Department of Informatics King's College London

6CCS3PRJ FINAL PROJECT REPORT

Topic: Robot Area Coverage

Supervisor: Prof. Simon Parsons

Author: Benjamin Buckley

KCL ID: 1328620

Originality Avowal

I, Benjamin Edward Buckley, hereby certify that I understand the nature of plagiarism and collusion, and that I am the sole author of this report, except where explicitly stated to the contrary. (25/04/2016)

Abstract

The intention of this project is to assess the different methods used to enable robots to traverse different environments at the same time as providing maximal area coverage. This includes: investigating current techniques of area coverage, implementing them in using agent-based modelling techniques, evaluating results after a period of experimentation and presenting the findings in a logical manner. More specifically, this will include producing an appropriate simulation system using the development environment NetLogo to test a number of different methods used in the research of multi-robot area coverage.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In the framework of my final year project, we shall attempt to make a contribution to the research of robot area coverage. Robot area coverage is an important task in mobile robotics. Indeed, the ideas of having a robot or team of robots move through an environment, whilst maximising total area covered in an efficient and structured manner is a well-researched topic.

Throughout the course of this project we will identify and review some of the current methods of robot area coverage. We will examine how robots can work as a single entity and in teams to cover an area to its maximum extent. After reviewing existing techniques, we will decide on their suitability and choose those perceived to be most relevant for the purposes of our project. From this we will attempt to implement them in the form of a series of software simulations. After several experiments, we will review the findings and compare how well each of the methods perform. The purpose of this report is to provide a complete description of every step taken as progress is made in regards to this project.

1.1 Motivation

The whole concept of the project is exceedingly relevant in today's world of fast-paced/constant technological advancement. Robots are becoming increasingly common nowadays and I feel the project will provide me with a good understanding of how they are able to perform different tasks in real world situations, as well as the algorithms and methods behind them.

What I am especially interested in is the altruistic use of computer science and robotics or in other words, how can robotics help and be useful to humanity and the planet. Aiding humans accomplish important tasks that would normally be too difficult or at times impossible is one such context where technology is indispensable. For example in the area of environmental monitoring and security, robots are already occupying roles that may be too hazardous for their human counterparts. One instance of this is in the event of a natural disaster such as the tsunami and following earthquake which occurred on the Fukushima power plant. Tokyo Electric Power Company Holdings and the International Research Institute for Nuclear Decommissioning have sent remote control mobile robots into the facility to remove the fuel rods which would have been a fatal mission if handled by a human [1]. Having mobile robots able to traverse treacherous terrains such as these to reach certain goals and take actions accordingly (or just report back any appropriate information) is exceedingly useful.

Another related topic in which I am extremely interested and in which I wish to engage in for future research is how computer science and robotics can help to alleviate the negative consequences of global warming and, more generally, how technology can help humanity achieve environmental sustainability. An exciting example of this is the first fully automated sustainable farm, created by a Japanese firm, named Spread that will open in 2017 [2].

Therefore, this project, by allowing me to further my knowledge in this field, is a small stepping stone towards these desired goals.

1.2 Scope

Overall, the scope of robot area coverage is vast. The techniques learnt from this can be implemented into numerous different applications, whether it is for networking, military use or waste management purposes. These are but a few among many other uses.

The actual scope of this project will include exploring the different methods that can be put in place to enable robots to work together and negotiate their surroundings as a team. This comes from the agents having no prior knowledge of a set map and having them work together to maximize their combined area of influence. Thus the scope of our report will include: testing the different methods of communication the agents can use, different movement techniques and comparing the efficiency between these methods, based on the time taken to converge and the total area covered when they converge.

1.3 Aims and Objectives

Throughout this project, we will aim to develop a better understanding of how robots can be used to traverse different environments to accomplish their assigned tasks, while maximizing the total area covered. Therefore, we will develop an understanding of how robots are able to work in multi-agent teams to negotiate an area and examine the different methods they use to communicate. Then, we will implement and test these different communication methods, analyzing the results from each of them. We will also compare multiple movement techniques involved with getting the agents to navigate through the environments; analyzing the strengths and weaknesses of each.

1.4 Report Structure

This report will start by providing an overview of the background behind this topic, where we will discuss why this subject is important as a whole. We will then proceed to a review of the existing literature which exhibit some of the techniques that we will endeavour to implement and test. This will also give a brief overview on how some of the methods can be used in real world scenarios.

Secondly, we will discuss the requirements and specification of the project. The former will outline what is tasked to be achieved by the end and what the final findings should include, both in the forms of software and the content produced from multiple tests. This chapter will also give a detailed and complete specification of the functionality of each algorithm that is to be implemented. It will put forward some questions that need to be answered in order to implement appropriate tests to assess the effectiveness of coverage and examine the thought process behind them. This will include some of the possible limitations that we may encounter throughout the duration of the project.

