian search methods to presuppose a prior knowledge or belief before exploring the dynamic problem.

Chapter 7 – Conclusion and Future Works - This chapter discusses how the thesis answers all the aims and objectives. A discussion on possible future improvement and works are presented.

Chapter 2

Background and Current Swarm Robotics Design

This chapter presents a review of the current practices in designing swarm robotics behaviour and an emerging epigenetic inheritance topic in evolutionary computation. Some sections of this chapter have been published in (Mukhlish, Page, and Bain 2018b). Firstly, a review of swarm robotics and its proceedings are discussed to explore the current capability in this area (Section 2.1). Then, the swarm design approaches are discussed (Section 2.2). The third topic is exploring an epigenetic-inspired method for robotics and computation design. Its source of inspiration and works are reviewed in section (Section 2.3). Several challenges in current swarm robotics design are presented (Section 2.4). Lastly, discussions and summary are presented in Section 2.5 and 2.6

2.1 Fundamental of Swarm Robotics

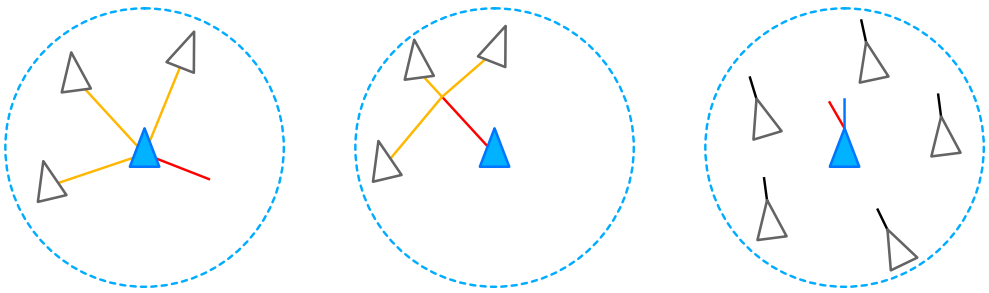
Intelligent robots have been widely studied and investigated to replace humans, fulfilling a mission in a hazardous or harsh environment. While developing a sophisticated robot to tackle a task is a common approach, a multi-robot system is currently gaining popularity (Spezzano 2019; Hamann 2018). Particularly in harsh and unpredictable environments, where a single robot system might struggle and fail to survive. On the other hand, a multi-robot system has a higher possibility of success in such condition through exploiting collective and possibility of success in such condition through exploiting collective and collaborative behaviour. These two behaviours in a multi-robot system are the foundational idea underlying a self-organising system known as swarm robotics.

Swarm robotics aims to control a multi-agent system to collectively and collaboratively tackle a task. Moreover, swarm robotics is firstly coined by Şahin as:

> *A novel approach to the coordination of large numbers of robots. . . [and] a study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among the agents and environment* (Şahin 2005, pp. 3)

Developing a self-organising swarm was initially inspired by animals' collective and collaborative behaviour. A simple individual such as ants shows a remarkably erudite collective behaviour in finding a food source and manage to realise the shortest path from their nest. This social behaviour was first observed by Forel (1874) and summarised by Wheeler (1910), an authoritative researcher of Ants. Later, studies in biology showed that there is exist a herd's intelligence that emerges from individual behaviour (Camazine 2001; Couzin et al. 2005; Ame et al. 2006).

Inspired by social animals, Reynolds (1987) proposed distributed behavioural models in his seminal work. The author investigated that a flocking behaviour of a multi-particle system, called (*boids*), can emerge naturally from simple rules at the individual level. There are cohesion, separation and alignment as can be seen in Figure 2.1. The first is a rule for a particle to move to the flock's centre (average position). The second steer the particle to avoid nearby particles. Moreover, the last is utilised to put all particles in the same heading.



(a) Separation (b) Alignment (c) Cohesion

Figure 2.1: Three simple rules for flocking behaviour

Largely, based in the first instance on ants and birds' behaviour models, algorithms have been developed mimicking social animals' behaviour and applied to artificial multi-agent systems. The resulting mathematical formulation generally takes distributed and decentralised approaches to collectively and collaboratively tackle various tasks (Bayindir 2016).

Overall, self-organising swarm's characteristics are:

- The group is at least partially autonomous in organising themselves,
- the environment of the swarm is dynamic due to the interaction within the group and with the environment,
- each member has a limited sensory capability that can only reach the local area,
- each member has a communication distance that only transmits and received information from nearby members within the group,
- each member may have access to swarms' global behaviour,
- a self-organising swarm has a decentralised control system at the individual level; a centralised controller is not present in the swarm system,
- and all members cooperate to tackle a given task collaboratively.

2.1.1 Various tasks undertaken by Swarm Robotics

With the versatility of swarm robotics, there are various tasks have been achieved. An extensive review of swarm robotics tasks has been done by Bayindir (2016). The author reviewed that the achievable tasks are not limited to aggregation, flocking, foraging, object clustering, navigation, path formation, deployment, collaborative manipulation, task allocation, and many more. In this Subsection, aggregation, flocking and foraging will be overviewed to discuss a few application in the field.

2.1.1.1 Aggregation

Self-organised aggregation is one of the basic behaviours can be easily found in many animal species. This behaviour is defined as a tendency for individuals to group together and takes benefits from the resulting group. When aggregating, the interaction between group members can change the behaviour of the group as a whole. Based on these phenomena, several studies proposed a various aggregation mathematical model. Particularly in swarm robotics, to obtain aggregation in a decentralised manner, the following design methods are commonly used:

- Application of virtual forces (artificial physics)(Mogilner and Edelstein-Keshet 1999; Vanualailai and Sharma 2010; Fetecau 2011; Hackett-Jones, Landman, and Fellner 2012; Fetecau and Meskas 2013; Priolo 2013),
- stochastic control of robot behaviour (Kernbach et al. 2009; Schmickl et al. 2009),
- and artificial evolution (Trianni et al. 2003; Francesca et al. 2012; Gomes and Christensen 2013; Gauci et al. 2014).

2.1.1.2 Flocking

Flocking behaviour is commonly found in a group of birds that are flying together. Other behaviours that are similar to flocking can be found in fish schooling and grazing ungulates. Similar to aggregation, a flocking behaviour also emerges from local interaction in the group and changes the group's behaviour as a whole. Swarm researchers' typical approach to replicate flocking behaviour is by keeping the group in compact formation while moving together. To maintain the group shape, all agents have to measure their neighbours' distance and relative orientation with limited sensing capabilities. However, a single or groups of robots are not considered as a part of the flocking group when they are out of the communication or sensing range of any member in the flock. This raises the assumption that a group of agents is a subset of the flocking swarm if at least one agent is reachable by the group and the swarm.

Flocking has a characteristic that allows the group to move collectively in one direction which is lacking in aggregation (static). Alignment of the agents' heading is the main characteristic that shapes the flocking behaviour. From this characteristic, a flocking model is formulated. For example, the flocking behaviour which was proposed by Reynolds (1987) as discussed previously. Another way to achieve flocking is by endowing each agent to sense or predict nearby agents' movement. Once the neighbours' movement are known, aligning can be achieved by exploiting them to implement flocking, as can be found in these research (Turgut et al. 2008; Fetecau 2011; Çelikkanat and Şahin 2010; Ferrante et al. 2010; Ferrante et al. 2014; Virágh et al. 2014; Yasuda, Adachi, and Ohkura 2014). The flocking is also still achievable in cases where the knowledge of neighbours' heading is not available or unnecessary (Hayes and Dormiani-Tabatabaei 2002; Baldassarre, Nolfi, and Parisi 2003; Antonelli, Arrichiello, and Chiaverini 2010; Moeslinger, Schmickl, and Crailsheim 2010; Ferrante et al. 2012).

2.1.1.3 Foraging

The foraging behaviour's objective is to find items and bringing them to specific location. This behaviour is mainly derived from ants' explorative behaviour in spotting a food source and exploitative behaviour in bringing the food to their nest. Some applications can be done by swarm robotics utilising foraging behaviour are mining, waste cleaning, search and rescue mission, and space exploration. The main characteristics of foraging behaviour is communication between agent. Since foraging is considered as an exploring behaviour, a type of communication can be an environmental modification (stigmergy), direct communication (physical features) or a virtual memory (shared map).

Various studies have been done developing foraging behaviour for decentralised multi-agent system for a single task. For example, multiple foraging robots using stochastic petri-nets (Rongier and Liegeois 1999) and emergent bucket brigading methods (Ostergaard, Sukhatme, and Matari 2001). The analysis studies also have been conducted in this area, such as the effect of interfence between foraging agents (Lerman and Galstyan 2002). Later,

the analytical and spatial foraging model for swarm robotics was proposed by Hamann and Wörn (2007).

Extending the single foraging task, a multi-foraging behaviour is the emerging topic in the area. Multi-foraging task is a behaviour of collecting multiple objects to the predefiend "home" location (Campo and Dorigo 2007). The swarm can be deployed based on allocation time or space. Most of the studies in multi-foraging tasks focuse on minimising energy required to explore and retreive objects (Liu et al. 2007; Pini et al. 2013; Schroeder et al. 2017).

2.2 Swarm Design

The studies of self-organised swarm robotics in developing collective behaviour to tackle various tasks have been studied in more than two decades, as discussed in the previous section. In the past decades, methods to apply decentralised control to achieve collective behaviour has been reviewed by Brambilla et al. (2013). The author grouped them into two class of design: behaviour-based and automatic design. The first is known as a top-down approach since most of the mathematical models were built from the individual formulation. The second models the swarm's behaviour without explicit behaviour and the strategy is developed automatically. These two approaches are discussed in the following subsections.

2.2.1 Behaviour-based Design

The observations of social animals' collective behaviour are mainly the source of inspiration in a behaviour-based approach. Generally, the design routine comprises a cycle consists of behavioural modelling, implementation, evaluation, and model improvement. In recent decades, swarm researchers have proposed swarm behavioural design with this paradigm. For example, Labella, Dorigo, and Deneubourg (2006) modelled the division of labour from ants and applied it on swarm robotics to collectively retrieve an object in the environment extending the mathematical foraging behaviour of ants proposed by Deneubourg et al. (1987). Implementing a collective swarm behaviour to a group of simple robots using a behaviour-based approach is attainable since the models comprise simple rules (Bogue 2008). In this subsection, three forms of modelling approaches commonly used in behaviour-based design will be discussed; a probabilistic approach, particles approach and stigmergy.

2.2.1.1 Probability Finite State Machine

A probabilistic approach that is mostly used to model a class of behaviour was first coined by Minsky (1967), the author proposed it as a Probability Finite State Machine (PFSM). The model consists of states (internal or environmental condition), behaviours and transitions. Each agent selects behaviours based on a probability value influenced by

the environment or inner perception, including other agents' presence. Then, the agent is transitioned to the next state following a probabilistic threshold value. The value can be constant or vary based on the selected behaviour at the previous state as illustrated in Figure 2.2 and 2.3, respectively. A simple aggregation model investigated by Soysal and Şahin (2005) is one of the example with a fixed threshold value. The second can be found in a study of aggregation behaviour of a swarm of cockroach-like robots by Garnier et al. (2009).

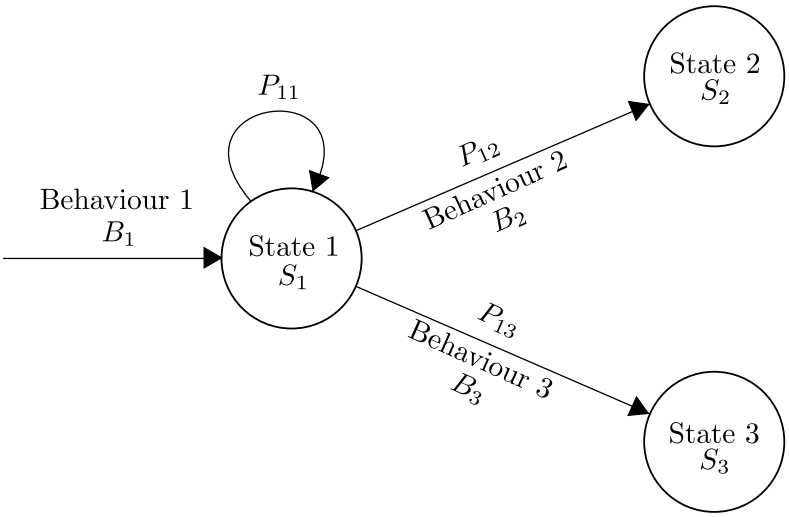


Figure 2.2: A state transition with a constant threshold value

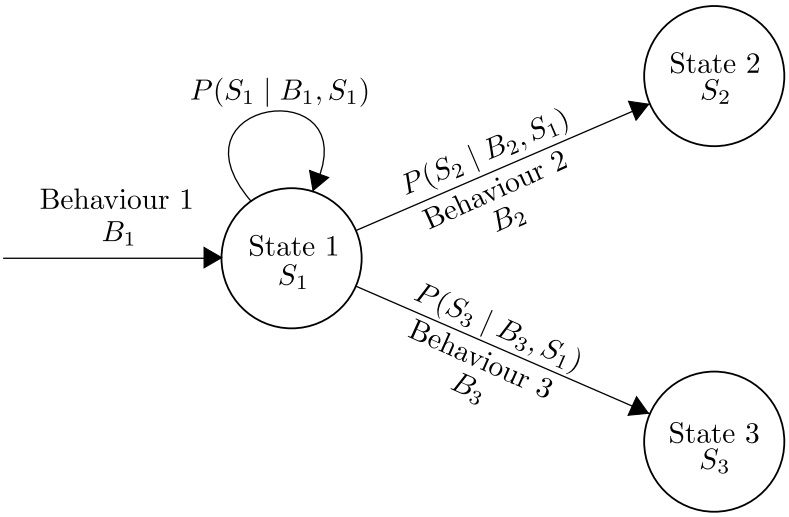


Figure 2.3: A state transition with a varying threshold value

2.2.1.2 Particles and Virtual Physics Design

Considering each agent as a moving particle affected by a virtual potential force is a typical virtual physic-based design. Usually, the force results from the particles' states (position and heading) and an artificial potential field. The resulting force summarises repulsive and attractive forces, allowing each agent to sense any nearby entities and distinguish whether they are robots, targets, or obstacles. Early application on this approach is in a seminal

work by Khatib (1986), which discussed dynamic avoidance in a group of vehicles. The study focuses on formulating a field of artificial forces from the presence of other vehicles. Another notable study was conducted by Reynolds (1987). Later, this design approach also can be found in physicomimetics frameworks (Spears et al. 2004) and a non-local kinetic method for aggregation behaviour (Fetecau 2011).

2.2.1.3 Stigmergy

Another design is by using a virtual medium to share information, such as maps or performance, between agents. The information-sharing of social animals, known as stigmergy, also inspires this particular approach. For example, ants share information using a pheromones trail to create paths to food sources, and bees use dancing to tell the swarm the direction to a nectar source. Several studies have mimicked animals' foraging and flocking behaviour using a characteristic of stigmergy (Labella, Dorigo, and Deneubourg 2006; Ranjbar-Sahraei, Weiss, and Nakisaee 2012).

These three models (PFSM, virtual physics, and stigmergy) are often used as a foundation of swarm behaviour because of their generality. Later with parametric computation, the model can be tuned automatically. Since both models are mostly represented by fixed mathematical formulation, the global behaviour (swarm level) can be easily predicted. However, a behaviour-based paradigm is restricted to a predefined problem to be solved.

2.2.2 Automatic Design

The automatic design aims to develop collaborative and collective swarm's behaviour without an explicit model defined beforehand. Evolutionary computation (EC) and reinforcement learning (RL) have been utilised to accomplish swarm robotics tasks automatically (Brambilla et al. 2013). The implementations of both methods are discussed in the following subsections.

2.2.2.1 Evolutionary Swarm Robotics

Evolutionary swarm robotics is a method to develop collective behaviours through the recombination and mutation processes. Articial chromosomes represents individual behaviour of the swarm and the performance evaluation is derived from its fitness value in accomplishing a task. The recurring improvement is mainly developed around the fittest chromosome at each evolution using a genetic operators: crossover, mutation and selection. A new strategy from each evolution will replace a chromosome with the least fitness value. Finally, the evolution ends when the fitness values meet the predefined criteria. The evolutionary process is illustrated in Figure 2.4.

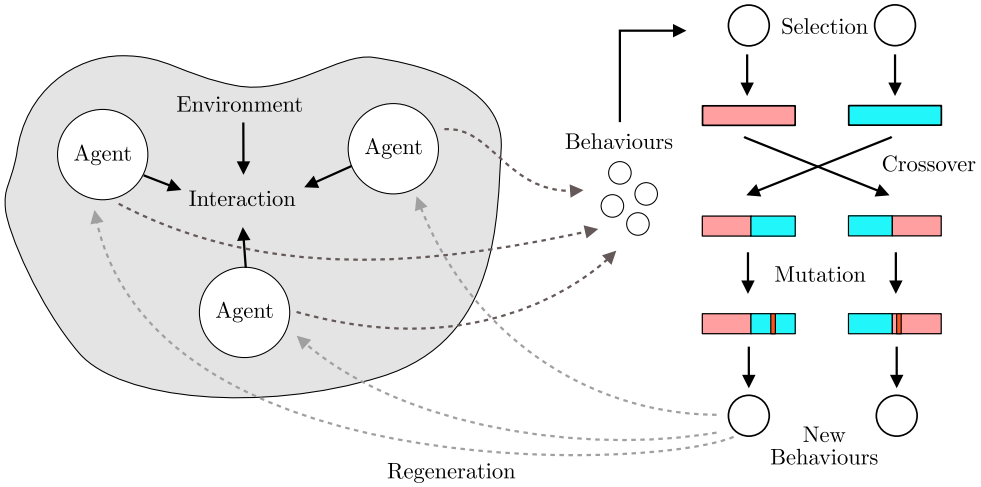


Figure 2.4: Evolutionary Swarm Robotics

Evolutionary computation has a versatility and flexibility to be implemented on various kind of problem and solve it in a reccursive manners (Eiben and Smith 2015). This approach is also beneficial in the development of swarm's collective behaviour despite of the system's non-linearities. Evolutionary approaches in swarm robotics have succesfully demonstrated a collective behaviour such as foraging (Trianni et al. 2003; Francesca et al. 2012; Gauci et al. 2014), flocking (Baldassarre, Nolfi, and Parisi 2003), path formation (Kuyucu, Tanev, and Shimohara 2012), clustering (Hartmann 2005; Gauci et al. 2014), collective object transport (Groß and Dorigo 2004, 2009), and task allocation (Tuci et al. 2008).

2.2.2.2 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) is an extended reinforcement learning method in which the formulation is tailored for a multi-agent system setting (MAS). A notable feature of MARL that uses the reinforcement learning principles is a trial-and-error interactions with the environment, and in the MAS case, the other agents (Kaelbling, Littman, and Moore 1996; Sen and Weiss 1999; Sutton and Barto 1998), as illustrated in Figure 2.5. In learning to choose a optimal action at each environmental state, each agent utilises rewards from all states to develop a policy mapping a set of actions to all available environmental states. Particular to MAS, the environmental state includes the presence of other agents and the decision at each state has to be collective and collaborative.

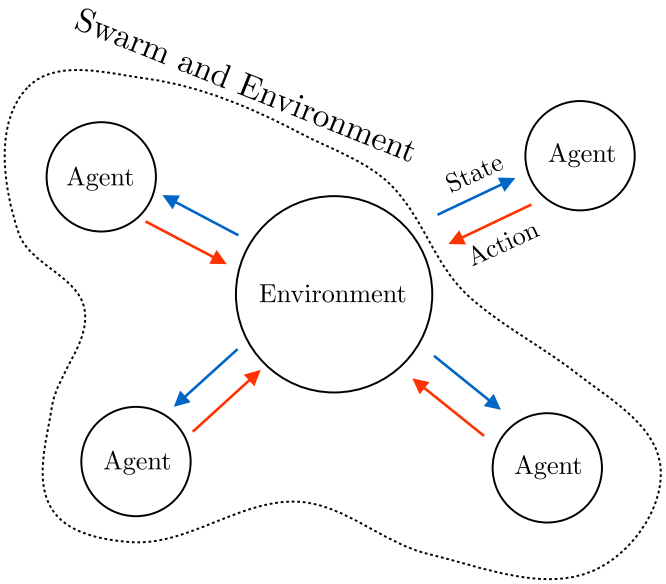


Figure 2.5: Multi-agent reinforcement learning

In a decentralised setting of swarm robotics, MARL mainly comprises four advantages. First, all member of swarms learn the collective behaviour in a parallel setting. Second, the agents shares their learnt best decisions with other agents along with the accumulated knowledge from the interactions. Third, each agent can imitate or teach other agent in the swarm. Last, when an agent fails to accomplished individual task, the other agent will take over performing redundancy to the swarm. A review by (Busoniu, Babuska, and De Schutter 2008) shows that multi-agent reinforcement learning has been implemented and performed quite well in multi-agent systems. For example, a distributive learning was investigated on a swarm robots localising a source of odour by Hayes, Martinoli, and Goodman (2003) and on a stick pulling robots by Li, Martinoli, and Abu-Mostafa (2004). Another application that is very popular in multi-agent reinforcement learning is RoboCup Soccer (Stone, Sutton, and Kuhlmann 2005; Kalyanakrishnan and Stone 2007; Riedmiller et al. 2009).

2.3 Epigenetic in Robotics and Computation

Evolutionary computation commonly uses a fitness function to evaluate solution candidates in solving a problem. The function gives natural pressure for the computational process to approach the optimal solution. The formulation needs to be well-defined and suitable for solving the problem. However, it is often difficult to define a fitness function that sufficiently represents dynamic and complex problems. Lehman and Stanley (2011) stated that the process of approaching a solution based on fitness function does not necessarily lead to the optimal solution, because it may not account the complexity of the problem. This aspect is vital in a computational search problem, since each step taken on the previous generation may be misguiding the search direction. So, a new paradigm to

evaluate the solution in each generation is required to avoid misguidance.

2.3.1 From Darwinism to Lamarckism

Recurring improvement in evolutionary computation has been modelled using the Darwinian model. This model was adapted and implemented in computation known as Genetic Algorithm (GA) (Goldberg 1989; Holland 1992). In this approach, a population of an artificial genetic sequence, commonly represented by bit-string and called a chromosome, is used as solution candidates. Each chromosome is evaluated using a fitness function to check its performance when applied to the problem. Each search iteration in evolutionary computation is known as the generation. After all chromosome's performance in the first generation is known, the selection process occurs. Two chromosomes are chosen in the selection process; a fitness-based selection is commonly used to select two best performing chromosomes. After the selection process, the two chromosomes then recombined through the crossover process. This Recombination process is inspired by the mating process in biology where progenies have the characteristics from both parents but generate a new solution candidate. Although the sequence of genetics in the population may change, it may reach a bounded search where innovation in the search direction is restricted by the same genetic value. To continue to evolve and explore the search space, random mutation is applied to the progenies. This random process gives heuristic search direction to the computation. The mutation process is controlled by stochastic mutation rate as a probability to flip a bit-value in each gene. The mutated progenies are then put back into the population replacing the two least perform chromosomes. The new population is now entering the second generation and repeats the same process from evaluation, selection, crossover, mutation, to regeneration. The computation continues until one of the candidate solutions reach a predetermined tolerance within the expected optimal solution.

From the previous passage, one operator that is used to explore the search space is random mutation. However, random mutation is not sufficient when the fitness function does not well-represent the problem. For complex and dynamic problems, the fitness function may generate a non-optimal direction, because the fitness function tends to be fixed and independent from the problem (Lehman and Stanley 2011). So, utilising random mutation on ill-defined fitness function is deceptive for the search direction. Particular in a dynamic environment, a fitness function is easily degraded because the formulation is gene-centric and untouchable by the environmental changes. Hence, the evolutionary operators and evaluation process should give open-ended improvement, environmentally aware, and offer unbounded innovation.

Another model that is promising in this area is the Lamarckian models. Lamarck proposed an evolution theory explaining that parental experience or adaptation is passed down to the next generation and improves their survival rate (Mayr 1972). Later, in 1984, Waddington showed that every gene has a layer regulating its expression (Waddington 2012). He defined it as Epigenetic, meaning "above gene". This layer regulates a pheno-

typic expression of a gene dependent on a stimulus, such as an environmental condition. The epigenetic layer also decides how and where a gene grows into cells and become a unique organ (ontogenesis). Then, the layer is inheritable to progenies passing down accumulated experience to develop adaptation. By using the concept of epigenetic inheritance, developmental robots and epigenetic-inspired computation were born.

Two body of works that apply an epigenetic mechanism of inheritance and development are epigenetic-inspired computation and epigenetic robotics, also defined as developmental robotics or morphogenesis robotics. The latter aims to study the developmental processes and architectural features in a restrictive environment (embodied machines). In this study, a robot is designed to obtained new skills and knowledge through continuous and open-ended learning. The former aims to develop genetic-based computation utilising inheritable epigenetic properties obtained from interaction with the environment as an external stimulus. The interaction forms a regulatory function to select optimal behaviour (phenotype) in response to the environment. In the following sections, both bodies of works will be reviewed.

2.3.2 Epigenetic robotics

Epigenetic robotics is a research area where the robot's cognitive development is the main focus, as stated by Jin and Meng (2011). The development consists of two mechanisms, ontogenesis and epigenesis as can be find in natural organisms (Berthouze and Ziemke 2003). Epigenesis is a learning process where the external stimulus is processed to strengthen the epigenetic robot's cognitive level. The first is a process of building a new set of cognitive which takes step-by-step learning starting from a simple problem to the complex one, mimicking a maturation process in postnatal development of natural organisms. The combination of the two provides open-ended improvement, unbounded innovation and environtmental awareness.

2.3.3 Epigenetic Computation

Epigenetic computation implements similar setting as evolutionary computation with an addition of inheritable knowledge of gene regulation. Generally, external stimulus from the environment is collected by an articial epigenetic layer to modify the corresponding gene's expression. The regulatory behaviour of the epigenetic layer consists of two mechanisms, introducing or altering a specific expression for a specific external stimulus. Both of the knowledge and regulatory function of epigenetic layer is passed down to the next generation in the evolutionary process, following an inheritance concept in biology (Wang, Liu, and Sun 2017). In summary, the general difference between genetic-based computation and epigenetic-based computation is how each genetic composition is evaluated, the first uses fitness test, while the second using external stimulus or feedback from environment, as can be seen in Figure 2.6. Several works applying this methodology are discussed in the following subsections.

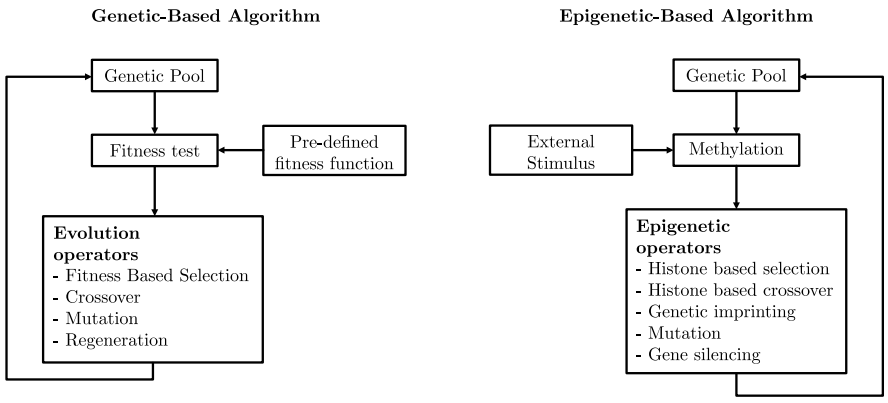


Figure 2.6: Flow diagram of genetic-based algorithm (left) and epigenetic-based algorithm (right).

2.3.3.1 Epigenetic Tracking (ET)

Epigenetic Tracking (ET) is an extended model of "Cell Tracking" utilising evolutionary-developmental techniques (Fontana 2007, 2008). In this work, evolution on genomes guides the development of a single cell to a preferred shape. The model contains two cells categories, namely "driver" cells and "normal" cells. The first cell is instructed by a genome, considered as developmental operators, to increase or induce apoptosis in the surrounding area. Apoptosis is a genetically regulated process leading to cells' death and triggered by the presence or absence of specific stimuli. While increasing, the driver cell mostly generates normal cells and driver cells around it. The area where proliferated around a "mother cell"-a proliferating driver cells is called a development area. A decision-making entity (genome) to increase or apoptosis depends on the cellular genetic type (CET) of the driver cell, in this case, the "mother cell". CET is an epigenetic memory that takes different values across driver cells and represents the source of differentiation during development. All driver cells know what type of cells they are and how their behaviour has to achieve a global target shape. Furthermore, recent work has been done to apply a distributed ET algorithm to generate shape formation in a self-organised swarm (Mishra, Semwal, and Nair 2018).

2.3.3.2 Intra-generational Epigenetic Algorithm (EGA)

Periyasamy, Gray, and Kille (2008) proposed an inra-generational epigenetic algorithm (EGA) which is an optimisation strategies utilising bio-molecules adaptive mechanisms. This work focuses on dynamic adaptive mechanisms that are exercised by biomolecules via epigenetic mechanisms. In the formulation, instead of using genetic sequence (chromosome) as a computational entity, the organisation of genetics' behaviour is used as the computational units. The organisation as a candidate solution is a composite entity that includes genetics combination, arrangement, interaction, and expression. Its structure and behaviour are embodied into computational entity analogous to bio-molecular species. In exploring the candidate solution, an autocatalytic is chosen as an operator to

bound biomolecules to obtain optimal internal organisation structure.

2.3.3.3 Epigenetic Programming (EP)

Tanev and Yuta Tanev and Yuta (2008) proposed a method to control gene expression called Epigenetic Programming. The author applied a histone modification in strongly-typed genetic programming (STGP). Each individual has a double-cell structure derived from the respective chromatin structures in their approach. As results, individuals with similar genotypic structures can express various phenotypic displays. The activation of a specific expression is due to whether it is beneficial to solve a problem. In their work, they introduced non-destructive operations to the individual; there are silencing and activation operators. Both operators help the individual to be preserved from the destructive crossover impact. In general, their study shows that by applying the epigenetic concept to genetic programming contributes to reducing more than 50% computational effort in a dynamic problem such as evolving the behaviour of collaborative predator agents in the predator-prey pursuit problem.

2.3.3.4 Epigenetic Algorithm (EpiAL)

Sousa and Costa (2011) proposed an epigenetic approach for artificial intelligence called Epigenetic Algorithm (EpiAL). The author investigated the mechanism of regulating gene expressions based on the dynamic of the environment. The gene expression is controlled by an activation rule based on the acquired experience accumulated by epigenetic marks similar to histone modification in the previous approaches. Each mark is attached to the respective gene, so the regulatory function is inheritable to the next generation; this mechanism is known as epigenetic inheritance. The result is that phenotypic regulations are preserved along with the acquired knowledge through consecutive generations. This approach is beneficial to adapt and cope with a dynamic environment by regulating each corresponding gene's expression. Thus, the key to developing a robust activation rule in this approach is by gaining more experience from the environment.

2.3.3.5 The Epigenetic Algorithm (epiGA)

Evolutionary operators based on the histone mark embedded in genetic information known as epigenetic mechanisms were proposed by Stolfi and Alba (2018). The operators are used to replace natural evolution in the classic evolutionary algorithm. Rather than recombine and modify genes based on fitness value and probabilistic approach, they utilise histone information as a basis. The authors showed that it is possible to do selection and recombination based on the histone values on gene structures. Similar to the evolutionary algorithm, their model is formulated as a search algorithm for solving complex optimisation problems. Moreover, the author also stated that their approach can be applied to solve different type of problems by tailoring the right epigenetic mechanisms to explore the solution space.

2.4 Challenges

Despite the advantages of both approaches, there are several challenges in the design process. As discussed in the previous section, a behaviour-based design is inspired mainly by the social animals' behaviours. There are two challenges for a behaviour-based design. The first is that the behavioural model is formulated for a specific task. Thus, the modelling process may be suffered in formulating complex behaviour. Secondly, the lack of self-improvement in the behavioural model raises a problem in a dynamic environment. To overcome these problems, the behavioural models are dynamically improved by applying automatic-design on top of it. However, despite the potential of obtaining collectiveness at the ongoing development of the swarm robotics area, up until now, swarm robotics systems with an automatic design have little been explored to tackle a real-world application and are still limited to the in-silico research. This is because there several challenges in applying a computational approach for the swarm system in a real dynamic environment.

2.4.1 Deception

Evolutionary computation and learning mechanisms which are utilised in developing swarm's behaviour often use the objective function as a performance evaluation. A problem arises when a dynamic problem is introduced. The computational process may be directed into a non optimal solution because of an ill-defined objective function, giving a deceitful solution (Lehman and Stanley 2011). To counter this, the predifined formulation has to consider the expected behaviour in the dynamic environmnet. However, as the problem becomes more complex, the formulation also becomes more challenging.

2.4.2 Exploration and exploitation dilemma

Following the deceptive problem of an objective function, exploitation problem arises. The recurring improvement in evolutionary computation is built around the fitness function, selecting the fittest individual in the group. This means the computation exploits current best solution to be developed further and disregard any inept solution. One way to alternate the exploitation is by introducing exploration into the computational process. Exploring a new solution or swarm behaviour is beneficial to open new possibility that may ultimately reach an optimal solution. However, overdoing the exploration process raises a dilemma problem. The behaviour of the swarm will be unstable if the agent perform more exploration than exploitation and the challenge is more apprent in a multi-agent learning system.

2.4.3 Non-stationary behaviour

A multi-agent setting of swarm allows all agent to evolve and learn in a parallel way at the same time in the group. When an agent tries to learn a better behaviour, it has to develop based on other agents' current behaviour. The compensation made by an agent

to improve itself may affect another agent's learning process in the group. Therefore, each agent in a learning swarm is faced with a moving-target or a non-stationary behaviour problem. This phenomena leads to the risk of unstable behaviour. Hence, developing the learning process of the learning swarm is challenging.

2.4.4 The curse of dimensionality

Since the learning swarm's aim is to realise an appropriate behaviour in a given situation, each agent has to map its individual behaviours to the expected environmental states. In a dynamic characteristic of swarm, the computational process suffers from the "cures of dimensionality" where the states grow exponentially as the complexity grow. For example, the states grow may due to the increasing number of agents. Thus, the computation to achieve behaviour for all states requires more computing resources.

2.5 Discussions

2.5.1 Sustaining behavioural diversity

A behavioural diversity should be maintained to overcome the problem of deception and exploitation in an objective-based computation. Solution diversity offers an alternative path for the computational process that may lead to better behaviour. Sustaining behavioural diversity has been proven to improve evolutionary computation efficiency in exploring better solution (Goldberg 1989; Mahfoud 1997; Sareni and Krahenbuhl 1998; Friedrich et al. 2008). An example of maintaining diversity is implementing a cluster of solutions based on the distance between solution, typically calculated using Hamming distance.

2.5.2 Novelty search

Exploring distant behaviours away from the currently available behaviours is one way to overcome the deception problem. The explored behaviour may be unique and beneficial to the solution space. The direction of exploration can use a novelty metric as suggested by Lehman and Stanley (2011). The metric is used to measures the novelty of the explored behaviour utilising the distance from the rest of the available behaviours. The novelty metric is also utilised to map the sparseness of the behavioural solution space. Hence, by applying a novelty search in the computational process of automatic design, the exploration can be focused on the novel area, and similar behaviours can be group together.

2.5.3 Balancing exploration-exploitation

The exploration-exploitation dilemma can be overcome by balancing when to exploit and when to explore. One way to proceed is by applying a stochastic event to the process, for

example, an $\varepsilon - greedy$ exploration method. An agent decides to exploit most of the time with a probability of $(1-\varepsilon)$ and with a small probability ε explores a new behaviour. Another probabilistic method can be used to balance the dilemma is a Boltzman distribution, a "SoftMax" distribution function.

2.5.4 Achieving a Nash equilibrium

Achieving an equilibrium criterion proposed by Nash (1950) in the learning process of a swarm behaviour can overcome the non-stationary behaviour problem. Bowling and Veloso (2001) stated that there are two necessary properties for multi-agent learning. The first is rationality: If an agent's behaviour converges to a stationary point, then the rest of the learning agents will converge to their best-suited behaviours. The second is converges: The agent will converge to a stationary behaviour in respect of other rational agents.

2.5.5 Adaptation

Adaptation is a significant challenge in swarm robotics to thrive in a dynamic environment, especially in developing collective behaviours through evolutionary computation. Incorporating the Lamarckian principle to the evolutionary swarm can provide adaptability in a dynamic environment (Mukhlish, Page, and Bain 2018b, 2018a). The external stimulus from the environment is perceived by a sensory component and a methylation process to build a regulatory function. Then, the regulatory function selects the appropriate behaviour.

2.6 Summary

In this chapter, the recent development of swarm robotics is presented with all the potential and advantages of swarm robotics. An overview to design swarm robotics is also discussed, and the summary of the current practices can be seen in Figure 2.7. Then, challenges in developing swarm behaviour and the improvement direction are also discussed. The discussion is mainly to open the possibility for future development and improvement for swarm robotics design.

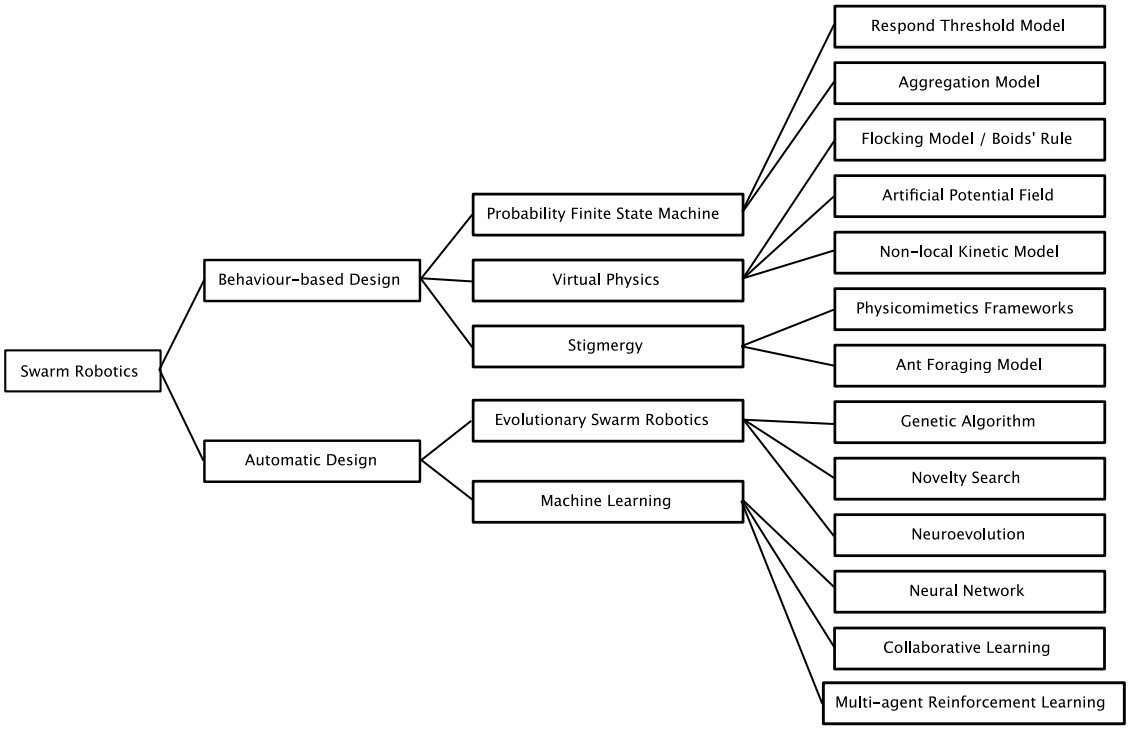


Figure 2.7: Classification of swarm robotics design

The future development of swarm design, especially the automatic design, should focus on the dynamic environment intrinsic to the swarm system. The dynamic environment arises several challenges discussed in this chapter such as deception, the exploration-exploitation dilemma, non-stationary behaviour and the curse of dimensionality. Furthermore, to overcome these challenges, future development has to consider the solution diversity and novelty, the balance between exploration and exploitation, the properties of learning equilibrium in the Nash criterion, and the adaptation capability in a dynamic environment.

Chapter 3

Reward-based Epigenetic Algorithm

In this chapter, reward-based evolutionary computation is discussed. The formulation takes the direction of epigenetic-inspired computation and focuses on accumulating external stimulus and using it as knowledge to obtain better behaviours in the future. There are four main parts of the formulation: namely adaptation framework, genetical representation, epigenetic mechanisms, and methylation process. The first is the foundation underlying the proposed method. Then, the genetic model for a reward-based evolutionary algorithm is discussed. The third demonstrates how the experience can be inherited to the next generation using epigenetic mechanisms. The epigenetic mechanisms utilise the epigenetic factor to recombine and restructure the behaviours. The last is the methylation process which is a process of how the external stimulus (reward) is propagated and embedded into the epigenetic layer. After the formulation, performance tests and results are discussed.

This chapter is organised as follows. The related works to the reward-based evolutionary computation are presented in Section 3.1. Then, the adaptation framework of the proposed method is investigated in Section 3.2. Thirdly, a novel genetic structure is proposed in Section 3.3. The methylation process is derived in Section 3.4. The formulation is finalised with the discussion of epigenetic mechanisms in Section 3.5. Finally, the simulation, results, discussion, conclusions, and summary are presented in Section 3.6, Section 3.7, Section 3.9, and Section 3.8.

3.1 Related Works

3.1.1 Epigenetic-based Algorithm

The early formulation was conducted by Periyasamy, Gray, and Kille (2008). The authors designed the algorithm based on the intra-generational epigenetic mechanism utilised by biological molecules. This algorithm optimises the organisation of molecules forming a cell, a set of solution to a given problem (combinatorial). Each molecule influences the next arrangement, which meets environmental demands. Adaptation operators in this algorithm are introducing (expressed) and eliminating (restrict) a cell molecule. These alterations are affecting the dissemination of the available molecules. This epigenetic strategy applies a different phenotype from the same genotype. Using both approaches, the authors were able to develop a SwarmCell, an autopoietic system to understand disease mechanisms at the sub-cellular level.

Similar works investigated that the regulatory structure, inspired by the epigenetic concept, can handle a dynamic environment (Tanev and Yuta 2008; La Cava and Spector 2015; Stolfi and Alba 2018). A formulation for a single agent was proposed by Sousa and Costa (2011) called as the Epigenetic Algorithm (*EpiAL*) illustrated in Figure 3.1. The figure shows that the epigenetic layer selects a genetic code based on the stimulus measured by the agent's sensor. The epigenetic layer in their work utilises a methyl value to regulate the gene's expression. The methyl value activates the underlying gene based on the activation rule. Each time step, each gene's methyl value is updated using activation value perceived from the sensor operation.

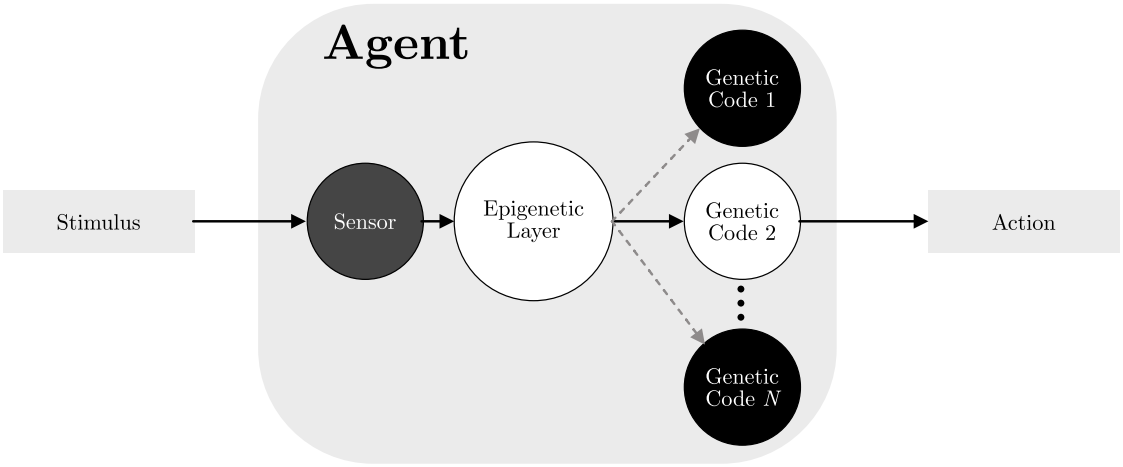


Figure 3.1: Epigenetic algorithm conceptual model (Sousa and Costa 2011)

3.1.2 Epigenetic Operators

An epigenetic layer's regulatory function is achieved from accumulating external stimulus or inherited from the previous generation. The developmental process of building a gene controlling is known as Epigenetic mechanisms (Stolfi and Alba 2018). The mechanisms

utilise histone values which regulate long-term gene's expression and also maintain the genotype over a genetic development and recombination in the subsequent generations.

In all epigenetics operators, the histone value is preserved on each gene. Hence, the epigenetic mechanisms can act as strategy exchange. There are seven epigenetic operators in each evolutionary process which are discussed in Section 3.5, namely histone-based gene selection, histone-based chromosome selection, crossover using histone mask, genomic imprinting, gene mutation, regeneration, and gene silencing.

3.2 Adaptation Framework

In this section, the extended Epigenetic Algorithm (Sousa and Costa 2011) is proposed in which the epigenetic layer is formulated to pick an action's behaviour based on its genetic code in response to the environmental measurement, as can be seen in Figure 3.2. The main objectives are to map the behaviour (genetic code) to the environmental state and improve the map through epigenetic operators as strategies recombination between members of the swarm. The map (solution space) is managed and activated by a regulatory function. Instead of always choosing a known optimal solution (greedy), the ε-greedy activation rule is used as the regulatory function to explore and exploit the solution space. Reward-based feedback or stimulus from the environment is used to update the methyl value while avoiding the performance evaluation by the fitness of objective function. The improved epigenetic algorithm consists of the sensory component, behaviour regulator and learning mechanism.

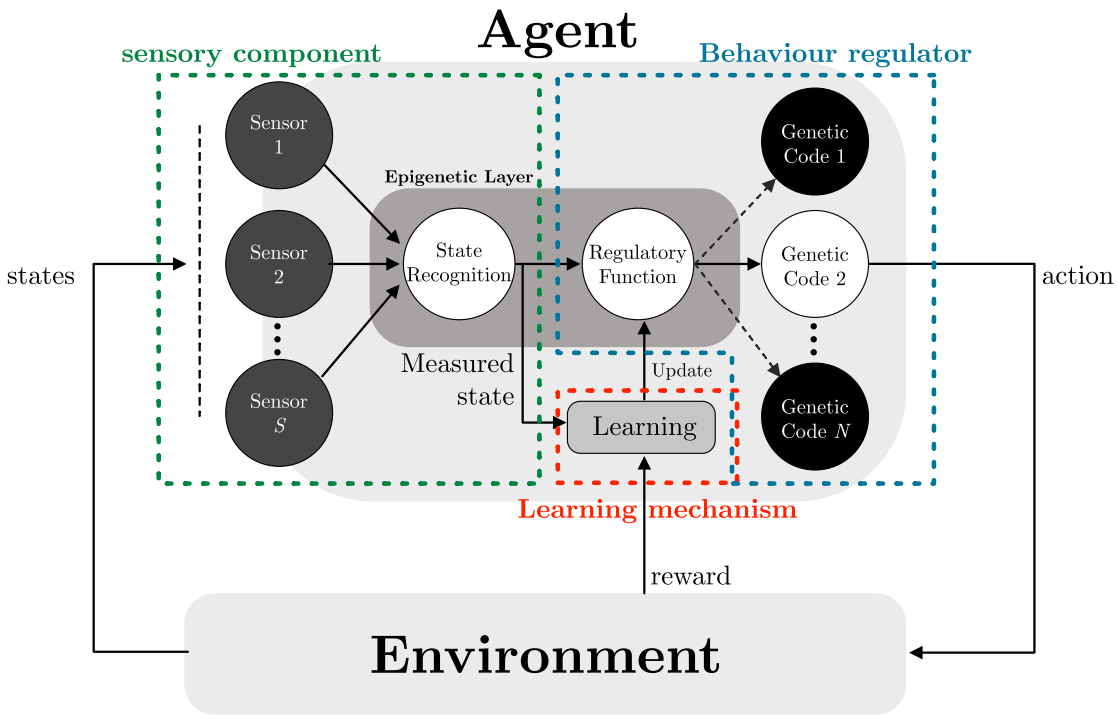


Figure 3.2: Adaptation framework for epigenetic layer

3.2.1 Sensory component

An environmental value can only be quantified using available sensors on board because of the agent's limited sensing capabilities. Generally, the sensor used by the agent is designed to measure a certain unit, such as a proximity sensor with a limited range. Since the knowledge of the environment is important to choose appropriate genetic's expression, the combination of measured values is often used to approximate the current state of the environment. If an instance of environmental dynamics is defined as a true state, then the state inferred from the sensory components at the current time step can be defined as measured state, see Figure 3.2.

The measurement using a sensor array function can be done by designing a state recognition analytically. However, it is challenging to craft a mathematical model of state recognition when environmental complexity increases. To counter this, machine learning is useful for predicting and observing a dynamic system (Pathak et al. 2017) or by inferring the environmental state using a Bayesian estimator. To generate this possibility automatically, the epigenetic layer concept should be improved by incorporating a learning mechanism to determine the dynamics of the environment.

3.2.2 Behaviour Regulator

Behaviour regulator is a component of the adaptation framework that selects an appropriate behaviour for an action given the measured environmental state. The regulatory function is derived from environmental knowledge. Such knowledge can be inferred from the agent's experience through the trial-and-error interaction with other agents and the environment. The agent's current decision is then modified with a recently acquired experience in the form of reward value and environmental state. The obtained rewards are utilised to map the relationship between behaviours and environmental state, yielding an adaptable and optimal decision making. The behaviour-state regulatory function can be done by extending the *EpiAL* as depicted in Figure 3.2.

In the epigenetic layer, phenotypic expressions of a gene are regulated based on the histone value, modified by methylation process. A gene with a higher histone value will be expressed more likely than the one with a lower histone value. Histone value for a gene is obtained from the accumulation of experience from the external stimulus or environmental dynamics. The environment gives a response/stimulus after the gene performs in a particular state of the environment. Then, if the stimulus is positive feedback, the histone value over the selected gene goes more substantial or apparent to the state, and this process is known as methylate phase. On the contrary, the histone value becomes weaker or altered when the stimulus is negative feedback. These methylation marks are then inherited to the next generation as a knowledge inheritance through epigenetics operator, namely selection, recombination, mutation, and regeneration.

3.2.3 Learning Mechanism

The epigenetic framework's learning mechanism is an experience backup process that accumulates reward values and improves behaviours selection in a set of environmental states. The experience acquired by each agent is done by applying an exploration and exploitation process. $\varepsilon - greedy$ is utilised to balance the process, where the exploitation is likely to happen with a probability of $1 - \varepsilon$. The exploration is done once in a while with a small probability ε.

3.3 Genetical representation

3.3.1 Genetic Structure

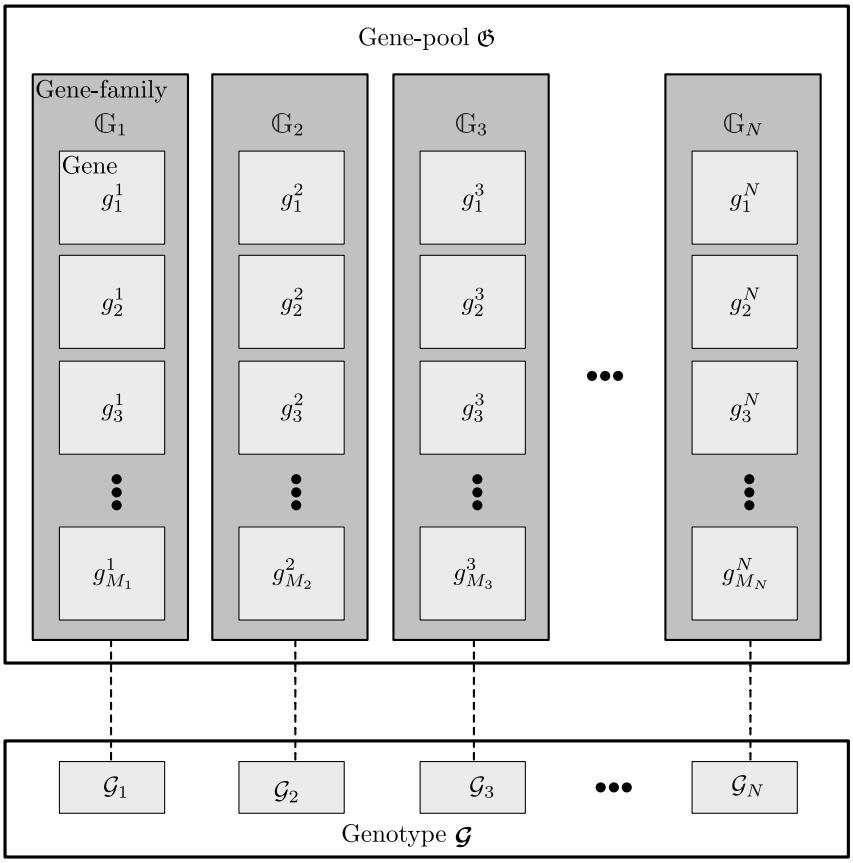


Figure 3.3: The diagram of genetic structure.

In this subsection, a genetic structure for an agent is introduced. The formulation comprises of three sets that correspond to each other, there are gene-pool (denoted by $\mathfrak{G}$), gene-families (denoted by $\mathfrak{G}$) and genotypes (denoted by $\boldsymbol{\mathcal{G}}$), with gene (denoted by g) as the smallest entity. The genetic information in each agent is represented by a genotype $\mathcal{G}(g)$ and a gene's value $\mathcal{V}(g)$ in form of *n-bits* binary code. The set of genotypes consists of several gene types that later is formalised as a sequence of solution (chromosome). A

group of genes with the same gene type is clustered in a gene-family. Then, the gene-pool contains all gene-families. Formally, all sets in the genetic structure are:

$$\text{genotypes } \boldsymbol{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, ..., \mathcal{G}_N\}, \tag{3.1}$$

$$\text{gene-pool } \mathfrak{G} = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, ..., \mathbb{G}_N\}, \tag{3.2}$$

$$\text{gene-family } \mathbb{G}_n = \{g_1^n, g_2^n, g_3^n, ..., g_M^n\}. \tag{3.3}$$

Where $g_m^n = \mathbb{G}_n(m) = \mathfrak{G}(n, m)$ with genotype $\mathcal{G}(g_m^n) = \mathcal{G}_n$. To simplify the location index of each gene in gene-family, index m is used where $m \in \{1, 2, 3, \ldots, M_n\}$. Moreover, Figure 3.3 illustrates the relationship between all the sets.

Binary values represent the genetic code because it allows a value modification by flipping the bit code at a specific location. However, in binary code, the distance between the decimal values is very large. For example, in a 4-bit representation, moving from integer value 11 to 12, represented as 1011 and 1100 respectively, requires three flipping steps, and this is inefficient from a computation standpoint. Another coding system preferred in this case is Gray-code, a binary sequence with a distance of one "flip" between adjacent integers. This cyclic properties can be seen in Table 3.1.

Table 3.1: Conversion from decimal value to Gray-code

Decimal	Binary code	Gray code	Decimal	Binary code	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

3.3.2 Chromosome Structure

A chromosome (C) consists of a sequence of genes with a different type from each gene-family. A gene's position in the chromosome follows the gene-family's index n in $\mathfrak{G}$, $\mathbb{G}_n$, as can be seen in Figure 3.4, where each $g_m^n \in \mathbb{G}_n$ and $m \in \{1, 2, 3, \ldots, M_n\}$.

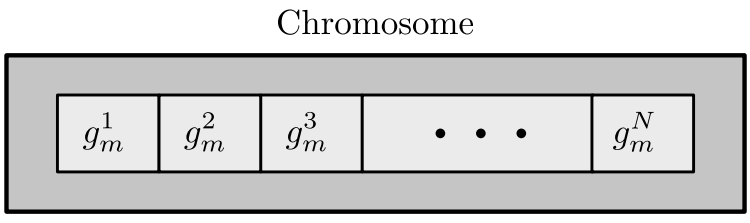


Figure 3.4: Chromosome structure.

A chromosome is a solution candidate containing decision variables to solve a given prob-

lem. To explore and open a new direction to approach the solution, several solution candidates are selected with a population L, as shown in Figure 3.5. The set of solution candidates is denoted as chromosome-pool $\mathfrak{C}$ where the member is defined as

$$\mathfrak{C} = \{C_l \mid l = \{1, 2, 3, \ldots, L\}\} = \{C_1, C_2, C_3, \ldots, C_L\} \tag{3.4}$$

and the chromosome C_l with a complete notation is defined as

$$C_l = \{{}^l g_m^n \mid \{n = 1, 2, 3, \ldots, N\}\} = \{{}^l g_m^1, {}^l g_m^2, {}^l g_m^3, \ldots, {}^l g_m^N\} \tag{3.5}$$

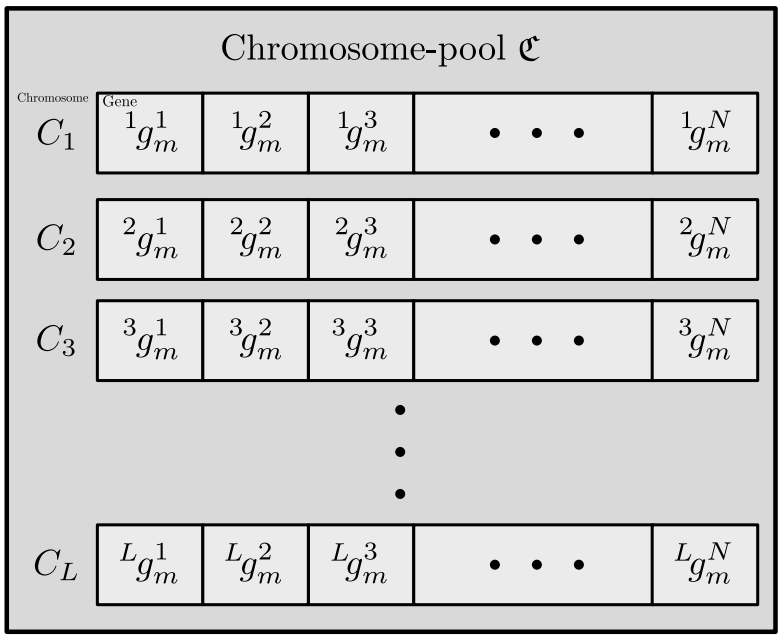


Figure 3.5: Chromosome-pool

3.3.3 Histone Layer

In the study of epigenetics, histone is a protein spooling around DNA and playing a role in gene regulation. Generally, DNA sequences that are wrapped and packed by histone are not expressed. On the other hand, the unpacked DNA sequences are expressed. By taking this principle, histone's behaviour is analogous to decision-making as which genes are worth expressing based on their performance value. Let's define a histone entity for each gene, denoted as $\mathcal{H}(g_m^n) = h_m^n$. As illustrated by Figure 3.6, histone values are embedded in each gene in gene-pool. In a gene-family, a set of histones is denoted as:

$$\mathbb{H}(\mathbb{G}_n) = \{h_m^n \mid m = \{1, 2, 3, \ldots, M_n\}\} \tag{3.6}$$

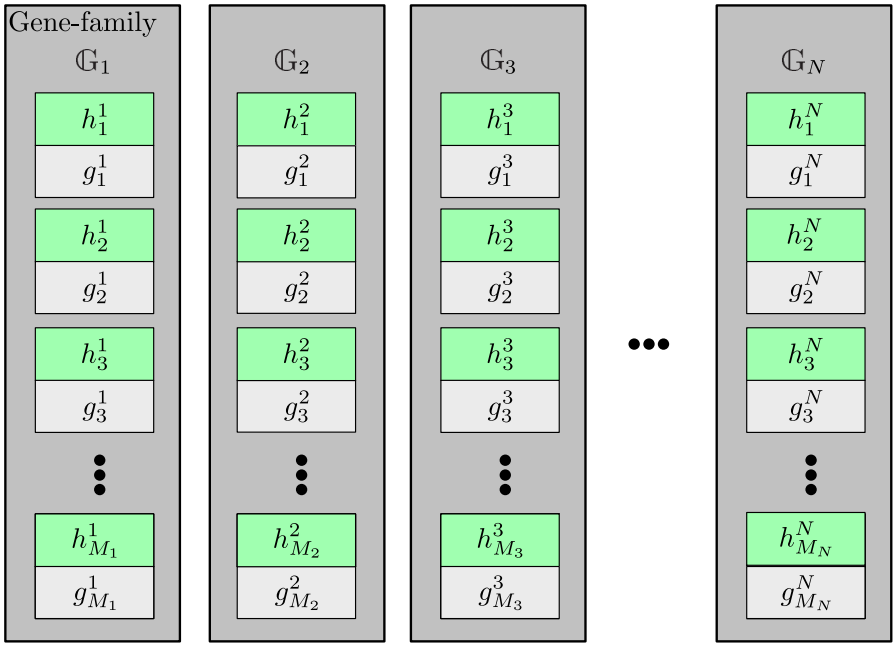


Figure 3.6: Histone values at each gene's location

For a chromosome, each selected gene is accomplished by the respected histone layer, see Figure 3.7

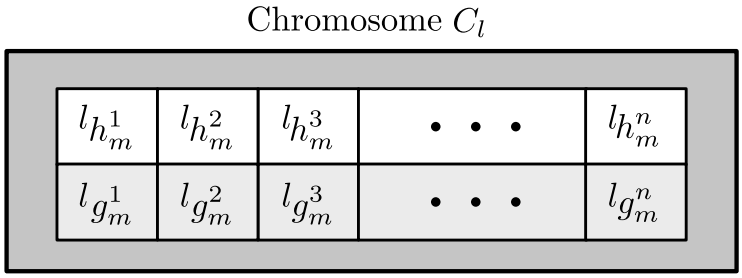


Figure 3.7: Chromosome with histone values

Since now each chromosome has a set of histone values, let us denote the set of histones for chromosome C_l as

$$\mathbb{H}(C_l) = \{{}^l h_m^n \mid n = \{1, 2, 3, \ldots, N\}, m = \{1, 2, 3, \ldots, M_n\}\} \tag{3.7}$$

where m_n is a selection index from the chromosome's perspective. Then, the overall histone value of each chromosome can be calculated as an average value of all histone values in

the chromosome, denoted as H and let us derive it as

$$H(C_l) \triangleq \bar{\mathbb{H}}(C_l) = \frac{\sum_{n=1}^{N} {}^{l}h_m^n}{N} \tag{3.8}$$

3.4 Methylation Process

Interaction between agent and environment can take many forms. It can be a direct feedback in the form of a fitness function or reinforcement function. The fitness function accounts for the agent's approximate performance given the environment, such as the absolute squared error or time to finish a task. One may argue that the derivative of the fitness function over each gene can be used to obtain a gradient-based update to each gene's histone value. However, formulating the contribution of each agent into a fitness function is very challenging. If the number of genes in the chromosome increase, then the numbers of dimensions and nonlinearities of the fitness function also increase. Hence, for the methylation process, incremental and accumulated feedback in the form of reinforcement is used rather than function-based feedback.

A reward is propagated and accumulated incrementally into a form of histone value, denoted as h at the gene level and H at the chromosome level. Since the histone value is updated incrementally, we can represent the current approximation update as $\tilde{H}$. Then, we can define the approximation error as the squared difference between final value H and approximate value $\tilde{H}$,

$$\delta_H = \frac{1}{2}\left(H - \tilde{H}\right)^2 \tag{3.9}$$

From equation (3.9), the approximation value $\tilde{H}$ will be equal to H when $\frac{d\delta_H}{d\tilde{H}} = 0$. The slight changes by each gene member ${}^{l}g^{n}{}_{m}$ over the approximation error is $\frac{\partial \delta_H}{\partial {}^{l}h^{n}{}_{m}}$. By using partial derivative, the gradient error is derived as

$$\begin{aligned}\frac{\partial \delta_H}{\partial {}^{l}h_m^n} &= \frac{\partial \delta_H}{\partial \tilde{H}} \cdot \frac{\partial \tilde{H}}{\partial {}^{l}h_m^n} = -2(H - \tilde{H}) \cdot \frac{\partial}{\partial {}^{l}h_m^n} \left(\frac{\sum_{n=1}^{N} {}^{l}h_m^n}{N} \right) \\ &= -\frac{2(H - \tilde{H})}{N}\end{aligned} \tag{3.10}$$

Equation (3.10) is used as an update for the histone at the gene level ${}^{l}h_m^n$. Because the gene ${}^{l}g_m^n$ may be expressed on the other chromosomes, a small update is required for the

approximate value $\tilde{H}$ to update incrementally. So, the update for histone at the gene level is

$$\begin{aligned} h' &= h - \eta \frac{\partial \delta_H}{\partial h} \\ &= h + \frac{2\eta}{N}(H - \tilde{H}) \end{aligned} \tag{3.11}$$

Where $\eta \in (0, 1]$ is a step-size incremental update and we can call it as methylation rate. Since the reward is utilised to update the histone value, then the final value is equivalent to the obtained reward $H \equiv R$. Then, equation (3.11) becomes

$$h' \equiv h + \frac{2\eta}{N}(R - \tilde{H}) \tag{3.12}$$

Mainly, if the reward is positive, +1, for example, the process is methylation. If a negative reward is achieved, -1, then demethylation occurs. If the reward is zero, then the process is a decaying process as happens in ageing cells. The overall methylation process is illustrated in Figure 3.8.

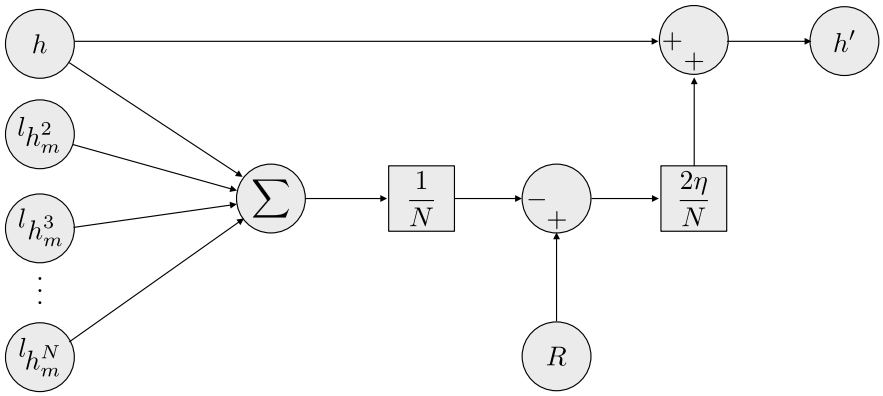


Figure 3.8: Methylation Process

Finally, the approximate value at the next step is defined as an average value of all methylated gene's histone as

$$\tilde{H}' = \frac{\sum_{n=1}^{N} {}^{l}h'^{n}_{m}}{N} \tag{3.13}$$

3.5 Epigenetic Mechanisms

The setting that is preferred in this research is a decentralised multi-agent system (swarm). Each agent has its gene-pool, and later through epigenetic mechanisms will exchange its strategy with other agents. Epigenetic mechanisms are used as operators to evolve and develop new behaviour. All the processes in the evolutionary procedure considers all histone values of a gene-family $\mathbb{H}(\mathbb{G}_n)$ and chromosome-pool $\mathbb{H}(\mathbb{C}_n)$ as a basis. Suppose a population of a swarm is P and each agent has L_p chromosomes, with p is the index of the agent. In the mating pool or available chromosome in the computational process is illustrated in Figure 3.9 below.

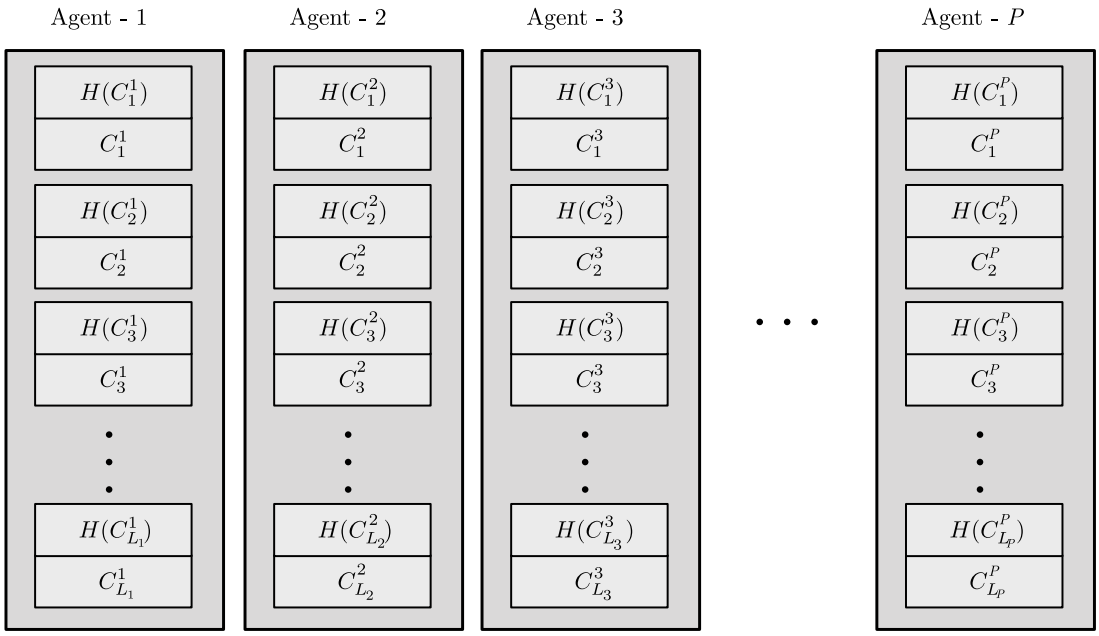


Figure 3.9: All available chromosomes at swarm setting

3.5.1 Histone-based Gene Selection

The selection process for all gene members in a chromosome utilises all available genes and histone values from the gene-pool. Each location (locus) of a chromosome is filled with a gene from the respected gene-family. At the initial condition of the computation process, the gene's composition is selected randomly from the gene-pool. At the gene selection process, the introducing (expressing a gene) follows exploration-exploitation rule using epsilon greedy selection, $\varepsilon_{\text{gene}}$. The selection exploits the gene-family to obtain a gene with maximum histone values with a probability $(1 - \varepsilon_{\text{gene}})$, and explore the gene-family with a probability of $\varepsilon_{\text{gene}}$ following a normal distribution $P_{\text{genes}} \in \mathcal{U}(0, 1)$ gives an equal chance for all available genes in a gene-family to be selected. Thus, the selection process

for each gene in chromosome C_l is formulated as:

$$ {}^{l}g_m^n = \begin{cases} \arg\max\limits_{g} \mathbb{H}(\mathbb{G}_n) & \text{, if } \varepsilon_{\text{gene}} \geq P_{\text{genes}} \\ g \in \mathbb{G}_n & \text{, otherwise} \end{cases} \tag{3.14} $$

3.5.2 Histone-based Chromosome Selection

The chromosome selection process utilises histone values at the chromosome-pool $\mathbb{H}(\mathfrak{C}^p)$. Firstly, each agent offers the best chromosome to the mating pool. Let us denote the set of all chromosome's average histone values for each agent is denoted as $\mathbb{H}(\mathfrak{C}^p)$. Each best chromosome is selected by $\arg\max\limits_{C_l} \mathbb{H}(\mathfrak{C}^p)$ if the selection rate $\varepsilon_{\text{chromosome}}$ is applied following $P_{\text{chromosomes}} \sim \mathcal{U}([0,1])$, the chromosome is selected randomly. Thus, the chromosome offered to the mating pool is

$$ C_*^p = \begin{cases} \arg\max\limits_{C} \mathbb{H}(\mathfrak{C}^p) & \text{, if } \varepsilon_{\text{chromosome}} \geq P_{\text{chromosomes}} \\ C \in \mathfrak{C}^p & \text{, otherwise.} \end{cases} \tag{3.15} $$

Then, we have a set of chromosomes in the mating pool ($\mathfrak{C}_*$), all average histone values $\mathbb{H}(\mathfrak{C}^*)$ and each chromosome's genetic histone values $\mathbb{H}(C_*^p)$ are

$$ \mathfrak{C}_* = \{C_*^1, C_*^2, C_*^3, \ldots, C_*^P\}, \tag{3.16} $$

$$ \mathbb{H}(\mathfrak{C}_*) = \{H(C_*^1), H(C_*^2), H(C_*^3), \ldots, H(C_*^P)\} \tag{3.17} $$

$$ \text{and } H(C_*^p) = \frac{1}{N}\sum_{n=1}^{N}{}^{*}h_m^n. \tag{3.18} $$

Then, each agent selects two chromosomes from the mating pool proportional to average histones $\mathbb{H}(\mathfrak{C}_*)$. The probability of each chromosome to be selected is defined by

$$ \Pr(C_*^p) = \frac{H(C_*^p)}{\sum\limits_{i=1}^{P} H(C_*^i)}. \tag{3.19} $$

Finally, the two selected chromosomes are denoted by $\hat{C}_1$ and $\hat{C}_2$ with gene members as follows:

$$ \hat{C}_1 = \{\hat{g}_1^1, \hat{g}_2^1, \hat{g}_3^1, \ldots, \hat{g}_N^1\} \tag{3.20} $$

$$ \hat{C}_2 = \{\hat{g}_1^2, \hat{g}_2^2, \hat{g}_3^2, \ldots, \hat{g}_N^2\} \tag{3.21} $$

3.5.3 Crossover Using Histone Mask

Crossover or recombination between two chromosomes is a process of exchanging information to get a new genetic sequence and yield a unique solution utterly different from its origin or parents. From the chromosome selection, the corresponding histone sequence for the selected chromsomes are

$$\hat{\mathbb{H}}_1 = \left\{\hat{h}_1^1, \hat{h}_2^1, \hat{h}_3^1, ..., \hat{h}_N^1\right\} \tag{3.22}$$

$$\text{and } \hat{\mathbb{H}}_2 = \left\{\hat{h}_1^2, \hat{h}_2^2, \hat{h}_3^2, ..., \hat{h}_N^2\right\}, \tag{3.23}$$

where their average histone values are

$$\tilde{H}\left(\hat{C}_1\right) \equiv \bar{\mathbb{H}}\left(\hat{C}_1\right) = \frac{1}{N}\sum_{n=1}^{N}\hat{h}_n^1 \tag{3.24}$$

$$, \text{and } \tilde{H}\left(\hat{C}_2\right) \equiv \bar{\mathbb{H}}\left(\hat{C}_2\right) = \frac{2}{N}\sum_{n=2}^{N}\hat{h}_n^2 \tag{3.25}$$

The chromosome recombination usually reconstructs a new chromosome by combining two sequences at a predefined crossover point. Due to the presence of histone values, crossover points can be specified for each gene by following a binary function

$$\mathbb{B}\left(\hat{h}_n\right) = \begin{cases} 1, & \text{if } \hat{h}_n \geq \tilde{H}\left(\hat{C}\right) \\ 0, & \text{otherwise .} \end{cases} \tag{3.26}$$

The sequence of value $\mathbb{B}\left(\hat{\mathbb{H}}\right)$ for both chromosomes is a histone mask. To keep the better genes, crossover operation only happens on the locus that yield "0" from "OR" operation between histone masks, see Figure 3.10. By using the crossover mask, two new

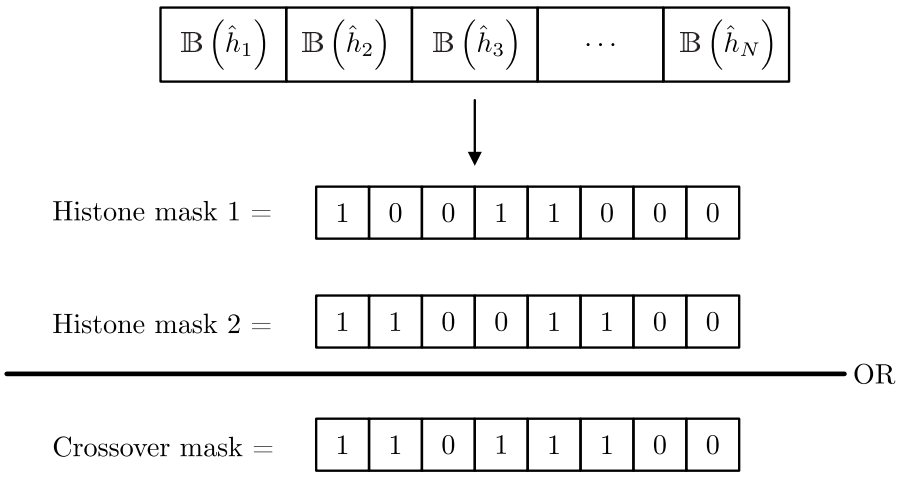


Figure 3.10: Crossover mask for recombination

chromosomes are obtained, $\hat{C}_1^{\text{child}}$ and $\hat{C}_2^{\text{child}}$. The crossover process is illustrated in Figure 3.11.

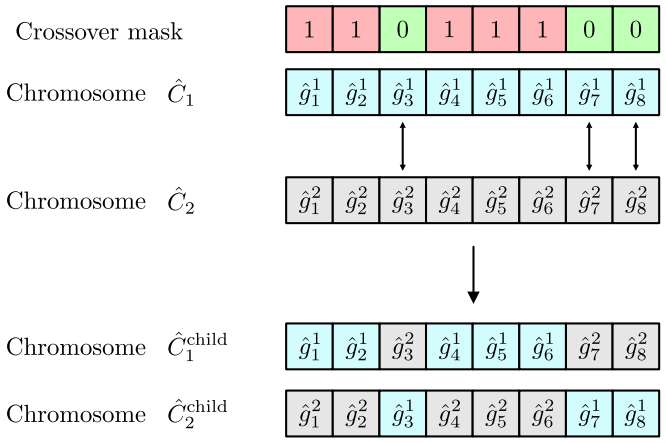


Figure 3.11: Crossover process using crossover mask

3.5.4 Genomic Imprinting

A genetic expression may depend on its origin, whether it is paternal or maternal. An individual typically has two genes from its parent, with the same genotypes, and it only expresses one gene. To apply the same effect, imprinting rate ($\varepsilon_{\text{imprinting}}$) with random process $P_{\text{imprinting}} \sim \mathcal{U}[0, 1]$ is introduced. The genomic imprinting is applied as an epigenetic mechanism to alter a crossover at the specific location, as illustrated in Figure 3.12.

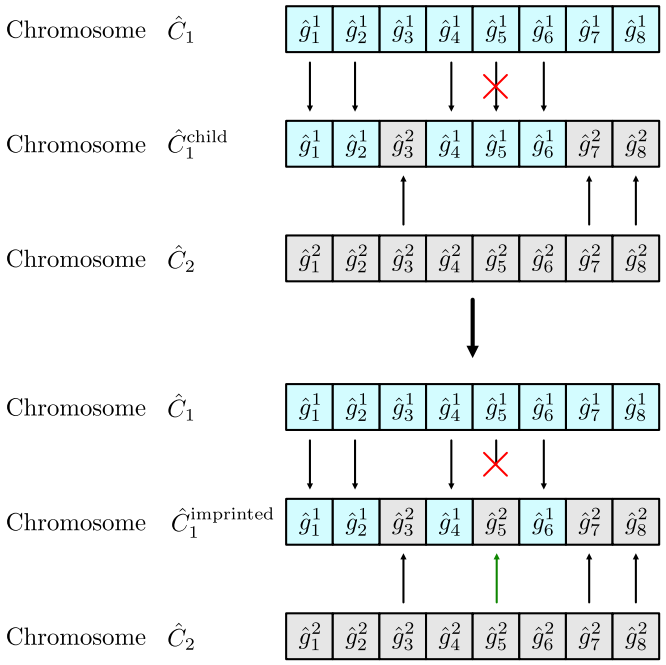


Figure 3.12: Genomic Imprinting process

3.5.5 Gene Mutation

The mutation process comprises two steps. Firstly, a gene is mutated according to mutation rate (μ) following a random process $P_{mutation} \sim \mathcal{U}([0,1])$. Secondly, the binary code is flipped following the same random process, changing incrementally based on the gray-code representation. For example, from Table 3.1, decimal number 7 is represented by grey-code as 0100, changing the decimal value to 8, 1100, only requires the first bit to be flipped. On the other hand, with binary code, changing 7 to 8 requires all bits to flip from 0111 to 1000. The mutation process is shown by Figure 3.13 and following the formulation:

$$\text{gene's value } \mathcal{V}(g_n) \longleftarrow \begin{cases} \acute{\mathcal{V}}(g_n), & \text{if } \mu \geq P_{mutation} \\ \mathcal{V}(g_n), & \text{otherwise} \end{cases} \tag{3.27}$$

$$\text{mutated value } \acute{\mathcal{V}}(g_n) = \{g_n(k) \in \mathbb{B} \mid \mu \geq P_{mutation} \rightarrow \neg g_n(k)\} \tag{3.28}$$

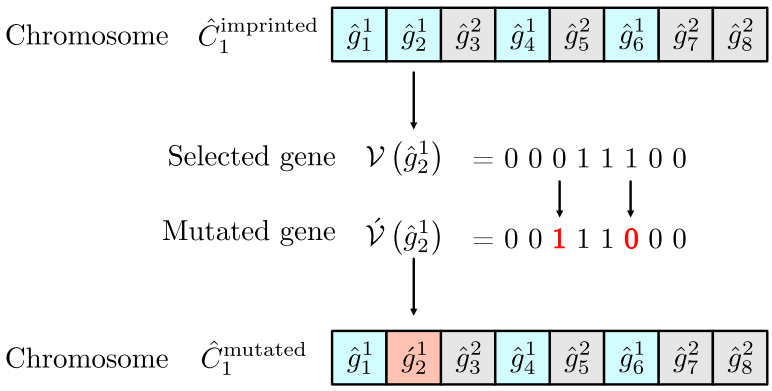


Figure 3.13: Mutation process.

3.5.6 Regeneration

The regeneration process replaces the chromosome in current chromosome-pool with a new chromosome from the epigenetic recombination and modification. The process follows a regeneration rate $\varepsilon_{regeneration}$ based on the random process $P_{regeneration} \sim \mathcal{U}([0,1])$ to decide whether accept the new chromosome or explore the current gene-pool to construct a new chromosome based on the gene selection in Subsection 3.5.1, the rule is defined as

$$\hat{C}^{\text{regeneration}} = \begin{cases} \hat{C}^{\text{mutated}} & \text{, if } \varepsilon_{regeneration} \geq P_{regeneration} \\ \hat{C}^{\text{selection}} \text{ by (3.14)} & \text{, otherwise.} \end{cases} \tag{3.29}$$

3.5.7 Gene Silencing

Silencing process is an epigenetic mechanism to "switching off" a gene in the gene-family and preventing it from being selected or expressed in a solution space, chromosome-pool. A gene with the least histone value compared to all genes in the gene-family is silenced following a silencing rate ($\varepsilon_{\text{silencing}}$) based on a random process $P_{\text{silencing}} \sim \mathcal{U}[0, 1]$. The silenced gene in a chromosome then is replaced by exploring or exploiting the gene-family based on the formulation in Subsection 3.5.1.

3.5.8 Summary of Epigenetic Mechanisms

The evolutionary learning algorithm consists of a genetic structure, a learning process, and an evolutionary process. At first, all agents initialise a gene-pool based on decision variables from the given problem. Then, the agent constructs a chromosome-pool to set off initial solutions to the problem. In the next stage, the agent enters the learning phase. The chromosomes are evaluated, and rewards are given by the environment (given problem). Each time step, the obtained reward is backed up to the histone value through methylation process. The learning process goes until the solution criteria is met. The evolutionary process using epigenetic mechanisms is a process of exchanging strategy and regeneration process. The mechanism involves histone-based operations such as selection, crossover, imprinting and silencing. Lastly, the parameters of the evolutionary process that need to be tuned are the gene and chromosome selection rate ($\varepsilon_{\text{gene}}, \varepsilon_{\text{chromosome}}$), genomic imprinting ($\varepsilon_{\text{imprinting}}$), mutation rate ($\mu$), silencing rate ($\varepsilon_{\text{silencing}}$), regeneration rate ($\varepsilon_{\text{regeneration}}$), and methylation rate (η). The evolutionary process can be seen in Figure 3.14.

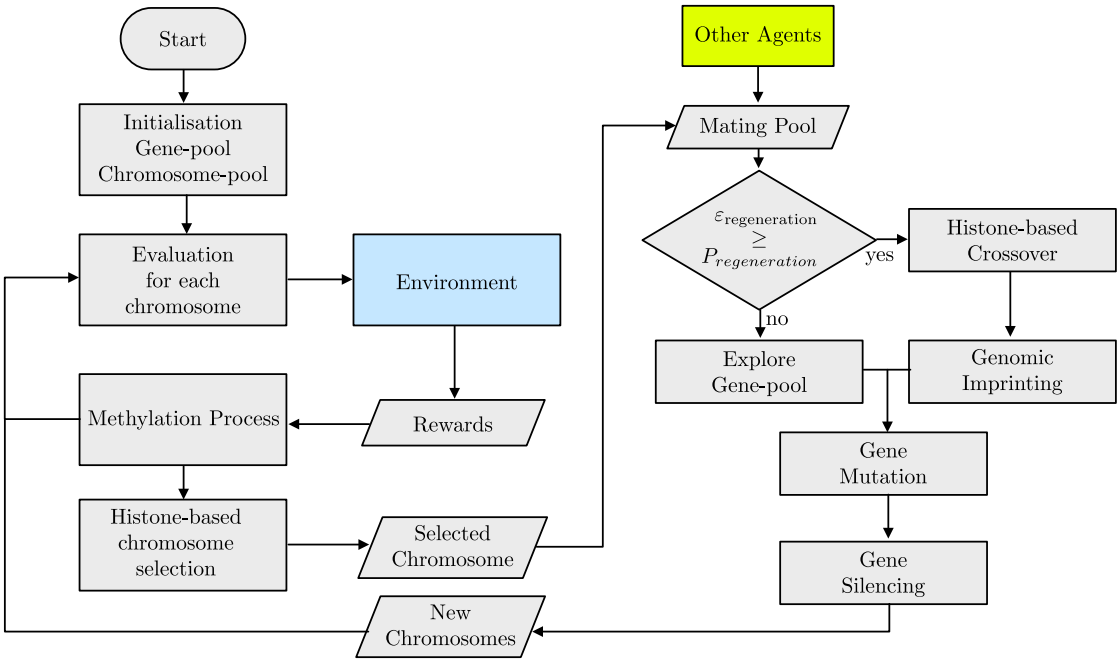


Figure 3.14: Evolutionary process of reward-based Epigenetic Algorithm

3.6 Performance Tests

Optimisation test functions are utilised to investigate the performance of the proposed evolutionary-learning algorithm. There are Sphere function, Levy function and Ackley function. Each function as a specific characteristic from single optimal value to multi-local minima. Parameters used for the performance test can be seen in Table 3.2

Table 3.2: Testing parameters

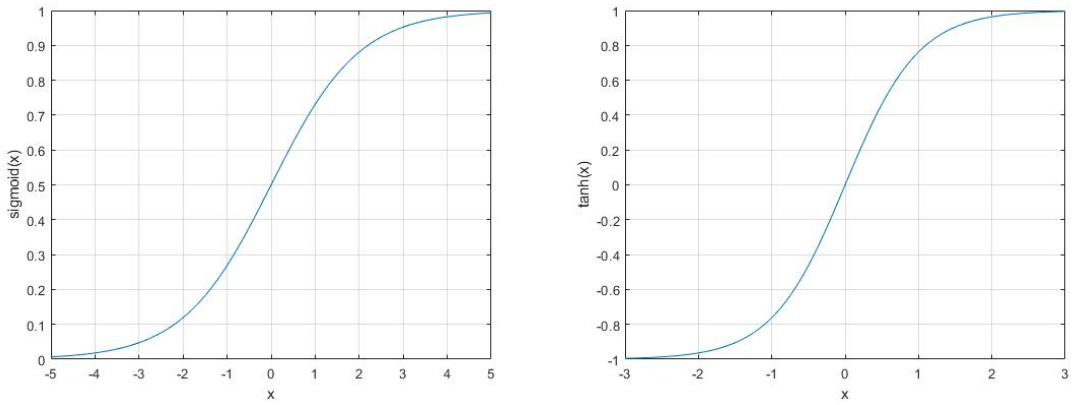
Parameters	Value	Parameters	Value
Generation	1000	$\mu_{mutation}$	0.1
Runs per Generation	10	$\eta_{methylation}$	0.1
Number of Agent	3	$\varepsilon_{chromosome}$	$[0, 1]$
$n - bits$	32	$\varepsilon_{regeneration}$	$[0, 1]$

Sigmoid and hyperbolic tangent function are selected as a rewarding function based on outputs difference between iterations. For sigmoid, see Figure 3.15a, the formulation is

$$R = \text{sigmoid}\left(f_k - f_{k-1}\right), \tag{3.30}$$

and the formulation of hyperbolic tangent,see Figure 3.15b is

$$R = \tanh\left(f_k - f_{k-1}\right). \tag{3.31}$$



(a) Sigmoid function (b) Hyperbolic tangent function

Figure 3.15: Rewarding function

3.6.1 Sphere function

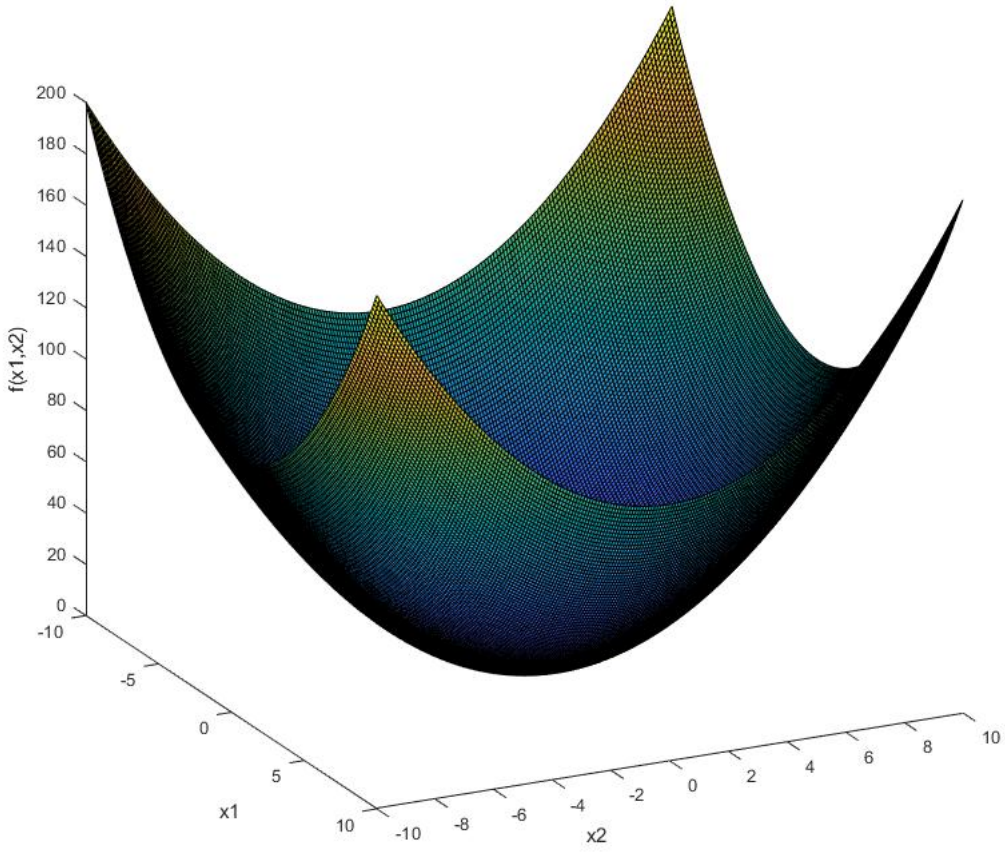


Figure 3.16: Sphere function with $d = 2$

The sphere function is a bowl-shaped function with a global minimum, see Figure 3.16. The function has a decision variables x_i for all $i = 1, 2, 3, \ldots, d$. The function is defined as

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{3.32}$$

The equation (3.32) is evaluated on the hypercude $x_i \in [-10, 10]$ with $d = 2$. The result with a hyperbolic tangent rewarding function is shown in Figure 3.17 and Figure 3.18 for sigmoid rewarding function.

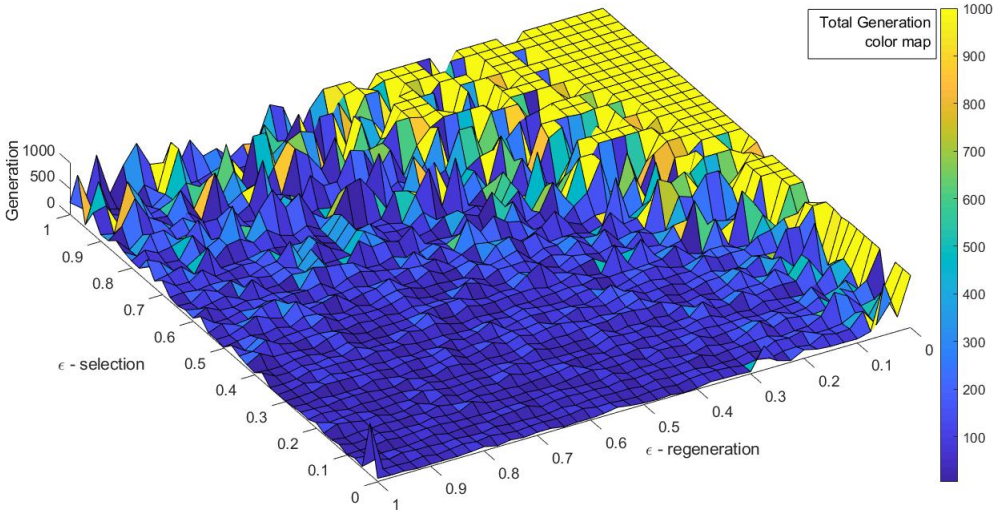


Figure 3.17: The result of the sphere test function using hyperbolic tangent rewarding function

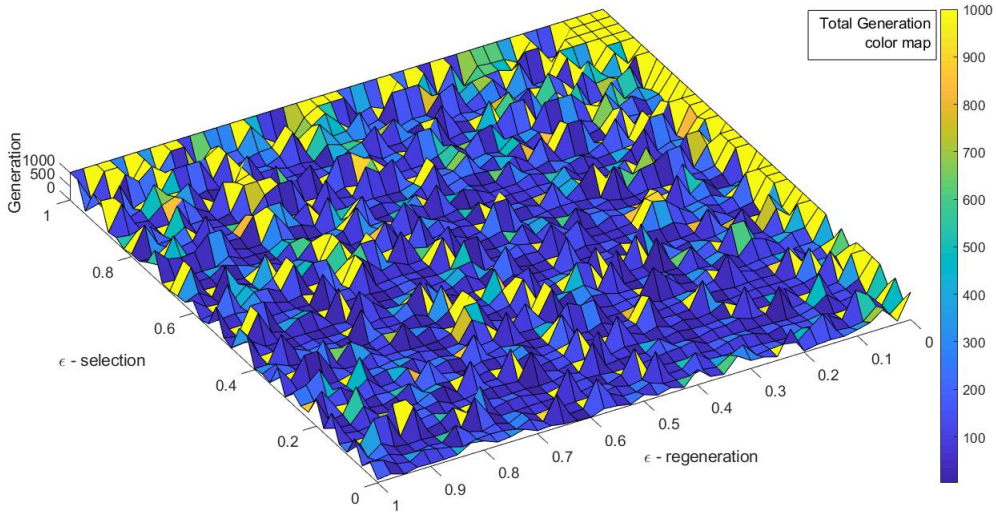


Figure 3.18: The result of the sphere test function using sigmoid rewarding function

3.6.2 Levy function

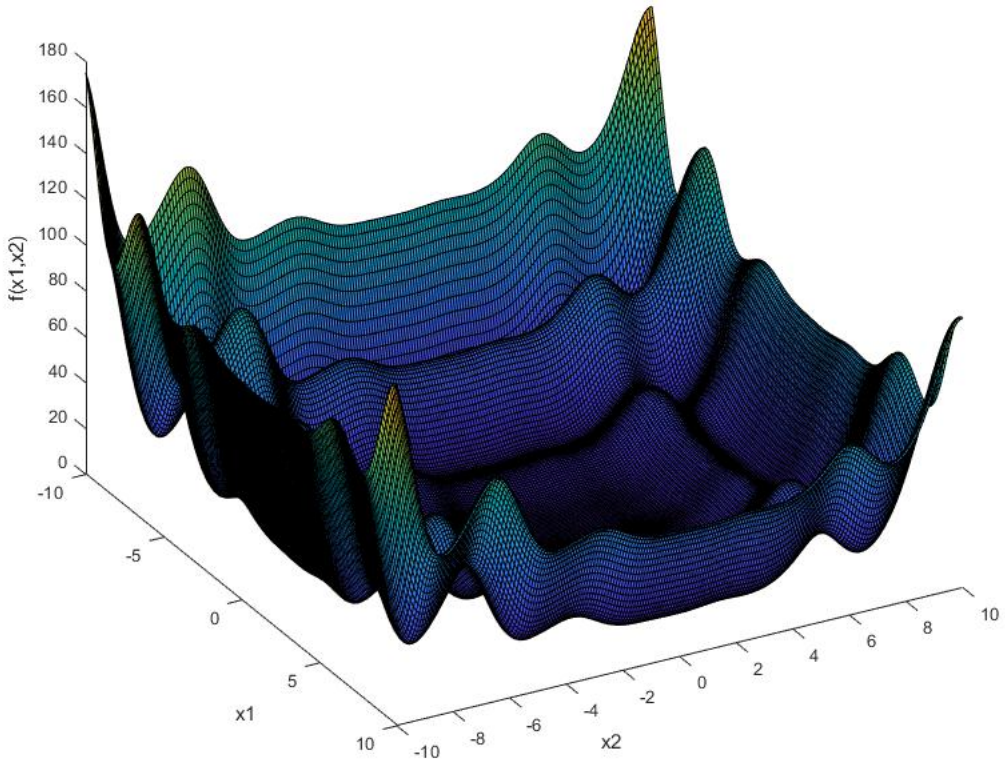


Figure 3.19: Levy function with $d = 2$

The Levy function is a standard function with a global minimum and many local minima, see Figure 3.19. The function has a decision variables x_i for all $i = 1, 2, 3, \ldots, d$. The function is defined as

$$f(x) = \sin(\pi\omega_i) + \sum_{i=1}^{d-1}(\omega_i - 1)^2 \left[1 + 10\sin^2(\pi\omega_i + 1)\right] + (\omega_d - 1)^2 \left[1 + \sin^2(2\pi\omega_d)\right] \quad (3.33)$$

Where $\omega_i = 1 + \frac{x_i - 1}{4}$, for all $i = 1, 2, 3, \ldots, d$. The equation (3.33) is evaluated on the hypercude $x_i \in [-10, 10]$ with $d = 2$. The result with a hyperbolic tangent rewarding function is shown in Figure 3.20 and Figure 3.21 for sigmoid rewarding function.