Next, the report will cover all the design aspects of the project. This includes: detailing the data flow of the system, providing a definition for the different parts of the graphical user interface and explain how my experimental results will be presented.

The following chapter will follow the different stages of implementation in a chronological fashion. It will detail the various different stages within the software development cycle, addressing the problems that were faced during each of them and the solutions that were put in place.

Subsequently, the experimentation phase of the project will be outlined. This chapter will detail the experimentation methods and parameters that were used, as well as present the data that was collected from each of the algorithms. The chapter following this will provide an evaluation of the software development and experimentation. It will analyse the strengths and weakness of the methods that have been implemented and provide a comparison.

The penultimate chapter will briefly discuss any professional issues that needed to be recognized and adhered to throughout the course of the project.

Finally, the concluding section of the report will provide a summary of our findings as well as an assessment of the contribution of our report, before outlining any possible future work.

Chapter 2: Background

In this section of the report we will provide a general background of the topic. Firstly, we shall briefly discuss the pertinence of research in this topic Then, we will move on to discuss some of the current methods used in the topic of robot area coverage. It will provide a review of some of the research papers read and evaluate the techniques discussed in each of them.

2.1 Why is research into this topic important?

We have briefly expressed a few reasons as to why this topic is important in today's society but we shall reiterate and build upon these.

The use of robots has significantly increased over recent years and the subject of robotics as a whole has become a much higher priority on the list of technological advancement, as we progress into the future. With this increase in interest it is evident that the topic of robot area coverage will come into play. All mobile robots incorporate some technique and algorithm to navigate its environment to report back various properties (e.g. location, temperature, radiation levels, etc).

Robot area coverage is an important research topic because the methods developed may serve a wide range of different applications, possible in many different scenarios. We have mentioned above search and rescue operations, natural disaster recovery, environmental monitoring, security, but other purposes like networking, military operations, assistive technology in industrial operations are equally essential.

One instance of where area coverage techniques are being used in mobile robotics is with Google's Project Loon. The project involves the use of high-altitude balloons being used to provide wireless signal to rural and remote areas and improve communication during natural disasters, when balloons have to adjust their altitude and float to a specified wind layer. The criteria to determine these layers includes desired speed and direction using wind data. Furthermore, the algorithms employed move each balloon to their specified wind layer and provides motion control to move with the wind. The balloons then need to provide the best coverage possible to the ground this means they need to be spaced correctly, leaving as few blank spots as possible [3].

Another fascinating case is that of the Centre for robot-assisted search and rescue: it has been dealing with concerns of disaster recovery since the infamous terrorist attack of 9/11. They have been creating robots that assist in the recovery operations during and after hurricanes, earthquakes, tsunamis and mudslides. The robots they create are autonomous and can maneuver on land, sea or air [4]. We have but listed a few reasons and examples to illustrate the pertinence of research in this field, but we reiterate our firm conviction that technological progress in this area can have decisive consequences beyond the mere realm of computer science and robotics. Now, we move on to a brief literature review of current methods for Robot Area Coverage.

2.2 Current Methods for Robot Area Coverage

The research being put into this topic is extensive and still ongoing. Many research organizations and universities have presented a number of different methods for efficient area coverage, either focusing on a specific domain within the topic, with their own set of parameters and test environments or improving upon existing techniques. The two main scenarios which have received scholarly attention thus far are navigating between known and unknown territories. Research has also been conducted in complex methods which take into account the possibility of obstacles.

Throughout our project, one of the tasks was to discover different methods proposed in scientific papers, articles, textbooks and existing software solutions. Therefore, the bulk of our work will be based on these existing solutions. We will now turn to a detailed review of the literature that was deemed relevant towards the project.

2.2.1 Behavior-based Formation Control for Multi-robot Teams [5]

The first paper we will review is an older publication by Balch and Arkin in 1999 which still contains relevant information. It discusses a number of different methodologies to enable a team of autonomous vehicles to navigate through different environments. The authors of the paper implemented formations in multi-robot teams. These methods are then united with other navigational behaviors, such as reaching goal states and avoiding obstacles, whilst attempting to maintain formation of the group. Balch and Arkin's work presented was intended to emulate the various types of formations used in human-led and communications-restricted applications.

It presents a number of different formations, that it adheres to throughout implementation and testing of each procedure. The formations used are as follows:

- Line: all robots form a single-file, horizontal line and travel line-abreast.
- Column: all robots form a single-file, vertical line (column) and travel one after the other.
- Diamond: the robots form and travel in a diamond shape.
- Wedge: the robots form and travel in a “V” shape.

To maintain the formation it is split into two steps: first, the robot must detect its proper formation position; second, it must attempt to maintain this position as the group moves throughout the environment.

In order to compute a robots proper position within a formation three techniques were tried and tested. The techniques used are as follows:

- Unit-center-referenced: the unit-center of the group is computed independently by each of the robots by taking an average of the x and y coordinates of all other robots involved in the formation. Each robot then defines its own position relative to the center.
- Leader-referenced: each of the robot's base their formation positions in relation to a lead robot (The lead robot will be picked according to the formation). The robots then attempt to remain in these positions as the leader moves throughout the environment (The leader does not maintain formation).
- Neighbour-referenced: The robots attempt to maintain position in relation to another neighbouring robot.

Once the proper positions of each of the robots are known, the team can begin to take action to move through its environment. Throughout this entire process, the robots attempt to maintain their formation positions by gathering their own position with respect to their team members or a centralised reference point and through the use of a motor schema-based formation control. There are several motor schemas which are used, **move-to-goal**, **avoid-static-obstacle**, **avoid-robot**, **maintain-formation** and **noise**. These schemas are used in conjunction to control the robots into reaching a final goal state, whilst avoiding obstacles, other robots and maintaining their positions within the formation. Each schema generates a behavioral vector (direction and magnitude of movement) based on its next desired state. These vectors are weighted to construct the importance of the next action. Firstly, the **noise** schema is used to filter out any problems created by the navigational methods. The **maintain-formation** schema creates a vector that gives indications to the robot to move toward the desired location within the formation. The intensity of the values generated within this vector is directly reflected by the distance from the desired position. There are three zones for generating the magnitude of the vector. These zones are:

- Ballistic Zone: this is the maximum deviation from the desired locations. The magnitude is set to the maximum, in turn giving it a higher weight,
- Controlled Zone: this area generates a linear magnitude vector ranging from the edges of the Ballistic and Dead Zones.
- Dead Zone: this is in the bounds of the desired location to maintain formation position. The magnitude of the vector is set to zero at this point.

Although the methods described in this paper produce some interesting results, we do not believe there is enough relevant material for us to attempt to implement formation based methods in our work. We may however consider some of the ideas presented here for future work, for example deciding how to prioritize the actions taken by agents and how to determine goal states.

2.2.2 Hybrid Insect-Inspired Multi-Robot Coverage in Complex Environments [6]

The second paper we will review discusses the techniques used by insect-inspired coordination to improve upon the performance, energy consumption, robustness and reliability of multi-agent robotic systems to perform complex tasks. Written by Broecker, Caliskanelli, Tuyls, Sklar and Hennes, the paper is about developing a new method of navigation to improve upon the effectiveness of previous solutions which consist of ant and bee inspired methods and achieves this by applying simple heuristics.

The ant inspired method or *StiCo*, uses an indirect method of communication within the agents to provide efficient area coverage. The robots using this method move in a circular fashion, and through the use of a virtual substance, which is emitted by each of them provides a method of communication between them. This trace acts as a repellant to other robots and tells them to alter their course accordingly. The circular movement of the robots is defined by a predefined magnitude. When the trace of another agent is detected by the interior sensor, the direction of the circular motion is reversed until no longer visible. When the trace of another agent is detected by the exterior sensor, the magnitude of the circular motion is increased. Due to the fact that the robots are moving in a constant circular motion, they can take an extended period of time to converge. The method also suffers from a lack of coverage redundancy.

The bee inspired method or *BeePCo*, however, uses a direct method of communication. The agents send out a signal notifying its neighbours of its current location within the environment. If a neighbouring agent receives this signal, it can then make a decision to move in the opposite direction from where it originated. The movement of the agents is vector based instead of circular. The movement vector consists of a direction and a magnitude. When the agents drift too far apart (out of range of neighbouring signals) the agent will cease moving.

The newly proposed method in this paper is a hybrid of the two previous methods. This is in the attempt to overcome the fragilities of each but maintain their respective effectiveness . This hybrid approach begins each process with the techniques by *BeePCo* but dynamically changes to the *StiCo* approach when they no longer receive a signal from any neighbouring agents. Once they are again within a distance as to transmit a signal, they revert back to *BeePCo*.All the simulated tests were attempted with predetermined obstacles in place. The agents however have no prior knowledge of their environment, therefore have to attempt to navigate around them, whilst continuing to maintain effective coverage.

Indeed, the methods discussed in the paper written by Broecker, Caliskanelli, Tuyls, Sklar and Hennes are exceedingly relevant to the project and we fully acknowledge that this will provide us a basis as to how to begin planning and implementing all work. We hope to test the benefits of biologically inspired methods, direct versus indirect communication and the different movement techniques involved.