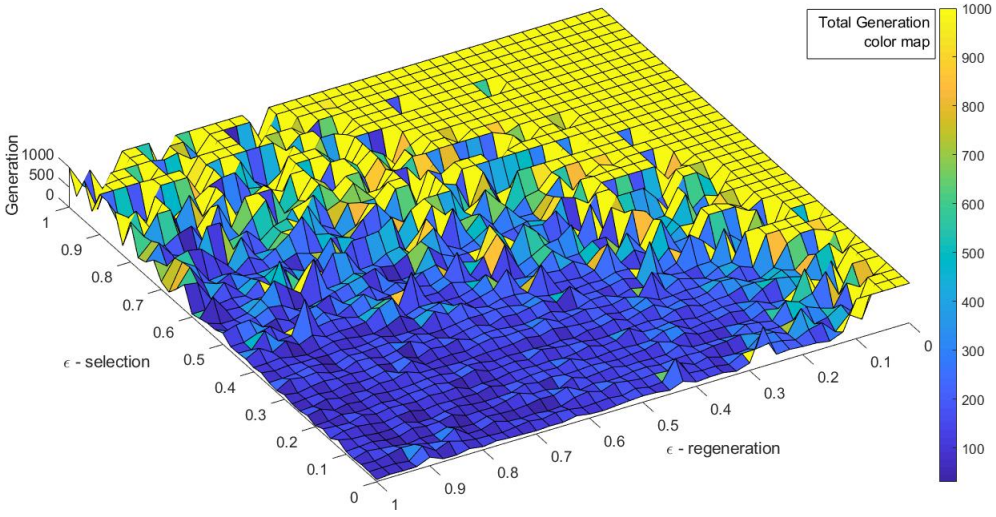


Figure 3.20: The result of the Levy test function using hyperbolic tangent rewarding function

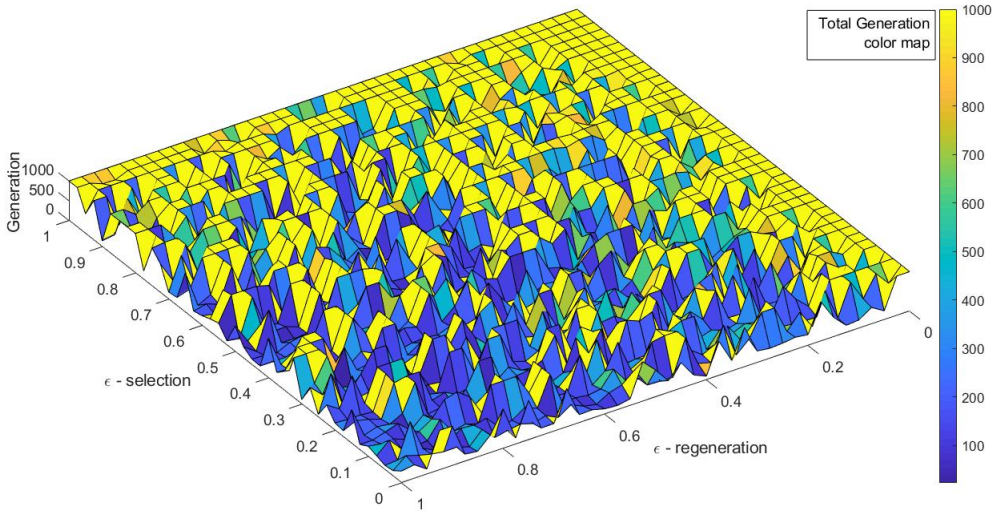


Figure 3.21: The result of the Levy test function using sigmoid rewarding function

3.6.3 Ackley Function

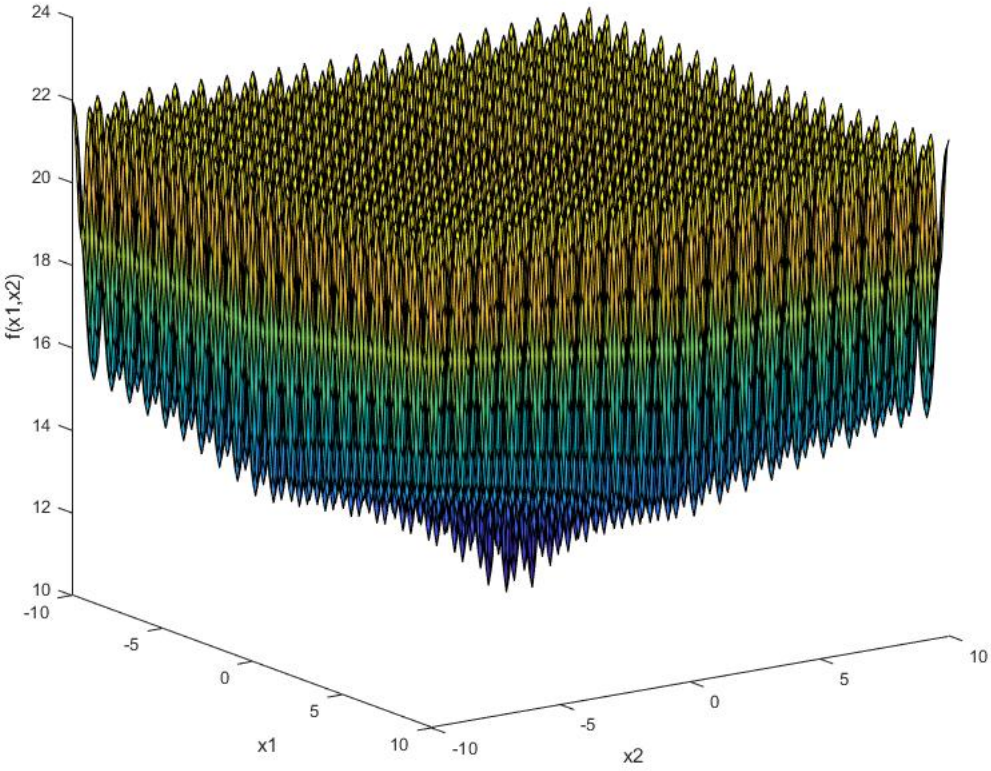


Figure 3.22: Ackley function with $d = 2$

The Ackley function is a standard function with a global minimum and many local minima, see Figure 3.22. The function has a decision variables x_i for all $i = 1, 2, 3, \ldots, d$. The function is defined as

$$f(x) = -a \exp\left(b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos\left(cx_i\right)\right) + a + \exp(1) \tag{3.34}$$

The equation (3.34) is evaluated on the hypercude $x_i \in [-10, 10]$ with $a = 20, b = 0.2, c = 2\pi$ and $d = 2$. The result with a hyperbolic tangent rewarding function is shown in Figure 3.23 and Figure 3.24 for sigmoid rewarding function.

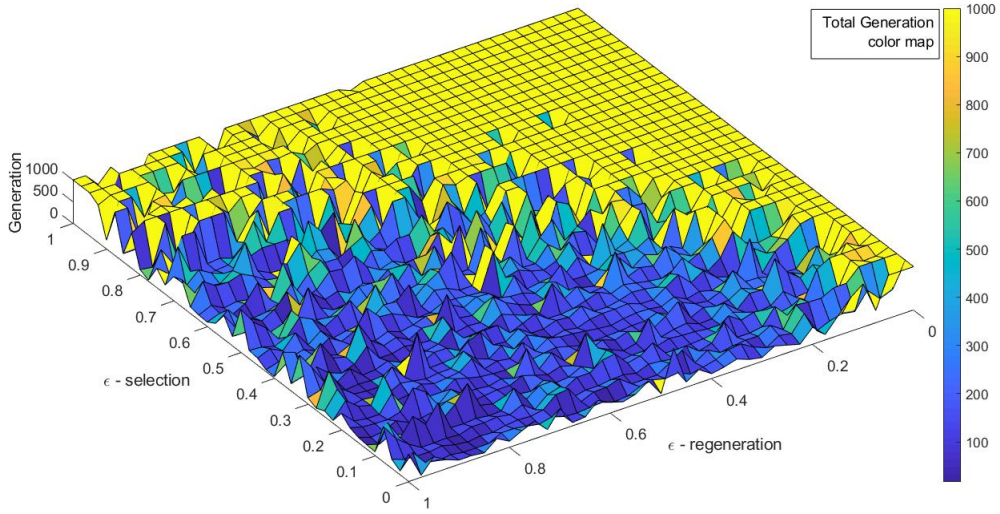


Figure 3.23: The result of the Ackley test function using hyperbolic tangent rewarding function

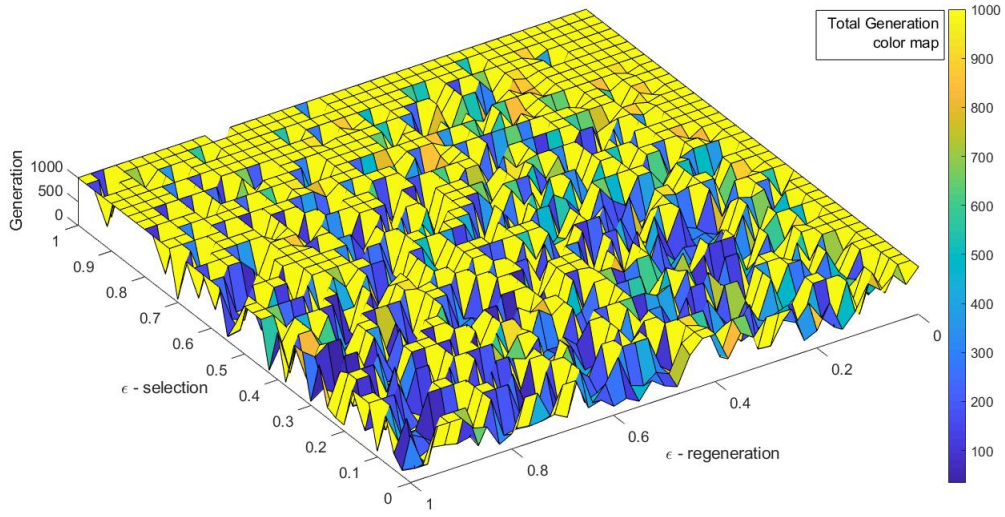


Figure 3.24: The result of the Ackley test function using sigmoid rewarding function

3.7 Results and Discussion

The results show consistent performance over the testing functions. The utilisation of reward-based feedback is used for the proposed algorithm to infer the test function's solution space. Then the reward is processed in the evolutionary process modifying histone value in the epigenetic layer. The performance metric is obtained by applying a range of selection ε-greedy rate ($\varepsilon_{chromosome}$) and regeneration ε-greedy rate ($\varepsilon_{regeneration}$) with range value $[0, 1]$.

The algorithm is tested using three functions. The results are presented in the form of a two-dimensional parameter metric, selection rate and regeneration rate. The first is Sphere function shown in Figure 3.18 and 3.17. The result using sigmoid reward function shows that the total generation for each parameter pairs is not consistent across the metric, while the hyperbolic tangent manages to show a convergence area. In the tangent one, the area that shows a low regeneration number is in a range of selection rate at [0,0.6] pairs with the regeneration at [0.4,1]. The result explains that in the selection process, the exploratory behaviour improves the computation's efficiency because each agent is exploring a new solution once in a while. For the regeneration process, the agent performs better if the strategy exchange is introduced in the process. In some area with the same regeneration range, even though the selection rate is exploiting at range [0.6,1], the agent is able to find an optimum solution rely only on regeneration operator. For the Levy function at Figure 3.21 and Figure 3.20, the performance metric is similar with the sphere function. The difference is in the convergence area, where the results show that the selection range is less than the sphere's results. This is due to the presence of local minima in the function, so the selection rate has to provide more exploratory behaviour in the system. As the problem getting more complex as in Ackley function, the parametric shows that the algorithm can manage to avoid local minima and ultimately ends up in the optimum point, see Figure 3.24 and Figure 3.23.

A rewarding function should be formulated accordingly to the problem. The value distribution of the rewards gives the computational process a current optimal direction approaching a better solution. Overall, the hyperbolic tangent rewarding function performs better than the sigmoid one. This is because the tangent function offers a negative reward (punishment) and suppress a non-optimal decision, while the sigmoid function outputs a decaying process which takes times to suppress the non-optimal solution. The suppression process is natural in methylation stage, as found in biology and known as demethylation. The demethylation and methylation rate controls the histone values within a range of punishment and reward value. Furthermore, the way rewarding process output reinforces values to the learning agent has to be tailored and has to provide the stepping-stone, ultimately leading to the goal.

3.8 Summary

A reward-based epigenetic algorithm for a decentralised swarm is able to solve optimisation problems in a collective way. The rewarding function and epigenetic rates are the operators to be tuned for the algorithm. The results show that as the problem becomes more complex, the algorithm with appropriate rates and reward function performs to obtain better behaviour or solution. The epigenetic layer demonstrates the benefit of the methylation process, regulatory function, and inheritance process. Thus, the available strategies in the swarm can be improve collectively and the genetic behaviour is improved based on external stimulus following epigenetic concept.

3.9 Conclusion

In this chapter, a reward-based epigenetic algorithm is proposed. The method is inspired by epigenetic concept such as histone modification, gene's expression regulator and epigenetic inheritance. As conclusion, the utilisation of environmental knowledge is the key to overcome a dynamic problem and an exploratory behaviour in finding solution offers a better efficiency in the computating efforts. The proposed method contributes and opens a possibility in achieving adaptation for evolutionary swarm robotics research area. Finally, the future reserch implementing a reward-based epigenetic algorithm is expected to advance swarm robotics design improving the swarm's adaptability, co-learning and co-evolve behaviour.

Chapter 4

Epigenetic Learning Framework

The evolutionary algorithm utilising epigenetic inheritance is able to solve a nonlinear problem as discussed in the previous chapter. The regulatory function of the epigenetic layer is obtained by collecting rewards from the problem and turning them into histone values which are utilised as an expression and alteration factors. Based on this, the epigenetic inheritance gives an opportunity to backup reward from a dynamic environment to the genetic level shaping the individual behaviour, which is the main topic of this chapter. Partial of this chapter has been published in (Mukhlish, Page, and Bain 2020b).

In a robotics problem, a task is considered as an episodic problem because a starting and a terminal point does exist. Developing a better behaviour for an episodic problem through an epigenetic algorithm can be done by combining the concept of episodic reinforcement learning with the methylation process on histone values. Since the episodic task consists of numbers of states and actions, a genetic value can be altered or expressed based on its histone value for a particular state-action pair. This approach expands the functionality of histone as a counterpart of an agent's experience or knowledge of the environment (Mukhlish, Page, and Bain 2020a). Then, based on the experience backup, a decision-making process is made. In this chapter, a novel decentralised multi-agent reward-based epigenetic learning (DMARBEL) is proposed to open the aforementioned possibility and apply swarm's decision-making adaptability in a dynamic environment.

This chapter is organised as follow. Firstly, a concept of agent-environment interaction is discussed in Section 4.1. Then, the addition of behaviour into an action is discussed in Section 4.2. The formulation of reward and return is presented in Section 4.3. Fourthly, the value function and policies of state, action and behaviour are investigated in Section 4.4. The following Section 4.5 and 4.6 discuss on the temporal learning and the experience backup of the epigenetic layer. The regulatory function is discussed Section 4.7 and the epigenetic operators are discussed in Section 4.8. Finally, simulation, results, conclusion, and summary are then presented in the following Section 4.9, Section 4.11 and Section 4.10.

4.1 Agent-Environment Interface

Robotics problems are always associated with interaction with an external environment. The surrounding environment is playing a pivotal role in how the robot decides to overcome any constraints. Any entity that can take action or decision is called an agent. Then, any constraints external to an agent is called the environment. For example, if we consider a robot includes all the physical features, then everything other than the robot is the environment. Similarly, if we consider an algorithm as the agent, then the physical features of the robot and everything other than the algorithm is the environment. In a swarm system, an agent is defined as a single member of the group, so other members and everything other than the member is an external environment. Selecting which one is an agent and which is an environment is a key to formulating the learning problem as an action is always taking into account of how the environment works.

A dynamic environment can be considered as a stochastic process from the agent's perspective. The mathematical framework that is often used to model such a problem is Finite Markov Decision Processes (MDPs), which is used as a basis to learn the environment throughout this research. MDPs consists of agent's decisions (actions) and environmental feedback (states and rewards) in the form of Markov Chains. The agent chooses an action a at each time step t and translate itself from current environmental state s to the next state s'. The agent's transition process from state s to the next state s' after taking an action a is denoted as P_a, the formulation is

$$P_a(s, s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a). \tag{4.1}$$

,

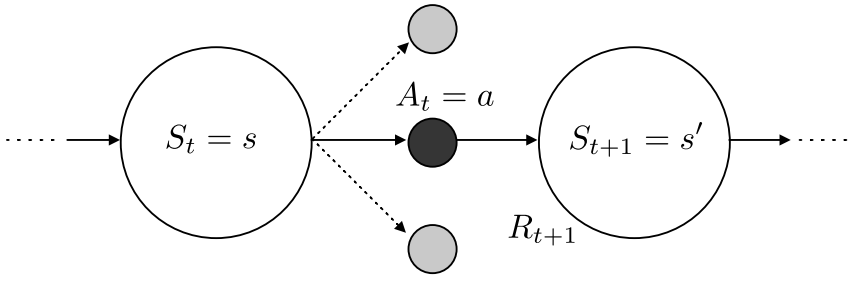


Figure 4.1: A Markov decision process

Where S is a random variable for a state and A is a random variable for an action. The equation (4.1) yields the probability of $S_{t+1} = s'$ given the condition of $S_t = s$ and $A_t = a$. Then, the agent receives an immediate reward from the environment denoted by R_{t+1} right after its arrival to the state s'. The transition is a Markovian process as illustrated in Figure 4.1. The environment will receive agent's action A_t and respond back with a changed state S_{t+1}. Then, in the next time step, the next state becomes current state S_t

for the agent. The interaction between agent and environment forms a cycle as illustrated in Figure 4.2.

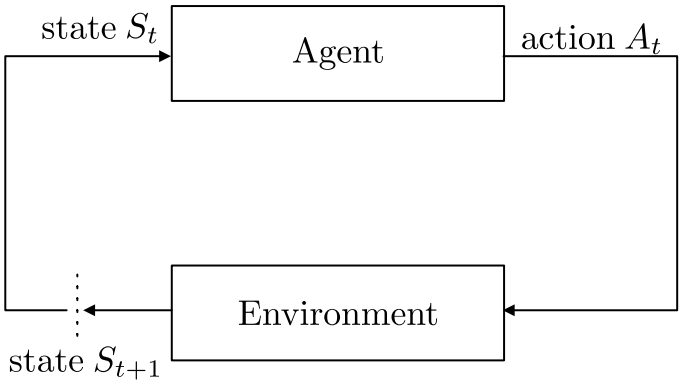


Figure 4.2: A cycle of agent-environment interaction in a MDP

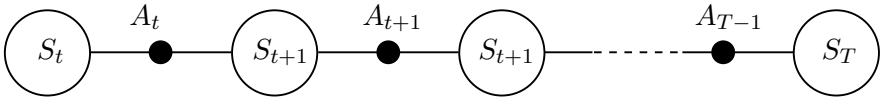


Figure 4.3: States and actions sequence in one episode.

The transition process stops when the agent's action does not change the environmental state, known as a terminal state, denoted by S_T. For example, the terminal state might be a condition that restricts the agent, such as out of energy or crashed. Thus, with the presence of the terminal condition, the process is considered as an episodic task. Moreover, the number of states is finite, which makes the transition process is considered as a Finite Markov Decision Process (FMDP). The episodic tasks experienced by an agent is illustrated in Figure 4.3. Thus, the set of states is finite, resulting in a set of finite actions and rewards, denoted as $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ respectively.

The episodic task is successfully done if the final goal, one of the terminal states, is reached. The strategy solving the episodic task is considered to be optimal if the accumulated of the obtained reward is maximised. The path an agent experienced is represented as a sequence of $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T$. Each state S_t and reward R_t in the sequence is a random variable with discrete probability distributions dependant only on the previous environmental state S_{t-1} and a chosen action A_{t-1}, recalling the MDPs definition (4.1). Hence, there is exist a probability of the next state and reward $(S_t = s', R_t = r)$ occuring at a time step t under the condition of the preceeding state and action pair $(S_{t-1} = s, A_{t-1} = a)$. The transition dynamics of the MDPs with a presence of immediate reward $R_t = r$ is defined as:

$$p(s', r \mid s, a) \doteq \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \tag{4.2}$$

For all states $(s', s) \in \mathcal{S}$, all rewards $r \in \mathcal{R}$ and all available actions in each state $a \in \mathcal{A}(s)$. The distribution of p for each choice of an action a in a state s is

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{for all } s \in \mathcal{S}, a \in \mathcal{A} \tag{4.3}$$

The dynamics of MDPs (4.2) gives a restriction not on the overall process but on the current event (S_t, R_t) and previous event (S_{t-1}, A_{t-1}), known as bootstrapping. Hence, for each state, the solution for (4.2) must include information of all previous experiences. If the final solution exists, then state has a Markov property or can be called a Markovian state. Prior assumption of this property gives benefits to solve each step rather than the overall process, which is a general way to solve episodic tasks.

4.2 Action and Behaviour

The term "action" is often interchangeably with "behaviour". However, a distinction between the two is made in the study "action, behavioural science, and the social science": An action is a meaningful activity with purpose and intent; A behaviour is an automatic and reflexive activity (Bates, Loyall, and Reilly 1994). In biology, a broader definition of behaviour is analogous to phenotypic[1] plasticity. For example, an action may takes characteristics, e.g., a walking action can take different pace such as slow, moderate or fast. If behaviours of an action $A_t = a$ at a state $S_t = s$ is stated as B_t, with the particular behaviour denoted as b, slight modification to equation (4.2) can be made as

$$p(s', r \mid s, a, b) \doteq \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a, B_{t-1} = b) \tag{4.4}$$

$$\text{and} \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a, b) = 1 \tag{4.5}$$

Where $s \in \mathcal{S}$ is all available states, $a \in \mathcal{A}(s)$ is all available actions at state s, and $b \in \mathcal{B}(s, a)$, all available behaviours in state-action pair. The updated Markov Decision Process consists of behaviour is illustrated by Figure 4.4 below.

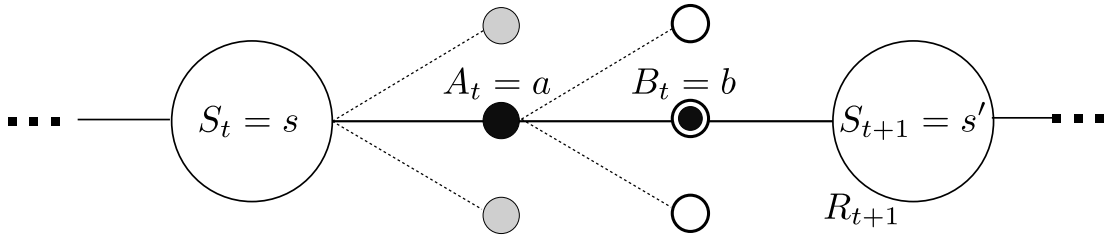


Figure 4.4: A Markov decision process with behaviour

1. Phenotype is an expression of genetic information. Phenotypic behaviour depends on how a gene behave in response of external stimulus.

4.3 Reward and Return

A reward R is a special signal passing from the environment represents the purpose of a goal. From the problem formulation above, reward R_t explains how good an agent's decision is in the environment at each time t, see Figure 4.5. The total rewards an agent obtained in one episode is called a return, denoted as G, from the reward sequence $R_0, R_1, R_2, \ldots, R_T$. Informally, the agent's objective is to select an optimal action at every state, maximising the total amount of reward (return) it obtains. The utilisation of reward to represent the idea of a goal to learn optimal action is the most distinctive features of reinforcement learning.

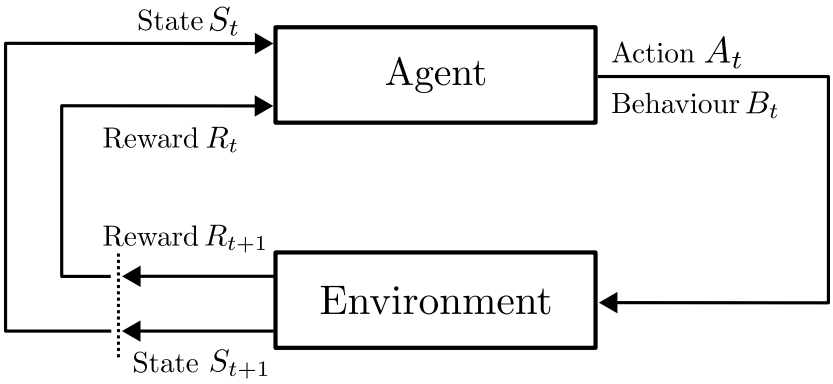


Figure 4.5: A cycle of agent-environment interaction with reward and behaviour

Formally, maximising the return G in one episode is the objective of developing the agent's decision making. The total rewards from a time step t to a terminal time step T, a time step at a terminal state in FMDP, is denoted as G_t and defined as

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T. \tag{4.6}$$

The total steps in (4.6) vary between episodes depending on the experience sequence until a terminal state is reached. To avoid an infinite return value, a discounted approach can be used reducing the effect of future reward with a discount rate γ, such as

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} R_{t+k+1} \tag{4.7}$$

A discount rate γ is in range of $0 \leq \gamma \leq 1$. To follow the bootstrapping process in the MDPs, G_t can be defined as a function of the next reward R_{t+1} and return G_{t+1}, following

(4.7). The relationship is derived by

$$\begin{aligned} G_t &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{4.8}$$

4.4 Value Functions and Policies

Reinforcement learning algorithms formulises functions to quantify the performance of agent's strategy. Two common functions that have been known in RL algorithms are state-value function $v(s)$ and action-value function $q(s, a)$. A state-value function explains how good a state s is in respect to the agent's strategy. The state-value is formulised as an expected value of return results at a particular state. For particular state $S_t = s$, the state-value function is derived as

$$v_s(s) \doteq \mathbb{E}[G_t \mid S_t = s] \tag{4.9}$$

$$= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \qquad \text{by (4.8)} \tag{4.10}$$

$$= \sum_{s'} \sum_{r} p(s', r \mid s, a, b) \left[r + \gamma \mathbb{E}\left[G_{t+1} \mid S_{t+1} = s' \right] \right] \tag{4.11}$$

$$= \sum_{s'} \sum_{r} p(s', r \mid s, a, b) \left[r + \gamma v_s(s') \right] \tag{4.12}$$

Equation (4.11) is known as the Bellman equation. This equation expresses the relationship between the current state and all successor states. It averages all possible successor states and rewards by weighting each sum of reward and discounted state-value function with its occurrence probability. For example, an initial state's value $v(S_0)$ must be equivalent to the discounted expected next state values $\mathbb{E}[v(S_1)]$ plus the expected rewards $\mathbb{E}[R_1]$.

To take action and behaviour, a set of rules is defined as policy. A rule to take action as a response to state is defined as an action policy π_a. Formally, all available action for a state s is defined as $a \in \mathcal{A}(s)$, and the policy is defined as $\pi_a(a \mid s)$. The policy for choosing behaviour for action is defined as a behaviour policy π_b. Similarly, all available action for an action a is defined as $b \in \mathcal{B}(a)$, and the behaviour policy is defined as $\pi_a(b \mid s, a)$. The total policy of an agent in taking a pair of behaviour-action as a response to a state s can be defined as $\pi(b, a \mid s)$. The relationship between the three policies is derived by formulating the probability chain as

$$\pi(b, a \mid s) = \pi(a \mid s).\pi(b \mid a, s) \tag{4.13}$$

Then, the state-value function in respect of policy π (4.13) is derived as:

$$v_\pi(s) \doteq \mathbb{E}\left[G_t \mid S_t = s\right] \tag{4.14}$$

$$= \sum_{a,b} \pi(b, a \mid s) \sum_{s'} \sum_{r} p(s', r \mid s, a, b) \left[r + \gamma v_\pi(s')\right] \tag{4.15}$$

$$= \sum_{a} \pi_a(a \mid s) \sum_{b} \pi_b(b \mid a, s) \sum_{s'} \sum_{r} p(s', r \mid s, a, b) \left[r + \gamma v_\pi(s')\right] \tag{4.16}$$

The formulation in (4.16) is a looking ahead process from state s to all possible successor s' by following policy π_a and π_b, as illustrated in Figure 4.6.

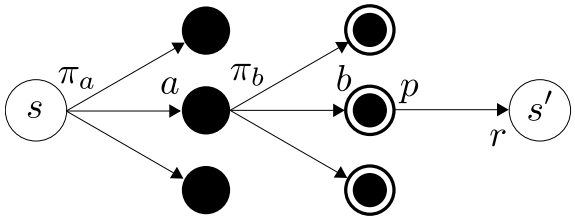


Figure 4.6: A backup diagram of $v_\pi(s)$

The state-value function v_π is useful to evaluate or predict the solution given by π and to help find a better policy π'. Particularly, to find a better policy for selecting a behaviour, a change in policy π_b needs to be evaluated. Hence, a value function for behaviour in a selected action at a particular state is needed, let us call it a behaviour-value function denoted as $h(s, a, b)$. The calculation begins from the behaviour node onward, see Figure 4.7.

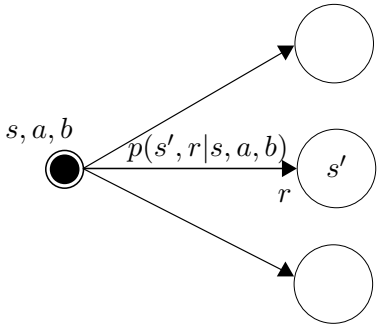


Figure 4.7: A backup diagram oh $h_\pi(s, a, b)$

Instead of using next $h(s', a', b')$ as successor, $v(s')$ is used for simplicity. By following π_b, the function is derived as:

$$h_{\pi_b}(s, a, b) \doteq \mathbb{E}_{\pi_b}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \tag{4.17}$$

$$= \sum_{s'} \sum_{r} p(s', r \mid s, a, b)[r + \gamma v_\pi(s')] \tag{4.18}$$

Similarly, to determine a better policy π_a, an action-value function is required, let us denote it as $q(s, a)$. This function defines how good an action a in a particular state s.

The action-value function is the expected value of all possible behaviour-values after taking action a, each behaviour-value is already defined in equation (4.18). The illustration for action-value function can be seen in Figure 4.8.

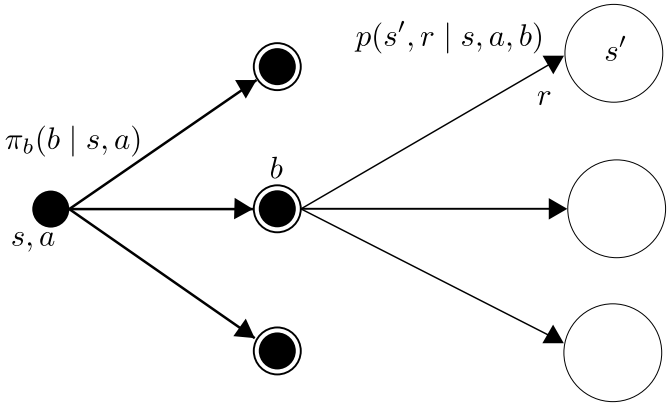


Figure 4.8: A backup diagram of q_π

The derivation of action-value function following policy π_a is:

$$q_{\pi_a}(s,a) \doteq \mathbb{E}_{\pi_a}[h_{\pi_b}(s,a,b) \mid S_t = s, A_t = a] \tag{4.19}$$

$$= \sum_b \pi_b(b \mid a, s) h_{\pi_b}(s,a,b) \tag{4.20}$$

$$= \sum_b \pi_b(b \mid a, s) \mathbb{E}_{\pi_b}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a, B_t = b] \text{ by (4.18)} \tag{4.21}$$

$$= \sum_b \pi_b(b \mid a, s) \sum_{s'} \sum_r p(s', r \mid s, a, b)[r + \gamma v_\pi(s')] \tag{4.22}$$

To maximise the total reward, optimal action and behaviour policy must exist. Let us denote the optimal action policy as π_{a_*}, the behaviour policy as π_{b_*}. Thus, the optimal value functions exist and they are denoted as v_*, q_* and h_*. A policy is said to be optimal if it is better than or equal to all other policies. Formally, it can be defined as:

$$v_{\pi_*}(s) \doteq \max_\pi v_\pi(s) \doteq v_*(s) \tag{4.23}$$

$$q_{\pi_*}(s,a) \doteq \max_\pi q_\pi(s,a) \doteq q_*(s,a) \tag{4.24}$$

$$h_{\pi_*}(s,a,b) \doteq \max_\pi h_\pi(s,a,b) \doteq h_*(s,a,b) \tag{4.25}$$

For all $s \in \mathcal{S}$. Intuitively, the state-value yielding under optimal policy π_* must equal to expected return for the best action-behaviour pair from that state, and the action-value under optimal policy must equal to the expected return for the best behaviour from that

action:

$$q_*(s,a) = \max_{b \in \mathcal{B}(s,a)} h_{\pi_*}(s,a,b) \tag{4.26}$$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) = \max_{\substack{b \in \mathcal{B}(s,a) \\ a \in \mathcal{A}(s)}} h_*(s,a,b) \tag{4.27}$$

To obtain all optimal value functions, let us first derive h_* as follows,

$$\begin{aligned} h_*(s,a,b) &= \max_{\pi} h_\pi(s,a,b) && (4.28)\\ &= \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a, B_t = b\right] && (4.29)\\ &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a, B_t = b\right] && (4.30)\\ &= \sum_{s',r} p(s', r \mid s, a, b)\left[r + \gamma v_*(s')\right] && (4.31) \end{aligned}$$

Then, from (4.26), we can derive optimal value q_* as:

$$\begin{aligned} q_*(s,a) &= \max_{b \in \mathcal{B}(s,a)} h_{\pi_*}(s,a,b) && (4.32)\\ &= \max_{b} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a, B_t = b\right] && (4.33)\\ &= \max_{b} \sum_{s',r} p(s', r \mid s, a, b)\left[r + \gamma v_*(s')\right] && (4.34) \end{aligned}$$

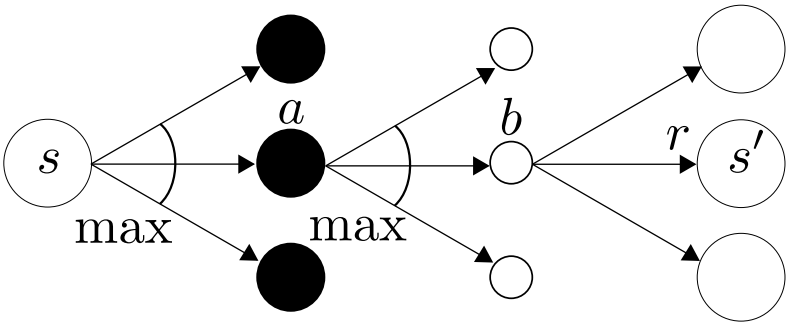
Finally, we can derive v_* as:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) && (4.35)\\ &= \max_{a}\left[\max_{b} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a, B_t = b\right]\right] \quad \text{(by (4.33))} && (4.36)\\ &= \max_{a,b} \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a, B_t = b\right] && (4.37)\\ &= \max_{a,b} \sum_{s',r} p(s', r \mid s, a, b)\left[r + \gamma v_*(s')\right] && (4.38) \end{aligned}$$

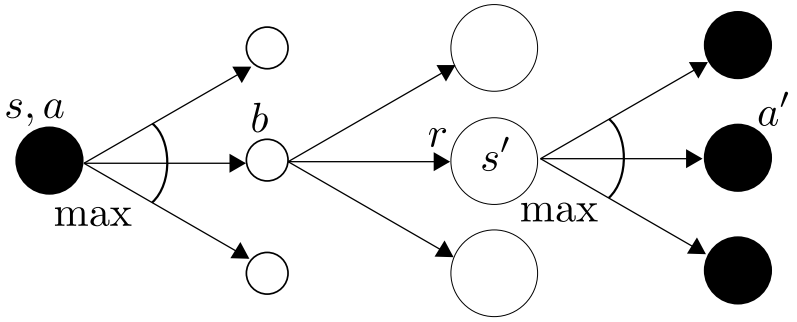
Equation (4.27) shows how optimal action-value and behaviour-value functions can be updated based on each next value as:

$$q_*(s,a) = \max_{b} \sum_{s',r} p(s', r \mid s, a, b)\left[r + \gamma \max_{a'} q_*(s', a')\right] \tag{4.39}$$

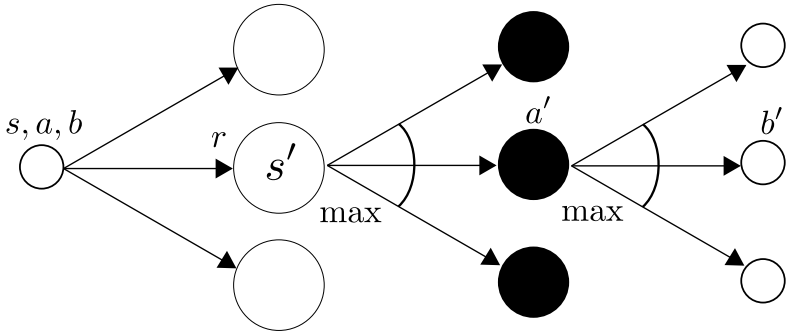
$$h_*(s,a,b) = \sum_{s',r} p(s', r \mid s, a, b)\left[r + \gamma \max_{a',b'} h_*(s', a', b)\right] \tag{4.40}$$



(a) Optimal state-value v_* backup diagram



(b) Optimal action-value q_* backup diagram



(c) Optimal behaviour-value h_* backup diagram

Figure 4.9: Optimal value function backup diagrams

An optimality diagram for each value function can be seen in Figure 4.9. Based on the characteristic of finite MDPs, if the environmental dynamic p are known, each Bellman equation (4.38), (4.39) and (4.40) can be solved yielding a unique solution. Moreover, there are three methods to solve finite Markov decision problems: dynamic programming, Monte Carlo methods and Temporal-difference learning. Each class of methods has its advantages and disadvantages (Sutton and Barto 1998). The first, dynamic programming methods require an accurate environmental model. Monte Carlo methods do not require any environmental models, but not well-suited for step-by-step incremental computation. Finally, Temporal-difference methods do not require an environmental model and are fully incremental. The latter is of interest in this research since the model is often non-trivial and the incremental computation is feasible to be merged with the iterative evolutionary process from (Mukhlish, Page, and Bain 2020b).

4.5 Temporal Difference Learning

Temporal-difference (TD) learning is a method to solve the finite Markov decision process in reinforcement learning problems. The learning process does not require a model of the environment's dynamics. TD-methods' updated estimates are acquired based on other learned estimates, without waiting for an outcome. In this section, the prediction and control problem are discussed in conjunction with of estimating $v_\pi, q_\pi, h_\pi, \pi_a$, and π_b.

4.5.1 Temporal Difference Prediction

In reinforcement learning, the prediction problem focuses on evaluation of a policy for a given problem. Policy evaluation allows predictable transition to proceeding state after taking an action and behaviour based on policy π from the preceding state as formulated in (4.14). The complete evaluation is embedded in value function v_π. TD-methods use experience from the environment to estimate v_π and solve the prediction problem. After following a policy π and get some experience, TD-methods update their estimate of v_π. Let us denote the estimate of v_π for a non-terminal state S_t as $V(S_t)$. For example, one of TD-methods such as a simple every-visit Monte Carlo method uses return G_t as a target for $V_(S_t)$ estimating equation (4.14):

$$V(S_t) \longleftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right]. \tag{4.41}$$

Where α is a constant step-size parameter. However, since return G_t from (4.7) is used, the total return will be available after the terminal state S_T is reached. To bootstrap and get the incremental process on a particular state S_t, a TD-method has to immediately update the estimate at time step $t + 1$. With the definition of discounted return, see equation (4.8), now the immediate update of the estimation becomes

$$V(S_t) \longleftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma G_{t+1} - V(S_t)\right], \tag{4.42}$$

Since the actual target of $V(S_t)$ is the return G_t, then incrementally, equation (4.42) can be modified by bootstrapping actual return G_{t+1} as estimated value $V(S_{t+1})$:

$$V(S_t) \longleftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right], \tag{4.43}$$

Equation (4.43) is called temporal difference, an updating method that uses the difference between time step (temporal). In this case, the TD-method is called one-step TD or TD(0) since the update is only for one time step. The pseudo-code for TD(0) in procedural form is listed in Algorithm 4.1.

Algorithm 4.1 Tubular TD(0) for estimating v_π

Input: the policy π_a and π_b to be evaluated
Parameter: step size $\alpha \in (0, 1]$, discount factor $\gamma \in [0, 1]$

initialise $V(s)$ ▷ For all $s \in \mathcal{S}$ and $V(S_T) = 0$
for all episodes **do**
    $S \leftarrow S_0$ ▷ Initialise start state
    for $t \leftarrow 1, T$ **do**
        $A \leftarrow$ action given by $\pi_a(a|S)$
        $B \leftarrow$ behaviour given by $\pi_b(b|S, A)$
        $S', R \leftarrow$ Agent take action A and behaviour B
        $V(S) \longleftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
        if $S == S_T$ **then**
            $t \leftarrow T$
        else
            $t \leftarrow t + 1$
        end if
    end for
end for

4.5.2 Temporal-Difference Control

In this subsection, the TD(0) method is used to solve control problem by extending the prediction formulation for v_π to estimate action-value q_π and behaviour-value h_π. The estimation of action value Q and behaviour value H allow the policy to select action and behaviour with better outcome. Firstly, let us recall that each episode consists of alternating sequence of state, actions and behaviours, see Figure 4.10.

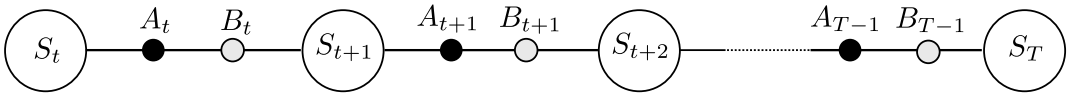


Figure 4.10: Alternating sequence of states, actions and behaviours in one episode

In prediction problem, transitions between state to state are examined to learn the state-value. Thus, the transition between a sequence of state-action-behaviour to the next sequence is examined to learn beahviour-value. Recall that the behaviour-value is an expected return defined as

$$h_{\pi_b}(s, a, b) = \sum_{s'} \sum_{r} p(s', r \mid s, a, b) \left[r + \gamma v_\pi(s')\right],$$

and if h has been learnt accurately, then the state-value can be obtained as:

$$v_*(s) = \max_{\substack{b \in \mathcal{B}(s,a) \\ a \in \mathcal{A}(s)}} h_{\pi_*}(s, a, b)$$

By following the equation (4.43), the update rule for H can be performed using the definition in equation (4.17):

$$H'(S_t, A_t, B_t) \leftarrow H(S_t, A_t, B_t) + \alpha \left[R_{t+1} + \gamma \max_{a,b} H(S_{t+1}, a, b) - H(S_t, A_t, B_t) \right]. \quad (4.44)$$

Since policy π_b is being evaluated in the update rule, the general formulation is defined as

$$H'(S_t, A_t, B_t) \leftarrow H(S_t, A_t, B_t) + \alpha \left[R_{t+1} + \gamma H(S_{t+1}, A_{t+1}, B_{t+1}) - H(S_t, A_t, B_t) \right]. \quad (4.45)$$

Since the update rule for behaviour-value is already obtained, recall the relation between action-value function and behaviour-values from equation (4.20),

$$\begin{aligned} q_{\pi_a}(s, a) &= \sum_b \pi_b(b|s, a) h_{\pi_b}(s, a, b) \\ &= \sum_b \pi_b(b|s, a) \sum_{s'} \sum_r p(s', r \mid s, a, b) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

and if q has also been learnt accurately, then the state-value can be obtained as:

$$\begin{aligned} v_*(s) &= \max_a q_{\pi_*}(s, a) \\ q_*(s, a) &= \max_b h_{\pi_*}(s, a, b) \end{aligned}$$

Then, the update rule for action-value function is

$$Q'(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[\max_b H(S_t, A_t, b) - Q(S_t, A_t) \right] \quad (4.46)$$

Because policy π_a is being evaluated, then the general form is

$$Q'(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[H'(S_t, A_t, B_t) - Q(S_t, A_t) \right] \quad (4.47)$$

or

$$Q'(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]. \quad (4.48)$$

The updated values H' and Q' now can be used to improve policy π_a and π_b. One way to improve each policy is by applying greedy policy where the improved policies give a probability of one for each strategy yielding maximum value of respective value functions. If the greedy policy is applied, the action and behaviour selections are defined as:

$$B_t \longleftarrow \arg\max_b H(S_t, A_t, b), \qquad \pi_b(b|S_t, A_t) \leftarrow \begin{cases} 1, & \text{if } b = B_* \\ 0, & \text{if } b \neq B_*, \end{cases} \quad (4.49)$$

$$A_t \longleftarrow \arg\max_a Q(S_t, a), \qquad \pi_a(a|S_t) \leftarrow \begin{cases} 1, & \text{if } a = A_* \\ 0, & \text{if } a \neq A_*. \end{cases} \quad (4.50)$$

Another policy is an $\varepsilon - greedy$, a policy will choose maximum strategy if the probabilities of choosing, P_b and P_a, are greater than or equal a probability $1 - \varepsilon$ and will choose randomly otherwise. If $\varepsilon - greedy$ is preferred, then the policies are

$$B_t \longleftarrow \begin{cases} \arg\max_b H(St, At, b), & \text{if } P_b \geq (1-\varepsilon) \\ B \in \mathcal{B}(S_t, A_t), & \text{otherwise,} \end{cases} \tag{4.51}$$

$$A_t \longleftarrow \begin{cases} \arg\max_a Q(St, a), & \text{if } P_a \geq (1-\varepsilon) \\ A \in \mathcal{A}(S_t), & \text{otherwise.} \end{cases} \tag{4.52}$$

The pseudo-code of temporal difference control for estimating h_π and q_π is listed in Algorithm 4.2 below.

Algorithm 4.2 TD control for estimating q_π, h_π

Parameter: $\alpha \in (0, 1]$, $\gamma \in [0, 1]$, small $\varepsilon > 0$
initialise $H(s, a, b), Q(s, a)$ arbitrarily ▷ For all $s \in \mathcal{S}, a \in \mathcal{A}(s), b \in \mathcal{B}(s, a)$ and $H(S_T, a, b) = Q(S_T, a) = 0$.
for all episodes **do**
    $S \leftarrow S_0$ ▷ Initialise start state
    $A \leftarrow$ action following π_a from $Q(S, a)$ ▷ (e.g.,$\varepsilon - greedy$)
    $B \leftarrow$ behaviour following π_b from $H(S, A, b)$ ▷ (e.g.,$\varepsilon - greedy$)
    for $t \leftarrow 1, T$ **do**
        $S', R \leftarrow$ Agent take action A and behaviour B
        $A' \leftarrow$ action following π_a from $Q(S', a)$ ▷ (e.g.,$\varepsilon - greedy$)
        $B' \leftarrow$ behaviour following π_b from $H(S', A', b)$ ▷ (e.g.,$\varepsilon - greedy$)
        $H(S, A, B) \longleftarrow H(S, A, B) + \alpha \left[R + \gamma H(S', A', B') - H(S, A, B)\right]$
        $Q(S, A) \longleftarrow Q(S, A) + \alpha \left[H(S, A, B) - Q(S, A)\right]$
        $S \leftarrow S'; A \leftarrow A'; B \leftarrow B'$
        if $S == S_T$ **then**
            $t \leftarrow T$
        else
            $t \leftarrow t + 1$
        end if
    end for
end for

This generates a set of discrete actions that correspond to all states in the MDP. However, a set of behaviour for delivering an action in particular state is not yet represented. Since swarm or multi-agents is the main interest in this research, exchanging learned behaviours between agents is one of the aims of the formulation. One method that allows the swarm to communicate their behaviour or strategy is evolutionary computation. The experience learned from previous formulation can be embedded into genetic information in form of epigenetic layer which is discussed in the next section.

4.6 Experience Backup of Epigenetic Layer

Based on the previous section, because swarm robotics tasks are episodic, epigenetic functions can be obtained by utilising an action and its chromosome, next state and next reward. A sequence of state, reward, action, and chromosome are feedback to the learning layer as reinforcement values. The interaction between epigenetic layer and the environment is illustrated by Figure 4.11. In the formulation, the agent's behaviour B_t is replaced with chromosome C_t.

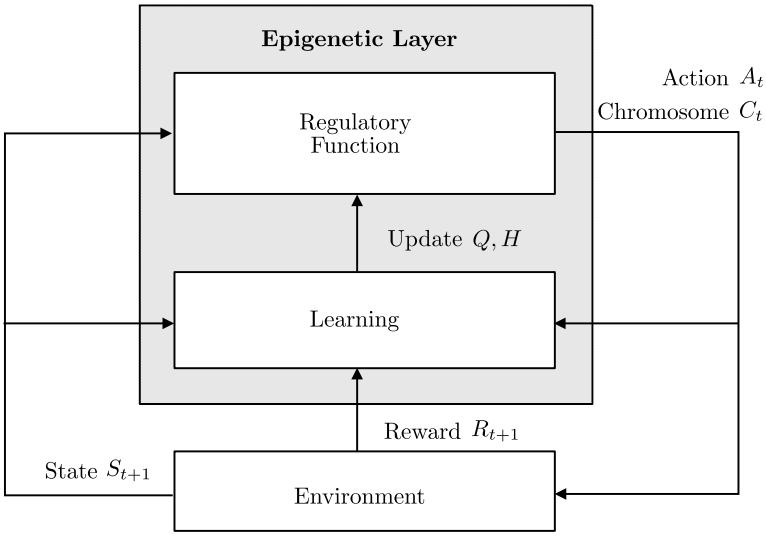


Figure 4.11: Components of Epigenetic layer

4.6.1 Chromosome Structure

From the formulation above, behaviour b which is a varying expression of an action a is analogous to a set of chromosomes C_L in the chromosome-pool $\mathfrak{C}$ which can be selected/-expressed based on their histone values. Thus, a chromosome for a state s and an action a is denoted as $C_l^{s,a}$ and all available chromosome is defined as a chromosome-set $\mathcal{C}$ with size L:

$$\mathcal{C}(s,a) = \left\{C_1^{s,a}, C_2^{s,a}, C_3^{s,a}, \ldots, C_L^{s,a}\right\}. \tag{4.53}$$

In a MDPs, where number of states and actions are finite, if the number of states and actions are donted by I and J respectively, then $\mathcal{S} = \left\{s^i \mid i = \{1, 2, 3, \ldots, I\}\right\}$ and $\mathcal{A} = \left\{a^j \mid j = \{= 1, 2, 3, \ldots, J\}\right\}$. Furthermore, for all combination of states and actions, we have $I \times J$ chromosome-sets and it redefines the chromosome-pool $\mathfrak{C}$ as

$$\mathfrak{C} = \{\mathcal{C}(s, a) \mid s \in \mathcal{S}, a \in \mathcal{A}\}, \tag{4.54}$$

and the diagram illustrating the chromosome-pool can be seen in Figure 4.12.

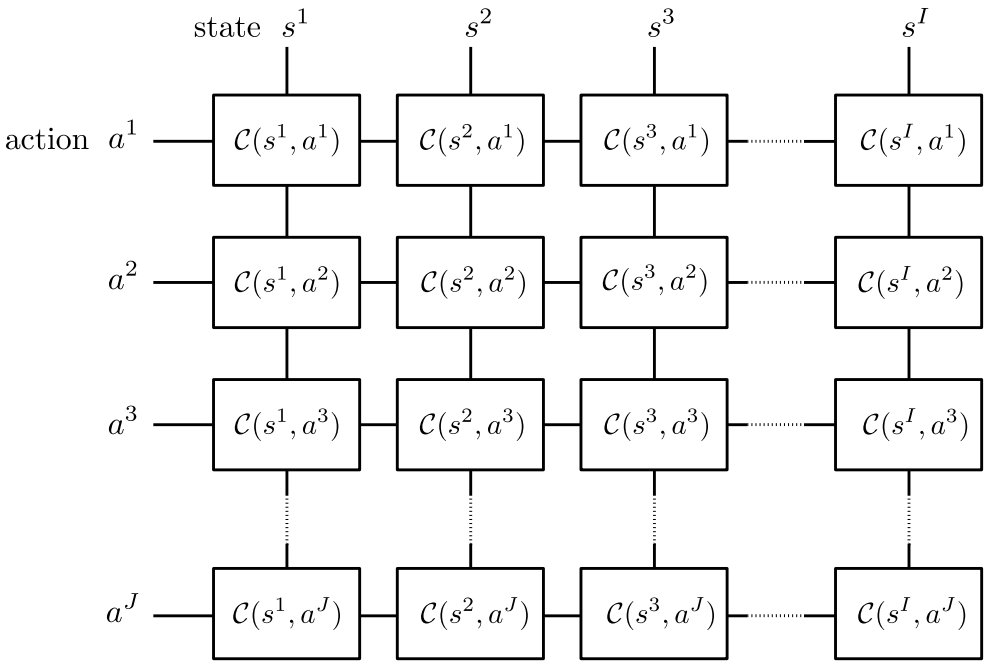


Figure 4.12: Chromosome-pool for all states and actions

4.6.2 Histone Map

From the formulation above, an action a takes form or several expressions of chromosome $C \in \mathcal{C}(s, a)$. The selection of chromosome C is based on histone value H. Recall that all chromosomes in a chromosome-set $\mathcal{C}(s, a)$ have a set of quantitative value (histone) $\left\{H(C_1^{s,a}), H(C_2^{s,a}), H(C_3^{s,a}), \ldots, H(C_L^{s,a})\right\}$, lets denote it as $\mathcal{H}(s, a)$. Then, all histones values in a chromosome-pool $\mathfrak{C}$ are

$$\mathbb{H}(\mathfrak{C}) = \{\mathcal{H}(s, a) \mid s \in \mathcal{S}, a \in \mathcal{A}\} \tag{4.55}$$

with

$$\mathcal{H}(s, a) = \left\{H(C_l^{s,a}) \mid l = \{1, 2, 3, \ldots, L\}\right\}. \tag{4.56}$$

A chromosome C_l has a set of genes $\{{}^l g_m^1, {}^l g_m^2, {}^l g_m^3, \ldots, {}^l g_m^N\}$ and corresponding histone value for each gene is $\{{}^l h_m^1, {}^l h_m^2, {}^l h_m^3, \ldots, {}^l h_m^N\}$. In order for a gene to have a phenotypic plasticity, the histone value has to accommodate various states and actions. Thus, the histone at gene level is represented as a matrix or a histone map $\mathbf{h}_n^m(S, A)$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$ such as

$$\mathbb{H}(g_m^n) = \mathbf{h}_n^m(S, A) = \begin{bmatrix} h_m^n(s^1, a^1) & h_m^n(s^2, a^1) & h_m^n(s^3, a^1) & \ldots & h_m^n(s^I, a^1) \\ h_m^n(s^1, a^2) & h_m^n(s^2, a^2) & h_m^n(s^3, a^2) & \ldots & h_m^n(s^I, a^2) \\ h_m^n(s^1, a^3) & h_m^n(s^2, a^3) & h_m^n(s^3, a^3) & \ldots & h_m^n(s^I, a^3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_m^n(s^1, a^J) & h_m^n(s^2, a^J) & h_m^n(s^3, a^J) & \ldots & h_m^n(s^I, a^J) \end{bmatrix}. \quad (4.57)$$

Hence, the general equation for average histone value is

$$H\left(C_l^{s,a}\right) = H\left(S = s, A = a, C_l\right) = \frac{1}{N} \sum_{n=1}^{N} {}^l h_m^n \left(S = s, A = a\right), \quad (4.58)$$

where m is a position of each gene in corresponding gene-family $\mathcal{G}_n$.

4.6.3 Methylation Process

Because both of methylation process and temporal-difference method utilise feedback from environment, the reward propagation can be formulised to accumulate temporal update and genetic methylation process by stacking them together. Since return G_t is used, then the update rule for equation (3.12) is

$${}^l h_m^n(S, A) \leftarrow {}^l h_m^n(S, A) + \frac{2\eta}{N} \left(G_t - H\left(S, A, C_l\right)\right) \quad (4.59)$$

and since the target of the behaviour value is $R_{t+1} + \gamma H(S', A', C')$, the incremnetal histone update at gene's level can be derived as

$${}^l h_m^n{}'(S, A) \leftarrow {}^l h_m^n(S, A) + \frac{2\eta}{N} \left(R_{t+1} + \gamma H(S', A', C') - H\left(S, A, C_l\right)\right) \quad (4.60)$$

$$\text{with } H'(S, A, C_l) = \frac{1}{N} \sum_{n=1}^{N} {}^l h_m^n{}'(S, A). \quad (4.61)$$

Where η is a methylation rate with $0 < \eta \leq 1$ and γ is a discount rate with $0 < \gamma < 1$. Since $\frac{2\eta}{N}$ is constant and η is small then we can assume that $0 < \frac{2\eta}{N} < 1$. Thus, the update rate $\frac{2\eta}{N}$ can be abbreviated simply as η, with $0 < \eta < 1$, and the update rule for each

histone (4.60) becomes

$$ {}^{l}h_m^n{}'(S,A) \leftarrow {}^{l}h_m^n(S,A) + \eta\left(R_{t+1} + \gamma H(S',A',C') - H\left(S,A,C_l\right)\right) \tag{4.62} $$

.

The methylation process in equation (4.62) is illustrated in Figure 4.13.

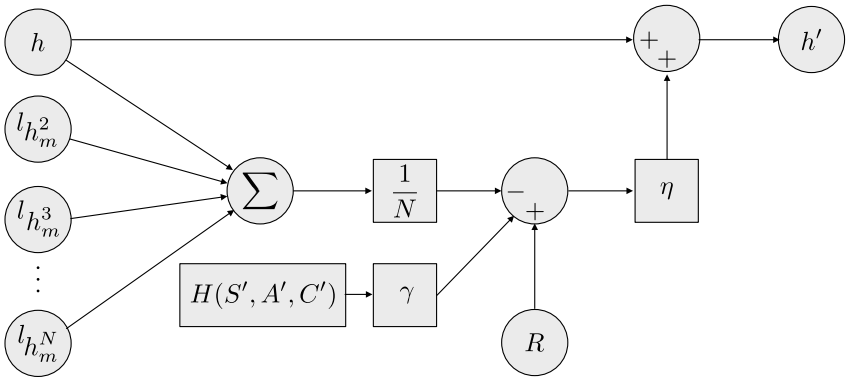


Figure 4.13: Methylation process for episodic task

Thus, the update rule for action value function is

$$ Q'(S,A) \leftarrow Q(S,A) + \alpha\left[H'(S,A,C_l) - Q(S_t,A_t)\right] \tag{4.63} $$

4.7 Epigenetic Regulatory Function

From the previous subsection, the epigenetic value represented by histone value can be derived from interaction with environment. The regulation function of epigenetic layer selects chromosome on a policy π_b, the policy replaced as π_c to avoid confusion. The policy can be a greedy policy where a chromosome with highest $\tilde{H}$ is selected:

$$ C_t = \arg\max_c \tilde{H}(S_t,A_t,C_c), \forall c \in \mathcal{C}(S_t,A_t) \tag{4.64} $$

However, greedy policy is prone to local optima because of lacking in exploration. Exploration allows the chromosome selection to look at other possibilities other than exploiting the same solution every time. Hence, the balancing between exploitation and exploration is needed. Exploration-exploitation policy allows the agent to explore new behaviour/chromosome that may give a better reward rather than exploit current best solution. One policy is of interest in the formulation is $\varepsilon - greedy$ policy which most of the time the agent chooses a chromosome C that has maximal estimated value $\tilde{H}$, but the agent instead selects other available chromosomes at random with probability ε. Suppose $C_*^{S_t,A_t}$ is a chromosome of an action A_t at a particular state S_t that gives maximal reward that follows equation (4.64), then the policy is

$$C_t = \begin{cases} \arg\max_c \tilde{H}(S_t, A_t, c), & \text{if} P_c \geq (1-\varepsilon) \\ C \in \mathcal{C}(S, A), & \text{otherwise} \end{cases} \tag{4.65}$$

$$A_t = \begin{cases} \arg\max_a Q(S_t, a), & \text{if} P_a \geq (1-\varepsilon) \\ a \in \mathcal{A}(S_t), & \text{otherwise} \end{cases} \tag{4.66}$$

Regulatory function is optimal if a value H_* that is equal or better than H exists. It implies that policy is improved as $\pi'_c \leq \pi_c$. The procedure of methylation process and regulatory function can be seen in Algorithm 4.3

Algorithm 4.3 Epigenetic regulator for estimating $\tilde{H}_{\pi_c}$,$\mathbb{H}$ and Q_{π_a}

Parameter: $\alpha \in (0,1]$, $\gamma \in [0,1]$, $\eta \in (0,1)$,small $\varepsilon > 0$
initialise $Q(s,a)$ arbitrarily ▷ For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ and $Q(S_T, a) = 0$.
initialise $\mathfrak{G}, \mathfrak{C}, \mathbb{H}(g), \tilde{H}(s,a,c)$ arbitrarily ▷ For all $g \in \mathfrak{G}, \mathfrak{G} \to \boldsymbol{\mathcal{G}}$, $c \in \mathfrak{C}, \mathfrak{C} \to \mathcal{C}(s,a)$
for all episodes **do**
    $S \leftarrow S_0$ ▷ Initialise start state
    $A \leftarrow$ action following π_a from $Q(S,a)$ ▷ (e.g.,$\varepsilon - greedy$)
    $C \leftarrow$ chromosome following π_c from $\tilde{H}(S,A,c)$ ▷ (e.g.,$\varepsilon - greedy$)
    $\tilde{H}(S,A,C) \leftarrow \frac{1}{N} \sum_{n=1}^{N} {}^{C}h_m^n(S,A)$
    for $t \leftarrow 1, T$ **do**
        $S', R \leftarrow$ Agent takes action A and chromosome C
        $A' \leftarrow$ action following π_a from $Q(S',a)$ ▷ (e.g.,$\varepsilon - greedy$)
        $C' \leftarrow$ chromosome following π_c from $H(S',A',c)$ ▷ (e.g.,$\varepsilon - greedy$)
        $\tilde{H}(S',A',C') \leftarrow \frac{1}{N} \sum_{n=1}^{N} {}^{C'}h_m^n(S',A')$
        for each g in C **do**
            ${}^{C}h_m^n(S,A) \leftarrow {}^{C}h_m^n(S,A) + \eta \left(R + \gamma \tilde{H}(S',A',C') - \tilde{H}(S,A,C) \right)$
        end for
        $\tilde{H}(S,A,C) \leftarrow \frac{1}{N} \sum_{n=1}^{N} {}^{C}h_m^n(S,A)$
        $Q(S,A) \longleftarrow Q(S,A) + \alpha \left[\tilde{H}(S,A,C) - Q(S,A) \right]$
        $S \leftarrow S'; A \leftarrow A'; C \leftarrow C'$
        if $S == S_T$ **then**
            $t \leftarrow T$
        else
            $t \leftarrow t+1$
        end if
    end for
end for

4.8 Evolutionary Learning Mechanism

The evolutionary-learning algorithm consists of experience backup, exchanging strategies and improving policies. Firstly, an agent interacts with the environment and get experience in taking an action for each state with immediate reward given by the environment. The experience is manifested into histone value through the methylation process for each gene. This process is analogous to biological process in obtaining external stimulus and store them into epigenetic layer. Then, genetic information along with their histone values are exchanged at swarm/group level with other members. The exchange process utilises the histone value as a basis. These processes include histone-based selection, histone-mask crossover, genomic imprinting, mutation, gene silencing, and regeneration. Finally, after new sequence of chromosomes are obtained by the agent, improving the policy based on new histone values of the gene-pool and chromosome-pool is carried out. Then, the process goes to the first step, experience backup, to evaluate and get new experience from the environment. The overall process is recurring until the plausible optimal policies are achieved. Figure 4.14 below illustrates the overall process of reinforcement evolutionary-learning using epigenetic inheritance.

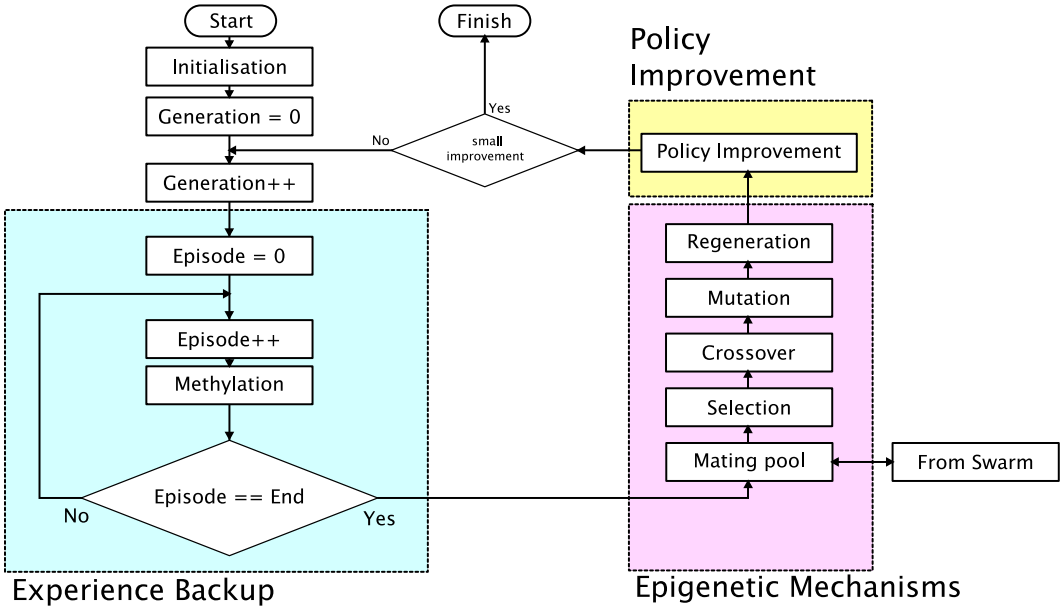


Figure 4.14: Diagram of evolutionary-learning algorithm using epigenetic layer

4.9 Simulation and Result

The case that is used in this chapter is a flocking of UAVs following a path. The dynamic used in the simulation based on the dynamic of small unmanned aerial vehicle (Beard and McLain 2012). Each UAV in the swarm has three behaviours, namely point target attraction, UAVs avoidance and altitude control. Point target attraction is a behaviour to go to a given point in the environment based on distance and relative angle to the point target. UAVs avoidance is a behaviour to avoid UAV in vicinity given the distance and relative angle. Lastly, altitude control is a behaviour to retain height while searching the area.

4.9.1 Flocking

4.9.1.1 Waypoint Tracking

Target points are arranged forming a set of waypoints for UAV. To follow the waypoints, the UAV is moving to approach the course line between two points on the trajectory. The UAV has to move to approach a line between two points by changing its heading and altitude set point as depicted in 4.15.

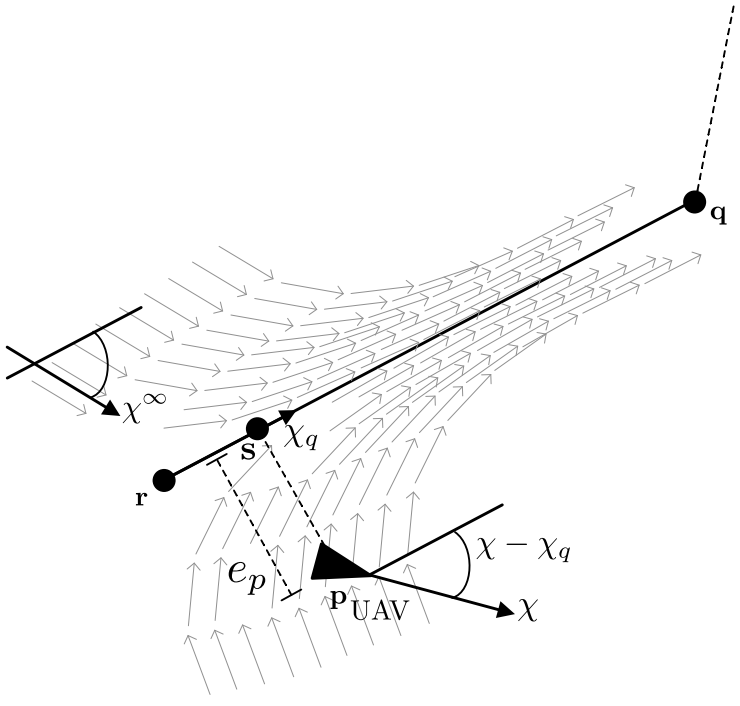


Figure 4.15: Vector field for two waypoints

From the figure above, χ is the UAV's heading and the course line heading is denoted by χ_q. Far away from the waypoint path, the vector field is directed with an angle χ^∞. The course line has path's origin denoted by $\mathbf{r} = \{r_n, r_e, r_d\} \in \mathbb{R}^3$ and a unit vector $\mathbf{q} = \{q_n, q_e, q_d\} \in \mathbb{R}^3$ as a direction indicates the disired of travel. The position of the UAV is $\mathbf{p} = \{p_n, p_e, p_d\} \in \mathbb{R}^3$ and its projection on the course line is $\mathbf{s} = \{s_n, s_e, s_d\} \in \mathbb{R}^3$. The

position error is derived by calculating the vector between $\mathbf{p}$ and $\mathbf{s}$ as $\mathbf{e}_p = \{e_{px}, e_{py}, e_{pz}\}$. The complete formulation to reduce the error position $\mathbf{e}_p$ is by calculating desired altitude h_d as

$$h_d(\mathbf{r}, \mathbf{p}, \mathbf{q}) = -r_d + \sqrt{s_n^2 + s_e^2} \left(\frac{q_d}{\sqrt{(q_n^2 + q_e^2)}} \right) \tag{4.67}$$

Then, the desired heading χ_d is

$$\begin{aligned} \chi_d(t) &= \chi_q - \chi^\infty \frac{2}{\pi} \tan^{-1}\left(k_{path} e_{py}(t)\right) \\ \text{with } \chi_q &= \arctan^2(q_e, q_n) + 2\pi m \\ \text{where } m \in \mathcal{N} &\rightarrow \pi \leq \chi_q - \chi \leq \pi \end{aligned} \tag{4.68}$$

where k_{path} is a positive constant that influences the rate of the transition from χ^∞ to zero.

4.9.1.2 Repulsion Force

To avoid collision between UAVs, an avoidance behaviour is applied. UAVs in vicinity are detected using proximity sensor shown in Figure 4.16. Repulsion force is calculated when neighbouring UAV is inside of outer radius of the proximity sensor. On the other hand, inner radius is used to detect a potential collision.

In order to avoid neighbouring UAV, relative position, as can be seen in Figure 4.17, is used to calculate repulsion heading χ_r and repulsion altitude h_r. First, the repulsion force magnitude between agent i and j, defined as $|\mathbf{F}_{ij}^r|$, is calculated as

$$|\mathbf{F}_{ij}^r| = \begin{cases} 0 & , |D_{ij}| > R_{\text{outer}} \\ K_r \left(\frac{R_{\text{outer}} - R_{\text{inner}}}{|D_{ij}| - R_{\text{inner}}} \right) & , R_{\text{inner}} < |D_{ij}| \leq R_{\text{outer}} \\ \text{collide} & , |D_{ij}| \leq R_{\text{inner}} \end{cases} \tag{4.69}$$

Where K_r is a positive constant of repulsion rate and $|D_{ij}|$ is derived as a relative neighbour position from the UAV. For the range of $R_{\text{inner}} < |D_{ij}| \leq R_{\text{outer}}$, the distance vector between agent i and any nearby UAV denoted by j is defined as

$$D_{ij} = \begin{bmatrix} p_e^j - p_e^i \\ p_n^j - p_n^i \\ p_d^j - p_d^i \end{bmatrix} \tag{4.70}$$

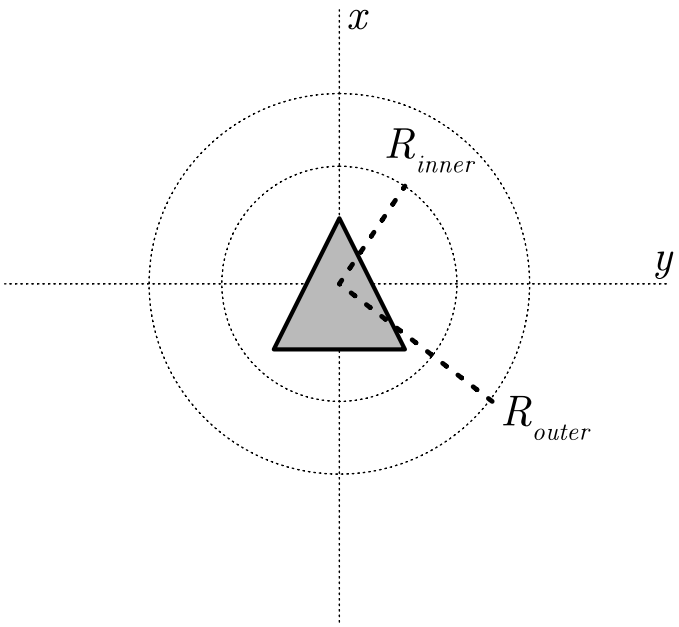


Figure 4.16: Perimeter of proximity sensor

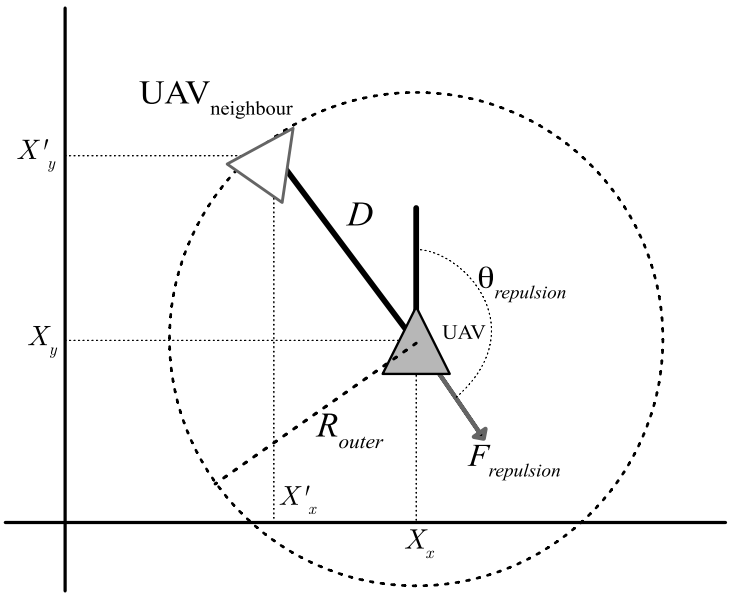


Figure 4.17: Repulsion force from neighbouring UAV

Then, the direction of the propulsion force is derived as a unit vector

$$\hat{\mathbf{f}}_{ij}^{r} = -\frac{D_{ij}}{|D_{ij}|} \tag{4.71}$$

So, the total repulsion force vector on agent i by neighbouring UAVs is obtained as

$$\begin{aligned}\sum \mathbf{F}_i^r &= K_r \begin{bmatrix} F_{r_e} \\ F_{r_n} \\ F_{r_d} \end{bmatrix}_i = K_r \sum_{j \neq i} \begin{bmatrix} F_{r_e} \\ F_{r_n} \\ F_{r_d} \end{bmatrix}_{ij} \\ &= K_r \sum_{j \neq i} |\mathbf{F}_{ij}^r| \cdot \hat{\mathbf{f}}_{ij}^r \\ &= K_r \sum_{j \neq i} \left(\frac{R_{\text{outer}} - R_{\text{inner}}}{|D_{ij}| - R_{\text{inner}}} \right) \cdot \hat{\mathbf{f}}_{ij}^r \end{aligned} \tag{4.72}$$

we can derive the desired heading for avoidance action

$$\chi_r = \pi - \arctan\left(\frac{F_{r_e}}{F_{r_n}} \right) \tag{4.73}$$

Finally, the desired altitude is proportional to $\hat{\mathbf{k}}$-component of the $\mathbf{F}_r$ as

$$h_r = K_r F_{r_d} \tag{4.74}$$

4.9.2 Velocity Alignment of Neighbours

The velocity alignment has been defiend as viscous friction-like interaction (Virágh et al. 2014; Cucker and Smale 2007; Helbing, Farkas, and Vicsek 2000) as

$$\mathbf{F}_{ij}^{v} = K_v \frac{v_j - v_i}{\left(\max\left\{r_{\min}, |D_{ij}|\right\}\right)^2} \tag{4.75}$$

Where K_v is the strength of the alignment and r_{min} defines a threshold to avoid division by close-to-zero distances. Then, the total velocity alignment is

$$\mathbf{F}_v = K_v \begin{pmatrix} F_{v_e} \\ F_{v_n} \\ F_{v_d} \end{pmatrix} = K_v \sum_{j \neq i} \begin{pmatrix} F_{v_e} \\ F_{v_n} \\ F_{v_d} \end{pmatrix}_{ij} = K_v \sum_{j \neq i} \frac{v_j - v_i}{\left(\max\left\{r_{\min}, |D_{ij}|\right\}\right)^2} \tag{4.76}$$

The resultant force is obtained as a sum of repulsion force and velocity alignment as

$$\mathbf{F}_R = \begin{pmatrix} F_{R_e} \\ F_{R_n} \\ F_{R_d} \end{pmatrix} = K_v \sum_{j \neq i} \frac{v_j - v_i}{\left(\max\left\{r_{\min}, |D_{ij}|\right\}\right)^2} - K_r \sum_{j \neq i} \left(\frac{R_{\text{outer}} - R_{\text{inner}}}{|D_{ij}| - R_{\text{inner}}} \right) \cdot \hat{\mathbf{f}}_{ij}^r \tag{4.77}$$

From equation (4.77), we can derive desired heading for each agent as

$$\chi_R = \arctan\left(\frac{F_{R_e}}{F_{R_n}}\right) \tag{4.78}$$

and the desired altitude as

$$h_R = h_d(t) + F_{R_d} \tag{4.79}$$

4.9.3 Result

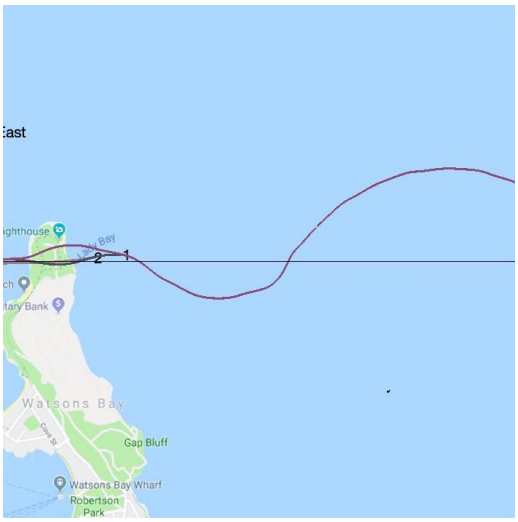
The mission of the swarm is to flock following a given pattern. These are a line pattern with multiple waypoints without impacting with other agents. The states of each UAV are the general information of its position relative to the path and other UAVs. The states of each UAV are "start", "far from path", "near path", "UAVs nearby", and "crash". The position of the UAV, especially distance vector D_{ij} is used to determine the distance between UAVs and the position error $\mathbf{e}_p$ is used to determine how close each UAV is to the path. The thresholds for distance vector are 1 meter for R_{inner} and 200 meters for R_{outer}. The tolerance for state "near path" is $\mathbf{e}_p < 5$ meters and "far from path" otherwise. The actions for the UAV are "go-to path" and "Avoid UAVs". Moreover, the actions or each UAVs are defined as activity in response to change in its state, namely "go-to path" and "avoid UAVs", following flocking scenarios discussed in section 4.9.1.

The simulation consists of experience backup process, epigenetic mechanisms and policy improvement. A number of episodes between processes are defined by evolution interval. In this particular simulation, 10 episodes is selected for the evolution interval to give 10 episodes of experience for the swarm and then evolve afterwards. A complete simulation parameters for the algorithm is listed in table 4.1 below.

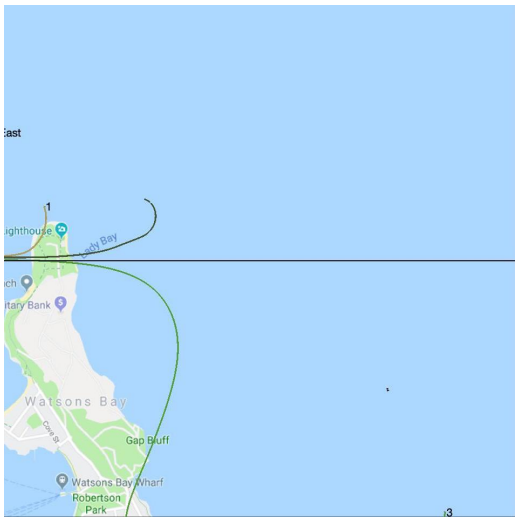
Table 4.1: Simulation parameters

Parameters	Value	Parameters	Value
Number of agents	3	Mutation rate	0.05
ε-greedy	0.75	Imprinting rate	0.01
Learning rate	0.1	Silence rate	0.05
Discount factor	0.1	Number of states	5
Methylation rate	0.1	Number of actions	2
Regeneration rate	0.75	Evolution interval	10 episodes
Histone selection rate	0.75		

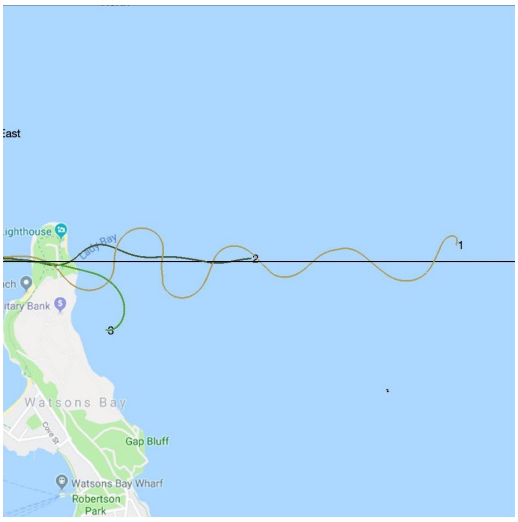
4.9.3.1 Line Pattern



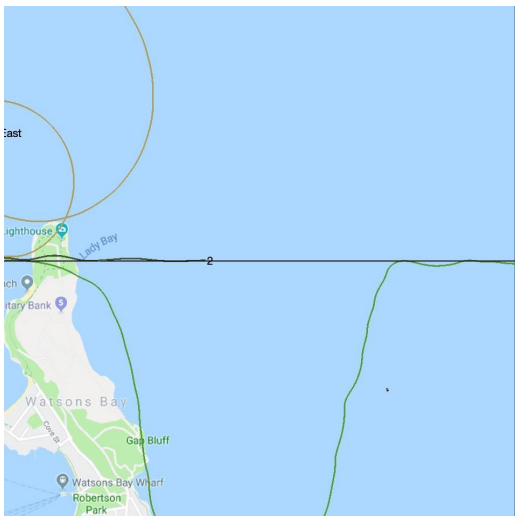
(a) Episode 1



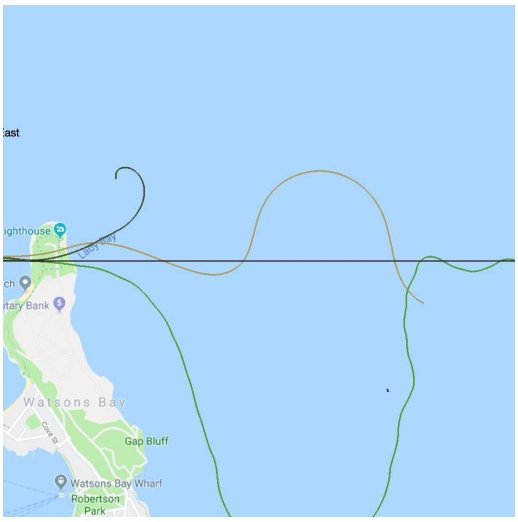
(b) Episode 10



(c) Episode 20

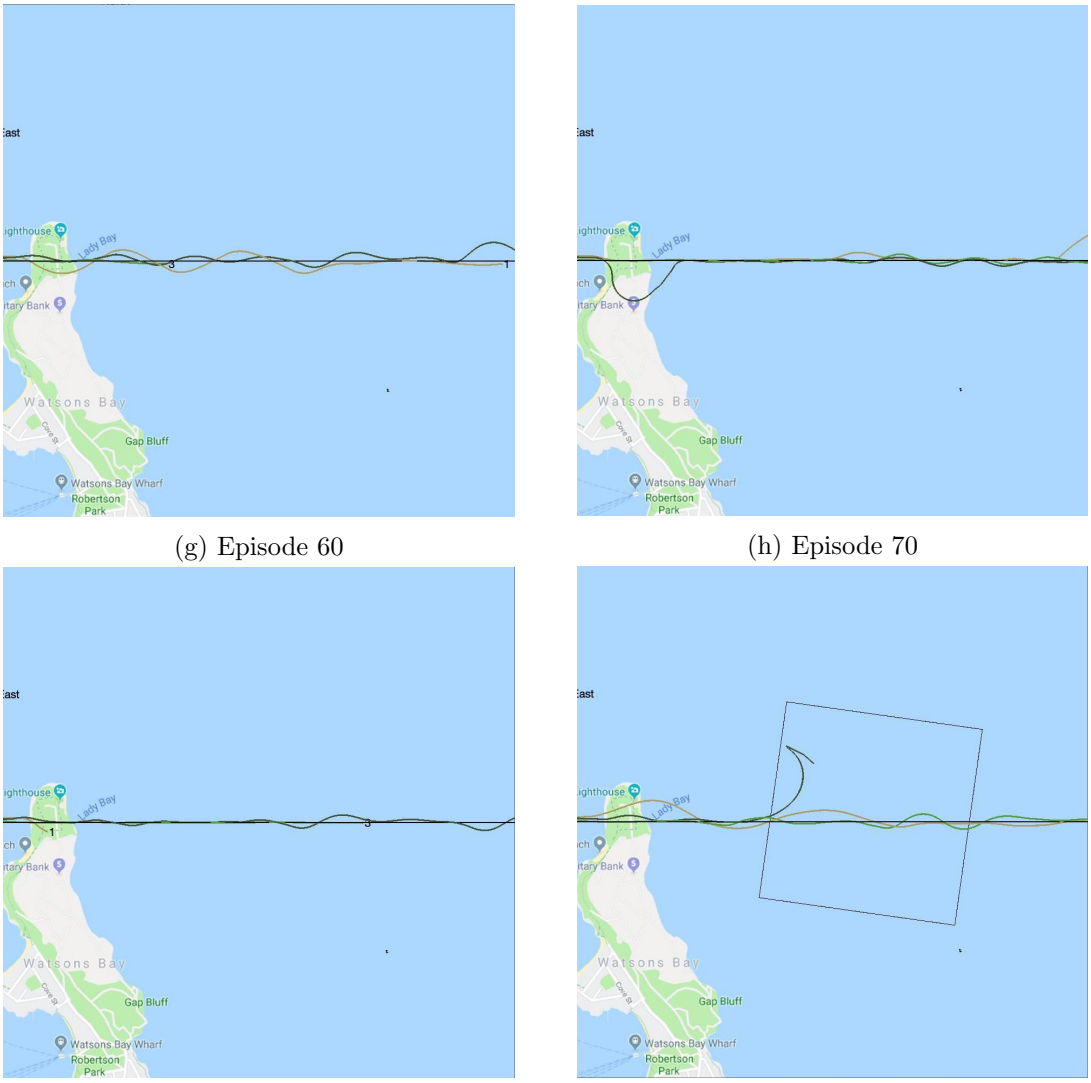


(d) Episode 30



(e) Episode 40

(f) Episode 50

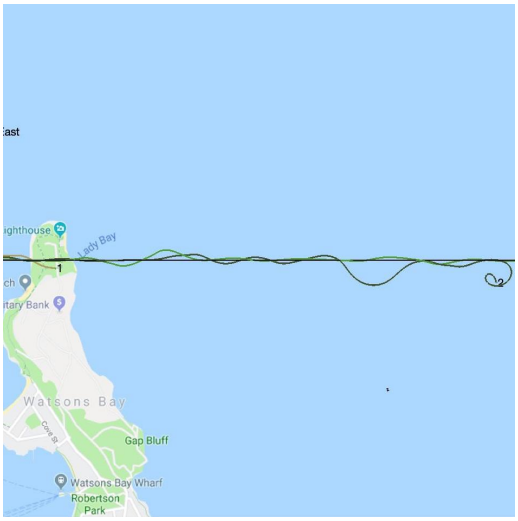


(g) Episode 60 (h) Episode 70

(i) Episode 80 (j) Episode 90

Figure 4.18: Flocking following straight line: from episode 1 - episode 90, with evolution interval 10 episodes.

From Figure 4.18, it can be seen the process in early stage of evolution and experience backup to follow a straight line. The flocking pattern from episode 1 to 50 shows unstable agent deviating from the course line. This is because the control parameters and the regulatory function have not yet figured out a better policy in early stage. Look at the development from episode 60 to 90, the following patterns show that the swarm flock forming a straight pattern following the course line although there is an agent crash in the middle of the way, see episode 60 where agent-3 is arrived in terminal state “crash”.



(a) Episode 100

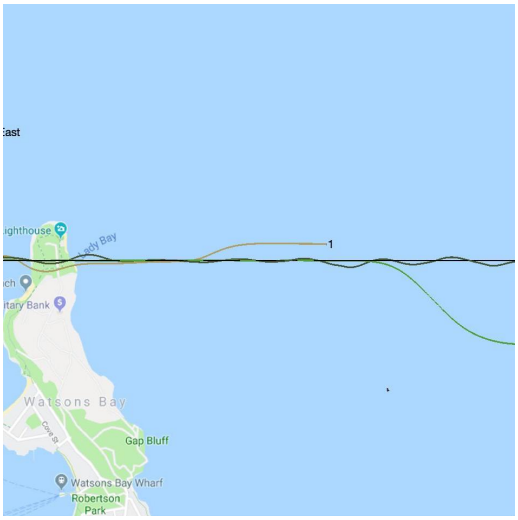
(b) Episode 200

(c) Episode 300

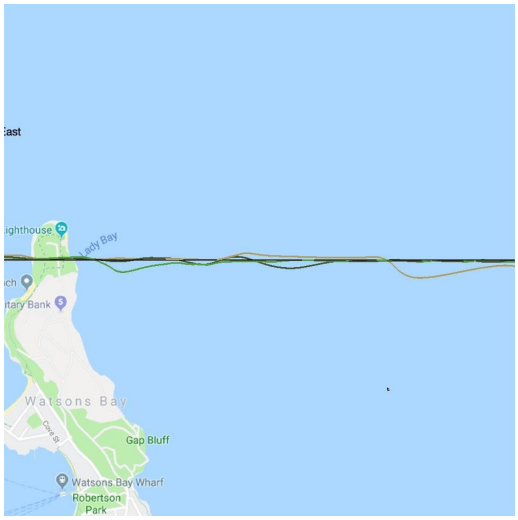
(d) Episode 400

(e) Episode 500

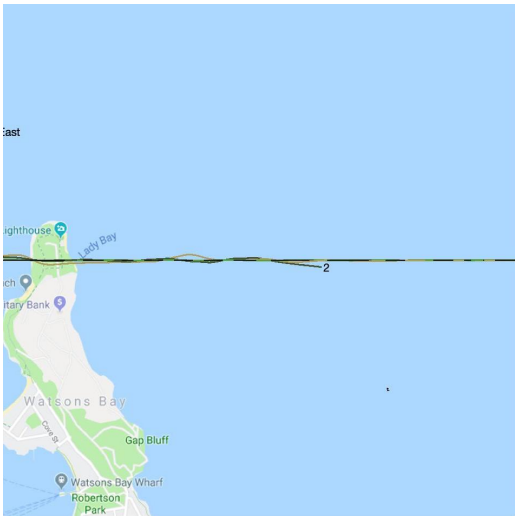
(f) Episode 600



(g) Episode 700



(h) Episode 800



(i) Episode 900

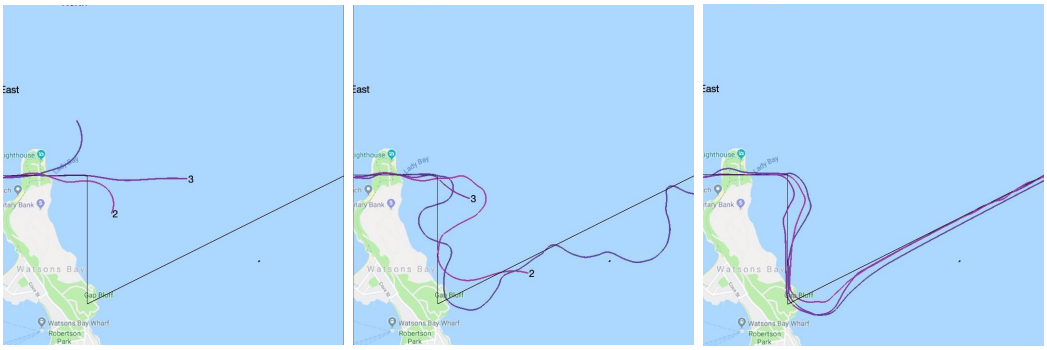
(j) Episode 1000

Figure 4.19: Flocking following straight line: from episode 100 - episode 1000, with evolution interval 10 episodes

Figure 4.19 above, the results are taken with 100 episodes interval to see the process of evolution in a longer period. The evolution progress shows that the swarm follow the straight waypoints, and the deviation showed in episode 500 and 800 explain that there is a UAV that avoids the collision with another UAV. Hence the actual trajectory is deviated and eventually fly back following the course line. The anomaly behaviour in episode 600 demonstrates that exploration behaviour is maintained to open the possibility of finding better behaviour from the existing solutions.

4.9.3.2 Multiple Waypoints

With similar process as the straight-line trajectories, the next simulation includes multiple waypoints resulting waypoints with different course heading. The result of the final evolution can be seen in Figure 4.20.



(a) Early Stage (b) Middle Stage (c) Final Stage

Figure 4.20: Flocking behaviour for multiple waypoints in multiple stages

4.9.3.3 Histone Values

It can be seen from Figure 4.21 and Figure 4.22 that the algorithm are improving the histone values, and converge to a certain value. The epigenetic layer regulates the actions selection based on its action-value functions derived from histone values. Then as results, the agent selects action "go-to path" in a state "far from the path". The agent selects the action "go-to path" over "avoid" based on actions average histone value 1.5 and 0.5, respectively. Thus, when the exploitation occurs, the agent will select "go-to path" at state "far from the path". By examining further, the action selection process is the same for other states. Moreover, since the movement of member of the swarm are dynamic, the histone values also changing overtime adapting to the current flying behaviour of the rest of the swarm.

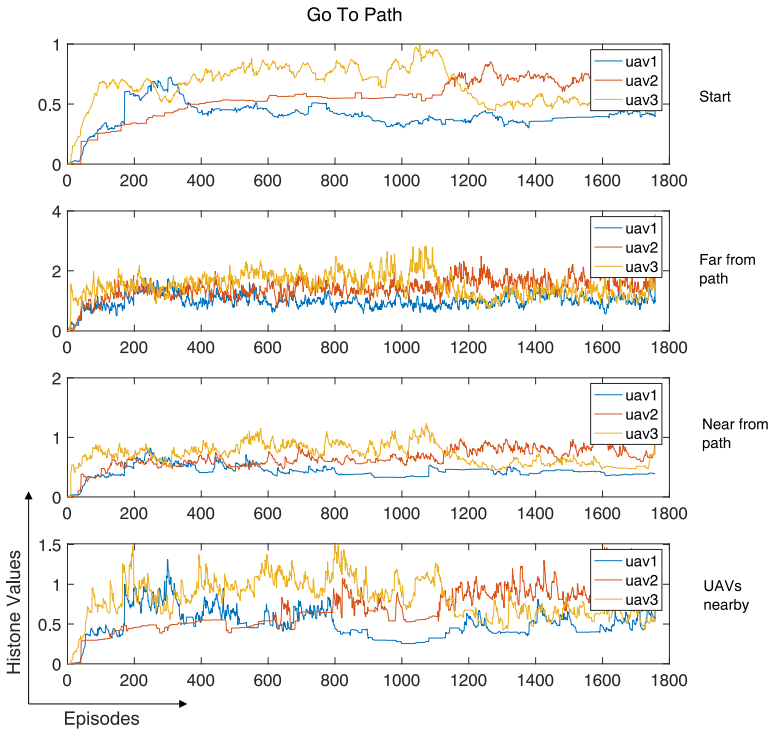


Figure 4.21: Histone values for best chromosome on "Go to path"

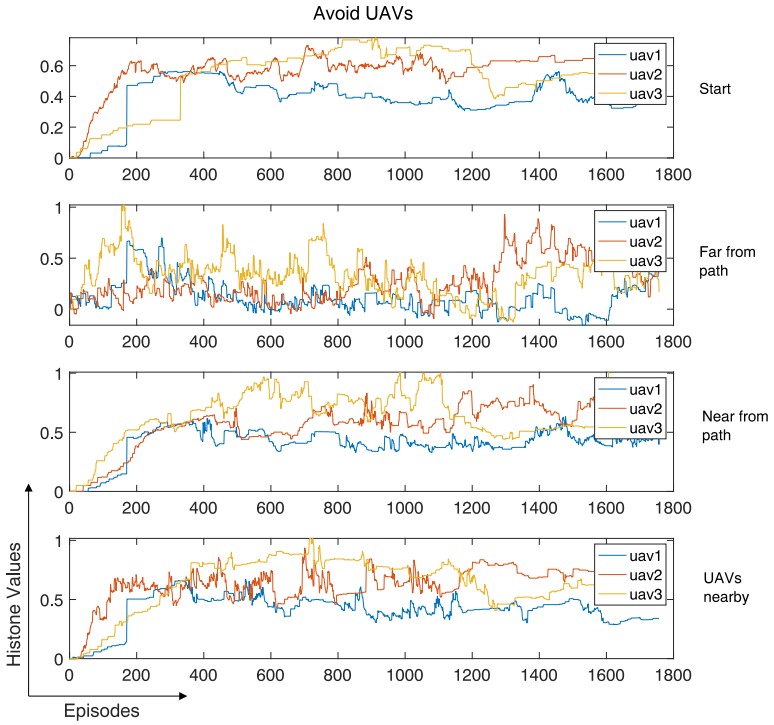


Figure 4.22: Histone values for best chromosome on "Avoid UAVs"

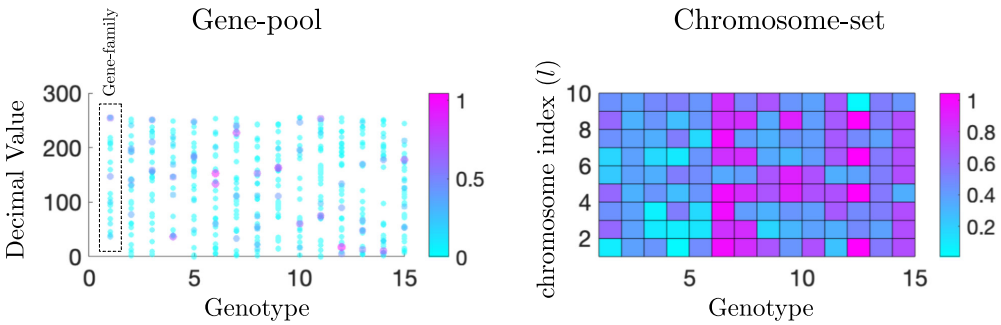


Figure 4.23: Chromosome-set and histone values

At the chromosome level (see Figure 4.23), on the right-hand side, a number of chromosomes for each state-action pair is 10, as defined in the simulation parameter. Each chromosome has a genetic structure that consists of a set of genes with different genotypes. Each gene corresponds to the gene-family in the gene-pool on the left-hand side of Figure 4.23. In the gene-pool metric, all the genes are positioned on the gene-family, x-axis, and their decimal value, derived from gray-code, along the y-axis. A coloured dot represents each gene's histone, and the value is defined by the colour bar. The relationship between chromosome-set and gene-pool is illustrated with the coloured histone value. As results, it can be seen that, through exploration and exploration selection, each chromosome is shaped differently. It shows that epigenetic learning successfully accumulates rewards to histone values and propagates the information to gene-level and chromosome-level.