2.2.3 Biologically Inspired Methods

As part of the lecture course, we undertook a module that pertains to creating a solution to the problem of maximizing the total area covered by a team of agents. Biologically Inspired Methods introduced a number of different computational methods that can be used in finding a solution or multiple solutions to optimization problems. A key issue to the problem at hand is to have the *agent system* navigate its environment in an efficient manner, whilst still providing reasonable coverage. This can be translated to an optimization problem and typically one that can be solved through the use of population based methods.

The topics that were introduced throughout the course of this module are as follows:

- Optimization concepts: this defined some of the key concepts and definitions of optimisation and showed how the traditional optimisation methods work. Detailing how to define the properties of a problems and how to formulate it as an optimization problem [7].
- Binary and continuous genetic algorithm: these algorithms are a subclass of Evolutionary Computing and are based on Charles Darwin's theory of evolution. Both GAs follow the same steps but the BGA has to encode the decision variables to binary, while CGA takes the actual values. Stages of each of the GAs are as follows: Evaluate decision variables and cost function, create initial population, use natural selection to remove solutions with higher cost, selection of parents is made from remaining solutions for crossover breeding, mutations are made and convergence is checked. GAs tends to a global optimum and can deal with large numbers of decision variables [8] [9].
- Evolution strategies: they use real-valued parameters and consider both genotypic and phenotypic evolution. Individuals are represented by genetics and strategy parameters and evolution of the population is based on strategy parameters and mutation (if successful). The stages of ES are as follows: initialise population, select parents and perform crossover breeding, mutate offspring, assess quality of the solution, use selection to check which individuals will survive next generation [10].
- Ant colony optimization: This swarm intelligence based method is flexible, robust, decentralised and self-organising. It uses stigmergy, a form of indirect communication between individuals which is conducted through the use of pheromone propagation and evaporation. These methods are generally used for path finding. It does this through the use of pheromones which attract ants along a certain path. The greater the number of ants moving along a path, the higher the propagation of the pheromone, thus the bigger the attraction. Then, the optimal path will be selected as the one with the most ants along it [11]. As stated previously, we will be taking inspiration from the *StiCo* algorithm which is based on this method of ant colony optimisation, with a crucial difference being that the pheromone trails act as a repellant.

- Particle swarm optimization: this swarm intelligence based method uses the principles of Coordinated Collective Behaviour: separation from neighbours, alignment to neighbours (if too far away) and cohesion (move towards an average position of neighbours). This method tends to converge towards either a local or global best solution. The velocity of an individual is based on the previous velocity, a cognitive component to attract to a personal best solution and a social component to attract to either a local or global best solution. Individual positions are updated based on current position and velocity [12]. The techniques used in this method are similar to those used in the *BeePCo* algorithm, which I will be attempting to implement.

Any one of these population based approaches falls under the scope of our project's requirements However, we have thus far only studied how they can be applied in terms of reaching a goal state. Further research would be required on our behalf to fully grasp how these methods can be implemented in a manner necessary to devise a solution to the project.

2.2.3 Cell Decomposition

Cell decomposition is one of the most extensively studied motion planning techniques. These methods are used to decompose the free space of the environment into separate regions, called cells, so that agents can move between them. These methods can be broken down further:

- Exact cell decomposition: this method assumes that the world is polygonal. It begins by creating vertices from each vertex until an obstacle is hit. This reduces the world to a union of trapezoidal-shaped cells. With the world converted to trapezoidal regions it mean that it can easily be covered with a back-and-forth sweeping motion [13].
- Approximate cell decomposition: for a 2D environment it uses quadtree decomposition. This method iteratively subdivides each (sub)region into 4 (more)equal sized regions. If the entire interior of a region is in free space then you can stop dividing. A predefined resolution can also be set, that once met will stop dividing the dividing process [13].

The method of exact cell decomposition provides a good basis of how the environment can be handled. We will need to conduct further research into which technique would be best suited to allow a team of robots to navigate through an environment if it was to be implemented.

Chapter 3: Requirements and Specification

In this section we will discuss the overall requirements for the project and put forward a detailed list of specifications to which I will adhere when developing any solution for the proposed problem. The purpose of the project is to develop a software simulation that will accurately represent the process that a multi-agent team takes to navigate an environment, in an attempt to maximize their combined coverage.

3.1 Requirements

The requirements for this project can be split up into to two sections. From these sections it is possible to split them up further into more specific subsections. The first two sections are: software and content requirements.

3.1.1 Software Requirements

The software requirements of the project can be split into three parts. The front end to manage the running of the application, the model of the environment and the agents.