4.10 Summary

This chapter investigated the possibility of learning the dynamic environment of swarm robotics by extending reward-based epigenetic algorithm with reinforcement approach. Thus, reinforcement evolutionary learning using epigenetic inheritance is proposed. The formulation is done by combining the reward-based learning with a backup experience for an episodic task as a recurring process. The proposed method successfully achieves automatic swarm behaviour in a dynamic environment. This method utilises reward, temporal difference and epigenetic inheritance to approximate optimal action and behaviour policies. Moreover, the methylation process and the temporal update is successful in accumulating the experience into the epigenetic layer, and the epigenetic is also successful in exchanging strategies between agents.

4.11 Conclusion

In this chapter, the results show that the evolutionary and learning process taking a receding approach in developing better behaviour over episodes, the exploration characteristics are maintained to open a possible novel behaviour that improves current optimal behaviour. A more complex task such as multiple waypoints that require the flocks to change heading together while maintaining the group is also demonstrated. The result also showed that the proposed method is able to achieve plausible behaviour. For future works, the improvement of the temporal difference backpropagation by applying TD(λ) to accumulate and propagate the reward back several time-steps back, so the transition between behaviour becomes smoother will be investigated.

Chapter 5

An Epigenetic Based Learning Swarm for Search and Rescue Mission

A Search and rescue (SAR) mission is the search to provide lifesaving support for people in distress, avoiding the danger of loss of life (T. Stone 2018). For the victims in a dynamic and harsh environment such as sea surface is challenging. The success of a SAR operation depends on the pace of deploying the operation. The number of searchers available and the search teams' organisation will be a major factor in determining search area coverage. The time required to search a vast area thoroughly is often critical when a number of search assets are limited. A group of search assets is often required to cover a large search area which is mainly the result of drifting, windage and a lack of certainty on the initial incident location. To overcome this problem, currently, a number of manned search assets are deployed using a predefined search pattern determined before the mission is initiated. However, this deploying solution is not cost-effective, and risks search crews' safety in a harsh environment (Allard and McNeilage 2014). The idea of deploying unmanned aerial vehicles would be preferable since the number of search assets is more flexible and results in less cost and risk to human assets. Extensive reviews for search and rescue missions using swarm robotics can be found in (Couceiro 2016; Tan 2016; Bakhshipour, Jabbari Ghadi, and Namdari 2017).

In this chapter, the simulation setup is discussed. Firstly, the search and rescue mission utilising swarm configuration is presented in Section 5.1. Then, the environmental dynamic is discussed in Section 5.2. The following Section 5.3 investigates the searching behaviour of each swarm member. Fourthly, a simulation setup and results are presented in Section 5.4 and Section 5.5, respectively. Finally, the summary and conclusion are presented in Section 5.6 and Section 5.7.

5.1 Search and Rescue Mission

In conducting a search mission, the SAR team follows several provisions summarised in the IAMSAR (The International Aeronautical and Maritime Search and Rescue) manual published by ICAO (International Civil Aeronautical Organisation) (Organization and Organization 2016). This guideline explains several search techniques and preparations are required to be implemented in mitigating accidents on land, sea and air. In the following subsections, search patterns will be discussed.

5.1.1 Searching Technique on SAR Mission

Search assets, such as airplanes, which are mainly used to carry out the search mission generally have sensors in the form of cameras, sonar, or radar that are attached to the bottom of the fuselage facing downward. The sensors available on the fuselage are configured to cover the search area below the search asset. The range and swath width of the sensor can be seen on Figure 5.1. The width of the swath range is twice the sensor range from the fuselage. The range of sensors using a camera and sonar depend on the altitude of the search asset. The higher the altitude, the more data is received but the resolution is reduced.

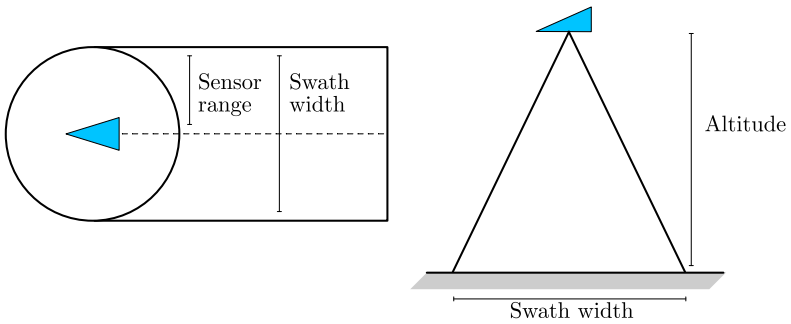


Figure 5.1: Sensor range and swath width

In a search mission, a swarm could be adopted as a search asset using existing search techniques. For example, by using flocking behaviour, the swath width will be even greater than single search asset as illustrated in Figure 5.2 below.

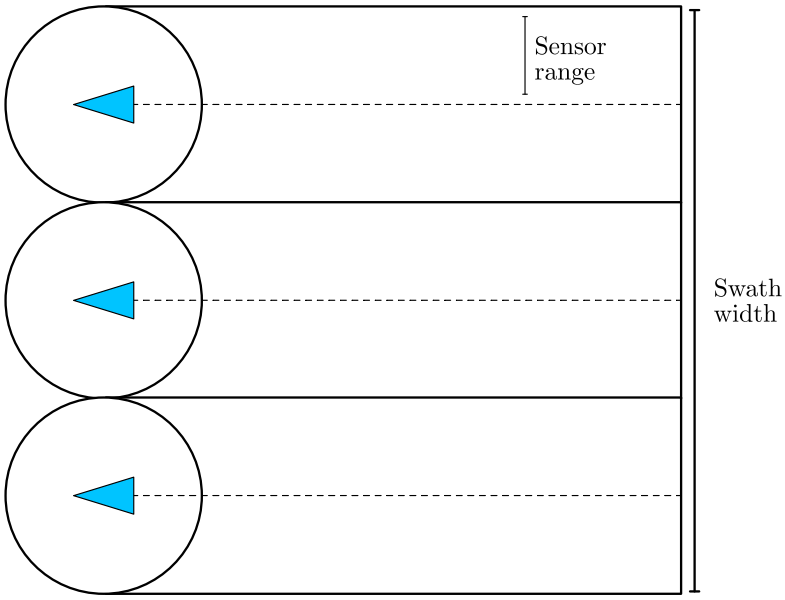


Figure 5.2: Swath width of swarm on SAR mission

Then, search pattern can be developed and have a wider detection area by applying flocking behaviour applying collision avoidance, alignment, attraction, and waypoints tracking.

5.1.1.1 Search Pattern

For the SAR mission, several search methods have been formulated by IMO standards (Organization and Organization 2016). In this subsection, several possible configuration of search patterns utilising swarm robotics are discussed.

5.1.1.2 Track Line Search

Track line search is a procedure used to perform searches by assets that follow a cruise line or flight path, as illustrated by Figure 5.3. The detectable area along the search path depends on the swath width of the search asset. The prediction of target's position is calculated in regard of cross wind or possible drifts.

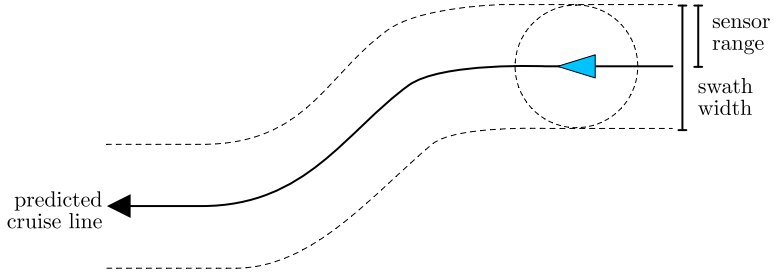


Figure 5.3: Track line search

If the flocking behaviour of swarm is applied, the swath width of track line-search can be

expanded by distributing member's swath width. To maximise the detection capability, the swath width is overlapped because the edge of sensor is the least sensitive and the distortion around the corner may affect the measurement. The configuration can be seen in Figure 5.4.

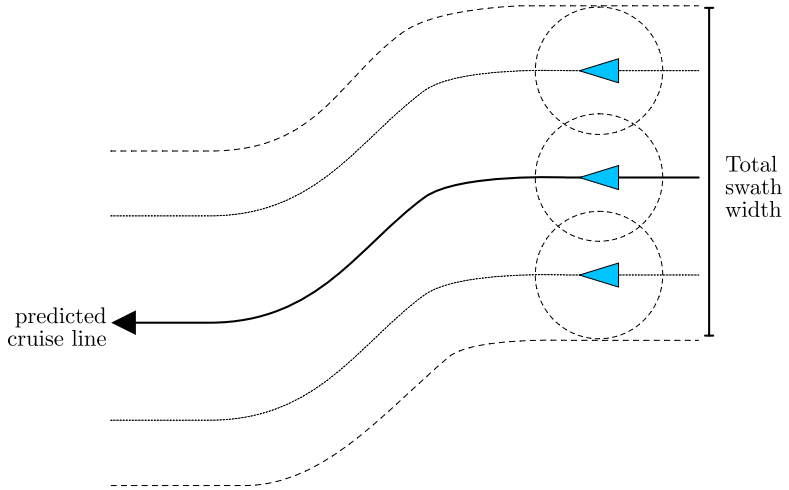


Figure 5.4: Swarm track line search

5.1.1.3 Swarm Creeping Line and Parallel Search

For a disaster happened in a fixed location, square search is preferred. Search techniques that are specifically design for a fixed location are creeping line, parallel search and expanding square search. The last is preferred for a disaster located on land, to search a flee behaviour of victims from disaster point. Creeping line and parallel search are similar in the configuration. The first is designated for a wide and long search area, while the second is commonly used for a wide search area only and conducted by following the longest side of the search area, both of the search methods are depicted by Figure 5.5 and Figure 5.6 respectively.

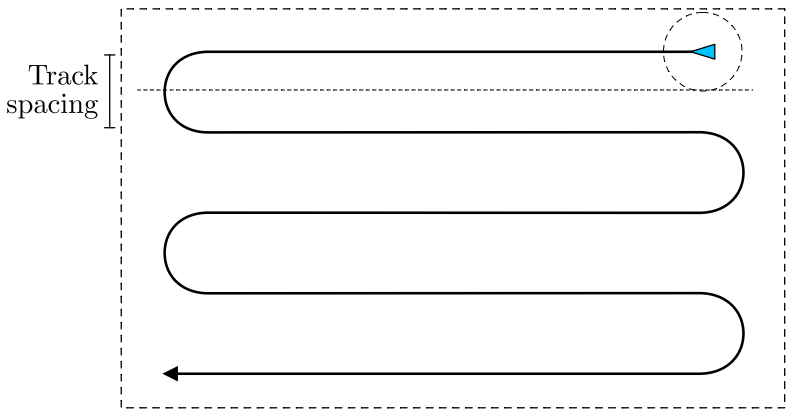


Figure 5.5: Parallel search

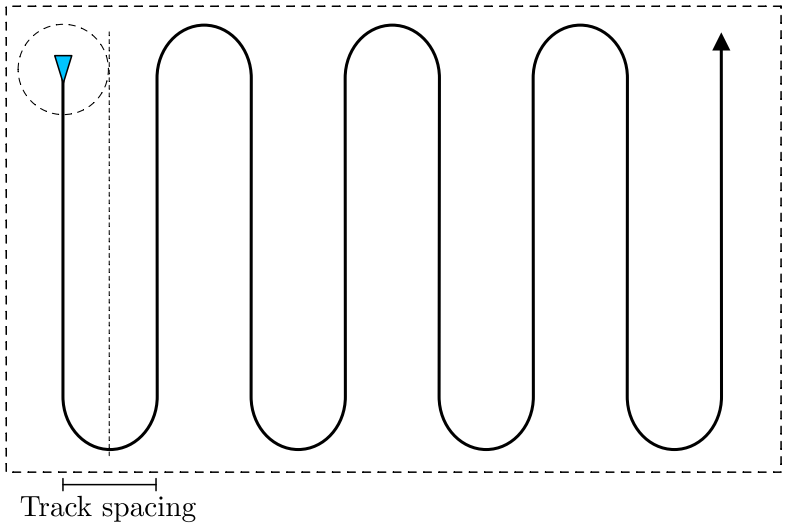


Figure 5.6: Creeping line search

From the figures above, the distance between paths depends on the swath width of the sensor to maximise the searchable area. The search path can be modified if there is a possibility drift caused by sea waves, currents or wind. The waypoints are skewed based on the speed of advance (SOA). The swath width configuration of square search for swarm can be modified by overlapping the search path while maintaining the group or flock integrity (Page, Armstrong, and Mukhlish 2019), see Figure 5.7.

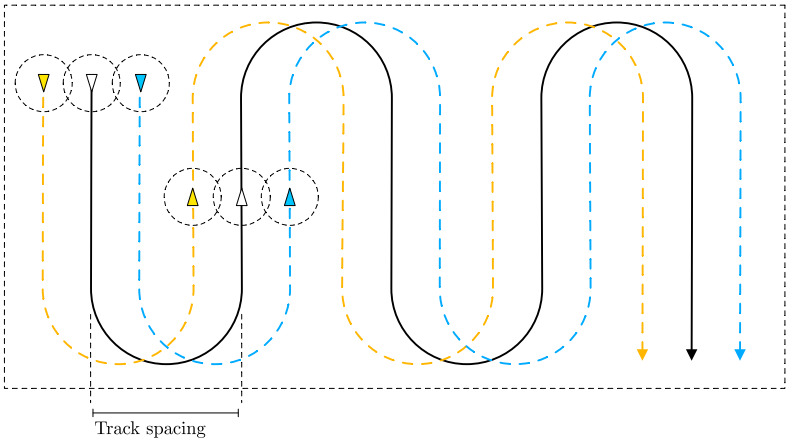


Figure 5.7: Swarm search for creeping line

Besides moving together as a flock, the use of swarm system can also be done by allocating the search area for each member as shown in Figure 5.8.

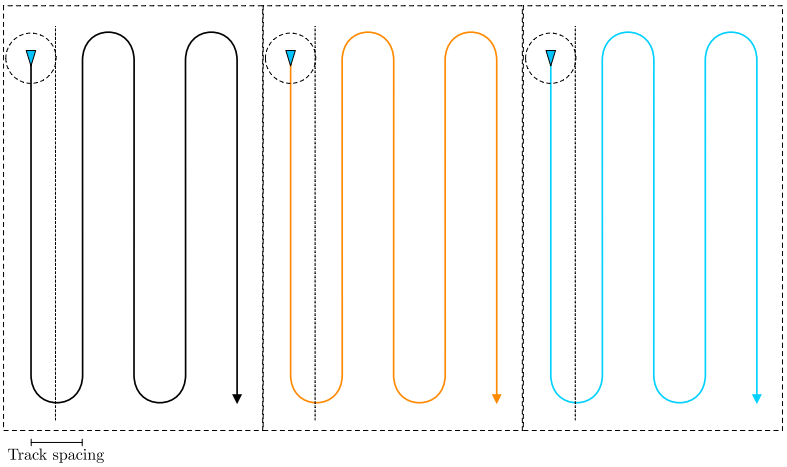


Figure 5.8: Swarm search allocation for parallel line

5.1.1.4 Consensus-based Tasks Allocation

The next allocation is to allocate different assignments based on group's consensus. For example, for a search mission that is full of uncertainties caused by the possibility of drifting victims by the sea currents, swarm robotics can be divided into several groups to carry out different tasks in covering the search area. Division of tasks could be square search team based on known datum and parallel search team based on ocean currents. The number of members in the division of tasks can be determined based on contact frequency of victims. The greater the probability or the detection rate at the allocated location, the greater the number of members deployed to the task. The example of division of tasks can be seen in the illustration in Figure 5.9.

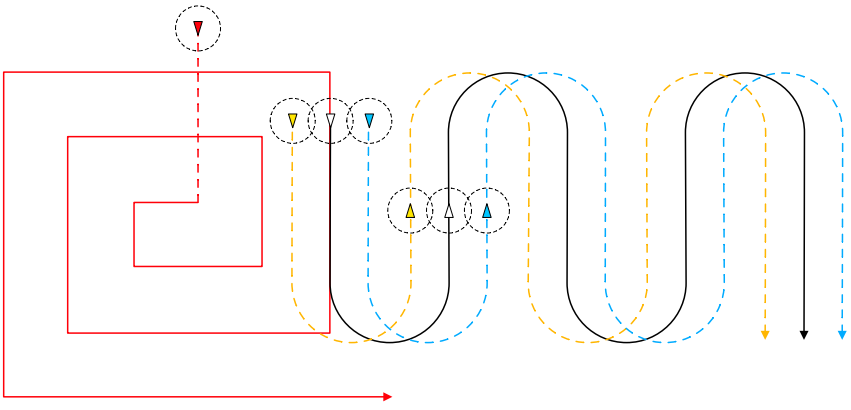


Figure 5.9: Example of consensus-based swarm search

5.2 Environmental Dynamic

The study case that is used in this work is a search and rescue mission over the ocean using swarm of unmanned aerial vehicles (UAVs). Moreover, the simulation consists of three components, namely UAV, Victim, and environment as depicted in Figure 5.10. Environmental dynamic settings in this simulation are composed of sun, wind and sea surface. These three components combined resulting a dynamic as follow. The Sun's radiance on the sea surface depends on position of the sun (zenith and azimuth angle), sky condition (cloudy or clear), and sea surface reflectivity. Both sea surface reflection value and sea wave's direction are dependant to wind direction.

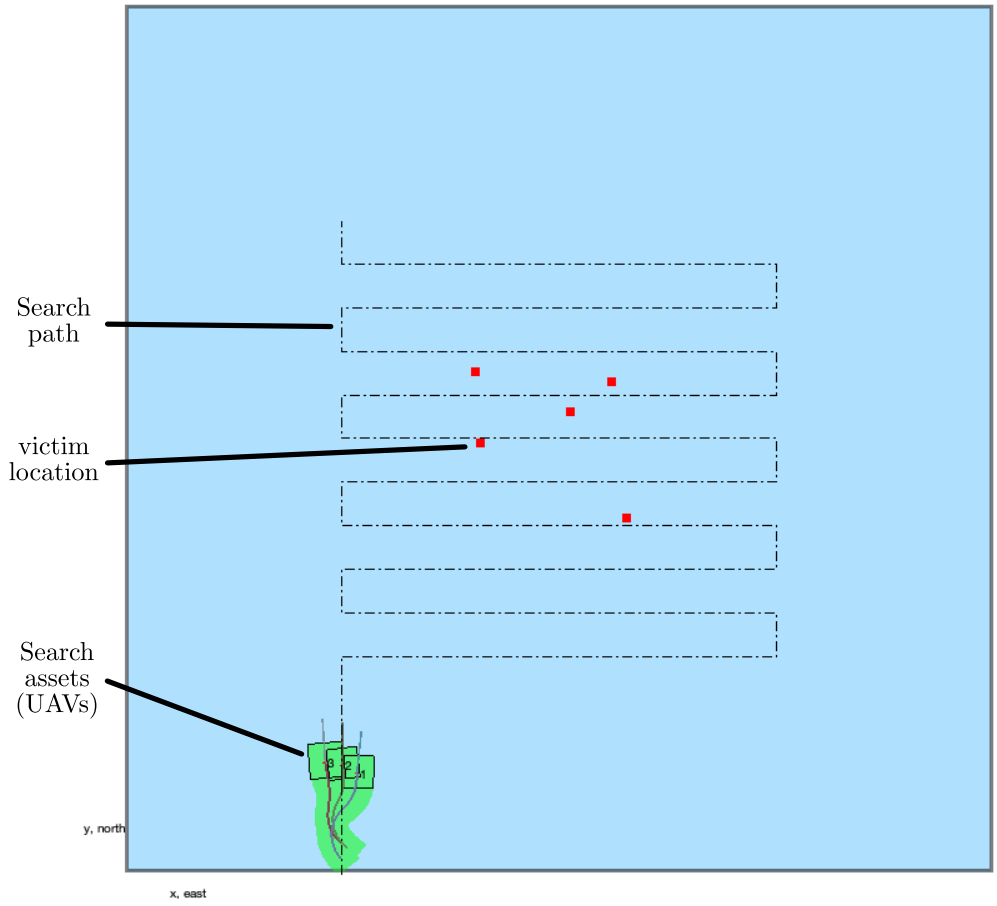


Figure 5.10: Area consists of UAVs, Victims and Environment

Wind's dynamic is regarded as vector which has components of scalar value and direction value described by:

$$\mathbf{W} = f(w, \psi) \tag{5.1}$$

Where w is the wind speed and ψ is the wind's direction. The direction is normaly distributed as $f(\psi \mid \mu_\psi, \sigma_\psi^2)$ with μ_ψ is the main direction and δ_ψ is a standard deviation by crosswind.

5.2.1 Sea Surface

In this simulation, sea surface has two components, namely reflective components and sea surface current. The sea surface current is taken to be proportional to wind's speed and direction. On the other hand, reflective component of the sea surface is derived by mathematical expression based on Fresnel's formula and Snell's law.

5.2.1.1 Sun Reflections

The real sea surface is inherently roughed with waves. The tempo-spatial features of the changing sea surface determine the reflection of the incident radiance L_i, which correspondently shows a directivity feature. The rate of the reflected radiance R_r from a direction is defined by using the Bidirectional Reflectivity Distribution Function (BRDF) (Ren, Liu, and Chen 2006)

$$R_r(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{dL_r(\theta_r, \phi_r)}{L_i(\theta_i, \phi_i)d\omega_i} \tag{5.2}$$

Where $L_r(\theta_r, \phi_r)$ is the reflected radiance in the direction θ_r, ϕ_r, $L_i(\theta_i, \phi_i)$ is the incident radiance in the direction θ_i, ϕ_i, ω_i is the solid angle of the incident radiance. The instructive view of the direction on a single wave facet is illustrated in

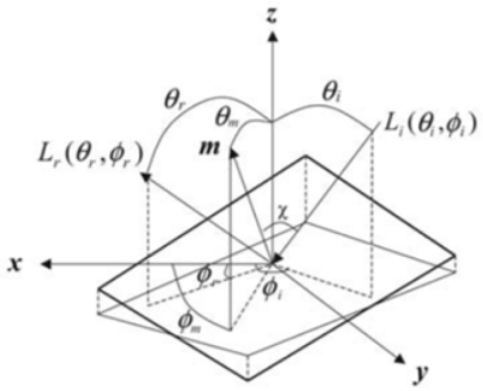


Figure 5.11: The instantaneous geometry on sea surface (Ren, Liu, and Chen 2006)

Then, because camera is commonly used to capture search area and most sensors used are unpolarised, the specular reflectivity at the air-sea interface can be expressed as:

$$\rho(\eta, \mathcal{X})\frac{|R_p|^2 + |R_s|^2}{2} \tag{5.3}$$

Where R_p and R_s are the reflected electromagnetic waves' magnitudes relative to incidence waves. According to the Fresnel's formula, R_p and R_s are given by

$$R_p = \frac{\eta \cos \mathcal{X} - \cos \mathcal{X}}{\eta \cos \mathcal{X} + \cos \mathcal{X}} \tag{5.4}$$

$$R_s = \frac{\cos \mathcal{X} - \eta \cos \mathcal{X}'}{\cos \mathcal{X} + \eta \cos \mathcal{X}'} \tag{5.5}$$

Where η is the seawater's reflective value, $\mathcal{X}$ is the incidence' angle, and $\mathcal{X}'$ is the refraction' angle given by the Snell's law as

$$\sin \mathcal{X}' = \frac{\sin \mathcal{X}}{\eta} \tag{5.6}$$

Although the real sea surface is featured as the roughness due to the inevitable waves, it is considered as a superposition of numerous wave facets. The area can be treated as a specular surface since the localised wave facet is so small. Accordingly, distribution of the reflective (BDRF) of the rough sea surface can be formulated by applying the specular reflectivity formula to each wave facet. Hence, the surface's facet slope can be approximated as a normal and isotropic distribution. The probability density function of the slopes is defined as

$$P(z_x, z_y) = \frac{1}{2\pi\zeta^2} \exp\left(-\frac{z_x^2 + z_y^2}{z\zeta^2}\right) \tag{5.7}$$

Where z_x and z_y are any orthogonal slope components at the sea surface. The value ζ^2 was founded to be proportional to the wind speed w at the range of 0-14 m/s (Ren, Liu, and Chen 2006), following calculation

$$2\zeta^2 = 0.003 + 0.00512w$$

The instantaneous $L_i(\theta_i, \phi_i)$ is reflected to the direction θ_r, ϕ_r at the wave facet, the slope of which is the function of parameters $\theta_i, \phi_i, \theta_r, \phi_r$,

$$z_x = -\frac{\sin \theta_i \cos \phi_i + \sin \theta_r \cos \phi_r}{\cos \theta_i + \cos \phi_r} \tag{5.8}$$

$$z_y = -\frac{\sin \theta_i \sin \phi_i + \sin \theta_r \sin \phi_r}{\cos \theta_i + \cos \phi_r} \tag{5.9}$$

Then, by using (5.3), the reflected radiance on the wave facet is transformed into

$L_i(\theta_i, \phi_i)\rho(\eta, \mathcal{X})$, where $\mathcal{X}$ is the incident angle defined as:

$$\cos \mathcal{X} = \sqrt{\frac{1 + \sin\theta_i \sin\phi_r \cos(\phi_i - \phi_r) + \cos\phi_i \cos\phi_r}{2}} \tag{5.10}$$

The mean reflected radiance is obtained as incidence radiance reflected by specular reflectivity times the sea surface's slope distribution. The formulation is

$$dL_r(\theta_r, \phi_r) = L_i(\theta_i, \phi_i)\rho(\eta, \mathcal{X})P(z_x, z_y)dz_x dz_y \tag{5.11}$$

By substituting (5.11)to (5.2) , the resulted BRDF reflectivity of the rough sea surface is

$$f_r(\theta_r, \phi_r, \theta_i, \phi_r) = \frac{\rho(\eta, \mathcal{X})P(z_x, z_y)dz_x dz_y}{\sin(\theta_i)d\theta_i d\phi_i} \tag{5.12}$$

since

$$dz_x dz_y = |\mathbf{J}| d\theta_i d\phi_i$$
$$|\mathbf{J}| = \begin{vmatrix} \frac{\partial z_x}{\partial \theta_i} & \frac{\partial z_x}{\partial \phi_i} \\ \frac{\partial z_y}{\partial \theta_i} & \frac{\partial z_y}{\partial \phi_i} \end{vmatrix} = -\frac{2\sin\theta_i \cos^2 \mathcal{X}}{(\cos\theta_i + \cos\theta_r)^3}$$

Equation (5.12) becomes

$$f_r(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{\rho(\eta, \mathcal{X}) \sin^2 \mathcal{X}}{\pi\zeta^2(\sin\theta_i + \cos\theta_r)^3} \exp\left(-\frac{z_x^2 + z_y^2}{2\zeta}\right) \tag{5.13}$$

This equation is the resulted mathematical expression for BRDF reflectivity of the rough sea surface, where ζ^2 is defined by equation (5.8) as a function of wind speed.

5.3 Swarm of Search Assets

Unmanned aerial vehicles (UAV) used in this simulation is Aerosonde™, see Figure 5.12. The UAV is chosen for the search and rescue simulation because it is capable of flying in extreme weather condition (Shakhatreh et al. 2019). The dynamic used in the simulation is based on the dynamic of small unmanned aerial vehicle with Aerosonde™specifications (Beard and McLain 2012).

Figure 5.12: Aerosonde™(Carey 2012)

5.3.1 Sensor Configuration

Each UAV is equipped with a camera capable of capturing image of the sea surface and victim detection. However, the image quality obtained by the UAV is affected by the sun's reflection on the sea surface, see Figure 5.13. If the reflection is too high, victim detection will suffer. Another factor affecting the image quality is the position of the camera relative to the sea surface, in this case is relative to UAV's altitude and the ground speed of the vehicle (Mukhlish, Page, and Bain 2019).

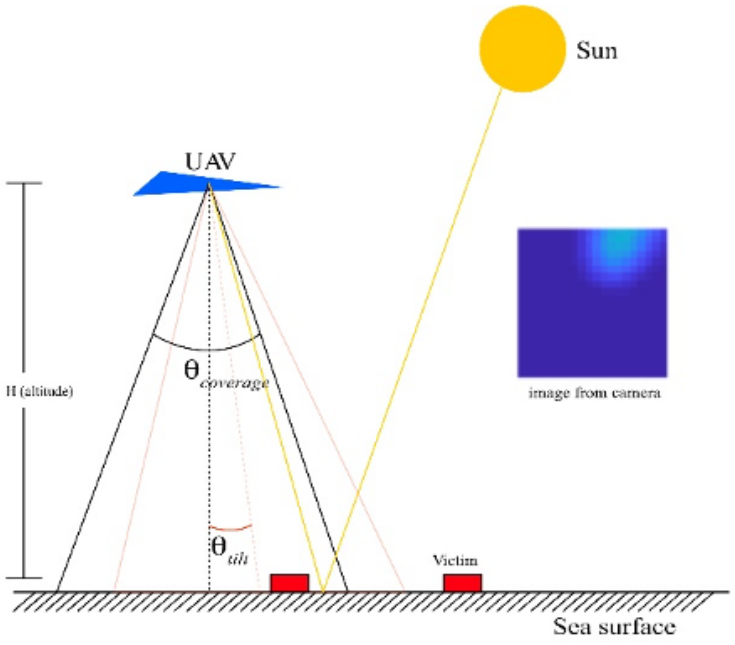


Figure 5.13: Sun reflection captured by UAV's camera

5.3.2 Search Behaviour

The search behaviour utilising swarm of UAVs is designed to maximise the total swath width. In order to do that, the swarm has to be aligned perpendicular to the searching

trajectory and maintain the distance between agents. Then the flock tracks the line between waypoints. The forces acted on each agent are illustrated in Figure 5.14.

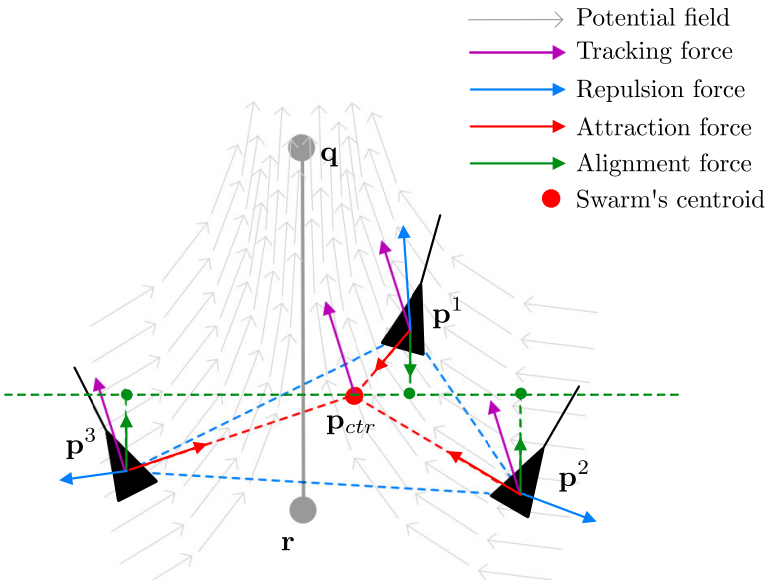


Figure 5.14: Forces of the collective tracking behaviour

There are four forces acted on each agent, namely tracking force, repulsion force, attraction force, and alignment force. As discussed in previous Chapter 4, each agent has a proximity sensor with certain range to detect nearby UAVs. From the detection, the repulsion force is formulated based on nearby agent's position while attraction, tracking and alignment forces are formulated in regard of the centre position of the swarm. From the proximity sensor's measurement, the centre of the swarm can be calculated based on the average value of all neighbours' position, known as swarm's centroid, calculated as average value of nearby UAVs' position.

$$\mathbf{p}^i_{\text{ctr}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}^i \tag{5.14}$$

Where N is the number of nearby UAVs and $\mathbf{p}$ is a position vector. Since the repulsion force is already discussed in previous Chapter 4, the formulations of the other forces are discussed in following subsections.

5.3.2.1 Tracking Force

Tracking force is a force that defines the direction of the flock. A potential field force is applied to define a force direction at any point around the path. In order to collectively flocking along the path, a potential force at the swarm's centroid is used as a tracking force, see Figure 5.15.

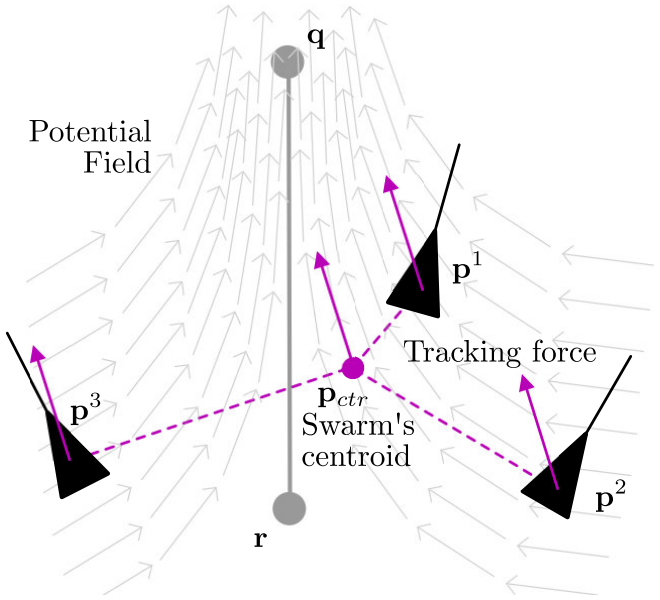


Figure 5.15: Tracking force of collective tracking

The swarm's centroid is calculated by each agent based on the average position of its nearby agents' positions, see (5.14). Hence, each agent may calculate different centroid in a presence of nearby UAVs, as depicted in Figure 5.16.

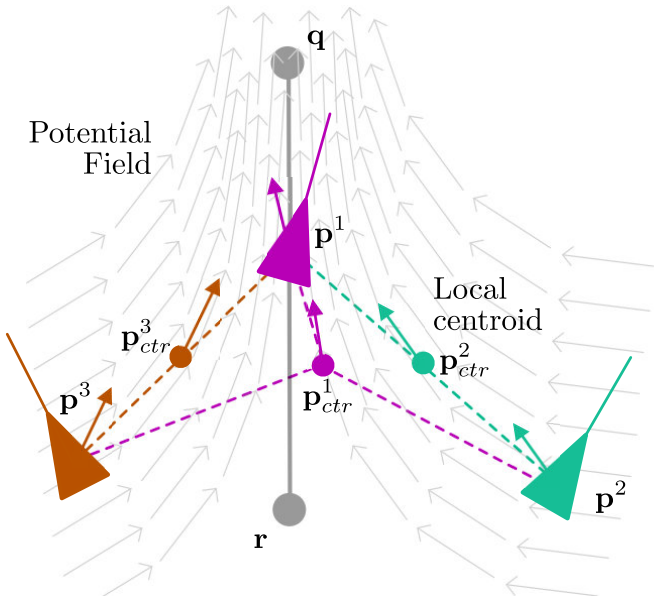


Figure 5.16: Tracking force caused by local centroid

When any UAVs are detected, then by following formulation (4.67) and (4.68) the tracking force yields desired altitude and desired heading on the centroid as

$$h_{\text{tracking}}\left(\mathbf{r}, \mathbf{p}^{i}_{\text{ctr},\mathbf{q}}\right) = -r_d + \sqrt{c_n^2 + c_e^2}\left(\frac{q_d}{\sqrt{(q_n^2 + q_e^2)}}\right) \tag{5.15}$$

Then, the desired heading χ_d is

$$\begin{aligned} \chi_{\text{tracking}}(t) &= \chi_q - \chi^{\infty}\frac{2}{\pi}\tan^{-1}\left(k_{\text{path}}e_{\text{ctr}_y}(t)\right) \\ \text{with}\chi_q &= \arctan^2(q_e, q_n) + 2\pi m \\ \text{where } m \in \mathcal{N} &\rightarrow \pi \leq \chi_q - \chi \leq \pi \end{aligned} \tag{5.16}$$

Where the course line has origin of the path denoted by $\mathbf{r} = \{r_n, r_e, r_d\} \in \mathbb{R}^3$ and a unit vector $\mathbf{q} = \{q_n, q_e, q_d\} \in \mathbb{R}^3$ as a direction indicates the desired of travel. The position of the centroid is $\mathbf{p}^i_{\text{ctr}} = \{p_{\text{ctr}_n}, p_{\text{ctr}_e}, p_{\text{ctr}_d}\} \in \mathbb{R}^3$ and its projection is $\mathbf{c} = \{c_n, c_e, c_d\} \in \mathbb{R}^3$, and the error is derived by calculating the vector between p and c as $e_{\text{ctr}} = \left\{e_{\text{ctr}_x}, e_{\text{ctr}_y}, e_{\text{ctr}_z}\right\}$. And k_{path} is a positive constant that influences the rate of the transition from χ^{∞} to zero. The tracking force is proportional to desired flocking speed V_{flock}.

5.3.2.2 Attraction Force

Attraction force is applied to maintain the integrity of the swarm. When a nearby UAV or more are in attraction radius, the attraction force points at the swarm's centroid, see Figure 5.17. Thus, all members are attracted to any nearby UAV.

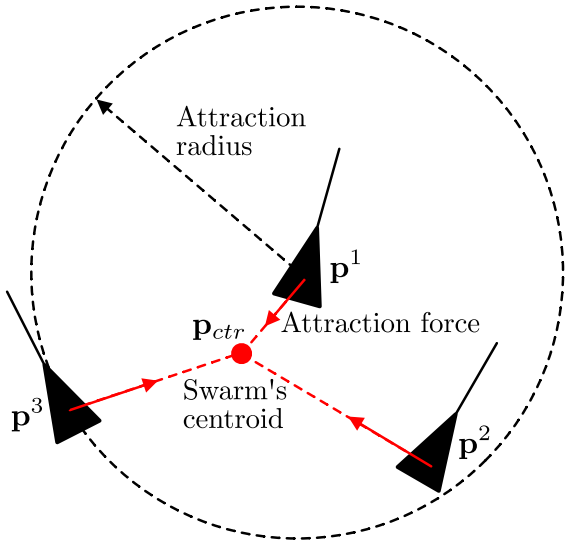


Figure 5.17: Attraction force of collective tracking

The attraction force is vector from an agent to centroid from its perspective and propor-

tional to the distance between centroid and agent's position.

$$\mathbf{F}_{\text{attr}} = \begin{cases} K_{\text{attr}} \left(\mathbf{p}^i_{\text{ctr}} - \mathbf{p}_i \right), & \text{if any UAVS nearby} \\ 0, & \text{otherwise} \end{cases} \tag{5.17}$$

Where K_{attr} is a rate of attraction that defines the strength of the attraction force.

5.3.2.3 Alignment Force

To align the search direction of the swarm, all agents are required to align perpendicular to the trajectory path. By utilising the swarm's centroid, the projection of the centroid on the path can be calculated. And based on the projection, an alignment line (dotted green line in Figure 5.18) is formulated as a direction of each alignment force on each agent.

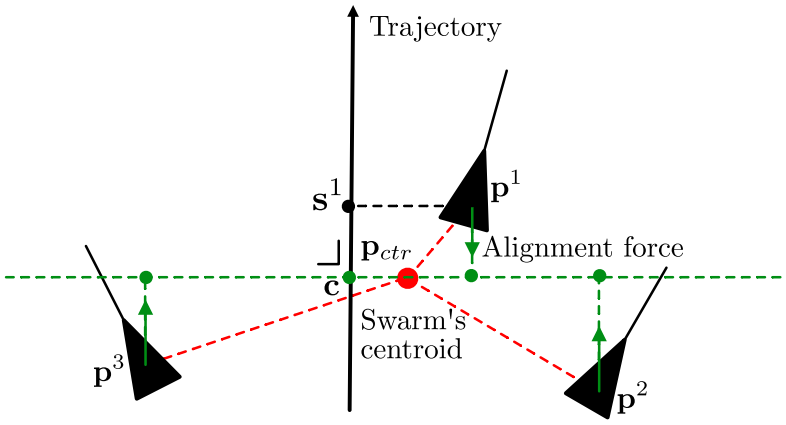


Figure 5.18: Alignment force of collective tracking

The alignment line is defined in regards of the centroid's projection $\mathbf{c} = \{c_n, c_e, c_d\} \in \mathbb{R}^3$ and UAV's projection $\mathbf{s} = \{s_n, s_e, s_d\} \in \mathbb{R}^3$ on the trajectory line. Then the alignment force is a vector from projection $\mathbf{s}$ to $\mathbf{c}$ and proportional to the distance between them as

$$\mathbf{F}_{\text{align}} = \begin{cases} K_{\text{align}} \left(\mathbf{c} - \mathbf{s}_i \right), & \text{if any UAVS nearby} \\ 0, & \text{otherwise} \end{cases} \tag{5.18}$$

Where K_{align} is a rate of alignment that defines the strength of the alignment force. Then, the desired heading of the UAV is calculated from resultant force acted on each UAV

$$\mathbf{F}_R = \begin{pmatrix} F_{R_e} \\ F_{R_n} \\ F_{R_d} \end{pmatrix} = F_{\text{traj}} + F_{\text{attr}} + F_{\text{align}} + F_{\text{rep}} \tag{5.19}$$

Since the heading direction of the UAV follows the earth axis, then only the lateral component of all forces in (5.19) is applicable on the desired heading.

$$\begin{aligned}\begin{pmatrix} F_{R_e} \\ F_{R_n} \end{pmatrix} = V_{\text{flock}} \begin{pmatrix} \cos\chi_d \\ \sin\chi_d \end{pmatrix} + K_{\text{attr}} \begin{pmatrix} p_{i_e} - p^i_{\text{ctr}_e} \\ p_{i_n} - p^i_{\text{ctr}_n} \end{pmatrix} + K_{\text{align}} \begin{pmatrix} p^i_{\text{ctr}_e} - s_{i_e} \\ p^i_{\text{ctr}_n} - s_{i_n} \end{pmatrix} \\ + K_{\text{rep}} \sum_{j \neq i} \left(\frac{R_{\text{outer}} - R_{\text{inner}}}{|D_{ij}| - R_{\text{outer}}} \right) \cdot \hat{\mathbf{f}}^r_{ij} \end{aligned} \tag{5.20}$$

Then, the desired heading is obtained as

$$\chi_R = \arctan \frac{F_{R_e}}{F_{R_n}} \tag{5.21}$$

The desired altitude follows the formulation in (5.15) and the desired flocking velocity is calculated and only affected by the alignment force as

$$V_R = \left\| V_{\text{flock}} \begin{pmatrix} \cos\chi_d \\ \sin\chi_d \end{pmatrix} + K_{\text{align}} \begin{pmatrix} p^i_{\text{ctr}_e} - s_{i_e} \\ p^i_{\text{ctr}_n} - s_{i_n} \end{pmatrix} \right\| \tag{5.22}$$

5.4 Simulation Setup

Target points are arranged forming a set of waypoints for UAVs to search the environment, the set of waypoints is calculated based on searching pattern given to the UAV, in this simulation a line search and creeping line pattern are used, see Figure 5.4 and Figure 5.7. The creeping line pattern is calculated dependent on the area size and swath width of the vehicle as can be seen on Figure 5.1. Swath width in this case is the area covered by UAV's camera pointing downward and is dependent on the altitude of the UAV. The simulation is conducted using computational software Matlab™, with following parameters in Table 5.1.

Table 5.1: Simulation parameters

Parameters	Value	Parameters	Value
Number of agents	3	Mutation rate	0.05
ε-greedy	0.75	Imprinting rate	0.01
Learning rate	0.1	Silence rate	0.05
Discount factor	0.1	Number of states	8
Methylation rate	0.1	Number of actions	2
Regeneration rate	0.75	Evolution interval	20 episodes
Histone selection rate	0.75		

The states consist of flocking behaviours, as discussed in Chapter 4, with the addition of group status and victim detection. The actions consist of actions discussed in Chapter 4 with the addition of searching behaviours. The complete states, terminal states and actions are listed in Table 5.2 below.

Table 5.2: States and actions

States	Actions
start	self-search
alone-near-path	wandering
found-uav-near-path	avoid-UAV
flocking-near-path	flocking
alone-far-path	flocking-and-search
found-uav-far-path	
flocking-far-path	
found-victim	
collide (terminal)	
crash (terminal)	
finish (terminal)	

The learning process goes through two phases, namely evaluation process and improvement process. The evaluation process is a process of getting or sampling experience to make sense of how the environment works. Then, after the experience is gained, the swarm enters the improvement process. The improvement process consists of experience exchanging based on the epigenetic mechanisms as discussed in previous chapters.

5.5 Results

The search patterns used for the search and rescue are line search and creeping line search. The swarm of UAVs evolves differently in each search pattern. For the line search pattern, the path is shown in Figure 5.19.

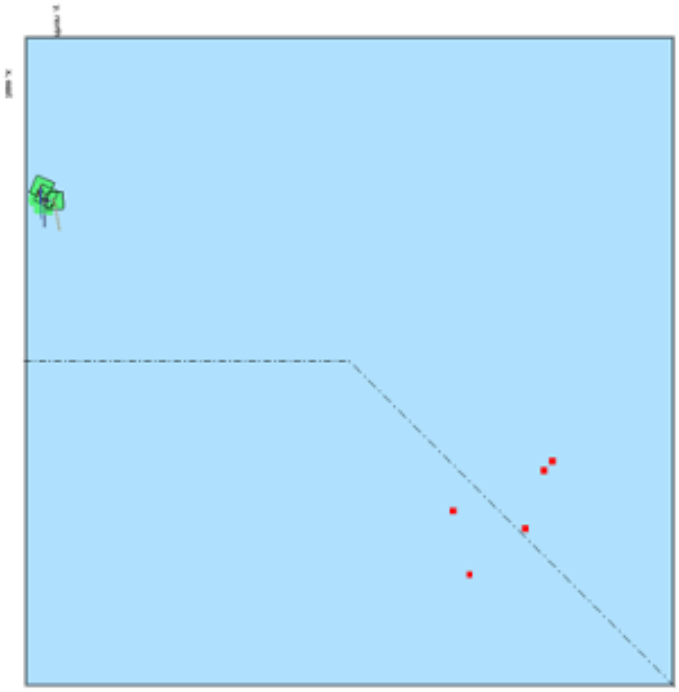
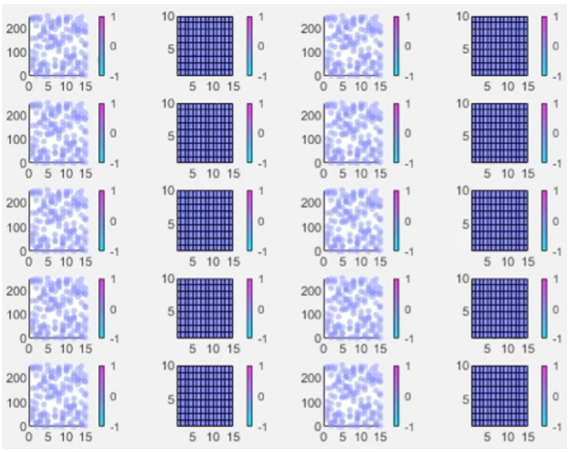
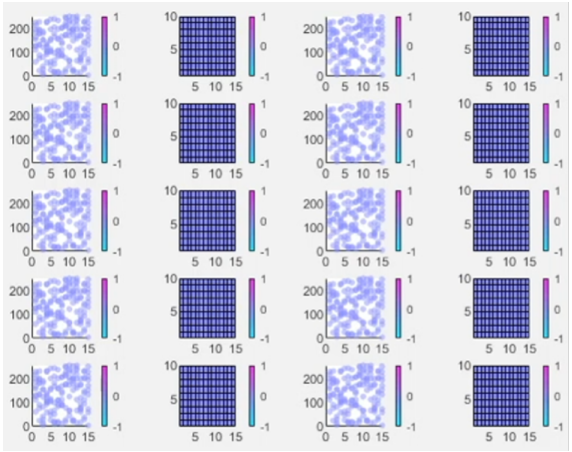


Figure 5.19: Line search pattern

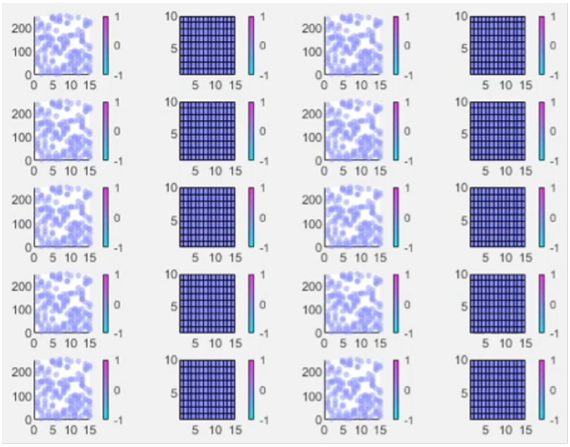
Since the problem parameters contain 8 states with three terminal states, the computationally worthy states are 5 states, namely start, far from path, near from path, UAVs nearby, and victim found. The initial gene-pool and chromosome-pools has equal histones values which are zero as depicted in Figure 5.20. The genetic population of all gene family are the same at the beginning, but the composition of genetic selection for each chromosome in the chromosome pool are picked randomly. The Figure 5.20 below shows that the dotted graphs on the left-side of each graph pair is the genetic population with the y-axis is the bit values and the x-axis is the gene family index. The grid graph is a chromosome set with y-axis is the chromosome index and the x-axis is the locus (gene's location) in respect to gene family index. The left-hand column represents the gene family and the right-hand column represents the gene selected in the given genome, as defined in Chapter 4.



(a) genepool of UAV-1



(b) genepool of UAV-2



(c) genepool of UAV-3

Figure 5.20: Initial genepool and chromosomepools for each UAVs

Search behaviour after 10 episodes of evaluation process and experiences in these episodic windows can be seen in Figure 5.21. In the initial step, the trial of all available chromosomes is conducted. Initial beliefs of the histone values are built from the equal chance of selection since the selection policy applies equal probability to all chromosomes.

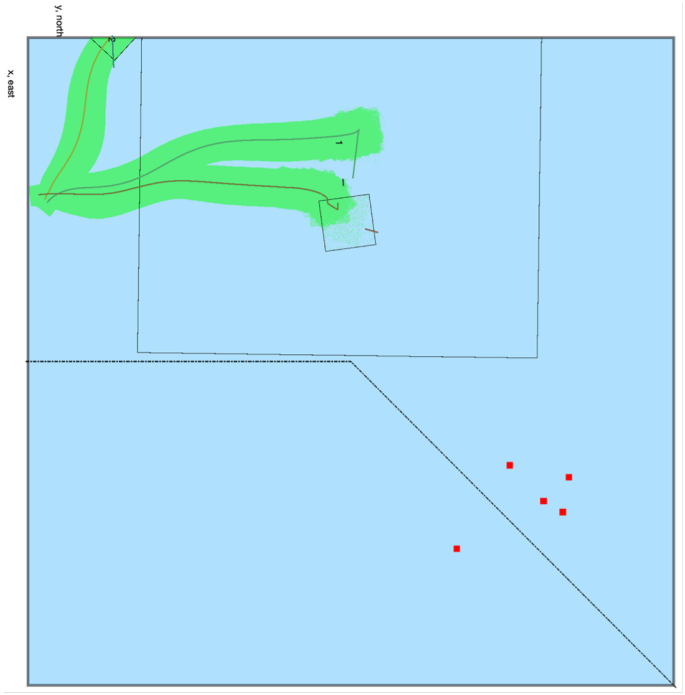
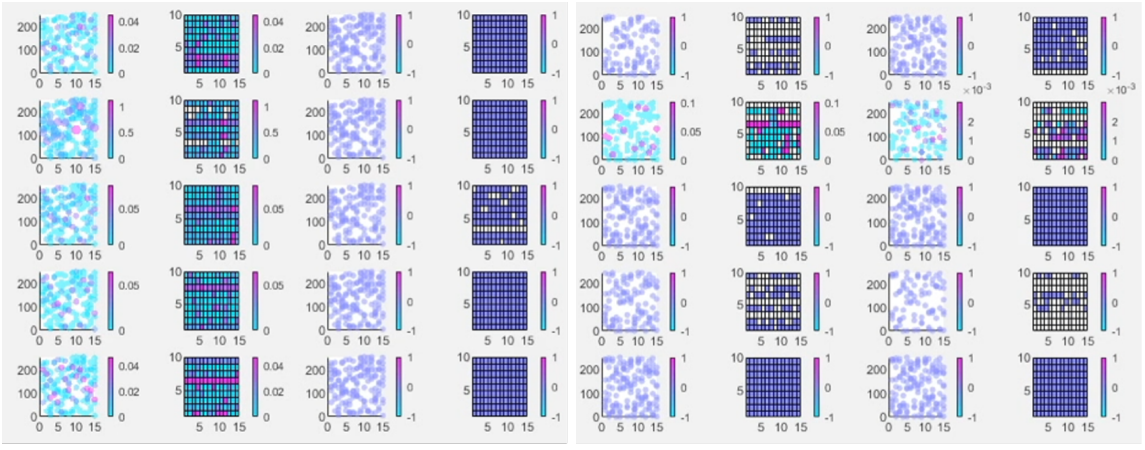


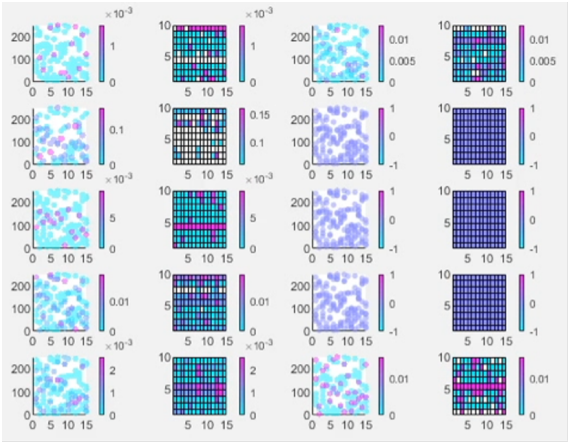
Figure 5.21: Search behaviour at episode-10

The histones values of each gene in the gene-pools and the chromosome-pools are obtained as depicted in Figure 5.22. It can be seen that after 10 episodes several experiences from each episode makes changes to the histone value through methylation process. And notice the colour changing at the light blue and magenta. The magenta one is the better performing gene, and the light blue is the least performing gene. Each agent explores and exploit independently and develops a unique belief of its own genetic strategy. The search behaviour of the swarm at episode-20 can be seen in Figure 5.23.



(a) genepool of UAV-1 (b) genepool of UAV-2

Figure 5.22: Episode 10 – Genepool and chromosomepool for each UAVs



(c) genepool of UAV-3

Figure 5.22: Episode 10 – Genepool and chromosomepool for each UAVs

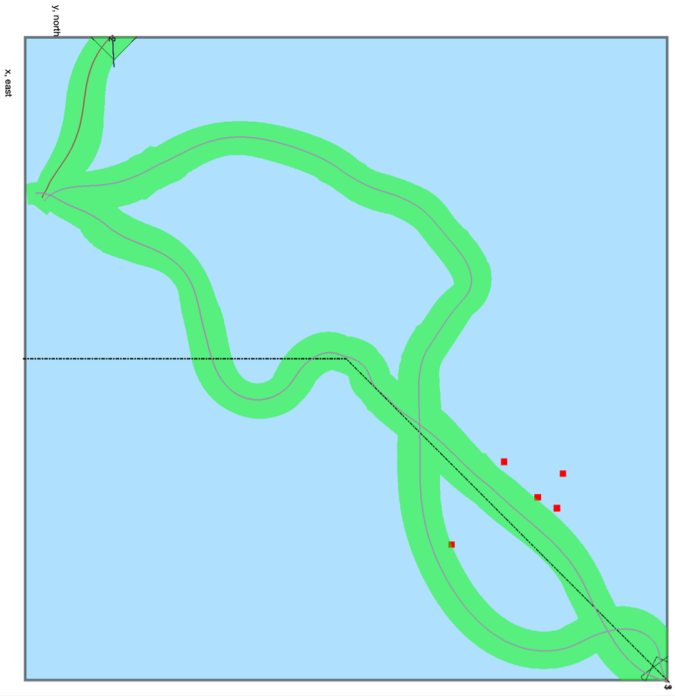
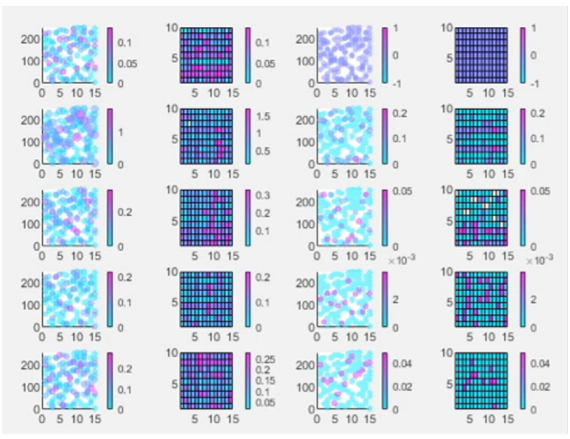
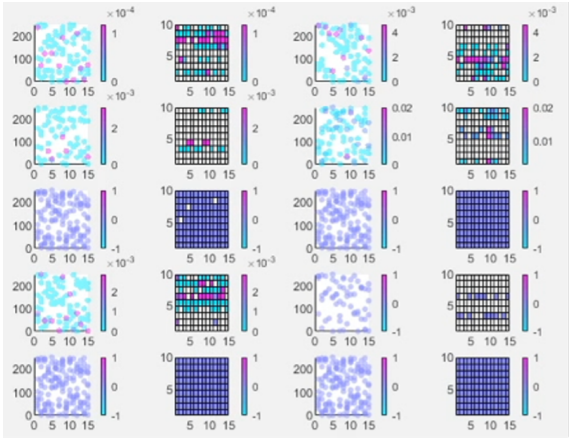


Figure 5.23: Search behaviour at episode-20

Then, after 20 episodes the differentiation between least and better performing genes become more apparent as depicted in Figure 5.24. The strategy exchange happened after the 10th episodes, and in the second window of evaluation (11th-20th episodes). It can be seen in Figure 5.24 that the epigenetic mechanisms is yielding a new histone values composition for each agent and promoting a better strategy from other agents.

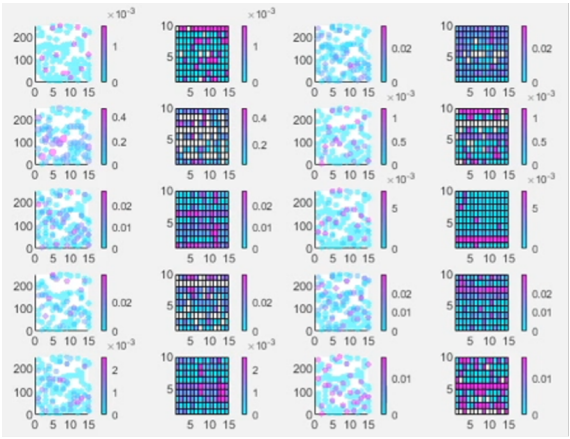


(a) genepool of UAV-1



(b) genepool of UAV-2

Figure 5.24: Episode 20 – Genepool and chromosomepool for each UAVs



(c) genepool of UAV-3

Figure 5.24: Episode 20 – Genepool and chromosomepool for each UAVs

In the next window, the swarm goes through the improvement process utilising the co-evolving using Epigenetic inheritance. It can be seen in Figure 5.23 that the behaviour of the swarm gets better after the first evolution after the 10th episode. They exchange experience and genetic strategy utilising the epigenetic mechanisms. From Figure 5.21 and 5.23 , if we compare the behaviour of the swarm, it is evident that the following behaviour and avoiding behaviours are getting better from episode to episode. Finally, after 200 episodes when the swarm done 20 improvement process or known as co-evolution process. The behaviour of the swarm is obtained as can be seen in following Figure 5.25.

Figure 5.25: Search behaviour at episode-200

The evolving decision making from episode to episode is summarised by how the histone values changing over episode to episode (see Figure 5.26).

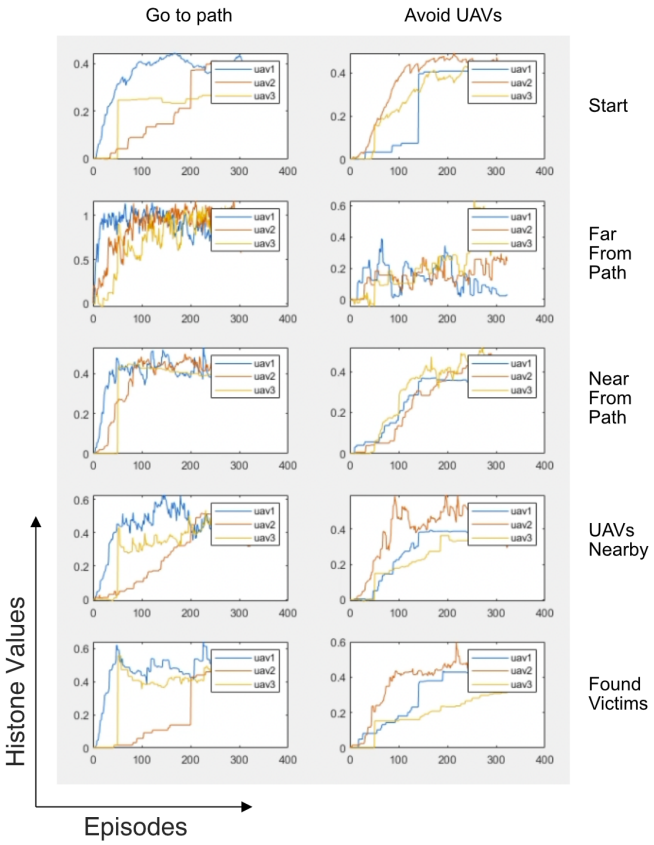


Figure 5.26: Histone values for line search

With the same process, the swarm behaviour for creeping line search can be seen in Figure 5.27 and Figure 5.28.

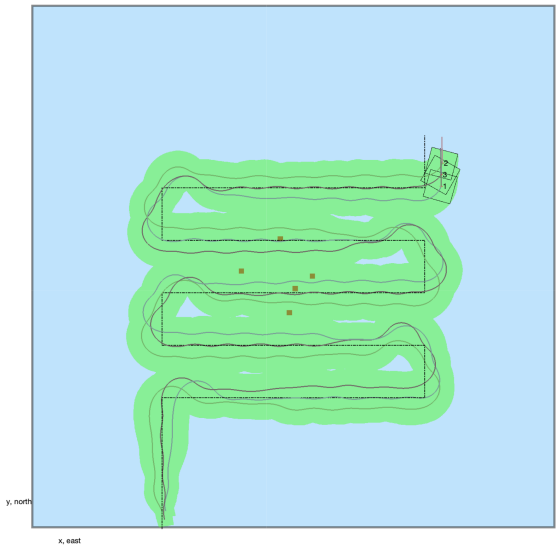


Figure 5.27: Resulted behaviour for creeping line search in 1000 Episodes

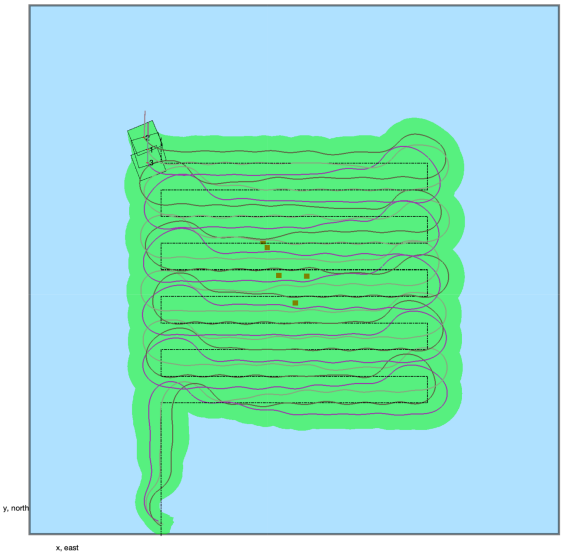


Figure 5.28: Resulted behaviour for creeping line search with reduced swath width

5.6 Summary

Search and rescue mission in this chapter is incorporated within a dynamic environment, namely wind, sea surface and reflection. The detection process with camera on board and affected by the sea reflection are also demonstrated. In this environmental setting, the evolutionary-learning framework using epigenetic layer is moving toward improved behaviour over each episode.

From the results, it is evident that the evaluation process and improvement process have distinctive mechanisms. The latter provides the communication and exchange process to improve each agent's strategy and resulted in improve swarm group behaviour. The first gives the swarm an ability to make sense of the dynamics of environment based on the action it took from episode to episode and then bootstrapped all achievable reward or feedback from the environment.

5.7 conclusion

The decentralised epigenetic learning for multi-agent system is successful in handling the highly dynamic problem such as a search and rescue mission. The simulation shows that the swarm can adapt to different search pattern, track line search and creeping line search. The exchange process through epigenetic mechanisms for a highly solution dimensional space (states = 11, actions = 5, and genotypes = 5), where the permutation problem applied. However, the problem presented in this chapter cannot be observed directly, because all simulations are done in one simulation software. Hence, for a more realistic scenario, future research with separated agents such as swam cluster simulator or real physical robots are necessary. This will develop the understanding of how well the strategy exchange is in a real time problem.

Chapter 6

A Bayesian Epigenetic Learning Swarm

This chapter presents a collective Epigenetic swarm Learning method that incorporates a recursive Bayesian filtering swarm to search and track multiple targets autonomously. The filtering method enables the learning process to stochastically update the target's probability density functions (PDFs). Since the targets' distribution is considered a non-linear problem, the filter method allows the proposed approach to handle the problem in the presence of non-Gaussian noise (Bourgault, Furukawa, and Durrant-Whyte 2006). In practical situations, a non-Gaussian signal frequently occurs because, in practice, the perceived data from the environment have been found to deviate strongly from a Gaussian characterisation.

6.1 Related Works

Search and tracking (SAT) have been studied for decades. The first search and rescue mission was performed by the British Royal Naval Air Service In 1915. Early work by Koopman (1980), which was firstly declassified in 1958, describes the primary issues in SAT during World War II. The early work was utilising area coverage technique by considering the search issue as an area coverage problem. Today, the proceeding works can be found in the Australian Search and Rescue Manual. Another development in the field is the introduction and utilisation of the probability of detection(L. D. Stone 1989b, 1989a; Yan and Blankenship 1988; T. Stone 2018). Later, Bourgault, Furukawa, and Durrant-Whyte (2004, 2003) formulated a Bayesian approach to search and tracking based on targets' prior and posterior distribution. Based on the Bayesian approach, a multi-vehicle search method was investigated (Bourgault, Furukawa, and Durrant-Whyte 2006). Besides Bayesian techniques, there are several filtering techniques to solve search and tracking problems such as the extended Kalman Filter by Jazwinski (1997), Unscented Kalman Filter by Julier and Uhlmann (2004), Gauss quadrature method by Ito and Xiong (2000), grid-based methods,

and Monte Carlo of particle filter methods by Arulampalam et al. (2002).

6.2 Bayes Filter

Recursive Bayesian Estimation (RBE) is a probabilistic approach to estimate an unknown probability density function (PDF). The estimation utilises measurements and a mathematical model recursively. In general, RBE is also known as Bayes Filter. A prior and posterior probabilities, which are known as Bayesian statistics, are used extensively as a process model of the estimation. The recursive process consists of two main parts, prediction and innovation.

RBE is used in robotics to estimate its state, such as position or orientation, based on the sensor's measurements. The estimation process allows an agent to conclude/deduce/infer its state over time. To be able to do this, the agent uses the most recently acquired sensor data to update their most likely state within an environment. For example, an agent starts with certainty that it is at the true position (x, y). As the robot wanders the space, it becomes less certain with its current true position. In this case, the agent can deduce its position overtime by updating its certainty using Bayes filter.

Since the true state is observed by the sensor and also estimated using Bayesian estimation, we can assume that the state is an unobserved Markov process, which means that the next state is only affected by the current state, not from the previous experiences. By assuming the process is Markovian, the Hidden Markov Model (HMM) can be used as a statistical model with an observational process. In HMM, the unobservable true state is defined as $\mathbf{x}$, and there is another observable process $\mathbf{z}$ assumed to be dependent on $\mathbf{x}$. For each time instance k, HMM stipulates the conditional probability distribution of $\mathbf{z}_k$ given the history $\{\mathbf{x}_n \in \mathcal{X}\}_{n \leq k}$, must not depend on $\{\mathbf{x}_n\}_{n<k}$. The following Figure 6.1 presents a Bayesian Network of an HMM.

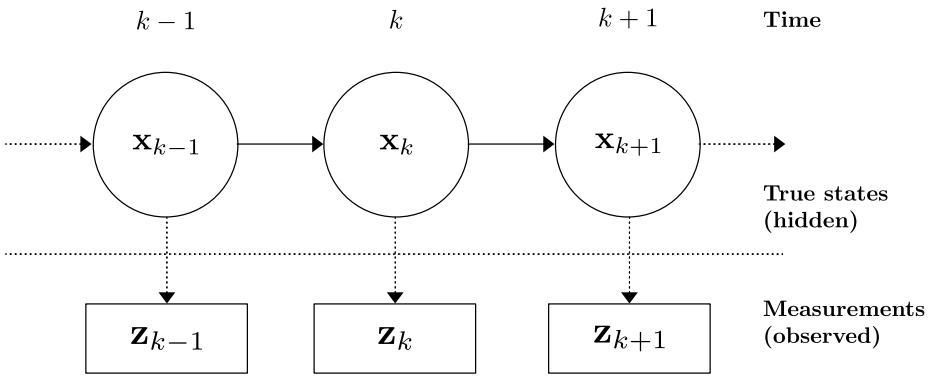


Figure 6.1: Bayesian network of hidden Markov model (HMM)

Then, the probability of true state at time step k is $\mathbf{x}_k$ given the previous state $\mathbf{x}_{k-1}$ is

defined following a Markovian rule as

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \ldots, \mathbf{x}_0) = p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) \tag{6.1}$$

Since the observed state $\mathbf{z}$ at the k-th timestep depends on the state $\mathbf{x}_k$, so the probability of observed state $\mathbf{z_k}$ given the true states is defined as

$$p(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{x}_{k-1}, \mathbf{x}_{k-2}, \ldots, \mathbf{x}_0) = p(\mathbf{z}_k \mid \mathbf{x}_k) \tag{6.2}$$

And the probability distribution of all states can be denoted as

$$p(\mathbf{x}_0, \ldots, \mathbf{x}_k, \mathbf{z}_1, \ldots, \mathbf{z}_k) = p(\mathbf{x}_0) \prod_{i=1}^{k} p(\mathbf{z}_i \mid \mathbf{x}_i) p(\mathbf{x}_i \mid \mathbf{x}_{i-1}) \tag{6.3}$$

When estimating the true state $\mathbf{x}_k$, the probability of interest is associated with the current true state given the measurement up to the $k-1$-th timestep, denoted by $\tilde{\mathbf{z}}_{1:k-1}$. The probability distribution is then equal to the sum (integral) of the true state transition probability from $k-1$-th timestep to the k-th and the probability of the previous state, to all possible $\mathbf{x}_{k-1}$.

$$p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) p(\mathbf{x}_{k-1} \mid \tilde{\mathbf{z}}_{1:k-1}) d\mathbf{x}_{k-1} \tag{6.4}$$

The updated probability of interest is proportional to the product of the observed state and the predicted state.

$$p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k}) = \frac{p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1})}{p(\mathbf{z}_k \mid \tilde{\mathbf{z}}_{1:k-1})} \propto p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1}) \tag{6.5}$$

The denominator, a probability of the predicted observed state is

$$p(\mathbf{z}_k \mid \tilde{\mathbf{z}}_{1:k-1}) = \int p(\mathbf{z}_k \mid \mathbf{x}_k) p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1}) d\mathbf{x}_{k-1}, \tag{6.6}$$

The numerator can be calculated and normalised since its integral must be unity. Hence, with likelihood $l(\mathbf{x}_k \mid \mathbf{z}_k) \triangleq p(\mathbf{z}_k \mid \mathbf{x}_k)$, the complete probability distribution of update is

$$p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k}) = \frac{l(\mathbf{x}_k \mid \mathbf{z}_k) p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1})}{\int l(\mathbf{x}_k \mid \mathbf{z}_k) p(\mathbf{x}_k \mid \tilde{\mathbf{z}}_{1:k-1}) d\mathbf{x}_{k-1}} \tag{6.7}$$

6.3 Bayesian Estimation

6.3.1 Target, Agent, and Sensor Platform Models

Consider a target, denoted by $\mathbf{t}$, is being searched for and being tracked. The motion is discretely given by:

$$\mathbf{x}_{k+1}^{\mathbf{t}} = f^{\mathbf{t}}(\mathbf{x}_k^{\mathbf{t}}, \mathbf{u}_k^{\mathbf{t}}, \mathbf{w}_k^{\mathbf{t}}) \tag{6.8}$$

Where $\mathbf{x}_k^{\mathbf{t}} \in \mathcal{X}^{\mathbf{t}}$ is the target' state at time step k, $\mathbf{u}_k^{\mathbf{t}} \in \mathcal{U}^{\mathbf{t}}$ is the set of target's inputs, and $\mathbf{w}_k^{\mathbf{t}} \in \mathcal{W}^{\mathbf{t}}$ is the external perturbation on the target such as wind and sea current in a search and rescue mission. The search asset is an autonomous agent, denoted by $\mathbf{a}$. The agent's motion model is given by

$$\mathbf{x}_{k+1}^{\mathbf{a}} = f^{\mathbf{a}}(\mathbf{x}_k^{\mathbf{a}}, \mathbf{u}_k^{\mathbf{a}}) \tag{6.9}$$

Where $\mathbf{x}_k^{\mathbf{a}} \in \mathcal{X}^{\mathbf{a}}$ represents the agent's state and $\mathbf{u}_k^{\mathbf{a}} \in \mathcal{U}^{\mathbf{a}}$ represents the agent's control input.

The agent carries a sensor, denoted as $\mathbf{s}$, with a limited coverage area as discussed in Chapter 5. Since only camera used in the previous Chapter, the terms "sensor platform" and "agent" are interchangeable. The probability of detection around the coverage area of the sensor is defined in a range of $0 \leq P_d(\mathbf{x}_k^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{s}}) \leq 1$. Hence, the observable area of the sensor can be expressed as ${}^{\mathbf{s}}\mathcal{X}_{\mathrm{o}}^{\mathbf{t}} = \{\mathbf{x}_k^{\mathbf{t}} \mid 0 \leq P_d(\mathbf{x}_k^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{s}}) \leq 1\}$. The observed state model ${}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}} \in \mathcal{X}^{\mathbf{t}}$ is defined as

$${}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}} = \begin{cases} {}^{\mathbf{s}}h_k^{\mathbf{t}}(\mathbf{x}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}}, {}^{\mathbf{s}}\mathbf{v}_k^{\mathbf{t}}), & \mathbf{x}_k^{\mathbf{t}} \in {}^{\mathbf{s}}\mathcal{X}_{\mathrm{o}}^{\mathbf{t}} \\ \emptyset, & \mathbf{x}_k^{\mathbf{t}} \notin {}^{\mathbf{s}}\mathcal{X}_{\mathrm{o}}^{\mathbf{t}} \end{cases} \tag{6.10}$$

Where ${}^{\mathbf{a}}\mathbf{v}_k^{\mathbf{t}}$ represents the measurement's noise and $\emptyset$ represents an event when target is undetected in the observable region.

6.3.2 Recursive Bayesian Estimation

Following the Recursive Bayesian Estimation method, the estimation of target's state $\mathbf{x}^{\mathbf{t}_k}$ can be predicted from the series of measurement event ${}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}} \equiv \{{}^{\mathbf{s}}\mathbf{z}_i^{\mathbf{t}} \mid \forall i \in \{1, \ldots, k\}\}$ and the set of sensor's state $\tilde{\mathbf{x}}_{1:k}^{\mathbf{s}} \equiv \{\mathbf{x}_i^{\mathbf{a}} \mid \forall i \in \{1, \ldots, k\}\}$. The target's probability density function at any time step k, denoted as $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$, is estimated recursively through update and prediction stage.

6.3.2.1 Update

The updating process computes the posterior probability density, denoted by $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$, given a new measurement ${}^{\mathbf{s}}\mathbf{z}_k^t$ and the target's prior probability density $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$. By applying (6.7), the update equation is defined as

$$\begin{aligned} p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) =& \frac{l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})}{p({}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})} \\ =& K\, l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_k^{\mathbf{t}}, \tilde{\mathbf{x}}_k^{\mathbf{s}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) \end{aligned} \tag{6.11}$$

with,

$$K = \frac{1}{\int l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) d\mathbf{x}_{k-1}^{\mathbf{s}}}$$

Where $l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}})$ represents the measurement likelihood given the information of the current agent's state defined by (6.10). Note that, at time step $k = 1$, the prior probability density $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$ is initialised as $p(\mathbf{x}_0^{\mathbf{t}})$.

6.3.2.2 Prediction

The prediction step computes the probability density of the target's next state, denoted as $p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$, given the target's current probability density $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) = p(\mathbf{x}_0^{\mathbf{t}})$. The prediction equation is carried out by applying Chapman-Kolmogorov's total probability theorm as

$$p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) = \int p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{t}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) d\mathbf{x}_k^{\mathbf{t}} \tag{6.12}$$

Where $p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{t}})$ is a probabilstic Markov notation model defined by (6.8) which maps the probability of transition from the current state $\mathbf{x}_k^{\mathbf{t}}$ to the next state $\mathbf{x}_{k+1}^{\mathbf{t}}$.

6.4 Bayesian Temporal Difference Learning

Based on the formulation in Chapter 4, learning is acquired by utilising reward at every time step R_k and return G_k from the environment. And later, make use of the methylation process to backup all the experience and infer the dynamics of the environment. The objective of a Bayesian search is to infer the target's states or the distribution of where the targets can be found with the addition of environmental noise. We can consider the observation dynamic (6.10) as a feedback from the environment to learn what to do $\mathbf{u}^{\mathbf{s}}_{k-1}$ at current agent's state $\mathbf{x}^{\mathbf{s}}_{k-1}$ after deducting the target's next states $\tilde{\mathbf{x}}^{\mathbf{t}}_{k:k+n_k}$ from n_k-th prediction $p(\mathbf{x}^{\mathbf{t}}_{k+n_k} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}^{\mathbf{t}}_{1:k}, \tilde{\mathbf{x}}^{\mathbf{s}}_{1:k})$.

The paths the agent will takes are the arguments that maximise given by the n_k-th prediction

$$\hat{\mathbf{x}}^{\mathbf{s}}_{k:k+n_k} = \arg\max_{\mathbf{x}^{\mathbf{t}}} p(\mathbf{x}^{\mathbf{t}}_{k+n_k} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}^{\mathbf{t}}_{1:k}, \tilde{\mathbf{x}}^{\mathbf{s}}_{1:k}) \tag{6.13}$$

If the reward is based on the belief of the the posterior distribution (6.11), then the reward value is $R_k \triangleq p(\mathbf{x}^{\mathbf{t}}_k = \hat{x}^{\mathbf{s}}_k \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}^{\mathbf{t}}_{1:k}, \tilde{\mathbf{x}}^{\mathbf{s}}_{1:k})$. Hence the return value G_k

$$G_k = p(\mathbf{x}^{\mathbf{t}}_k = \hat{x}^{\mathbf{s}}_k \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}^{\mathbf{t}}_{1:k}, \tilde{\mathbf{x}}^{\mathbf{s}}_{1:k}) + \gamma G_{k+1} \tag{6.14}$$

Then, the control input of the agent's motion can be replaced by action and behaviour defined in the previous chapter

$$\mathbf{x}^{\mathbf{a}}_{k+1} = f^{\mathbf{a}}(\mathbf{x}^{\mathbf{a}}_k, a_k, b_k) \tag{6.15}$$

with Markov probability state transition

$$\sum_{\mathbf{x}^{\mathbf{a}}_{k+1}} \sum_{r} p(\mathbf{x}^{\mathbf{a}}_{k+1}, r_k \mid \mathbf{x}^{\mathbf{a}}_k, a_k, b_k) = 1 \tag{6.16}$$

then, the update rule for behaviour-value function is:

$$H'(\mathbf{x}^{\mathbf{a}}_k, a_k, b_k) \leftarrow H(\mathbf{x}^{\mathbf{a}}_k, a_k, b_k) + \alpha \left[R_{k+1} + \gamma H(\mathbf{x}^{\mathbf{a}}_{k+1}, a_{k+1}, b_{k+1}) - H(\mathbf{x}^{\mathbf{a}}_k, a_k, b_k) \right] \tag{6.17}$$

hence, the update rule for action-value function is:

$$Q'(\mathbf{x}_k^{\mathbf{a}}, a_k) \longleftarrow Q(\mathbf{x}_k^{\mathbf{a}}, a_k) + \alpha \left[H'(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k) - Q(\mathbf{x}_k^{\mathbf{a}}, a_k) \right] \tag{6.18}$$

6.4.1 Implementation

From the formulation above, the complete procedure of recursive bayesian and the experience backup for epigenetic learning can be seen in Algorithm 6.1. After the experience has backed up, the epigenetic mechanisms is done as discussed in previous chapter.

Algorithm 6.1 Experience Backup for Recursive Bayesian Epigenetic Learning

for all k=1:N **do**
  $\mathbf{x}_k^{\mathbf{s}} \leftarrow$ agent take action $a_{k-1}^{\mathbf{s}}$ and behaviour $b_{k-1}^{\mathbf{s}}$
  ${}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}} \leftarrow$ agent observes target based on ${}^{\mathbf{s}}h_k^{\mathbf{t}}(\mathbf{x}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}}, {}^{\mathbf{s}}\mathbf{v}_k^{\mathbf{t}})$
  if ${}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}}$ is exist **then**
    $l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^t, \mathbf{x}_k^{\mathbf{s}}) = 1 - P_{\mathrm{d}}(\mathbf{x}_k^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{s}})$ ▷ Detected
  else
    $l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^t, \mathbf{x}_k^{\mathbf{s}}) = p({}^{\mathbf{s}}\mathbf{z}_k^t \mid \mathbf{x}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}})$ ▷ Not Detected
  end if
  $p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) \leftarrow \frac{l(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}}, \mathbf{x}_k^{\mathbf{s}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})}{p({}^{\mathbf{s}}\mathbf{z}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k-1}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})}$ ▷ Bayes update
  $p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) \leftarrow \int p(\mathbf{x}_{k+1}^{\mathbf{t}} \mid \mathbf{x}_k^{\mathbf{t}}) p(\mathbf{x}_k^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}}) d\mathbf{x}_k^{\mathbf{t}}$ ▷ Bayes prediction
  $\hat{\mathbf{x}}_{k:k+n_k}^{\mathbf{s}} = \arg\max_{\mathbf{x}^{\mathbf{t}}} p(\mathbf{x}_{k+n_k}^{\mathbf{t}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$ ▷ Path planning
  $R_k \leftarrow p(\mathbf{x}_k^{\mathbf{t}} = \hat{x}_k^{\mathbf{s}} \mid {}^{\mathbf{s}}\tilde{\mathbf{z}}_{1:k}^{\mathbf{t}}, \tilde{\mathbf{x}}_{1:k}^{\mathbf{s}})$ ▷ Current Reward
  for each g in b_k **do**
    $\mathsf{lh}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k, g) \leftarrow \mathsf{lh}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k, g) + \eta \left(R + \gamma \tilde{H}(\mathbf{x}_{k+1}^{\mathbf{a}}, a_{k+1}, b_{k+1}) - \tilde{H}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k) \right)$
  end for
  $\tilde{H}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k) \leftarrow \frac{1}{N} \sum_{n=1}^{N} \mathsf{lh}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k, g^n)$
  $Q(\mathbf{x}_k^{\mathbf{a}}, a_k) \longleftarrow Q(\mathbf{x}_k^{\mathbf{a}}, a_k) + \alpha \left[\tilde{H}(\mathbf{x}_k^{\mathbf{a}}, a_k, b_k) - Q(\mathbf{x}_k^{\mathbf{a}}, a_k) \right]$
  $a_{k-1}^{\mathbf{s}}, b_{k-1}^{\mathbf{s}} \leftarrow$ agent pick based on H and Q
end for

6.5 Summary

A Bayesian technique that enables autonomous path planning for a search problem is discussed in this chapter. Rather than define the search routine manually, it is possible to utilise the experience and the prior belief to act accordingly. It has been shown that the recursive Bayesian estimation (RBE) provides sufficient reward estimation in the form of the probabilistic distribution of the search map. Since the belief of targets' position changing every time an agent moves, the states, action, and behaviour value function are also changing dynamically. This gives a robust adaptation to the system.

The product of this approach is the belief distribution map, agent's state and action map, agent's action and behaviour map. Co-evolving settings from the previous chapter display the possibility to exchange and improve genetic information based on Epigenetic inheritance. This covers the state-action and action-behaviour decision making matrices. Furthermore, the joint distribution of the agent's belief can also be exchanged.

6.6 Conclusion

The concept of combining updating and predicting the search is to obtain an immediate reward for Epigenetic Swarm Learning is accomplished. The formulation shows that epigenetic learning aims to develop optimal behaviour in a dynamic state of the environment, and the Recursive Bayesian Estimation aims to predict the future state that gives immediate reward for the agent. One area that is of interest in applying the proposed method is how well the swarm can exchange their belief and their regulatory function through epigenetic inheritance. Thus, to see the real benefits of this approach, further research is needed.

Chapter 7

Conclusion and Future Works

In this thesis, the formulation and the implementation of evolutionary learning swarm robotics using epigenetic inheritance in the dynamic environment was investigated. This thesis's main contribution is to improve evolutionary swarm capability by proposing a novel learning strategy using epigenetic inheritance. There are two frameworks proposed in this thesis: (1) A reward-based epigenetic learning, and (2) A decentralised multi-agent reward-based epigenetic learning. In general, both algorithms are able to deduct the environmental dynamics by utilising external feedback or stimulus in the form of reward or reinforcement.

This thesis begins with an investigation and overview of what is the current proceeding of swarm robotics design. The challenges, along with necessary approaches to overcome the challenges, are presented. The discussion leads to the topic in proposing a novel way to apply adaptation to evolutionary swarm using the epigenetic concept. There are two formulations done in this thesis. The first is the inclusion of stimulus data as the basis for evolution for the evolutionary swarm and entirely abandoning fitness function. The investigation shows that the artificial epigenetic function is beneficial to represent environmental awareness in the evolution process. Moreover, by applying epigenetic inheritance, experience and knowledge can be passed down to the next progenies. The proposed method displays two activities, rewards accumulation and evolutionary improvement. A methylation process allows each agent to craft regulatory functions based on the rewards (co-learning), and the evolutionary improvement with histone-based operations provides a medium to exchange strategies for the swarm (co-evolution). The reward-based epigenetic algorithm opens a new possibility to improve swarm's adaptability utilising external stimulus from the environment.

The second is a decentralised epigenetic learning framework which is the extended reward-based epigenetic algorithm. The second attempt is done by adding experience backup using a temporal-different approach on top of the methylation process. In the formulation, the genetic structure is tailored to comply with a dynamic environment. The product of this

approach is the decision-making of selecting action and behaviour based on environmental state, namely action policy and behaviour policy. In this approach, the regulatory function activates genetic expression based on the selection of state and action pair. The results in Chapter 4 and 5 show that the extended method can overcome the dynamic environment to prove that the epigenetic concept is beneficial to advance evolutionary swarm's adaptability. Thus, the epigenetic learning framework proceeds a new possibility to solve swarm robotics tasks with high dynamic constraints through epigenetic concept.

There are several directions to take in the future to investigate and improve the capability further. As been discussed in Chapter 6, the proposed methods can be extended with another mechanism, Bayes filter. The core concept of artificial epigenetic learning and inheritance are flexible to be extended with other state recognition methods, such as artificial neural networks, fuzzy logic and Kalman filter. For the proposed method, the simulated environment of this research is contained in one machine and program. Thus, the cyclic process of the computation is constrained with limited threads at the same random pooling. A viable option is to investigate each gene pool and chromosome pool separately by deploying the decentralised algorithm to a cluster of machines/CPU. This will open up more understanding of how the epigenetic layer is communicating with the rest of the group. It is also essential to establish what is the best data framework to transmit meaningful data between agents. Since temporal difference learning was chosen as the reinforcement learning method, it is possible to apply a prediction N-step ahead by applying the Monte Carlo method and backed up the experience to N-step backwards, known as TD-λ. The implementation for a group of robots or homogeneous system requires further study on how this approach would perform in a real situation.

Appendices

Appendix A

Program

Main.m

```
clear
clc


% opengl hardwarebasic

import ReBEL.ReBEL;
import testFunction.Sphere;
import testFunction.Levy;
import testFunction.Ackley;

genotypes = ["x1","x2"];

states = ["s1"];
actions = ["a1"];

nbits = 8;
minValue = -10;
maxValue = 10;
% minValue = -32.768;
% maxValue = 32.768;


agentsPopulation = 5;
genePopulation = 10;
chromosomePopulation = 5;
egreedyPolicy = 0.5;
methylationRate = 0.1;
epsilonSelection = 0.1;
imprintingRate = 0.1;
mutationRate = 0.1;
regenerationRate = 0.1;
silenceRate = 0.1;

```

```
span = 0:0.05:1;
[X,Y] = meshgrid(span,span);
optimalGeneration = zeros(length(span),length(span));
datax = 0;
for regenerationRate = span
    datax = datax + 1;
    datay = 0;
    for egreedyPolicy = span
        datay = datay + 1;
        rebel = ReBEL(agentsPopulation,genePopulation,chromosomePopulation,
    genotypes,states,actions,methylationRate,nbits,minValue,maxValue,
    egreedyPolicy,epsilonSelection,imprintingRate,mutationRate,
    regenerationRate,silenceRate);

        trial = 20;
        sphere = Sphere(genotypes,minValue,maxValue,nbits,1);
        rebel.setFunction(sphere,trial,"min");

    %     levy = Levy(genotypes,minValue,maxValue,nbits,1);
    %     rebel.setFunction(levy,trial,"min");

        % ackley = Ackley(genotypes,minValue,maxValue,nbits,1);
        % rebel.setFunction(ackley,20,"min");


        figure(2);
        clf
        legendString = strings(agentsPopulation,1) + "agent-" + (1:1:
    agentsPopulation)';
        % errorData = plot(zeros(generation,agentsPopulation));
        generationData = line(1,zeros(1,agentsPopulation),'LineWidth',1);
        % trialData = line(1,zeros(1,agentsPopulation),'LineStyle',':','
    LineWidth',0.001);
        titleText = title(sprintf("generation - %d",0));
        legend(legendString);
        set(gca,'Yscale','log')

        i = 0;
    %     for i=1:generation
        while true
            i = i + 1;
            fprintf(sprintf("generation - %d, greedy policy = %f",i,
    egreedyPolicy));
            titleText.String = sprintf("generation - %d",i);
            [valueGeneration,valueTrial,isOptimal] = rebel.evaluate();
            rebel.evolve();
            for j=1:agentsPopulation
        %         initialStamp = (trial*(i-1) + 1);
        %         generationData(j).XData(end:end+trial-1) = initialStamp:(
    initialStamp+trial-1);
        %         generationData(j).YData(end:end+trial-1) =
    valueGeneration(j)*ones(1,trial);
```

```
79          %
80          %           trialData(j).XData(end:end+trial-1) = initialStamp:(
    initialStamp+trial-1);
81          %           trialData(j).YData(end:end+trial-1) = valueTrial(j,:);
82                      generationData(j).XData(i) = i;
83                      generationData(j).YData(i) = valueGeneration(j);
84          %             trialData(j).XData(i) = i;
85          %             trialData(j).YData(i) = mean(valueTrial(j,:));
86
87              end
88  %             drawnow;
89              clc
90          %     rebel.printResult
91              disp(isOptimal)
92              if isOptimal
93                  disp("all optimal")
94                  currentGeneration = i;
95                  break
96              end
97
98              if (i>=500)
99                  disp("all optimal")
100                 currentGeneration = i;
101                 break
102             end
103             currentGeneration = i;
104         end
105         optimalGeneration(datax,datay) = currentGeneration;
106         figure(3)
107         clf
108         surf(X,Y,optimalGeneration)
109     end
110 end
111
112 rebel.printResult
```

+ReBEL/Agent.m

```
1  classdef Agent < handle
2
3      properties
4          genepool
5          chromosomepool
6          evolution
7          evaluation
8      end
9
10     methods
11         function this = Agent(genePopulation,chromosomePopulation,genotypes
       ,states,actions,methylationRate,nbits,min,max,egreedyPolicy,
       epsilonSelection,imprintingRate,mutationRate,regenerationRate,
       silenceRate)
12             import ReBEL.geneticStructure.GenePool;
```

```
            import ReBEL.chromosomeStructure.ChromosomePool;
            import ReBEL.Evolution;
            import ReBEL.Evaluation;

            if nargin > 0
                this.genepool = GenePool(genePopulation,genotypes,states,
    actions,methylationRate,nbits,min,max);
                this.chromosomepool = ChromosomePool(this.genepool,
    chromosomePopulation,states,actions);
                this.evolution = Evolution(epsilonSelection,imprintingRate,
    mutationRate,regenerationRate,silenceRate);
                this.evaluation = Evaluation(egreedyPolicy);
            end
        end

        function [chromosomeValues,value] = evaluate(this,func,mode)
            [chromosomeValues,value] = this.evaluation.evaluate(this.
    chromosomepool,func,mode);
        end

        function evolve(this,otherChromosomes)
            this.evolution.evolve(this.genepool,this.chromosomepool,
    otherChromosomes);
        end

        function [chromosomes,status] = getChromosomes(this)
            [chromosomes,status] = this.evolution.selection(this.
    chromosomepool);
        end
    end
end
```

+ReBEL/Evaluation.m

```
classdef Evaluation < handle
    %EVALUATION Summary of this class goes here
    %   Detailed explanation goes here

    properties
        egreedyPolicy
        currentChromosome
        currentState
        currentAction
        currentIndex
        bestValue
    end

    methods
        function this = Evaluation(egreedyPolicy)
            this.egreedyPolicy = egreedyPolicy;
        end

        function [chromosomeValues,value] = evaluate(this,chromosomepool,
```

```
    func,mode)
20          [chromosomeValues,histone,index,status] = this.getChromosome(
    chromosomepool,"s1","a1");
21          %calculate reward
22          value = func.calculate(chromosomeValues);
23          if mode == "min"
24              reward = minMode(this,value);
25          elseif mode == "max"
26              reward = maxMode(this,value);
27          end
28
29          this.setReward(reward);
30      end
31
32      function [chromosomeValues,histone,index,status] = getChromosome(
    this,chromosomepool,state,action)
33          randomNumber = rand();
34          this.currentState = state;
35          this.currentAction = action;
36          if randomNumber < (1 - this.egreedyPolicy)
37              status = "exploit";
38              [this.currentChromosome,this.currentIndex] = chromosomepool
    .exploit(state,action);
39          else
40              status = "explore";
41              [this.currentChromosome,this.currentIndex] = chromosomepool
    .explore(state,action);
42          end
43
44          chromosomeValues = this.currentChromosome.values;
45          histone = this.currentChromosome.histones();
46  %         histone = this.currentChromosome.histones(state,action);
47          index = this.currentIndex;
48      end
49
50      function reward = minMode(this,value)
51          if isempty(this.bestValue)
52              this.bestValue = value;
53              reward = 1;
54          elseif this.bestValue == value
55              reward = 1;
56          else
57              reward = 0;
58              reward = this.relu(this.bestValue - value);
59  %             reward = tanh(this.bestValue - value);
60              if reward > 0
61                  this.bestValue = value;
62              end
63          end
64      end
65
66      function reward = maxMode(this,value)
```

```
67              if isempty(this.bestValue)
68                  this.bestValue = value;
69                  reward = 1;
70              elseif this.bestValue == value
71                  reward = 1;
72              else
73                  reward = 0;
74                  reward = this.relu(value-this.bestValue);
75  %                 reward = tanh(value-this.bestValue);
76                  if reward > 0
77                      this.bestValue = value;
78                  end
79              end
80          end
81
82          function setReward(this,returnValue)
83              this.currentChromosome.methylation(this.currentState,this.
    currentAction,returnValue);
84          end
85
86          function output = relu(this,value)
87
88              output = max(0,value);
89          end
90      end
91  end
```

+ReBEL/Evolution.m

```
1   classdef Evolution < handle
2       %EVOLUTION Summary of this class goes here
3       %   Detailed explanation goes here
4
5       properties
6           epsilonSelection
7           imprintingRate
8           mutationRate
9           regenerationRate
10          silenceRate
11          selectionChromosomes
12      end
13
14      methods
15          function this = Evolution(epsilonSelection,imprintingRate,
    mutationRate,regenerationRate,silenceRate)
16              this.epsilonSelection = epsilonSelection;
17              this.imprintingRate = imprintingRate;
18              this.mutationRate = mutationRate;
19              this.regenerationRate = regenerationRate;
20              this.silenceRate = silenceRate;
21          end
22
23          function [chromosomes,status] = selection(this,chromosomepool)
```

```
24              [chromosomes,status] = ReBEL.evolutionaryOperator.
    histoneBasedSelection(chromosomepool,this.epsilonSelection);
25              this.selectionChromosomes = chromosomes;
26          end
27
28          function evolve(this,genepool,chromosomepool,otherChromosomes)
29              %% selection
30
31              if rand < this.regenerationRate
32                  % from this agent
33  %                 chromosomes1 = ReBEL.evolutionaryOperator.
    histoneBasedSelection(chromosomepool,this.epsilonSelection);
34                  chromosomes1 = this.selectionChromosomes;
35                  % from other agent
36                  chromosomes2 = ReBEL.evolutionaryOperator.
    proportionalTournament(otherChromosomes);
37
38                  % crossover
39                  [newchromosomes1,newchromosomes2] = ReBEL.
    evolutionaryOperator.histoneBasedCrossover(chromosomes1,chromosomes2);
40                  % genomic imprinting
41                  [imprintedChildChromosomes1,imprintedChildChromosomes2] =
    ReBEL.evolutionaryOperator.genomicImprinting(newchromosomes1,
    newchromosomes2,this.imprintingRate);
42                  % mutation
43                  mutatedChildChromosomes1 = ReBEL.evolutionaryOperator.
    mutation(imprintedChildChromosomes1,this.mutationRate);
44                  mutatedChildChromosomes2 = ReBEL.evolutionaryOperator.
    mutation(imprintedChildChromosomes2,this.mutationRate);
45              else
46                  chromosomes1 = chromosomepool.exploreGenepool(genepool);
47                  chromosomes2 = chromosomepool.exploreGenepool(genepool);
48                  mutatedChildChromosomes1 = ReBEL.evolutionaryOperator.
    mutation(chromosomes1,this.mutationRate);
49                  mutatedChildChromosomes2 = ReBEL.evolutionaryOperator.
    mutation(chromosomes2,this.mutationRate);
50              end
51                  %regeneration
52                  ReBEL.evolutionaryOperator.regeneration(genepool,
    chromosomepool,mutatedChildChromosomes1,mutatedChildChromosomes2);
53                  % cek integration
54                  ReBEL.evolutionaryOperator.cekIntegration(chromosomepool,
    genepool);
55                  % gene silencing
56                  ReBEL.evolutionaryOperator.geneSilencing(chromosomepool,
    genepool,this.silenceRate);
57                  ReBEL.evolutionaryOperator.geneDeletion(chromosomepool,
    genepool);
58                  % cek integration
59                  ReBEL.evolutionaryOperator.cekIntegration(chromosomepool,
    genepool);
60
```

```
        end
    end
end
```