3.1.1.1 Front End

- The front end must be able to initialise the environment and agents.
- The front end must be able to run and stop the model.
- The front end must be able to output suitable graphs and data, indicating appropriate measurements for testing efficiency of the methods in practice.
- The front end must have controls for configuring the entities within the models and certain variables.

3.1.1.2 Environment

- The environment must be able to represent a somewhat realistic space.
- The environment must allow for definitions of different area types.
- The randomly generated environments must appropriately place obstacles and select starting positions for the agents, as not to impede the entire flow of the simulation.

- The predetermined environments must be structured in a suitable and unique way to provide an exhaustive series of tests for the agents.

3.1.1.3 Agents

- The agents must be able to move around the environment in accordance to the algorithm implemented.
- The agents must be able to discern different objects such as other agents, obstacles and boundaries and react accordingly.
- The agents must be able to learn from their movements to take further actions.
- The agents must be able to emit; in an appropriate form, their area of influence.
- The agents must be able to transmit their knowledge to other agents.

3.1.2 Content Requirements

The content requirements of this project consists of collecting all relevant data after running the simulations for several iterations and analyse and present said data in a logical and well structured fashion. This part of the project is crucial as it is our main deliverable in the end and truly shows that we have understood the objectives of the project. This report should display our findings from several tests of the different method types. By running the simulations over several iterations, we shall obtain a more accurate representation of the methods involved. This report should also include a detailed analysis of the comparison of the different methods.

3.2 Specifications

3.2.1 Development Environment

There are many different development environments that can be used to accurately and efficiently simulate the process of agent-based models. The project's main requirement is in the form of this software report, which details my findings,. Thus, with regards to the design and interface of the models, specifications were less detailed. The simulations that are developed and tested need to produce accurate results to form a representative analysis of the specific methods involved.

Early on within the project it became apparent that the development environment that would be used was NetLogo. This is a multi-agent programmable modelling environment, which works cross-platform [14]. It is fully programmable and has a simplistic design for ease of use.

3.2.2 Software Specification

3.2.2.1 Front End

The front end of the application is intended to be as simplistic as possible. The reason for this is in an attempt to reduce the computational power being taken away from the actual simulations. If the application is spending too much time or is slowed down by what is happening on the front end, this may have an adverse effect on the simulation process, thus producing inaccurate results and possible anomalies within these results.

The front end of the program will allow for a certain amount of customization within the program. Sliders to adjust the agent population and control parameters will be implemented. There will also be buttons to initiate the environment and agents (according to the control parameters) and to start at stop the simulation process if necessary.

It will also include the options to set the starting positions of the agents. Indeed, it is important to test how the starting positions affect the outcome of the process. We will thus be testing the differences between having all agents initiate from a single location, compared to when they are initiated in separate, unique locations. Then, we will also test the differences in efficiency when we spawn the agents all at the same time rather than sequentially.

It will also display any relevant data in the form of text and graphs as output. Which will consist of useful measurements such as time taken for the agents to converge, percentage of the total environment covered and number of patches covered.

3.2.2.2 Environment

For the purpose of this project all environments will be restricted solely to a two-dimensional plane. We will not take into account the ‘height’ of a physical space as this will require further research into the development environment.

One important detail to take into account is how to design an environment that is representative of a somewhat ‘real-world’ scenario. The environment has to allow the agents to move throughout it without excessive impedance, but still providing enough of a challenge (at times) to provide us with a varied set of results. There has to be a balance between the accuracy of the simulation and the efficiency of the agents when it comes to completing it.

The development environment that we will be using provides us with a reasonable state space to work with. Where deemed necessary, the properties of the environment will be altered,. As the environment is set up in a grid structure, each cell, or "patch" can have its individual properties altered. Through the purposeful modification of these properties we will be able to construct obstacles and boundaries for the agents to interact with. Indeed, the amount of obstacles can either be generated randomly, customized through the front end or predetermined by a set map. A change in the patch properties will also indicate whether or not they are within the area of influence of any of the agents.

3.2.2.3 Agents

Actions

The agents will be able to move in any direction within the environment (in accordance with the running algorithm) so long as it is not already being occupied by either an obstacle, boundary or another agent. There will be a collision detection system in place to notify the agent that something is obstructing its path and that it should attempt to find an alternative route to reach its desired state.

The desired states for each of the agents will vary depending on the different methods being run. These will be elaborated on in a different section.

Communication

We shall implement two different types of communication methods., which are direct and indirect communication. The details of these two methods were expressed earlier and in [6]. The method for direct communication as seen in the *BeePCo* algorithm will consist of creating network links between the agents for signal transmittance, whereas the method for the latter used in the *StiCo* algorithm will see the agents drop artificial chemicals called pheromones onto its surrounding patches. There will be no form of communication in the *Random Walk* algorithm.

Choosing

The way in which the agents make movement decisions will vary depending on the method in place. However, two common factors for determining movement choices by all agents are the proximity of its neighbours and the strength of the emitting signal or pheromone (apart from *Random Walk*).

- *StiCo*: agents move in a rotary fashion until a pheromone trail is detected. If the exterior sensor senses a pheromone,the circling magnitude is increased. If the interior sensor senses a pheromone, the direction of motion is reversed.
- *BeePCo*: for every signal that is received from a neighbouring agent, move with a magnitude proportional to the distance the signal has travelled and in the opposite direction to its origin.
- *Random Walk*: agents will move forward at a constant velocity and turn with a random angle determined every time step.

Success

The goal of the simulations is to maximise the total area covered by the combined efforts of the agents, as well as keeping the total time for all agents to converge to a minimum. The success rate of each simulation will be based on these results.

3.2.3 Content Specification

As mentioned previously the main deliverable from this project will be in the form of this software report. This report endeavours to detail the results of my simulations and present them in a logical way. The two main ways of testing the efficiency of these simulations will be through collecting the data regarding the time taken for the agents to converge and the percentage of the environment affected by the combined area of influence from the agents. Furthemore, this report will record these statistics over several iterations of each simulation to obtain a more accurate result. The data recorded will include the total number of patches within the environment, a mean for the number of patches covered, a mean for the percentage of area covered, the standard deviation for the number of patches covered, a mean for the time taken to converge, the standard deviation for time taken to converge and the performance from each iteration.

Chapter 4: Design

The purpose of this chapter is to provide detailed steps with suitable figures, on how the simulation process will take place at an abstract level. This overview of the design will give us an idea of how the system will eventually be implemented.Due to the fact that we will potentially be implementing multiple methods for these simulations, the design may vary somewhat depending on the respective requirements. This will be made apparent within separate headings.

4.1 Data Flow

The data flow for agents, upon execution of the models is as follows:

4.1.1 StiCo

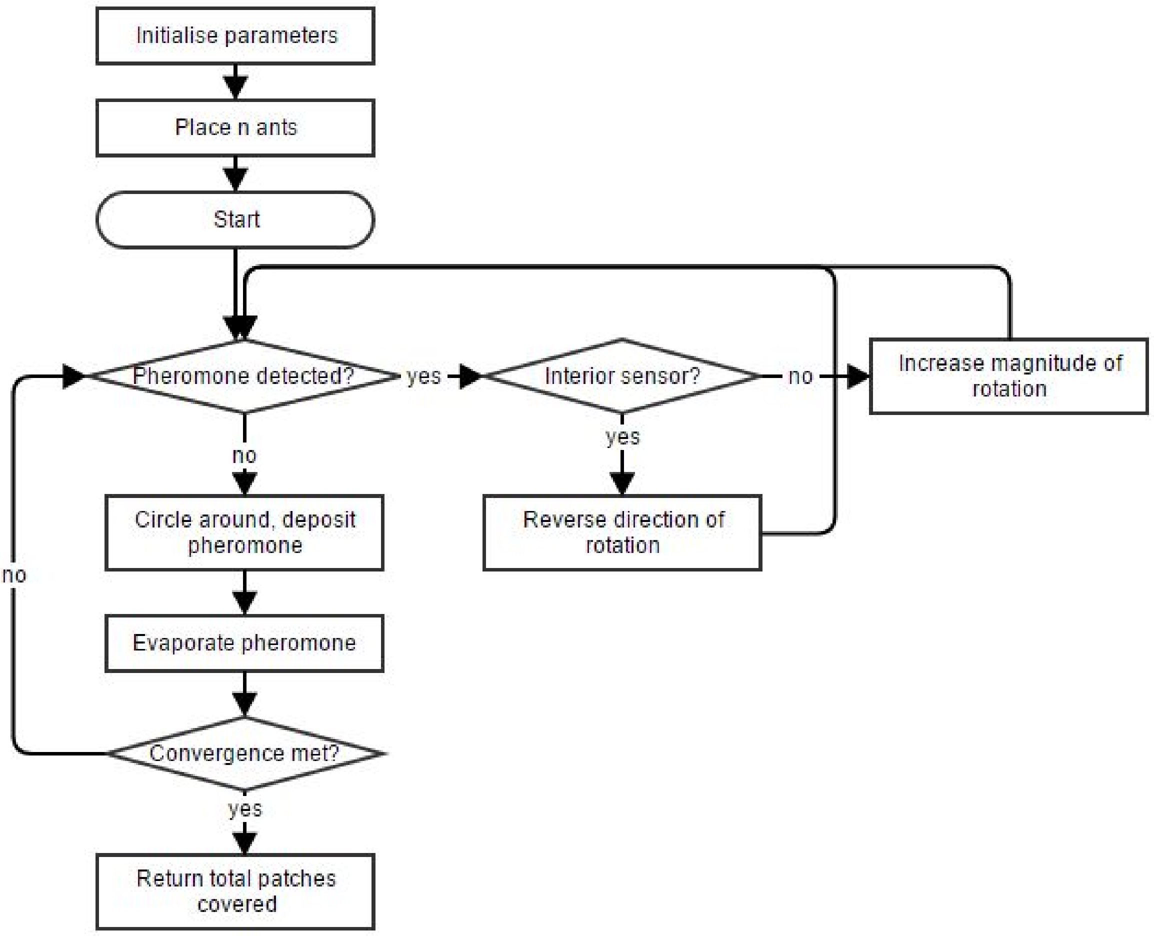
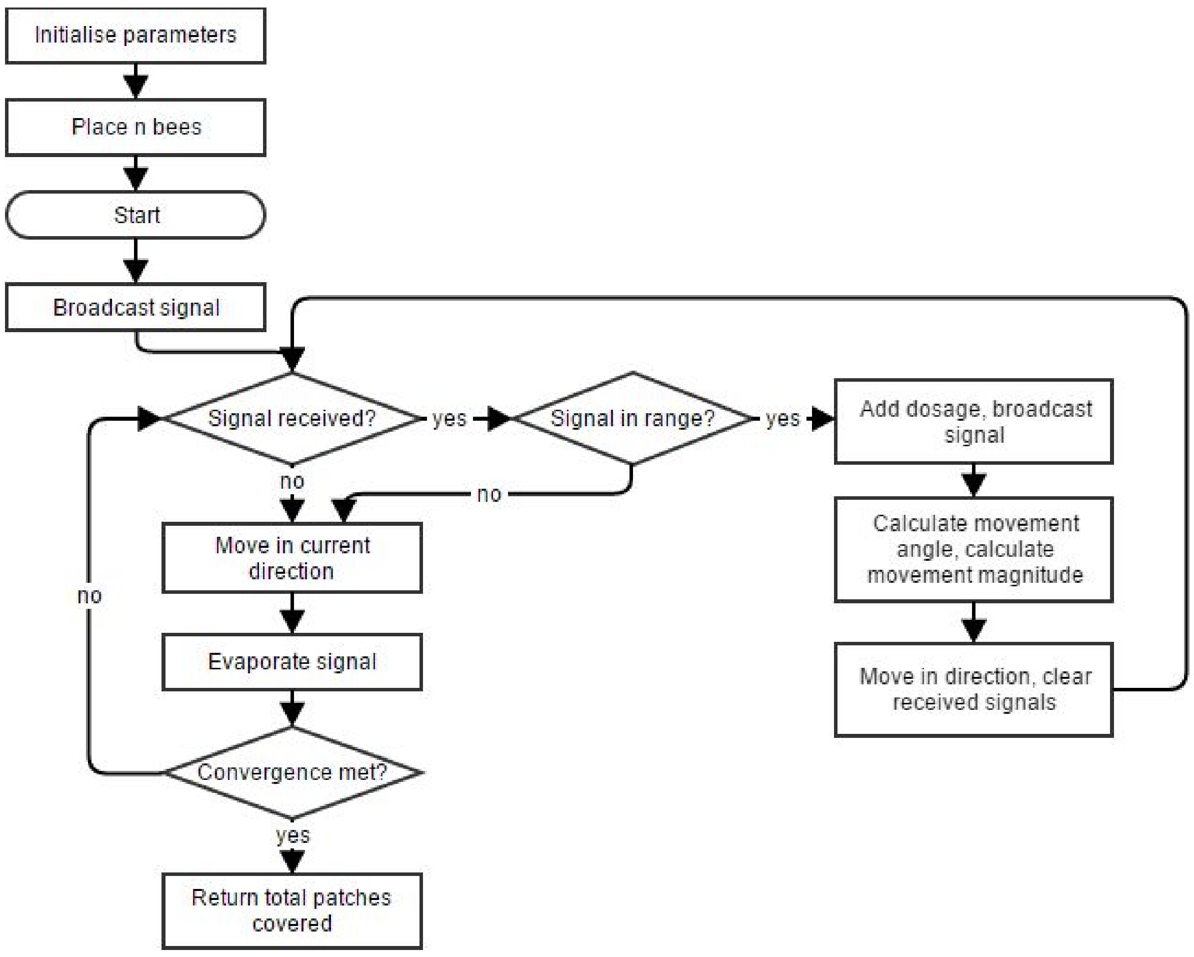