+ReBEL/ReBEL.m

```
classdef ReBEL < handle
    %REBEL Summary of this class goes here
    %   Detailed explanation goes here

    properties
        agents
        evolution
        evaluation
        func
        trial
        mode
        logisticFunction
    end

    methods
        function this = ReBEL(agentsPopulation,genePopulation,
    chromosomePopulation,genotypes,states,actions,methylationRate,nbits,min
    ,max,egreedyPolicy,epsilonSelection,imprintingRate,mutationRate,
    regenerationRate,silenceRate)
            import ReBEL.Agent;
            this.agents = Agent.empty(agentsPopulation,0);
            for i = 1:agentsPopulation
                this.agents(i,1) = Agent(genePopulation,
    chromosomePopulation,genotypes,states,actions,methylationRate,nbits,min
    ,max,egreedyPolicy,epsilonSelection,imprintingRate,mutationRate,
    regenerationRate,silenceRate);
            end
        end

        function setFunction(this,func,trial,mode)
            this.func = func;
            this.trial = trial;
            this.mode = mode;
        end

        function reset(self,agentsPopulation,genePopulation,
    chromosomePopulation,genotypes,states,actions,methylationRate,nbits,min
    ,max,egreedyPolicy,epsilonSelection,imprintingRate,mutationRate,
    regenerationRate,silenceRate)
            import ReBEL.Agent;
            this.agents = Agent.empty(agentsPopulation,0);
            for i = 1:agentsPopulation
                this.agents(i,1) = Agent(genePopulation,
    chromosomePopulation,genotypes,states,actions,methylationRate,nbits,min
    ,max,egreedyPolicy,epsilonSelection,imprintingRate,mutationRate,
    regenerationRate,silenceRate);
            end
```

```
36        end
37
38        function [valueGeneration,valueTrial,isOptimal] = evaluate(this)
39
40            valueTrial = zeros(length(this.agents),this.trial);
41            for i = 1:this.trial
42                for j = 1 : length(this.agents)
43                    [chromosomeValues,value] = this.agents(j).evaluate(this
    .func,this.mode);
44                    this.func.draw(j,chromosomeValues);
45                    valueTrial(j,i) = value;
46                end
47            end
48
49
50            valueGeneration = zeros(length(this.agents),1);
51            isOptimal = false(length(this.agents),1);
52            for i=1:length(this.agents)
53                valueGeneration(i)= (this.agents(i).evaluation.bestValue);
54                if valueGeneration(i) == this.func.optimalValue
55                    isOptimal(i) = true;
56                end
57            end
58        end
59
60        function printResult(this)
61            str_data = string();
62            for i=1:length(this.agents)
63                for j=1:length(this.agents(i).chromosomepool.chromosomeSet.
    chromosomes)
64                    str_data = str_data + sprintf("%d.%02d. ",i,j);
65                    values = this.agents(i).chromosomepool.chromosomeSet.
    chromosomes(j).values;
66                    for k = 1:length(values)
67                        str_data = str_data + sprintf("%+.3f ",values(k));
68                    end
69                    str_data = str_data + sprintf(" %+.2f\t",this.agents(i)
    .chromosomepool.chromosomeSet.chromosomes(j).averageHistone);
70
71                    for k =1:length(this.agents(i).chromosomepool.
    chromosomeSet.chromosomes(j).genes)
72                        str_data = str_data + sprintf("%s ",this.agents(i).
    chromosomepool.chromosomeSet.chromosomes(j).genes(k).getCode());
73                    end
74                    if this.func.calculate(values) == this.func.
    optimalValue
75                        str_data = str_data + sprintf(" Optimum");
76                    end
77                    str_data = str_data + newline;
78                end
79            end
80            fprintf(str_data);
```

```
81          end
82
83          function evolve(this)
84              import ReBEL.chromosomeStructure.Chromosome;
85              statesNum = this.agents(1).chromosomepool.getStatesNum();
86              actionsNum = this.agents(1).chromosomepool.getActionsNum();
87              swarmChromosomes = Chromosome.empty(statesNum,actionsNum,0);
88              status = string.empty(statesNum,actionsNum,0);
89
90              for i = 1 : length(this.agents)
91                  [swarmChromosomes(:,:,i),status(:,:,i)] = this.agents(i).
    getChromosomes();
92              end
93
94              for i = 1 : length(this.agents)
95                  status = ismember(this.agents,this.agents(i));
96                  this.agents(i).evolve(swarmChromosomes(:,:,~status));
97              end
98          end
99      end
100 end
```

+ReBEL/+chromosomeStructure/Chromosome.m

```
1  classdef Chromosome < handle & matlab.mixin.Copyable & matlab.mixin.
       CustomDisplay
2
3      properties (Access = private)
4          geneMember
5          chromosomeLength
6          state
7          action
8      end
9
10     methods (Static)
11         function chromosome = explore(genepool,state,action)
12             import ReBEL.chromosomeStructure.Chromosome;
13             import ReBEL.geneticStructure.Gene;
14             geneMember = Gene.empty(0,0);
15             chromosomeLength = length(genepool.genefamilies);
16             genotypes = genepool.genotypes;
17             for i=1:chromosomeLength
18                 randomIndex = randi(length(genepool.genefamilies(genotypes(
       i)).genes));
19                 geneMember(i) = genepool.genefamilies(genotypes(i)).genes(
       randomIndex);
20             end
21
22             chromosome = Chromosome(geneMember,state,action);
23         end
24
25         function chromosome = exploit(genepool)
26
```

```
27          end
28      end
29
30      methods
31          function this = Chromosome(genes,state,action)
32              if nargin > 0
33                  this.geneMember = genes;
34                  this.chromosomeLength = length(genes);
35                  this.state = state;
36                  this.action = action;
37              end
38          end
39
40          function exploreGene(this,index,genepool)
41              randomIndex = randi(length(genepool.genefamilies(this.
    geneMember(index).genotype).genes));
42              randomGene = genepool.genefamilies(this.geneMember(index).
    genotype).genes(randomIndex);
43              while (this.geneMember(index) == randomGene)
44                  randomIndex = randi(length(genepool.genefamilies(this.
    geneMember(index).genotype).genes));
45                  randomGene = genepool.genefamilies(this.geneMember(index).
    genotype).genes(randomIndex);
46              end
47              this.geneMember(index) = genepool.genefamilies(this.geneMember(
    index).genotype).genes(randomIndex);
48          end
49
50          function genotypes = genotypes(this)
51              genotypes = string.empty(0,0);
52              for i = 1:this.chromosomeLength
53                  genotypes(i) = this.geneMember(i).genotype;
54              end
55          end
56
57          function genes = genes(this,index)
58              if nargin > 1
59                  genes = this.geneMember(index);
60              else
61                  genes = this.geneMember;
62              end
63          end
64
65          function setGene(this,index,genes)
66              this.geneMember(index) = genes;
67          end
68
69          function action = getAction(this)
70              action = this.action;
71          end
72
73          function state = getState(this)
```

```
74              state = this.state;
75          end
76
77          function values = values(this,genotype)
78              values = double.empty(0,this.chromosomeLength);
79              if nargin <=1
80                  for i=1:this.chromosomeLength
81                      values(i) = this.geneMember(i).value;
82                  end
83              else
84                  [status,index] = ismember(genotype,this.genotypes);
85                  if status
86                      values = this.geneMember(index).value;
87                  else
88                      error("genotype not found in this chromosome");
89                  end
90              end
91          end
92
93          function histones = histones(this)
94              histones = double.empty(0);
95              for i=1:this.chromosomeLength
96                  histones(i) = this.geneMember(i).histone(this.state,this.
    action);
97              end
98          end
99
100         function histone = averageHistone(this)
101             histone = mean(this.histones());
102         end
103
104 %         function histones = histones(this,state,action)
105 %             [stateNum,actionNum] = size(this.geneMember(1).histone);
106 %             if nargin <= 1
107 %                 if stateNum*actionNum ~= 1
108 %                     histones = double.empty(stateNum,actionNum,0);
109 %                     for i=1:this.chromosomeLength
110 %                         histones(:,:,i) = this.geneMember(i).histone;
111 %                     end
112 %                 elseif nargin <= 1 && stateNum*actionNum == 1
113 %                     histones = double.empty(0,0);
114 %                     for i=1:this.chromosomeLength
115 %                         histones(i) = this.geneMember(i).histone;
116 %                     end
117 %                 end
118 %             else
119 %                 histones = double.empty(0,0);
120 %                 for i=1:this.chromosomeLength
121 %                     histones(i) = this.geneMember(i).histone(state,action
    );
122 %                 end
123 %             end
```

```
124 %         end
125
126 %         function histone = averageHistone(this,state,action)
127 %             if nargin <= 1
128 %                 histone = mean(histones(this),3);
129 %             else
130 %                 histone = mean(histones(this,state,action));
131 %             end
132 %         end
133
134         function methylation(this,state,action,returnValue)
135             averageHistoneAtSA = this.averageHistone();
136 %             averageHistoneAtSA = averageHistone(this,state,action);
137             for i=1:this.chromosomeLength
138                 this.geneMember(i).methylation(state,action,this.
    chromosomeLength,averageHistoneAtSA,returnValue);
139             end
140         end
141
142         function chromosome = mutate(this,k,mutationRate)
143             this.geneMember(k).mutate
144         end
145     end
146
147     methods (Access = protected)
148         function copyThis = copyElement(this)
149             copyThis = copyElement@matlab.mixin.Copyable(this);
150             copyThis.geneMember = copy(this.geneMember);
151         end
152
153         function propgrp = getPropertyGroups(this)
154             if ~isscalar(this)
155                 propgrp = getPropertyGroups@matlab.mixin.CustomDisplay(this
    );
156             else
157                 propList = struct(...
158                     'genes',this.geneMember,...
159                     'histones',this.histones,...
160                     'averageHistones',this.averageHistone,...
161                     'state',this.state,...
162                     'action',this.action);
163                 propgrp = matlab.mixin.util.PropertyGroup(propList);
164             end
165         end
166     end
167 end
```

+ReBEL/+chromosomeStructure/ChromosomePool.m

```
1 classdef ChromosomePool < handle & matlab.mixin.CustomDisplay
2     %CHROMOSOMEPOOL Summary of this class goes here
3     %   Constructor: ChromosomePool(genepool,chromosomePopulation,states,
    actions)
```

```
4       %   Getter:
5       %       - chromosomeSet(stateName, actionName)
6       %       - chromosomeSet()
7       properties (Access = private)
8           chromosomeSetMember
9           states
10          actions
11      end
12
13      methods
14          function this = ChromosomePool(genepool,chromosomePopulation,states
    ,actions)
15              import ReBEL.chromosomeStructure.ChromosomeSet;
16              this.actions = actions;
17              this.states = states;
18              stateNum = length(states);
19              actionNum = length(actions);
20              this.chromosomeSetMember = ChromosomeSet.empty(stateNum,
    actionNum,0);
21              for i = 1:stateNum
22                  for j = 1:actionNum
23                      this.chromosomeSetMember(i,j,1) = ChromosomeSet(
    genepool,chromosomePopulation,states(i),actions(j));
24                  end
25              end
26          end
27
28          function chromosomeset = chromosomeSet(this,state,action)
29              if nargin <= 1
30                  chromosomeset = this.chromosomeSetMember;
31              else
32                  [status,indexState] = ismember(state,this.states);
33                  if ~status
34                      error("state not found");
35                  end
36                  [status,indexAction] = ismember(action,this.actions);
37                  if ~status
38                      error("state not found");
39                  end
40                  chromosomeset = this.chromosomeSetMember(indexState,
    indexAction);
41              end
42          end
43
44          function chromosomes = exploreGenepool(this,genepool)
45              import ReBEL.chromosomeStructure.Chromosome;
46              statesNum = length(this.states);
47              actionsNum = length(this.actions);
48              chromosomes = Chromosome.empty(statesNum,actionsNum,0);
49              for i = 1:statesNum
50                  for j = 1:actionsNum
51                      chromosomes(i,j,1) = copy(Chromosome.explore(genepool,
```

```
    this.states(i),this.actions(j)));
52              end
53          end
54      end
55
56      function [chromosome,index] = exploit(this,state,action)
57          [chromosome,index] = this.chromosomeSet(state,action).exploit()
    ;
58      end
59
60      function [chromosome,index] = explore(this,state,action)
61          [chromosome,index] = this.chromosomeSet(state,action).explore()
    ;
62      end
63
64      function genes = getGenes(this,index)
65          import ReBEL.geneticStructure.Gene;
66          statesNum = length(this.states);
67          actionsNum = length(this.actions);
68          genes = Gene.empty(statesNum,actionsNum,0);
69          for i=1:statesNum
70              for j=1:actionsNum
71                  for k=1:length(this.chromosomeSetMember(i,j).
    chromosomes)
72                      genes(i,j,k) = this.chromosomeSetMember(i,j).
    chromosomes(k).genes(index);
73                  end
74              end
75          end
76          genes = squeeze(genes)';
77      end
78
79      function values = getStatesNum(this)
80          values = length(this.states);
81      end
82
83      function values = getActionsNum(this)
84          values = length(this.actions);
85      end
86
87      function values = getStates(this)
88          values = this.states;
89      end
90
91      function values = getActions(this)
92          values = this.actions;
93      end
94  end
95
96  methods (Access = protected)
97      function propgrp = getPropertyGroups(this)
98          if ~isscalar(this)
```

```
                propgrp = getPropertyGroups@matlab.mixin.CustomDisplay(this
    );
            else
                propList = struct(...
                    'chromosomeSet',this.chromosomeSetMember,...
                    'states',this.states,...
                    'actions',this.actions);
                propgrp = matlab.mixin.util.PropertyGroup(propList);
            end
        end
    end
end
```

+ReBEL/+chromosomeStructure/ChromosomeSet.m

```
classdef ChromosomeSet < handle & matlab.mixin.CustomDisplay
    %CHROMOSOMESET Summary of this class goes here
    %   Detailed explanation goes here
    %   Constructor: ChromosomeSet(genepool,chromosomePopulation,state,
    action)
    %   Getter:
    %       - histones(index)
    %       - histones()
    %       - chromosomes(index)
    %       - chromosomes()
    %       - exploit()
    %       - explore()

    properties (Access = private)
        chromosomeMember
        action
        state
    end

    methods
        function this = ChromosomeSet(genepool,chromosomePopulation,state,
    action)
            import ReBEL.chromosomeStructure.Chromosome;
            if nargin > 0
                this.chromosomeMember = Chromosome.empty(0,0);
                this.state = state;
                this.action = action;
                for i = 1:chromosomePopulation
                    this.chromosomeMember(i) = Chromosome.explore(genepool,
    state,action);
                end
            end
        end

        function histones = histones(this,index)
            if nargin > 1
                histones = this.chromosomeMember(index).averageHistone();
%                 histones = this.chromosomeMember(index).averageHistone(
```

```
    this.state,this.action);
36              else
37                  chromosomeLength = length(this.chromosomeMember);
38                  histones = double.empty(0,chromosomeLength);
39                  for i=1:chromosomeLength
40                      histones(i) = this.chromosomeMember(i).averageHistone()
    ;
41  %                     histones(i) = this.chromosomeMember(i).averageHistone
    (this.state,this.action);
42                  end
43              end
44          end
45
46          function chromosome = chromosomes(this,index)
47              if nargin > 1
48                  chromosome = this.chromosomeMember(index);
49              else
50                  chromosome = this.chromosomeMember;
51              end
52          end
53
54          function [chromosome,index] = exploit(this)
55              [~,index] = max(this.histones);
56              chromosome = this.chromosomeMember(index);
57          end
58
59          function [chromosome,index] = explore(this)
60              chromosomeLength = length(this.chromosomeMember);
61              index = randi(chromosomeLength);
62              chromosome = this.chromosomeMember(index);
63          end
64
65          function methylation(this,index,returnValue)
66              this.chromosomeMember(index).methylation(this.state,this.action
    ,returnValue);
67          end
68
69          function setChromosome(this,index,chromosome)
70              this.chromosomeMember(index) = chromosome;
71          end
72      end
73
74      methods (Access = protected)
75          function propgrp = getPropertyGroups(this)
76              if ~isscalar(this)
77                  propgrp = getPropertyGroups@matlab.mixin.CustomDisplay(this
    );
78              else
79                  propList = struct(...
80                      'chromosomes',this.chromosomeMember,...
81                      'histones',this.histones,...
82                      'state',this.state,...
```

```
                    'action',this.action);
                propgrp = matlab.mixin.util.PropertyGroup(propList);
            end
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Crossover.m

```
classdef Crossover
    %CROSSOVER Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = Crossover(inputArg1,inputArg2)
            %CROSSOVER Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Evolver.m

```
classdef Evolver < handle
    %EVOLVE Summary of this class goes here
    %   Detailed explanation goes here

    properties
       epsilon_selection
       eta_genomic_imprinting
       mu_genetic_mutation
       epsilon_regeneration
    end

    methods
        function this = Evolver(e_selection,eta_imprint,mu_mutation,
    e_regeneration)

        end
    end
end
```

+ReBEL/+epigeneticMechanisms/GeneSilencing.m

```
classdef GeneSilencing
    %GENESILENCING Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = GeneSilencing(inputArg1,inputArg2)
            %GENESILENCING Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/GenomicImprinting.m

```
classdef GenomicImprinting
    %GENOMICIMPRINTING Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = GenomicImprinting(inputArg1,inputArg2)
            %GENOMICIMPRINTING Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Mutation.m

```
classdef untitled
```

```
    %UNTITLED Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = untitled(inputArg1,inputArg2)
            %UNTITLED Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Paramutation.m

```
classdef untitled
    %UNTITLED Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = untitled(inputArg1,inputArg2)
            %UNTITLED Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Regeneration.m

```
classdef Regeneration
    %REGENERATION Summary of this class goes here
    %   Detailed explanation goes here

```

```
    properties
        Property1
    end

    methods
        function obj = Regeneration(inputArg1,inputArg2)
            %REGENERATION Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+epigeneticMechanisms/Selection.m

```
classdef Selection
    %SELECTION Summary of this class goes here
    %   Detailed explanation goes here

    properties
        Property1
    end

    methods
        function obj = Selection(inputArg1,inputArg2)
            %SELECTION Construct an instance of this class
            %   Detailed explanation goes here
            obj.Property1 = inputArg1 + inputArg2;
        end

        function outputArg = method1(obj,inputArg)
            %METHOD1 Summary of this method goes here
            %   Detailed explanation goes here
            outputArg = obj.Property1 + inputArg;
        end
    end
end
```

+ReBEL/+evolutionaryOperator/cekIntegration.m

```
function cekIntegration(chromosomepool,genepool)
%CEKINTEGRATION Summary of this function goes here
%   Detailed explanation goes here
states = chromosomepool.getStates;
actions = chromosomepool.getActions;
statesNum = chromosomepool.getStatesNum;
actionsNum = chromosomepool.getActionsNum;
```

```
8
9  for i=1:statesNum
10     for j=1:actionsNum
11         for k=1:length(chromosomepool.chromosomeSet(states(i),actions(j)).
       chromosomes)
12             for l=1:length(chromosomepool.chromosomeSet(states(i),actions(j
       )).chromosomes(k).genes)
13                 genotype = chromosomepool.chromosomeSet(states(i),actions(j
       )).chromosomes(k).genes(l).genotype;
14                 status = ismember(chromosomepool.chromosomeSet(states(i),
       actions(j)).chromosomes(k).genes(l),...
15                                     genepool.genefamilies(genotype).genes);
16                 if ~status
17                     error("integrity failed");
18                 end
19             end
20         end
21     end
22 end
23 end
```

+ReBEL/+evolutionaryOperator/geneDeletion.m

```
1  function geneDeletion(chromosomepool,genepool)
2  genefamilies = genepool.genefamilies;
3
4  for i=1:length(genefamilies)
5      if length(genefamilies(i).genes) > 20
6          genes = chromosomepool.getGenes(i);
7          [status,index] = ismember(genefamilies(i).genes,genes);
8          [value,leastIndex] = min(status);
9
10         if ~value
11             genefamilies(i).deleteGene(leastIndex);
12         end
13     end
14 end
15
16 end
```

+ReBEL/+evolutionaryOperator/geneSilencing.m

```
1  function geneSilencing(chromosomepool,genepool,silenceRate)
2  states = chromosomepool.getStates;
3  actions = chromosomepool.getActions;
4  statesNum = chromosomepool.getStatesNum;
5  actionsNum = chromosomepool.getActionsNum;
6
7  genefamilies = genepool.genefamilies;
8
9  for i=1:length(genefamilies)
10     for j=1:statesNum
11         for k=1:actionsNum
```

```
12              [~,indexMin] = min(genefamilies(i).histones(states(j),actions(k
    )));
13              geneMin = genefamilies(i).genes(indexMin);
14
15              for l = 1:length(chromosomepool.chromosomeSet(states(j),actions
    (k)).chromosomes)
16                  geneChromosome = chromosomepool.chromosomeSet(states(j),
    actions(k)).chromosomes(l).genes(i);
17
18                  if (geneMin == geneChromosome) && (rand < silenceRate)
19                      % explore new genes
20                      chromosomepool.chromosomeSet(states(j),actions(k)).
    chromosomes(l).exploreGene(i,genepool)
21                      if geneMin == chromosomepool.chromosomeSet(states(j),
    actions(k)).chromosomes(l).genes(i)
22                          disp("wrong");
23                      end
24                  end
25              end
26          end
27      end
28  end
29
30
31
32  % for i=1:statesNum
33  %     for j=1:actionsNum
34  %         for k=1:length(chromosomepool.chromosomeSet(states(i),actions(j))
    .chromosomes)
35  %             for l=1:length(chromosomepool.chromosomeSet(states(i),actions
    (j)).chromosomes(k).genes)
36  %                 gene = chromosomepool.chromosomeSet(states(i),actions(j))
    .chromosomes(k).genes(l);
37  %                 [~,indexMin] = min(genepool.genefamilies(gene.genotype).
    histones(states(i),actions(i)));
38  %                 [status,indexLocation] = ismember(gene,genepool.
    genefamilies(gene.genotype).genes);
39  %                 if ~status
40  %                     error("integrity failed");
41  %                 end
42  %                 if indexMin == indexLocation
43  %                     gene.silence(states(i),actions(j));
44  %                     % explore new genes
45  %                     chromosomepool.chromosomeSet(states(i),actions(j)).
    chromosomes(k).exploreGene(l,genepool)
46  %                 else
47  %                     gene.activate(states(i),actions(j));
48  %                 end
49  %             end
50  %         end
51  %     end
52  % end
```

```
53
54 end
```

+ReBEL/+evolutionaryOperator/genomicImprinting.m

```
1  function [imprintedChildChromosomes1,imprintedChildChromosomes2] =
       genomicImprinting(childchromosomes1,childchromosomes2,imprintingRate)
2      geneLength = length(childchromosomes1(1,1).genes);
3      imprintedChildChromosomes1 = copy(childchromosomes1);
4      imprintedChildChromosomes2 = copy(childchromosomes2);
5      [stateNum,actionNum] = size(childchromosomes1);
6      for i = 1:stateNum
7          for j = 1:actionNum
8              for k = 1:geneLength
9                  randomNumber = rand;
10                 if randomNumber < imprintingRate
11                     imprintedChildChromosomes1(i,j).setGene(k,
       childchromosomes2(i,j).genes(k));
12                     imprintedChildChromosomes2(i,j).setGene(k,
       childchromosomes1(i,j).genes(k));
13             end
14         end
15     end
16     end
17 end
```

+ReBEL/+evolutionaryOperator/histoneBasedCrossover.m

```
1  function [childChromosomes1,childChromosomes2] = histoneBasedCrossover(
       chromosomes1,chromosomes2)
2      parentChromosomes1 = copy(chromosomes1);
3      parentChromosomes2 = copy(chromosomes2);
4      childChromosomes1 = copy(parentChromosomes1);
5      childChromosomes2 = copy(parentChromosomes2);
6      [statenum,actionnum] = size(parentChromosomes1);
7
8      for i=1:statenum
9          for j = 1:actionnum
10             histone1 = parentChromosomes1(i,j).histones;
11             histone2 = parentChromosomes2(i,j).histones;
12             averageHistone1 = parentChromosomes1(i,j).averageHistone;
13             averageHistone2 = parentChromosomes2(i,j).averageHistone;
14
15             booleanHistone1 = averageHistone1 <= histone1;
16             booleanHistone2 = averageHistone2 <= histone2;
17
18             maskBooleanHistone = booleanHistone1 | booleanHistone2;
19
20             childChromosomes1(i,j).setGene(maskBooleanHistone,
       parentChromosomes2(i,j).genes(maskBooleanHistone));
21             childChromosomes2(i,j).setGene(maskBooleanHistone,
       parentChromosomes1(i,j).genes(maskBooleanHistone));
22         end
```

```
    end
end
```

+ReBEL/+evolutionaryOperator/histoneBasedSelection.m

```
function [chromosomes,status] = histoneBasedSelection(chromosomepool,
    epsilonSelection)

    statesNum = chromosomepool.getStatesNum();
    actionsNum = chromosomepool.getActionsNum();

    states = chromosomepool.getStates();
    actions = chromosomepool.getActions();

    import ReBEL.chromosomeStructure.Chromosome;
    chromosomes = Chromosome.empty(statesNum,actionsNum,0);
    status = string.empty(statesNum,actionsNum,0);
    for i = 1:statesNum
        for j = 1:actionsNum
            randomValue = rand();
            if randomValue > (1 - epsilonSelection)
                % exploit
                chromosome = chromosomepool.exploit(states(i),actions(j));
                chromosomes(i,j,1) = copy(chromosome);
                status(i,j,1) = "exploit";
            else
                % explore
                chromosomes(i,j,1) = copy(chromosomepool.explore(states(i),
    actions(j)));
                status(i,j,1) = "explore";
            end
        end
    end
end
```

+ReBEL/+evolutionaryOperator/mutation.m

```
function [mutatedChildChromosomes] = mutation(imprintedChildChromosomes,
    mutationRate)
    geneLength = length(imprintedChildChromosomes(1,1).genes);
    mutatedChildChromosomes = copy(imprintedChildChromosomes);
    [stateNum,actionNum] = size(imprintedChildChromosomes);
    for i = 1:stateNum
        for j = 1:actionNum
            for k = 1:geneLength
                randomNumber = rand;
                if randomNumber < mutationRate
                    geneticCode = imprintedChildChromosomes(i,j).genes(k).
    getCode();

                    mutantCode = geneticCode;

                    for lCode = 1:length(mutantCode)
```

```
                        if rand() < mutationRate

                            if mutantCode(lCode) == '0'
                                mutantCode(lCode) = '1';
                            else
                                mutantCode(lCode) = '0';
                            end

                        end
                    end

                    if ~strcmp(mutantCode,geneticCode)
                        mutatedChildChromosomes(i,j).genes(k).setCode(
    mutantCode);
                    end
                end
            end
        end
    end
end
```

+ReBEL/+evolutionaryOperator/proportionalTournament.m

```
function selectedChromosomes = proportionalTournament(chromosomes)
    [stateNum,actionNum,agentNum] = size(chromosomes);
    allHistones = double.empty(stateNum,actionNum,0,1);
    for i = 1:agentNum
        for j = 1:stateNum
            for k = 1:actionNum
                averageHistones = chromosomes(j,k,i).averageHistone();
                allHistones(j,k,i) = averageHistones;
            end
        end
    end
    import ReBEL.chromosomeStructure.Chromosome;
    selectedChromosomes = Chromosome(stateNum,actionNum,0);
    for i=1:stateNum
        for j = 1:actionNum
            histones = squeeze(allHistones(i,j,:));
            % normalised histones range
            if any(histones(:) < 0)
                histones = histones - min(histones);
            end
            probabilities = histones./sum(histones);
            [sortedProbabilities,index] = sort(probabilities);
            randomNumber = rand();
            previousProbability = 0;
            for k = 1:agentNum
                previousProbability = previousProbability +
    sortedProbabilities(k);
                if randomNumber < previousProbability
                    selectedChromosomes(i,j,1) = chromosomes(i,j,index(k));
                    break
```

```
                end
                if k == agentNum
                    disp("not selecting any chromosomes");
                end
            end
        end
    end
end
```

+ReBEL/+evolutionaryOperator/regeneration.m

```
function regeneration(genepool,chromosomepool,mutatedChildChromosomes1,
    mutatedChildChromosomes2)

    stateNum = chromosomepool.getStatesNum();
    actionNum = chromosomepool.getActionsNum();
    states = chromosomepool.getStates();
    actions = chromosomepool.getActions();

    for i=1:stateNum
        for j=1:actionNum
            chromosomeset = chromosomepool.chromosomeSet(states(i),actions(
    j));
            chromosomesethistone = chromosomeset.histones;
            [sorted,index] = sort(chromosomesethistone);
            chromosome1 = mutatedChildChromosomes1(i,j);
            chromosome2 = mutatedChildChromosomes2(i,j);
            chromosomeHistone1 = chromosome1.averageHistone;
            chromosomeHistone2 = chromosome2.averageHistone;
            chromosomeset.setChromosome(index(1),chromosome1);
            chromosomeset.setChromosome(index(2),chromosome2);
            genepool.addChromosome(chromosome1);
            genepool.addChromosome(chromosome2);
%             if chromosomeHistone1 > sorted(1)
%                 chromosomeset.setChromosome(index(1),chromosome1);
%                 genepool.addChromosome(chromosome1);
%             end
        end
    end
end
```

+ReBEL/+geneticStructure/Gene.m

```
classdef Gene < handle & matlab.mixin.Copyable & matlab.mixin.CustomDisplay
    %GENE Summary of this class goes here
    %   Constructor: Gene(genotype,states,actions,nbits,max,min)
    %   Getter:
    %       - histone(state,action)
    %       - histone
    %       - value
    %       - genotype
    properties (Access = private)
        geneValue
```

```
        geneHistone
        geneGenotype
        geneSilence
    end

    methods
        function this = Gene(genotype,states,actions,methylationRate,nbits,
    max,min)
            import ReBEL.geneticStructure.GeneValue;
            import ReBEL.geneticStructure.GeneHistone;
            import ReBEL.geneticStructure.Genotype;
            import ReBEL.geneticStructure.GeneSilence;

            if nargin > 0
                this.geneValue = GeneValue(nbits,max,min);
                this.geneHistone = GeneHistone(states,actions,
    methylationRate);
                this.geneGenotype = genotype;
                this.geneSilence = GeneSilence(states,actions);
            end
        end

        function silence(this,state,action)
            this.geneSilence.silence(state,action);
        end

        function activate(this,state,action)
            this.geneSilence.activate(state,action);
        end

        function silenceStatus = getSilenceStatus(this)
            silenceStatus = this.geneSilence.values;
        end

        function histone = histone(this,state,action)
            if nargin <=1
                histone = this.geneHistone.value();
            else
                histone = this.geneHistone.value(state,action);
            end
        end

        function setCode(this,code)
            this.geneValue.setCode(code);
        end

        function setHistone(this,value,state,action)
            this.geneHistone.setValue(value,state,action);
        end

        function output = value(this)
            output= this.geneValue.value();
```

```
        end

        function genotype = genotype(this)
            genotype = this.geneGenotype;
        end

        function binary = getBinary(this)
            binary = this.geneValue.binary();
        end

        function code = getCode(this)
            code = this.geneValue.getCode();
        end

        function methylation(this,state,action,chromosomeLength,
    averageHistoneAtSA,returnValue)
            this.geneHistone.methylation(state,action,chromosomeLength,
    averageHistoneAtSA,returnValue);
        end
    end

    methods (Access = protected)
        function copyThis = copyElement(this)
            copyThis = copyElement@matlab.mixin.Copyable(this);
            copyThis.geneValue = copy(this.geneValue);
            copyThis.geneHistone = copy(this.geneHistone);
        end

        function propgrp = getPropertyGroups(this)
            if ~isscalar(this)
                propgrp = getPropertyGroups@matlab.mixin.CustomDisplay(this
    );
            else
                propList = struct(...
                    'Genotype',this.genotype,...
                    'Value',value(this),...
                    'Histone',this.histone);
                propgrp = matlab.mixin.util.PropertyGroup(propList);
            end
        end
    end
end
```

+ReBEL/+geneticStructure/GeneFamily.m

```
classdef GeneFamily < handle & matlab.mixin.CustomDisplay
    %GENEFAMILY Summary of this class goes here
    %   Constructor: GeneFamily(population,genotype,states,actions,nbits,
    min,max)
    %   Getter:
    %       - genes(index)
    %       - values
    %       - histones(state,action)
```

```
8       %       -
9
10      properties (Access = private)
11          genotype
12          geneMembers
13          population
14      end
15
16      methods
17          function this = GeneFamily(population,genotype,states,actions,
    methylationRate,nbits,min,max)
18              import ReBEL.geneticStructure.Gene;
19              if nargin ~= 0
20                  this.population = population;
21                  this.geneMembers = Gene.empty(0,population);
22                  this.genotype = genotype;
23                  for i=1:population
24                      this.geneMembers(i) = Gene(genotype,states,actions,
    methylationRate,nbits,min,max);
25                  end
26              end
27          end
28
29          function gene = genes(this,index)
30              if nargin == 1
31                  gene = this.geneMembers;
32              else
33                  gene = this.geneMembers(index);
34              end
35          end
36
37          function registeredgene = addGene(this,gene,state,action)
38              [status,index] = ismember(gene.value,this.values);
39
40              if ~status
41                  this.population = this.population + 1;
42                  this.geneMembers(end+1) = gene;
43                  registeredgene = this.geneMembers(end);
44              else
45                  this.geneMembers(index).setHistone(gene.histone(state,
    action),state,action);
46                  registeredgene = this.geneMembers(index);
47              end
48          end
49
50          function genotype = getGenotype(this)
51              genotype = this.genotype;
52          end
53
54          function values = values(this)
55              values = double.empty(0,this.population);
56              for i=1:this.population
```

```
57                  values(i) = this.geneMembers(i).value;
58              end
59          end
60
61          function silenceStatus = getSilenceStatus(this)
62              silenceStatus = double.empty(0,this.population);
63              for i=1:this.population
64                  silenceStatus(i) = this.geneMembers(i).getSilenceStatus;
65              end
66          end
67
68          function deleteGene(this,index)
69              this.geneMembers(index) = [];
70              this.population = this.population - 1;
71          end
72
73          function histones = histones(this,state,action)
74              [stateNum,actionNum] = size(this.geneMembers(1).histone);
75              if nargin <= 1
76                  if stateNum*actionNum ~= 1
77                      histones = double.empty(stateNum,actionNum,0);
78                      for i=1:this.population
79                          histones(:,:,i) = this.geneMembers(i).histone;
80                      end
81                  elseif nargin <= 1 && stateNum*actionNum == 1
82                      histones = double.empty(0,0);
83                      for i=1:this.population
84                          histones(i) = this.geneMembers(i).histone;
85                      end
86                  end
87              else
88                  histones = double.empty(0,0);
89                  for i=1:this.population
90                      histones(i) = this.geneMembers(i).histone(state,action)
    ;
91                  end
92              end
93          end
94      end
95
96      methods (Access = protected)
97          function propgrp = getPropertyGroups(this)
98              if ~isscalar(this)
99                  propgrp = getPropertyGroups@matlab.mixin.CustomDisplay(this
    );
100             else
101                 propList = struct(...
102                     'genotype',this.genotype,...
103                     'genes',this.geneMembers);
104                 propgrp = matlab.mixin.util.PropertyGroup(propList);
105             end
106         end
```

```
    end
end
```

+ReBEL/+geneticStructure/GeneHistone.m

```
classdef GeneHistone < handle & matlab.mixin.Copyable
    %GENEHISTONE Summary of this class goes here
    %   Constructor: GeneHistone(states,actions)

    properties (Access = private)
        histone_value
        methylationRate
        states
        actions
    end

    methods
        function this = GeneHistone(states,actions,methylationRate)
            this.states = states;
            this.actions = actions;
            this.histone_value = rand(length(states),length(actions));
            this.methylationRate = methylationRate;
        end

        function value = value(this,state,action)
            if nargin <= 1
                value = this.histone_value;
            else
                [status,stateIndex] = ismember(state,this.states);
                if status ~= 1
                    error("no state founded")
                end

                [status,actionIndex] = ismember(action,this.actions);
                if status ~= 1
                    error("no action founded")
                end

                value = this.histone_value(stateIndex,actionIndex);
            end
        end

        function setValue(this,value,state,action)
            [status,stateIndex] = ismember(state,this.states);
            if status ~= 1
                error("no state founded")
            end

            [status,actionIndex] = ismember(action,this.actions);
            if status ~= 1
                error("no action founded")
            end

```

```
            this.histone_value(stateIndex,actionIndex) = ...
                this.histone_value(stateIndex,actionIndex) + ...
                this.methylationRate*(value - this.histone_value(stateIndex
    ,actionIndex));
        end

        function methylation(this,state,action,chromosomeLength,
    averageHistone,returnValue)
            [status,stateIndex] = ismember(state,this.states);
            [status,actionIndex] = ismember(action,this.actions);
            this.histone_value(stateIndex,actionIndex) = ...
                this.histone_value(stateIndex,actionIndex) + ...
                this.methylationRate*(2/chromosomeLength)*((1+
    chromosomeLength)*returnValue - averageHistone - chromosomeLength*this.
    histone_value(stateIndex,actionIndex));
        end
    end

    methods (Access = protected)
        function copyThis = copyElement(this)
            copyThis = copyElement@matlab.mixin.Copyable(this);
        end
    end
end
```

+ReBEL/+geneticStructure/GenePool.m

```
classdef GenePool < handle
    %GENEPOOL Summary of this class goes here
    %   Detailed explanation goes here

    properties (Access = private)
        geneFamilyMember
        allGenotypes
    end

    methods
        function this = GenePool(population,genotypes,states,actions,
    methylationRate,nbits,min,max)
            import ReBEL.geneticStructure.GeneFamily;
            this.allGenotypes = genotypes;
            this.geneFamilyMember = GeneFamily.empty(0,0);
            for i=1:length(genotypes)
                this.geneFamilyMember(i) = GeneFamily(population,genotypes(
    i),states,actions,methylationRate,nbits,min,max);
            end
        end

        function genefamily = genefamilies(this,genotype)
            if nargin <= 1
                genefamily = this.geneFamilyMember;
            else
                [status,index] = ismember(genotype,this.genotypes);
```

```
                if status
                    genefamily = this.geneFamilyMember(index);
                else
                    error("genotype not found");
                end
            end
        end

        function genotypes = genotypes(this)
            genotypes = this.allGenotypes;
        end

        function addChromosome(this,chromosome)
            for i=1:length(chromosome.genes)
                gene = this.geneFamilyMember(i).addGene(chromosome.genes(i)
    ,chromosome.getState(),chromosome.getAction());
                chromosome.setGene(i,gene);
            end
        end
    end
end
```

+ReBEL/+geneticStructure/GeneSilence.m

```
classdef GeneSilence < handle
    %GENESILENCE Summary of this class goes here
    %   Detailed explanation goes here

    properties
        states
        actions
        values
    end

    methods
        function this = GeneSilence(states,actions)
            this.states = states;
            this.actions = actions;
            this.values = false(length(states),length(actions));
        end

        function silence(this,state,action)
            [status,stateNum] = ismember(state,this.states);
            [status,actionNum] = ismember(action,this.actions);
            this.values(stateNum,actionNum) = true;
        end

        function activate(this,state,action)
            [status,stateNum] = ismember(state,this.states);
            [status,actionNum] = ismember(action,this.actions);
            this.values(stateNum,actionNum) = false;
        end
    end
```

```
end
```

+ReBEL/+geneticStructure/GeneValue.m

```
classdef GeneValue < handle & matlab.mixin.Copyable
    %GENEVALUE Summary of this class goes here
    %   Detailed explanation goes here

    properties (Access = private)
        g_value
        g_decimal
        g_binary
        g_grey
        nbits
        min_value
        max_value
        min_decimal
        max_decimal
    end

    methods
        function this = GeneValue(nbits,min,max)
            if nargin > 0
                this.nbits = nbits;
                this.min_value = min;
                this.max_value = max;

                % calculating decimal constraints
                this.min_decimal = 0;
                this.max_decimal = 2^nbits - 1;

                % set initial value on other properties
                randomValue = min + (max-min)*rand();
                this.g_decimal = val2dec(this,randomValue);
                this.g_value = dec2val(this,this.g_decimal);
                this.g_binary = dec2bin(this.g_decimal,this.nbits);
                this.g_grey = bin2grey(this,this.g_binary);
                this.g_binary = this.grey2bin(this.g_grey);
                this.g_decimal = bin2dec(this.g_binary);
                this.g_value = this.dec2val(this.g_decimal);
            end
        end

        function value = value(this)
            value = this.g_value;
        end

        function decimal = decimal(this)
            decimal = this.g_decimal;
        end

        function binary = binary(this)
            binary = this.g_binary;
```

```
50          end
51
52          function grey = grey(this)
53              grey = this.g_grey;
54          end
55
56          function code = getCode(this)
57              code = this.g_grey;
58  %             code = this.g_binary;
59          end
60          function setCode(this,code)
61              this.g_grey = code;
62              this.g_binary = this.grey2bin(this.g_grey);
63              this.g_decimal = bin2dec(this.g_binary);
64              this.g_value = this.dec2val(this.g_decimal);
65          end
66      end
67
68      methods (Access = private)
69          function decimal = val2dec(this,value)
70              decimal = ((this.max_decimal - this.min_decimal)/...
71                         (this.max_value - this.min_value))...
72                         *(value - this.min_value) + this.min_decimal;
73              decimal = round(decimal);
74          end
75
76          function value = dec2val(this,decimal)
77              value = ((this.max_value - this.min_value)/...
78                       (this.max_decimal - this.min_decimal))...
79                       *(decimal - this.min_decimal) + this.min_value;
80          end
81
82          function grey = bin2grey(this,binary)
83              n = length(binary);
84              grey = zeros(1,n);
85              grey(1) = binary(1);
86              for i = 2:n
87                  grey(i) = xorChar(this,binary(i-1),binary(i));
88              end
89
90              grey = char(grey);
91          end
92
93          function binary = grey2bin(this,grey_code)
94              n = length(grey_code);
95              binary(1) = grey_code(1);
96
97              for i = 2:n
98                  if grey_code(i) == '0'
99                      binary(i) = binary(i - 1);
100                 else
101                     binary(i) = flipChar(this,binary(i -1));
```

```
                end
            end

            binary = char(binary);
        end

        % xor operator for char
        function output = xorChar(~,a,b)
            if (a == b)
                output = '0';
            else
                output = '1';
            end
        end

        function output = flipChar(~,a)
            if (a == '0')
                output = '1';
            else
                output = '0';
            end
        end
    end

    methods (Access = protected)
        function copyThis = copyElement(this)
            copyThis = copyElement@matlab.mixin.Copyable(this);
        end
    end
end
```

+ReBEL/+geneticStructure/Genotype.m

```
classdef Genotype < handle & matlab.mixin.Copyable
    %GENOTYPE Summary of this class goes here
    %   Detailed explanation goes here

    properties
        name
    end

    methods
        function this = Genotype(name)
            this.name = name;
        end
    end

    methods (Access = protected)
        function copyThis = copyElement(this)
            copyThis = copyElement@matlab.mixin.Copyable(this);
            copyThis.name = copy(this.name);
        end
    end
```

```
end
```

+testFunction/Ackley.m

```
classdef Ackley < handle
    %LEVY Summary of this class goes here
    %   Detailed explanation goes here

    properties
        variables
        g_value
        g_decimal
        g_binary
        g_grey
        nbits
        min_value
        max_value
        min_decimal
        max_decimal
        optimalValue
        fig
        dots

        a = 20;
        b = 0.2;
        c = pi/2;
    end

    methods
        function this = Ackley(variables, minValue, maxValue, nbits,
    figureNum)
            this.variables = variables;
            this.nbits = nbits;
            this.min_value = minValue;
            this.max_value = maxValue;

            % calculating decimal constraints
            this.min_decimal = 0;
            this.max_decimal = 2^nbits - 1;

            % set initial value on other properties
            optimal_value = zeros(length(variables),1);
            this.g_decimal = val2dec(this,optimal_value);
            this.g_value = dec2val(this,this.g_decimal);
            this.g_binary = dec2bin(this.g_decimal,this.nbits);
            this.g_grey = bin2grey(this,this.g_binary);
            this.g_binary = this.grey2bin(this.g_grey);
            this.g_decimal = bin2dec(this.g_binary);
            this.g_value = this.dec2val(this.g_decimal);
            this.optimalValue = this.calculate(this.g_value);

            if length(variables) == 2
                this.initShow(figureNum);
```

```
49                  drawnow;
50              end
51          end
52
53          function output = calculate(this,variables)
54              output = -this.a*exp(-this.b*sqrt(sum(variables.^2)./length(
    this.variables))) - exp(sum(cos(this.c*variables))./length(this.
    variables)) + this.a + exp(1);
55          end
56
57          function initShow(this,figureNum)
58              this.fig = figure(figureNum);
59              clf(this.fig);
60               X = repmat(this.min_value:0.5:this.max_value,[length(this.
    variables),1]);
61              [x1,x2] = meshgrid(X(1,:),X(2,:));
62
63              y = -this.a*exp(-this.b*sqrt(((x1.^2) + (x2.^2))./length(this.
    variables))) - exp((cos(this.c*x1) + cos(this.c*x2))./length(this.
    variables)) + this.a + exp(1);
64              subplot(1,2,1);
65              surface(x1,x2,y);
66              xlabel("x1");
67              ylabel("x2");
68              zlabel("f(x1,x2)");
69              ax = gca;
70              set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1
    0.5]);
71              ax.View=[50,15];
72
73              X = repmat(this.min_value:0.1:this.max_value,[length(this.
    variables),1]);
74              [x1,x2] = meshgrid(X(1,:),X(2,:));
75
76              y = -this.a*exp(-this.b*sqrt(((x1.^2) + (x2.^2))./length(this.
    variables))) - exp((cos(this.c*x1) + cos(this.c*x2))./length(this.
    variables)) + this.a + exp(1);
77
78              subplot(1,2,2);
79              [C,h]= contour(x1,x2,y);
80              clabel(C,h);
81              xlabel("x1");
82              ylabel("x2");
83              zlabel("f(x1,x2)");
84              ax = gca;
85              set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1 1]);
86              hold on
87              this.dots = plot(zeros(length(this.variables),1),zeros(length(
    this.variables),1),'Marker','x','Color','red','LineStyle','none');
88          end
89
90          function draw(this,index,values)
```

```
91              this.dots.XData(index) = values(1);
92              this.dots.YData(index) = values(2);
93              drawnow;
94          end
95
96      end
97
98      methods (Access = private)
99          function decimal = val2dec(this,value)
100             decimal = ((this.max_decimal - this.min_decimal)/...
101                        (this.max_value - this.min_value))...
102                        .*(value - this.min_value) + this.min_decimal;
103             decimal = round(decimal);
104         end
105
106         function value = dec2val(this,decimal)
107             value = ((this.max_value - this.min_value)/...
108                      (this.max_decimal - this.min_decimal))...
109                      .*(decimal - this.min_decimal) + this.min_value;
110         end
111
112         function grey = bin2grey(this,binary)
113             [m,n] = size(binary);
114             grey = zeros(m,n);
115             grey(:,1) = binary(:,1);
116             for j = 1:m
117                 for i = 2:n
118                     grey(j,i) = xorChar(this,binary(j,i-1),binary(j,i));
119                 end
120             end
121
122             grey = char(grey);
123         end
124
125         function binary = grey2bin(this,grey_code)
126             [m,n] = size(grey_code);
127             binary(:,1) = grey_code(:,1);
128
129             for j = 1:m
130                 for i = 2:n
131                     if grey_code(j,i) == '0'
132                         binary(j,i) = binary(j,i - 1);
133                     else
134                         binary(j,i) = flipChar(this,binary(j,i -1));
135                     end
136                 end
137             end
138
139             binary = char(binary);
140         end
141
142         % xor operator for char
```

```
143         function output = xorChar(~,a,b)
144             if (a == b)
145                 output = '0';
146             else
147                 output = '1';
148             end
149         end
150 
151         function output = flipChar(~,a)
152             if (a == '0')
153                 output = '1';
154             else
155                 output = '0';
156             end
157         end
158     end
159 end
```

+testFunction/Levy.m

```
1  classdef Levy < handle
2      %LEVY Summary of this class goes here
3      %   Detailed explanation goes here
4  
5      properties
6          variables
7          g_value
8          g_decimal
9          g_binary
10         g_grey
11         nbits
12         min_value
13         max_value
14         min_decimal
15         max_decimal
16         optimalValue
17         fig
18         dots
19     end
20 
21     methods
22         function this = Levy(variables, minValue, maxValue, nbits,
       figureNum)
23             this.variables = variables;
24             this.nbits = nbits;
25             this.min_value = minValue;
26             this.max_value = maxValue;
27 
28             % calculating decimal constraints
29             this.min_decimal = 0;
30             this.max_decimal = 2^nbits - 1;
31 
32             % set initial value on other properties
```

```
33            optimal_value = ones(length(variables),1);
34            this.g_decimal = val2dec(this,optimal_value);
35            this.g_value = dec2val(this,this.g_decimal);
36            this.g_binary = dec2bin(this.g_decimal,this.nbits);
37            this.g_grey = bin2grey(this,this.g_binary);
38            this.g_binary = this.grey2bin(this.g_grey);
39            this.g_decimal = bin2dec(this.g_binary);
40            this.g_value = this.dec2val(this.g_decimal);
41            this.optimalValue = this.calculate(this.g_value);
42
43            if length(variables) == 2
44                this.initShow(figureNum);
45                drawnow;
46            end
47        end
48
49        function output = calculate(this,variables)
50            w = 1 + (variables - 1)./4;
51
52            term0 = sin(pi*w(1))^2;
53            term1 = (w(1:(end-1))-1).^2;
54            term2 = 1 + 10*(sin(pi*w(1:(end-1)) + 1).^2);
55            term3 = ((w(end)-1).^2).*(1 + sin(2*pi*w(end)).^2);
56            output = term0 + sum(term1.*term2 + term3);
57        end
58
59        function initShow(this,figureNum)
60            this.fig = figure(figureNum);
61            clf(this.fig);
62            X = repmat(this.min_value:0.25:this.max_value,[length(this.
    variables),1]);
63            [x1,x2] = meshgrid(X(1,:),X(2,:));
64
65            dataW = 1 + (X - 1)./4;
66            [w1,w2] = meshgrid(dataW(1,:),dataW(2,:));
67            term1 = (w1-1).^2;
68            term2 = 1 + 10*(sin(pi*w1 + 1).^2);
69            term3 = ((w2-1).^2).*(1 + sin(2*pi*w2).^2);
70            y = sin(pi*w1).^2 + term1.*term2 + term3;
71            subplot(1,2,1);
72            surface(x1,x2,y);
73            xlabel("x1");
74            ylabel("x2");
75            zlabel("f(x1,x2)");
76            ax = gca;
77            set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1 5]);
78            ax.View=[70,35];
79
80             X = repmat(this.min_value:0.01:this.max_value,[length(this.
    variables),1]);
81            [x1,x2] = meshgrid(X(1,:),X(2,:));
82
```

```
83              dataW = 1 + (X - 1)./4;
84              [w1,w2] = meshgrid(dataW(1,:),dataW(2,:));
85              term1 = (w1-1).^2;
86              term2 = 1 + 10*(sin(pi*w1 + 1).^2);
87              term3 = ((w2-1).^2).*(1 + sin(2*pi*w2).^2);
88              y = sin(pi*w1).^2 + term1.*term2 + term3;
89              subplot(1,2,2);
90              [C,h]= contour(x1,x2,y);
91              visible = [10,20,30,40,50];
92              clabel(C,h,visible);
93              xlabel("x1");
94              ylabel("x2");
95              zlabel("f(x1,x2)");
96              ax = gca;
97              set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1 1]);
98              hold on
99              this.dots = plot(zeros(length(this.variables),1),zeros(length(
    this.variables),1),'Marker','x','Color','red','LineStyle','none');
100         end
101
102         function draw(this,index,values)
103             this.dots.XData(index) = values(1);
104             this.dots.YData(index) = values(2);
105         end
106
107     end
108
109     methods (Access = private)
110         function decimal = val2dec(this,value)
111             decimal = ((this.max_decimal - this.min_decimal)/...
112                        (this.max_value - this.min_value))...
113                        .*(value - this.min_value) + this.min_decimal;
114             decimal = round(decimal);
115         end
116
117         function value = dec2val(this,decimal)
118             value = ((this.max_value - this.min_value)/...
119                      (this.max_decimal - this.min_decimal))...
120                      .*(decimal - this.min_decimal) + this.min_value;
121         end
122
123         function grey = bin2grey(this,binary)
124             [m,n] = size(binary);
125             grey = zeros(m,n);
126             grey(:,1) = binary(:,1);
127             for j = 1:m
128                 for i = 2:n
129                     grey(j,i) = xorChar(this,binary(j,i-1),binary(j,i));
130                 end
131             end
132
133             grey = char(grey);
```

```
134         end
135
136         function binary = grey2bin(this,grey_code)
137             [m,n] = size(grey_code);
138             binary(:,1) = grey_code(:,1);
139
140             for j = 1:m
141                 for i = 2:n
142                     if grey_code(j,i) == '0'
143                         binary(j,i) = binary(j,i - 1);
144                     else
145                         binary(j,i) = flipChar(this,binary(j,i -1));
146                     end
147                 end
148             end
149
150             binary = char(binary);
151         end
152
153         % xor operator for char
154         function output = xorChar(~,a,b)
155             if (a == b)
156                 output = '0';
157             else
158                 output = '1';
159             end
160         end
161
162         function output = flipChar(~,a)
163             if (a == '0')
164                 output = '1';
165             else
166                 output = '0';
167             end
168         end
169     end
170 end
```

+testFunction/Sphere.m

```
1  classdef Sphere < handle
2      properties
3          variables
4          g_value
5          g_decimal
6          g_binary
7          g_grey
8          nbits
9          min_value
10         max_value
11         min_decimal
12         max_decimal
13         optimalValue
```

```
14          fig
15          dots
16      end
17
18      methods
19          function this = Sphere(variables, minValue, maxValue, nbits,
    figureNum)
20              this.variables = variables;
21              this.nbits = nbits;
22              this.min_value = minValue;
23              this.max_value = maxValue;
24
25              % calculating decimal constraints
26              this.min_decimal = 0;
27              this.max_decimal = 2^nbits - 1;
28
29              % set initial value on other properties
30              optimal_value = zeros(length(variables),1);
31              this.g_decimal = val2dec(this,optimal_value);
32              this.g_value = dec2val(this,this.g_decimal);
33              this.g_binary = dec2bin(this.g_decimal,this.nbits);
34              this.g_grey = bin2grey(this,this.g_binary);
35              this.g_binary = this.grey2bin(this.g_grey);
36              this.g_decimal = bin2dec(this.g_binary);
37              this.g_value = this.dec2val(this.g_decimal);
38              this.optimalValue = this.calculate(this.g_value);
39
40              if length(variables) == 2
41                  this.initShow(figureNum);
42                  drawnow;
43              end
44          end
45
46          function output = calculate(this,variables)
47              output = sum(variables.^2);
48          end
49
50          function initShow(this,figureNum)
51              this.fig = figure(figureNum);
52              clf(this.fig);
53              X = repmat(this.min_value:0.5:this.max_value,[length(this.
    variables),1]);
54              [x1,x2] = meshgrid(X(1,:),X(2,:));
55              y = x1.^2 + x2.^2;
56              subplot(1,2,1);
57              surface(x1,x2,y);
58              xlabel("x1");
59              ylabel("x2");
60              zlabel("f(x1,x2)");
61              ax = gca;
62              set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1 10])
    ;
```

```
63              ax.View=[70,35];
64              subplot(1,2,2);
65              [C,h]= contour(x1,x2,y);
66              clabel(C,h);
67              xlabel("x1");
68              ylabel("x2");
69              zlabel("f(x1,x2)");
70              ax = gca;
71              set(ax,'PlotBoxAspectRatio',[1 1 1],'DataAspectRatio',[1 1 1]);
72              hold on
73              this.dots = plot(zeros(length(this.variables),1),zeros(length(
    this.variables),1),'Marker','x','Color','red','LineStyle','none');
74          end
75
76          function draw(this,index,values)
77              this.dots.XData(index) = values(1);
78              this.dots.YData(index) = values(2);
79  %             drawnow;
80          end
81      end
82
83      methods (Access = private)
84          function decimal = val2dec(this,value)
85              decimal = ((this.max_decimal - this.min_decimal)/...
86                         (this.max_value - this.min_value))...
87                         .*(value - this.min_value) + this.min_decimal;
88              decimal = round(decimal);
89          end
90
91          function value = dec2val(this,decimal)
92              value = ((this.max_value - this.min_value)/...
93                       (this.max_decimal - this.min_decimal))...
94                       .*(decimal - this.min_decimal) + this.min_value;
95          end
96
97          function grey = bin2grey(this,binary)
98              [m,n] = size(binary);
99              grey = zeros(m,n);
100             grey(:,1) = binary(:,1);
101             for j = 1:m
102                 for i = 2:n
103                     grey(j,i) = xorChar(this,binary(j,i-1),binary(j,i));
104                 end
105             end
106
107             grey = char(grey);
108         end
109
110         function binary = grey2bin(this,grey_code)
111             [m,n] = size(grey_code);
112             binary(:,1) = grey_code(:,1);
113
```

```
114             for j = 1:m
115                 for i = 2:n
116                     if grey_code(j,i) == '0'
117                         binary(j,i) = binary(j,i - 1);
118                     else
119                         binary(j,i) = flipChar(this,binary(j,i -1));
120                     end
121                 end
122             end
123 
124             binary = char(binary);
125         end
126 
127         % xor operator for char
128         function output = xorChar(~,a,b)
129             if (a == b)
130                 output = '0';
131             else
132                 output = '1';
133             end
134         end
135 
136         function output = flipChar(~,a)
137             if (a == '0')
138                 output = '1';
139             else
140                 output = '0';
141             end
142         end
143     end
144 end
```

Bibliography

Allard, Tom, and Amy McNeilage. 2014. "Most Expensive Aviation Search: $53 Million to Find Flight MH370," April 5, 2014. Accessed December 8, 2018. https://www.smh.com.au/national/most-expensive-aviation-search-53-million-to-find-flight-mh370-20140404-36463.html.

Ame, J. M., J. Halloy, C. Rivault, C. Detrain, and J. L. Deneubourg. 2006. "Collegial Decision Making Based on Social Amplification Leads to Optimal Group Formation." *Proceedings of the National Academy of Sciences* 103, no. 15 (April 11, 2006): 5835–5840. ISSN: 0027-8424, 1091-6490, accessed October 2, 2017. https://doi.org/10.1073/pnas.0507877103. http://www.pnas.org/cgi/doi/10.1073/pnas.0507877103.

Antonelli, Gianluca, Filippo Arrichiello, and Stefano Chiaverini. 2010. "Flocking for Multi-Robot Systems via the Null-Space-Based Behavioral Control." *Swarm Intelligence* 4, no. 1 (March): 37–56. ISSN: 1935-3812, 1935-3820, accessed August 9, 2018. https://doi.org/10.1007/s11721-009-0036-6. http://link.springer.com/10.1007/s11721-009-0036-6.

Arulampalam, M.S., S. Maskell, N. Gordon, and T. Clapp. 2002. "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking." *IEEE Transactions on Signal Processing* 50, no. 2 (February): 174–188. ISSN: 1941-0476. https://doi.org/10.1109/78.978374.

Bakhshipour, M., M. Jabbari Ghadi, and F. Namdari. 2017. "Swarm Robotics Search & Rescue: A Novel Artificial Intelligence-Inspired Optimization Approach." *Applied Soft Computing* 57 (August 1, 2017): 708–726. ISSN: 1568-4946, accessed June 6, 2021. https://doi.org/10.1016/j.asoc.2017.02.028. https://www.sciencedirect.com/science/article/pii/S1568494617301072.

Baldassarre, Gianluca, Stefano Nolfi, and Domenico Parisi. 2003. "Evolving Mobile Robots Able to Display Collective Behaviors." *Artificial Life* 9, no. 3 (July): 255–267. ISSN: 1064-5462, 1530-9185, accessed October 2, 2017. https://doi.org/10.1162/106454603322392460. http://www.mitpressjournals.org/doi/10.1162/106454603322392460.

Bates, Joseph, A. Bryan Loyall, and W. Scott Reilly. 1994. "An Architecture for Action, Emotion, and Social Behavior." In *Artificial Social Systems,* edited by Cristiano Castelfranchi and Eric Werner, redacted by J. G. Carbonell, J. Siekmann, G. Goos, and J. Hartmanis, 830:55–68. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-58266-3 978-3-540-48589-6, accessed September 12, 2020. http://link.springer.com/10.1007/3-540-58266-5_4.

Bayindir, Levent. 2016. "A Review of Swarm Robotics Tasks." *Neurocomputing* 172 (January): 292–321. ISSN: 09252312, accessed October 2, 2017. https://doi.org/10.1016/j.neucom.2015.05.116. http://linkinghub.elsevier.com/retrieve/pii/S0925231215010486.

Beard, Randal W., and Timothy W. McLain. 2012. *Small Unmanned Aircraft: Theory and Practice.* Princeton, N.J: Princeton University Press. ISBN: 978-0-691-14921-9.

Berthouze, Luc, and Tom Ziemke. 2003. "Epigenetic Robotics—Modelling Cognitive Development in Robotic Systems." *Connection Science* 15, no. 4 (December 1, 2003): 147–150. ISSN: 0954-0091, accessed September 20, 2019. https://doi.org/10.1080/09540090310001658063. https://doi.org/10.1080/09540090310001658063.

Bogue, Robert. 2008. "Swarm Intelligence and Robotics." Edited by David Sanders. *Industrial Robot: An International Journal* 35, no. 6 (October 17, 2008): 488–495. ISSN: 0143-991X, accessed October 2, 2017. https://doi.org/10.1108/01439910810909475. http://www.emeraldinsight.com/doi/10.1108/01439910810909475.

Bourgault, Frédéric, Tomonari Furukawa, and Hugh F. Durrant-Whyte. 2003. "Coordinated Decentralized Search for a Lost Target in a Bayesian World." In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453),* vol. 1, 48–53 vol.1. October. https://doi.org/10.1109/IROS.2003.1250604.

———. 2004. "Process Model, Constraints, and the Coordinated Search Strategy." In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004,* 5:5256–5261. April. https://doi.org/10.1109/ROBOT.2004.1302552.

———. 2006. "Optimal Search for a Lost Target in a Bayesian World." In *Field and Service Robotics: Recent Advances in Reserch and Applications,* edited by Shin'ichi Yuta, Hajima Asama, Erwin Prassler, Takashi Tsubouchi, and Sebastian Thrun, 209–222. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer. ISBN: 978-3-540-32854-4, accessed September 30, 2020. https://doi.org/10.1007/10991459_21. https://doi.org/10.1007/10991459_21.

Bowling, Michael, and Manuela Veloso. 2001. "Rational and Convergent Learning in Stochastic Games." In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2,* 1021–1026. IJCAI'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 978-1-55860-812-2, accessed October 16, 2017. http://dl.acm.org/citation.cfm?id=1642194.1642231.

Brambilla, Manuele, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. 2013. "Swarm Robotics: A Review from the Swarm Engineering Perspective." *Swarm Intelligence* 7, no. 1 (March): 1–41. ISSN: 1935-3812, 1935-3820, accessed October 2, 2017. https://doi.org/10.1007/s11721-012-0075-2. http://link.springer.com/10.1007/s11721-012-0075-2.

Busoniu, L., R. Babuska, and B. De Schutter. 2008. "A Comprehensive Survey of Multiagent Reinforcement Learning." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, no. 2 (March): 156–172. ISSN: 1094-6977. https://doi.org/10.1109/TSMCC.2007.913919.

Camazine, Scott, ed. 2001. *Self-Organization in Biological Systems.* Princeton Studies in Complexity. Princeton, N.J: Princeton University Press. ISBN: 978-0-691-01211-7.

Campo, Alexandre, and Marco Dorigo. 2007. "Efficient Multi-Foraging in Swarm Robotics." In *Advances in Artificial Life,* edited by Fernando Almeida e Costa, Luis Mateus Rocha, Ernesto Costa, Inman Harvey, and António Coutinho, 4648:696–705. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-74912-7, accessed February 7, 2021. https://doi.org/10.1007/978-3-540-74913-4_70. http://link.springer.com/10.1007/978-3-540-74913-4_70.

Carey, Bill. 2012. "U.S. Chooses Aerosonde, Other UAVs for ISR Services." Aviation International News, March 16, 2012. Accessed February 4, 2021. https://www.ainonline.com/aviation-news/defense/2012-03-16/us-chooses-aerosonde-other-uavs-isr-services.

Çelikkanat, Hande, and Erol Şahin. 2010. "Steering Self-Organized Robot Flocks through Externally Guided Individuals." *Neural Computing and Applications* 19, no. 6 (September 1, 2010): 849–865. ISSN: 1433-3058, accessed October 10, 2019. https://doi.org/10.1007/s00521-010-0355-y. https://doi.org/10.1007/s00521-010-0355-y.

Couceiro, M. 2016. "An Overview of Swarm Robotics for Search and Rescue Applications." https://doi.org/10.4018/978-1-5225-1759-7.ch061.

Couzin, Iain D., Jens Krause, Nigel R. Franks, and Simon A. Levin. 2005. "Effective Leadership and Decision-Making in Animal Groups on the Move." *Nature* 433, no. 7025 (February 3, 2005): 513–516. ISSN: 0028-0836, 1476-4679, accessed October 2, 2017. https://doi.org/10.1038/nature03236. http://www.nature.com/doifinder/10.1038/nature03236.

Cucker, Felipe, and Steve Smale. 2007. "Emergent Behavior in Flocks." *IEEE Transactions on Automatic Control* 52, no. 5 (May): 852–862. ISSN: 0018-9286, accessed October 2, 2017. https://doi.org/10.1109/TAC.2007.895842. http://ieeexplore.ieee.org/document/4200853/.

Deneubourg, J.-L., Simon Goss, Jacques M Pasteels, D Fresneau, and J.-P Lachaud. 1987. "Self-Organization Mechanisms in Ant Societies (II): Learning in Foraging and Division of Labor." In *From Individual to Collective Behavior in Social Insects,* by Jacques M Pasteels and J.-L. Deneubourg, 54:177–196. Eds. Experientia Supplementum. Basel, Switzerland: Birkhäuser.

Eiben, A.E., and J.E. Smith. 2015. *Introduction to Evolutionary Computing.* Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-662-44873-1 978-3-662-44874-8, accessed October 2, 2017. http://link.springer.com/10.1007/978-3-662-44874-8.

Ferrante, Eliseo, Ali Emre Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, and Marco Dorigo. 2012. "Self-Organized Flocking with a Mobile Robot Swarm: A Novel Motion Control Method." *Adaptive Behavior* 20, no. 6 (December): 460–477. ISSN: 1059-7123, 1741-2633, accessed October 10, 2019. https://doi.org/10.1177/1059712312462248. http://journals.sagepub.com/doi/10.1177/1059712312462248.

Ferrante, Eliseo, Ali Emre Turgut, Nithin Mathews, Mauro Birattari, and Marco Dorigo. 2010. "Flocking in Stationary and Non-Stationary Environments: A Novel Communication Strategy for Heading Alignment." In *Parallel Problem Solving from Nature, PPSN XI,* edited by Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, 331–340. Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN: 978-3-642-15871-1.

Ferrante, Eliseo, Ali Emre Turgut, Alessandro Stranieri, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. 2014. "A Self-Adaptive Communication Strategy for Flocking in Stationary and Non-Stationary Environments." *Natural Computing* 13, no. 2 (June 1, 2014): 225–245. ISSN: 1572-9796, accessed October 10, 2019. https://doi.org/10.1007/s11047-013-9390-9. https://doi.org/10.1007/s11047-013-9390-9.

Fetecau, R. C. 2011. "Collective Behavior of Biological Aggregations in Two Dimensions: A Nonlocal Kinetic Model." *Mathematical Models and Methods in Applied Sciences* 21, no. 07 (July): 1539–1569. ISSN: 0218-2025, 1793-6314, accessed October 2, 2017. https://doi.org/10.1142/S0218202511005489. http://www.worldscientific.com/doi/abs/10.1142/S0218202511005489.

Fetecau, R. C., and J. Meskas. 2013. "A Nonlocal Kinetic Model for Predator–Prey Interactions." *Swarm Intelligence* 7, no. 4 (December): 279–305. ISSN: 1935-3812, 1935-3820, accessed October 2, 2017. https://doi.org/10.1007/s11721-013-0084-9. http://link.springer.com/10.1007/s11721-013-0084-9.

Fontana, Alessandro. 2007. "Cell Tracking: Genesis and Epigenesis in an Artificial Organism." In *Advances in Artificial Life,* edited by Fernando Almeida e Costa, Luis Mateus Rocha, Ernesto Costa, Inman Harvey, and António Coutinho, 4648:163–171. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-74912-7, accessed September 23, 2019. http://link.springer.com/10.1007/978-3-540-74913-4_17.

Fontana, Alessandro. 2008. "Epigenetic Tracking, a Method to Generate Arbitrary Shapes By Using Evolutionary-Developmental Techniques," May 16, 2008. Accessed September 23, 2019. arXiv: 0805.2522 [nlin, q-bio]. http://arxiv.org/abs/0805.2522.

Forel. 1874. "Les Fourmis de La Suisse. Systématique, Notices Anatomiques et Physiologiques, Architecture, Distribution Géographique, Nouvelles Expériences et Observations de Moeurs." *Neue Denkschriften der Allgemeinen Schweizerischen Gesellschaft für die Gesammten Naturwissenschaften* 26:1–452. Accessed October 2, 2019. http://antcat.org/references/124988.

Francesca, Gianpiero, Manuele Brambilla, Vito Trianni, Marco Dorigo, and Mauro Birattari. 2012. "Analysing an Evolved Robotic Behaviour Using a Biological Model of Collegial Decision Making." In *From Animals to Animats 12,* edited by Tom Ziemke, Christian Balkenius, and John Hallam, 7426:381–390. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-33092-6 978-3-642-33093-3, accessed October 2, 2017. http://link.springer.com/10.1007/978-3-642-33093-3_38.

Friedrich, Tobias, Pietro S. Oliveto, Dirk Sudholt, and Carsten Witt. 2008. "Theoretical Analysis of Diversity Mechanisms for Global Exploration," 945. ACM Press. ISBN: 978-1-60558-130-9, accessed February 26, 2018. https://doi.org/10.1145/1389095.1389276. http://portal.acm.org/citation.cfm?doid=1389095.1389276.

Garnier, Simon, Jacques Gautrais, Masoud Asadpour, Christian Jost, and Guy Theraulaz. 2009. "Self-Organized Aggregation Triggers Collective Decision Making in a Group of Cockroach-Like Robots." *Adaptive Behavior* 17, no. 2 (April): 109–133. ISSN: 1059-7123, 1741-2633, accessed October 2, 2017. https://doi.org/10.1177/1059712309103430. http://journals.sagepub.com/doi/10.1177/1059712309103430.

Gauci, Melvin, Jianing Chen, Wei Li, Tony J. Dodd, and Roderich Gross. 2014. "Clustering Objects with Robots That Do Not Compute." In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems,* 421–428. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN: 978-1-4503-2738-1, accessed October 15, 2017. http://dl.acm.org/citation.cfm?id=2615731.2615800.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, Mass: Addison-Wesley Pub. Co. ISBN: 978-0-201-15767-3.

Gomes, Jorge, and Anders L. Christensen. 2013. "Generic Behaviour Similarity Measures for Evolutionary Swarm Robotics." In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation,* 199–206. GECCO '13. New York, NY, USA: ACM. ISBN: 978-1-4503-1963-8, accessed October 2, 2019. https://doi.org/10.1145/2463372.2463398. http://doi.acm.org/10.1145/2463372.2463398.

Groß, Roderich, and Marco Dorigo. 2004. "Cooperative Transport of Objects of Different Shapes and Sizes." In *Ant Colony Optimization and Swarm Intelligence,* 106–117. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, September 5, 2004. ISBN: 978-3-540-22672-7 978-3-540-28646-2, accessed October 15, 2017. https://link.springer.com/chapter/10.1007/978-3-540-28646-2_10.

———. 2009. "Towards Group Transport by Swarms of Robots." *International Journal of Bio-Inspired Computation* 1 (1/2): 1. ISSN: 1758-0366, 1758-0374, accessed October 23, 2017. https://doi.org/10.1504/IJBIC.2009.022770. http://www.inderscience.com/link.php?id=22770.

Hackett-Jones, Emily J., Kerry A. Landman, and Klemens Fellner. 2012. "Aggregation Patterns from Nonlocal Interactions: Discrete Stochastic and Continuum Modeling." *Physical Review E* 85, no. 4 (April 17, 2012). ISSN: 1539-3755, 1550-2376, accessed October 2, 2017. https://doi.org/10.1103/PhysRevE.85.041912. https://link.aps.org/doi/10.1103/PhysRevE.85.041912.

Hamann, Heiko. 2018. *Swarm Robotics: A Formal Approach.* Cham: Springer International Publishing. ISBN: 978-3-319-74526-8 978-3-319-74528-2, accessed July 15, 2019. https://doi.org/10.1007/978-3-319-74528-2. http://link.springer.com/10.1007/978-3-319-74528-2.

Hamann, Heiko, and Heinz Wörn. 2007. "An Analytical and Spatial Model of Foraging in a Swarm of Robots." In *Swarm Robotics,* edited by Erol Şahin, William M. Spears, and Alan F. T. Winfield, 4433:43–55. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-71540-5 978-3-540-71541-2, accessed October 11, 2019. https://doi.org/10.1007/978-3-540-71541-2_4. http://link.springer.com/10.1007/978-3-540-71541-2_4.

Hartmann, Vegard. 2005. "Evolving Agent Swarms for Clustering and Sorting." In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation,* 217–224. GECCO '05. New York, NY, USA: ACM. ISBN: 978-1-59593-010-1, accessed October 15, 2017. https://doi.org/10.1145/1068009.1068042. http://doi.acm.org/10.1145/1068009.1068042.

Hayes, A. T., and P. Dormiani-Tabatabaei. 2002. "Self-Organized Flocking with Agent Failure: Off-Line Optimization and Demonstration with Real Robots." In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292),* vol. 4, 3900–3905 vol.4. May. https://doi.org/10.1109/ROBOT.2002.1014331.

Hayes, Adam T., Alcherio Martinoli, and Rodney M. Goodman. 2003. "Swarm Robotic Odor Localization: Off-Line Optimization and Validation with Real Robots." *Robotica* 21, no. 4 (August): 427–441. ISSN: 1469-8668, 0263-5747, accessed October 16, 2017. https://doi.org/10.1017/S0263574703004946. https://www.cambridge.org/core/journals/robotica/article/swarm-robotic-odor-localization-offline-optimization-and-validation-with-real-robots/00A5AE0791542848945A5C9CC2527D1D.

Helbing, Dirk, Illés Farkas, and Tamás Vicsek. 2000. "Simulating Dynamical Features of Escape Panic." *Nature* 407, no. 6803 (6803): 487–490. ISSN: 1476-4687, accessed January 17, 2021. https://doi.org/10.1038/35035023. https://www.nature.com/articles/35035023.

Holland, John H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* 1st MIT Press ed. Complex Adaptive Systems. Cambridge, Mass: MIT Press. ISBN: 978-0-262-08213-6 978-0-262-58111-0.

Holliday, R. 1987. "The Inheritance of Epigenetic Defects." *Science (New York, N.Y.)* 238, no. 4824 (October 9, 1987): 163–170. ISSN: 0036-8075. pmid: 3310230.

Ito, K., and K. Xiong. 2000. "Gaussian Filters for Nonlinear Filtering Problems." *IEEE Transactions on Automatic Control* 45, no. 5 (May): 910–927. ISSN: 1558-2523. https://doi.org/10.1109/9.855552.

Jazwinski, Andrew H. 1997. *Stochastic Processes and Filtering Theory.* 12. [print]. Mathematics in Science and Engineering 64. San Diego: Acad. Press. ISBN: 978-0-12-381550-7.

Jin, Yaochu, and Yan Meng. 2011. "Morphogenetic Robotics: An Emerging New Field in Developmental Robotics." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41, no. 2 (March): 145–160. ISSN: 1094-6977, 1558-2442, accessed August 8, 2018. https://doi.org/10.1109/TSMCC.2010.2057424. http://ieeexplore.ieee.org/document/5555989/.

Julier, S.J., and J.K. Uhlmann. 2004. "Unscented Filtering and Nonlinear Estimation." *Proceedings of the IEEE* 92, no. 3 (March): 401–422. ISSN: 0018-9219, accessed September 30, 2020. https://doi.org/10.1109/JPROC.2003.823141. http://ieeexplore.ieee.org/document/1271397/.

Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. 1996. "Reinforcement Learning: A Survey." *J. Artif. Int. Res.* 4, no. 1 (May): 237–285. ISSN: 1076-9757, accessed October 2, 2017. http://dl.acm.org/citation.cfm?id=1622737.1622748.

Kalyanakrishnan, Shivaram, and Peter Stone. 2007. "Batch Reinforcement Learning in a Complex Domain," 1. ACM Press. ISBN: 978-81-904262-7-5, accessed October 2, 2017. https://doi.org/10.1145/1329125.1329241. http://portal.acm.org/citation.cfm?doid=1329125.1329241.

Kernbach, Serge, Ronald Thenius, Olga Kernbach, and Thomas Schmickl. 2009. "Re-Embodiment of Honeybee Aggregation Behavior in an Artificial Micro-Robotic System." *Adaptive Behavior* 17, no. 3 (June 1, 2009): 237–259. ISSN: 1059-7123, accessed October 1, 2019. https://doi.org/10.1177/1059712309104966. https://doi.org/10.1177/1059712309104966.

Khatib, Oussama. 1986. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." *The International Journal of Robotics Research* 5, no. 1 (March 1, 1986): 90–98. ISSN: 0278-3649, accessed March 29, 2018. https://doi.org/10.1177/027836498600500106. https://doi.org/10.1177/027836498600500106.

Koopman, Bernard Osgood. 1980. *Search and Screening: General Principles with Historical Applications.* Elmsford, N.Y: Pergamon Press. ISBN: 978-0-08-023136-5 978-0-08-023135-8.

Kuyucu, Tüze, Ivan Tanev, and Katsunori Shimohara. 2012. "Evolutionary Optimization of Pheromone-Based Stigmergic Communication." In *European Conference on the Applications of Evolutionary Computation,* 63–72. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, April 11, 2012. ISBN: 978-3-642-29177-7 978-3-642-29178-4, accessed October 15, 2017. https://link.springer.com/chapter/10.1007/978-3-642-29178-4_7.

La Cava, William, and Lee Spector. 2015. "Inheritable Epigenetics in Genetic Programming." In *Genetic Programming Theory and Practice XII,* edited by Rick Riolo, William P. Worzel, and Mark Kotanchek, 37–51. Cham: Springer International Publishing. ISBN: 978-3-319-16029-0 978-3-319-16030-6, accessed October 2, 2017. http://link.springer.com/10.1007/978-3-319-16030-6_3.

Labella, Thomas H., Marco Dorigo, and Jean-Louis Deneubourg. 2006. "Division of Labor in a Group of Robots Inspired by Ants' Foraging Behavior." *ACM Trans. Auton. Adapt. Syst.* 1, no. 1 (September): 4–25. ISSN: 1556-4665, accessed October 4, 2017. https://doi.org/10.1145/1152934.1152936. http://doi.acm.org/10.1145/1152934.1152936.

Lehman, Joel, and Kenneth O. Stanley. 2011. "Abandoning Objectives: Evolution Through the Search for Novelty Alone." *Evolutionary Computation* 19, no. 2 (June): 189–223. ISSN: 1063-6560, 1530-9304, accessed October 2, 2017. http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00025.

Lerman, Kristina, and Aram Galstyan. 2002. "Mathematical Model of Foraging in a Group of Robots: Effect of Interference." *Autonomous Robots* 13, no. 2 (September 1, 2002): 127–141. ISSN: 1573-7527, accessed October 11, 2019. https://doi.org/10.1023/A:1019633424543. https://doi.org/10.1023/A:1019633424543.

Li, Ling, Alcherio Martinoli, and Yaser S. Abu-Mostafa. 2004. "Learning and Measuring Specialization in Collaborative Swarm Systems." *Adaptive Behavior* 12, nos. 3-4 (December 1, 2004): 199–212. ISSN: 1059-7123, accessed October 16, 2017. https://doi.org/10.1177/105971230401200306. https://doi.org/10.1177/105971230401200306.

Liu, Wenguo, Alan F. T. Winfield, Jin Sa, Jie Chen, and Lihua Dou. 2007. "Towards Energy Optimization: Emergent Task Allocation in a Swarm of Foraging Robots." *Adaptive Behavior* 15, no. 3 (September 1, 2007): 289–305. ISSN: 1059-7123, accessed October 4, 2017. https://doi.org/10.1177/1059712307082088. https://doi.org/10.1177/1059712307082088.

Mahfoud, S. 1997. "Niching Methods." In *Handbook of Evolutionary Computation,* edited by Thomas Bäck, David B. Fogel, and Zbigniew Michalewics, C6.1:1–C6.1:4. Bristol ; Philadelphia : New York: Institute of Physics Pub. ; Oxford University Press. ISBN: 978-0-7503-0392-7.

Mayr, Ernst. 1972. "Lamarck Revisited." *Journal of the History of Biology* 5 (1): 55–94. ISSN: 0022-5010. JSTOR: 4330569.

Minsky, Marvin. 1967. *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice-Hall. ISBN: 978-0-13-165563-8.

Mishra, Rahul Shivnarayan, Tushar Semwal, and Shivashankar B. Nair. 2018. "A Distributed Epigenetic Shape Formation and Regeneration Algorithm for a Swarm of Robots." In *Proceedings of the Genetic and Evolutionary Computation Conference Companion,* 1505–1512. GECCO '18. New York, NY, USA: ACM. ISBN: 978-1-4503-5764-7, accessed September 20, 2019. https://doi.org/10.1145/3205651.3208300. http://doi.acm.org/10.1145/3205651.3208300.

Moeslinger, Christoph, Thomas Schmickl, and Karl Crailsheim. 2010. "Emergent Flocking with Low-End Swarm Robots." In *Swarm Intelligence,* edited by Marco Dorigo, Mauro Birattari, Gianni A. Di Caro, René Doursat, Andries P. Engelbrecht, Dario Floreano, Luca Maria Gambardella, et al., 424–431. Lecture Notes in Computer Science. Springer Berlin Heidelberg. ISBN: 978-3-642-15461-4.

Mogilner, Alexander, and Leah Edelstein-Keshet. 1999. "A Non-Local Model for a Swarm." *Journal of Mathematical Biology* 38, no. 6 (June 23, 1999): 534–570. ISSN: 0303-6812, 1432-1416, accessed October 2, 2017. https://doi.org/10.1007/s002850050158. http://link.springer.com/10.1007/s002850050158.

Mukhlish, Faqihza, John Page, and Michael Bain. 2018a. "Evolutionary-Learning Framework for Swarm Robotics Using Epigenetics Layer," vol. 14, P86. Jeju, South Korea: International Society of Intelligent Unmanned System.

———. 2018b. "Evolutionary-Learning Framework: Improving Automatic Swarm Robotics Design." *International Journal of Intelligent Unmanned Systems* 6, no. 4 (October 8, 2018): 197–215. ISSN: 2049-6427, accessed October 22, 2018. https://doi.org/10.1108/IJIUS-06-2018-0016. https://www.emeraldinsight.com/doi/10.1108/IJIUS-06-2018-0016.

———. 2019. "Evolving Swarm for Search and Rescue Mission: Potential, Development and Risk." Beijing, August.

Mukhlish, Faqihza, John Page, and Michael Bain. 2020a. "Reward-Based Epigenetic Learning Algorithm for a Decentralised Multi-Agent System." *International Journal of Intelligent Unmanned Systems* 8, no. 3 (April 13, 2020): 201–224. ISSN: 2049-6427, accessed February 10, 2021. https://doi.org/10.1108/IJIUS-12-2018-0036. https://www.emerald.com/insight/content/doi/10.1108/IJIUS-12-2018-0036/full/html.

———. 2020b. "Reward-Based Epigenetic Learning Algorithm for a Decentralised Multi-Agent System." *International Journal of Intelligent Unmanned Systems* ahead-of-print (ahead-of-print 2020). ISSN: 2049-6427, accessed May 3, 2020. https://doi.org/10.1108/IJIUS-12-2018-0036. https://doi.org/10.1108/IJIUS-12-2018-0036.

Nash, John F. 1950. "Equilibrium Points in N-Person Games." *Proceedings of the National Academy of Sciences* 36, no. 1 (January 1, 1950): 48–49. ISSN: 0027-8424, 1091-6490, accessed April 9, 2018. https://doi.org/10.1073/pnas.36.1.48. pmid: 16588946. http://www.pnas.org/content/36/1/48.

Organization, International Maritime, and International Civil Aviation Organization, eds. 2016. *International Aeronautical and Maritime Search and Rescue Manual: IAMSAR Manual. Volume 1: Organization and Management.* 2016 edition, tenth edition. IMO Publication. London: International Maritime Organization. ISBN: 978-92-801-1639-7.

Ostergaard, Esben H., Gaurav S. Sukhatme, and Maja J. Matari. 2001. "Emergent Bucket Brigading: A Simple Mechanisms for Improving Performance in Multi-Robot Constrained-Space Foraging Tasks." In *Proceedings of the Fifth International Conference on Autonomous Agents,* 29–30. AGENTS '01. New York, NY, USA: ACM. ISBN: 978-1-58113-326-4, accessed October 11, 2019. https://doi.org/10.1145/375735.375825. http://doi.acm.org/10.1145/375735.375825.

Page, John, Robert Armstrong, and Faqihza Mukhlish. 2019. "Simulating Search and Rescue Operations Using Swarm Technology to Determine How Many Searchers Are Needed to Locate Missing Persons/Objects in the Shortest Time." In *Intersections in Simulation and Gaming: Disruption and Balance,* edited by Anjum Naweed, Lorelle Bowditch, and Cyle Sprick, 1067:106–112. Communications in Computer and Information Science. Singapore: Springer Singapore. ISBN: 978-981-329-581-0 978-981-329-582-7, accessed February 4, 2021. https://doi.org/10.1007/978-981-32-9582-7_8. http://link.springer.com/10.1007/978-981-32-9582-7_8.

Pathak, Jaideep, Zhixin Lu, Brian R. Hunt, Michelle Girvan, and Edward Ott. 2017. "Using Machine Learning to Replicate Chaotic Attractors and Calculate Lyapunov Exponents from Data." *Chaos: An Interdisciplinary Journal of Nonlinear Science* 27, no. 12 (December): 121102. ISSN: 1054-1500, 1089-7682, accessed April 30, 2018. https://doi.org/10.1063/1.5010300. arXiv: 1710.07313. http://arxiv.org/abs/1710.07313.

Periyasamy, Sathish, Alex Gray, and Peter Kille. 2008. "The Epigenetic Algorithm." In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence),* 3228–3236. Hong Kong, China: IEEE, June 1–6, 2008. ISBN: 978-1-4244-1822-0, accessed October 2, 2017. https://doi.org/10.1109/CEC.2008.4631235. https://ieeexplore.ieee.org/document/4631235.

Pini, Giovanni, Arne Brutschy, Carlo Pinciroli, Marco Dorigo, and Mauro Birattari. 2013. "Autonomous Task Partitioning in Robot Foraging: An Approach Based on Cost Estimation." *Adaptive Behavior* 21, no. 2 (April): 118–136. ISSN: 1059-7123, 1741-2633, accessed October 11, 2019. https://doi.org/10.1177/1059712313484771. http://journals.sagepub.com/doi/10.1177/1059712313484771.

Priolo, Attilio. 2013. "Swarm Aggregation Algorithms for Multi-Robot Systems." Doctor of Philosophy, Engineering Departement, Faculty of Computer Science Engineering, University of Roma Tre.

Ranjbar-Sahraei, Bijan, Gerhard Weiss, and Ali Nakisaee. 2012. "A Multi-Robot Coverage Approach Based on Stigmergic Communication." In *Multiagent System Technologies,* 126–138. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, October 10, 2012. ISBN: 978-3-642-33689-8 978-3-642-33690-4, accessed August 10, 2018. https://doi.org/10.1007/978-3-642-33690-4_13. https://link.springer.com/chapter/10.1007/978-3-642-33690-4_13.

Ren, HaiGang, YingZheng Liu, and HanPin Chen. 2006. "Calculation of Routh Sea Surface Reflection." *International Journal of Infrared and Millimeter Waves* 27, no. 7 (July 1, 2006): 1019–1026. ISSN: 0195-9271, 1572-9559, accessed March 4, 2018. https://doi.org/10.1007/s10762-006-9087-6. https://link.springer.com/article/10.1007/s10762-006-9087-6.

Reynolds, Craig W. 1987. "Flocks, Herds and Schools: A Distributed Behavioral Model." *ACM SIGGRAPH Computer Graphics* 21, no. 4 (August 1, 1987): 25–34. ISSN: 00978930, accessed October 2, 2017. https://doi.org/10.1145/37402.37406. http://portal.acm.org/citation.cfm?doid=37402.37406.

Riedmiller, Martin, Thomas Gabel, Roland Hafner, and Sascha Lange. 2009. "Reinforcement Learning for Robot Soccer." *Autonomous Robots* 27, no. 1 (July): 55–73. ISSN: 0929-5593, 1573-7527, accessed October 2, 2017. https://doi.org/10.1007/s10514-009-9120-4. http://link.springer.com/10.1007/s10514-009-9120-4.

Rongier, P., and A. Liegeois. 1999. "Analysis and Prediction of the Behavior of One Class of Multiple Foraging Robots with the Help of Stochastic Petri Nets." In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028),* vol. 5, 143–148 vol.5. October. https://doi.org/10.1109/ICSMC.1999.815537.

Şahin, Erol. 2005. "Swarm Robotics: From Sources of Inspiration to Domains of Application." In *Swarm Robotics,* edited by Erol Şahin and William M. Spears, 3342:10–20. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-24296-3 978-3-540-30552-1, accessed August 8, 2018. http://link.springer.com/10.1007/978-3-540-30552-1_2.

Sareni, B., and L. Krahenbuhl. 1998. "Fitness Sharing and Niching Methods Revisited." *IEEE Transactions on Evolutionary Computation* 2 (3): 97–106. ISSN: 1089778X, accessed February 26, 2018. https://doi.org/10.1109/4235.735432. http://ieeexplore.ieee.org/document/735432/.

Schmickl, Thomas, Heiko Hamann, Heinz Wörn, and Karl Crailsheim. 2009. "Two Different Approaches to a Macroscopic Model of a Bio-Inspired Robotic Swarm." *Robotics and Autonomous Systems* 57, no. 9 (September 30, 2009): 913–921. ISSN: 0921-8890, accessed October 2, 2019. https://doi.org/10.1016/j.robot.2009.06.002. http://www.sciencedirect.com/science/article/pii/S0921889009000815.

Schroeder, Adam, Subramanian Ramakrishnan, Manish Kumar, and Brian Trease. 2017. "Efficient Spatial Coverage by a Robot Swarm Based on an Ant Foraging Model and the Lévy Distribution." *Swarm Intelligence* 11, no. 1 (March): 39–69. ISSN: 1935-3812, 1935-3820, accessed October 2, 2017. https://doi.org/10.1007/s11721-017-0132-y. http://link.springer.com/10.1007/s11721-017-0132-y.

Sen, Sandip, and Gerhard Weiss. 1999. "Learning in Multiagent Systems." In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,* edited by Gerhard Weiss, 259–298.

Shakhatreh, Hazim, Ahmad Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. 2019. "Unmanned Aerial Vehicles: A Survey on Civil Applications and Key Research Challenges." *IEEE Access* 7:48572–48634. ISSN: 2169-3536, accessed February 4, 2021. https://doi.org/10.1109/ACCESS.2019.2909530. arXiv: 1805.00881. http://arxiv.org/abs/1805.00881.

Sousa, Jorge A. B., and Ernesto Costa. 2011. "Designing an Epigenetic Approach in Artificial Life: The EpiAL Model." In *Agents and Artificial Intelligence,* edited by Joaquim Filipe, Ana Fred, and Bernadette Sharp, 129:78–90. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-19889-2 978-3-642-19890-8, accessed October 2, 2017. http://link.springer.com/10.1007/978-3-642-19890-8_6.

Soysal, O., and E. Şahin. 2005. "Probabilistic Aggregation Strategies in Swarm Robotic Systems." In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005,* 325–332. Pasadena, CA, USA: IEEE. ISBN: 978-0-7803-8916-8, accessed October 2, 2017. https://doi.org/10.1109/SIS.2005.1501639. http://ieeexplore.ieee.org/document/1501639/.

Spears, William M., Diana F. Spears, Jerry C. Hamann, and Rodney Heil. 2004. "Distributed, Physics-Based Control of Swarms of Vehicles." *Autonomous Robots* 17, nos. 2-3 (September 1, 2004): 137–162. ISSN: 0929-5593, 1573-7527, accessed March 29, 2018. https://doi.org/10.1023/B:AURO.0000033970.96785.f2. https://link.springer.com/article/10.1023/B:AURO.0000033970.96785.f2.

Spezzano, Giandomenico, ed. 2019. *Swarm Robotics.* MDPI, May 13, 2019. ISBN: 978-3-03897-923-4, accessed October 1, 2019. https://doi.org/10.3390/books978-3-03897-923-4. http://www.mdpi.com/books/pdfview/book/1294.

Stolfi, Daniel H., and Enrique Alba. 2018. "Epigenetic Algorithms: A New Way of Building GAs Based on Epigenetics." *Information Sciences* 424 (January 1, 2018): 250–272. ISSN: 0020-0255, accessed April 23, 2018. https://doi.org/10.1016/j.ins.2017.10.005. http://www.sciencedirect.com/science/article/pii/S0020025517309921.

Stone, Lawrence D. 1989a. "OR Forum—What's Happened in Search Theory Since the 1975 Lanchester Prize?" *Operations Research* 37, no. 3 (June): 501–506. ISSN: 0030-364X, 1526-5463, accessed September 30, 2020. https://doi.org/10.1287/opre.37.3.501. http://pubsonline.informs.org/doi/abs/10.1287/opre.37.3.501.

———. 1989b. *Theory of Optimal Search.* 2nd ed. Arlington, Va: Military Applications Section, Operations Research Society of America. ISBN: 978-1-877640-00-1.

Stone, Peter, Richard S. Sutton, and Gregory Kuhlmann. 2005. "Reinforcement Learning for RoboCup Soccer Keepaway." *Adaptive Behavior* 13, no. 3 (September 1, 2005): 165–188. ISSN: 1059-7123, accessed October 16, 2017. https://doi.org/10.1177/105971230501300301. https://doi.org/10.1177/105971230501300301.

Stone, Toby, ed. 2018. *National Search and Rescue Manual.* 1st ed. Canbera, ACT, Australia: Australian Maritime Safety Authority (AMSA), September 1, 2018. Accessed October 3, 2018. https://natsar.amsa.gov.au/documents/NATSAR-Manual/NATSAR-Manual-Sept-2018.pdf.

Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. ISBN: 978-0-262-19398-6.

Tan, Ying, ed. 2016. *Handbook of Research on Design, Control, and Modeling of Swarm Robotics:* redacted by Ivan Giannoccaro. Advances in Computational Intelligence and Robotics. IGI Global. ISBN: 978-1-4666-9572-6 978-1-4666-9573-3, accessed June 10, 2021. https://doi.org/10.4018/978-1-4666-9572-6. http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-9572-6.

Tanev, I, and K Yuta. 2008. "Epigenetic Programming: Genetic Programming Incorporating Epigenetic Learning through Modification of Histones." *Information Sciences* 178, no. 23 (December 1, 2008): 4469–4481. ISSN: 00200255, accessed October 2, 2017. https://doi.org/10.1016/j.ins.2008.07.027. http://linkinghub.elsevier.com/retrieve/pii/S0020025508002880.

Trianni, Vito, Roderich Groß, Thomas H. Labella, Erol Şahin, and Marco Dorigo. 2003. "Evolving Aggregation Behaviors in a Swarm of Robots." In *Advances in Artificial Life,* 865–874. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. Accessed October 15, 2017. https://link.springer.com/chapter/10.1007/978-3-540-39432-7_93.

Tuci, Elio, Christos Ampatzis, Vito Trianni, Anders Lyhne Christensen, and Marco Dorigo. 2008. "Self-Assembly in Physical Autonomous Robots-the Evolutionary Robotics Approach." In *ALIFE,* 616–623. Accessed October 15, 2017. http://iridia0.ulb.ac.be/IridiaTrSeries/IridiaTr2008-007r001.pdf.

Turgut, Ali E., Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. 2008. "Self-Organized Flocking in Mobile Robot Swarms." *Swarm Intelligence* 2, no. 2 (December 1, 2008): 97–120. ISSN: 1935-3820, accessed October 10, 2019. https://doi.org/10.1007/s11721-008-0016-2. https://doi.org/10.1007/s11721-008-0016-2.

Vanualailai, Jito, and Bibhya Sharma. 2010. "A Lagrangian-Based Swarming Behavior in the Absence of Obstacles." In *Workshop on Mathematical Control Theory, Kobe University,* 8–10. Accessed October 2, 2017. http://www.research.kobe-u.ac.jp/csi-applmath/proc/workshopKobe/final/Section12.pdf.

Virágh, Csaba, Gábor Vásárhelyi, Norbert Tarcai, Tamás Szörényi, Gergő Somorjai, Tamás Nepusz, and Tamás Vicsek. 2014. "Flocking Algorithm for Autonomous Flying Robots." *Bioinspiration & Biomimetics* 9, no. 2 (May): 025012. ISSN: 1748-3190, accessed October 10, 2019. https://doi.org/10.1088/1748-3182/9/2/025012. https://doi.org/10.1088%2F1748-3182%2F9%2F2%2F025012.

Waddington, C. H. 2012. "The Epigenotype." *International Journal of Epidemiology* 41, no. 1 (February): 10–13. ISSN: 0300-5771, 1464-3685, accessed October 2, 2017. https://doi.org/10.1093/ije/dyr184. https://academic.oup.com/ije/article-lookup/doi/10.1093/ije/dyr184.

Wang, Yan, Huijie Liu, and Zhongsheng Sun. 2017. "Lamarck Rises from His Grave: Parental Environment-Induced Epigenetic Inheritance in Model Organisms and Humans: Parental Experience-Induced Epigenetic Inheritance." *Biological Reviews* (February). ISSN: 14647931, accessed October 2, 2017. https://doi.org/10.1111/brv.12322. http://doi.wiley.com/10.1111/brv.12322.

Wheeler, William Morton. 1910. *Ants: Their Structure, Development and Behaviour.* 1st ed. Columbia University Biological Series IX. New York: The Columbia University Press. ISBN: 978-1-120-26274-5.

Yan, I., and G.L. Blankenship. 1988. "Numerical Methods in Search Path Planning." In *Proceedings of the 27th IEEE Conference on Decision and Control,* 1563–1569 vol.2. December. https://doi.org/10.1109/CDC.1988.194592.

Yasuda, T., A. Adachi, and K. Ohkura. 2014. "Self-Organized Flocking of a Mobile Robot Swarm by Topological Distance-Based Interactions." In *2014 IEEE/SICE International Symposium on System Integration,* 106–111. December. https://doi.org/10.1109/SII.2014.7028020.

Yi, Xin, Anmin Zhu, Simon X. Yang, and Chaomin Luo. 2017. "A Bio-Inspired Approach to Task Assignment of Swarm Robots in 3-D Dynamic Environments." *IEEE Transactions on Cybernetics* 47, no. 4 (April): 974–983. ISSN: 2168-2267, 2168-2275, accessed October 2, 2017. https://doi.org/10.1109/TCYB.2016.2535153. http://ieeexplore.ieee.org/document/7434030/.