Figure 4.1: Data flow for the *StiCo* algorithm

4.1.2 BeePCo



Initialise parameters
Place n bees
Start
Broadcast signal
Signal received?
yes
Signal in range?
yes
Add dosage, broadcast signal
no
no
Move in current direction
no
Calculate movement angle, calculate movement magnitude
Evaporate signal
Move in direction, clear received signals
Convergence met?
yes
Return total patches covered

Figure 4.2: Data flow for the *BeePCo* algorithm

4.1.3 Random Walk

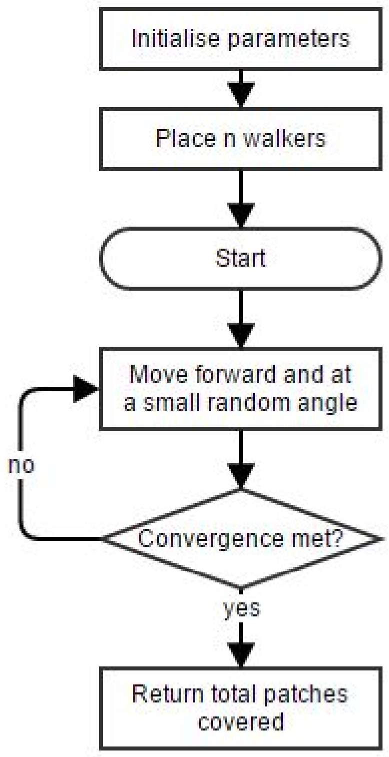

Initialise parameters
Place n walkers
Start
Move forward and at a small random angle
no
Convergence met?
yes
Return total patches covered

Figure 4.3: Data flow for the *Random Walk* algorithm

4.2 Graphical User Interface

As the graphical user interface is not the main priority of this project, it will be made to be as simple as possible. This is not only for the purpose of saving time but this could also result in more accurate results from the simulations. We want the interface to be as little computationally taxing as possible, as to not have it affect the simulation in any manner. If the application is spending too much time rendering the models, this may have an adverse affect on the outcome of the final test data.

4.3 Agents

The agent procedures for the different models vary. The pseudocode for each of the agent procedures are given in algorithms 1 through 6, which provide a brief understanding of how each of the models will operate during execution.

The agent procedures for the *StiCo* algorithm can be seen in algorithm 1. All agents are required to have the ability to detect and deposit pheromones. The initialization of the model sets a global circling direction for all of the agents to begin with (either clockwise or counterclockwise). The main loop of the program will then run until convergence has been met.

Algorithm 1: StiCo Algorithm

```
Requires: each robot can detect/deposit pheromones
Initialization: set circling direction (CW/CCW)
loop
 while no pheromone detected do
    Circle around;
    Deposite pheromone;
 if interior sensor detects pheromone then
    Reverse direction of rotation;
 else
    while pheromone is detected do
       Increase magnitude of rotation;
end loop
```

The agent procedures for the *BeePCo* algorithm can be seen in algorithms 2 to 6. Algorithms 2 & 6 are time based and occur every time step, whereas algorithms 3, 4 & 5 are event triggered.

Algorithm 2: Differentiation Cycle

```
loop
 if current pheromone level < set threshold then
    Set queen to true;
    Broadcast signal;
 else
    Set queen to false;
end loop
```

Algorithm 3: Pheromone Propagation Cycle

```
while pheromone is received do
   if pheromone is within a set distance then
      Increase current pheromone level by received dosage;
      Broadcast signal;
   else
      Ignore received pheromone;
   Call Move Cycle;
```

```
Algorithm 4: Move Cycle
if pheromone is received then
    Call Movement Decision
else
    Move with current heading
```

```
Algorithm 5: Moving Decision
if current pheromone level $< 0$ then
    for all received pheromones do
        Calculate the difference between the senders coordinates and own;
        Calculate the angle by taking the arctangent of these differences;
        Resolve vertical and horizontal components based on the received pheromone level;
        Sum these components;
Calculate movement magnitude by taking the average of the summed components of all received phermones;
Calculate movement angle by taking the arctargent of the summed components;
Apply $180°$ shift to angle;
Clear all received pheromones;
```

```
Algorithm 6: Evaporation Cycle
loop
    Multiple current pheromone level by a factor $> 0$ and $< 1$
end loop
```

The agent procedures for the *Random Walk* method can be seen in algorithm 7.

```
Algorithm 7: Random Walk Algorithm
loop
    Move forward;
    Rotate by small random float;
end loop
```

4.3.1 Positioning

The agents will have a position in the environment which is indicated by the agents x and y coordinates and the heading. An agent’s position is perfectly known and can be measured from the center of a patch or the exact position on a patch. The values of each parameter (where applicable) must be updated as they proceed to move through the world.

- *StiCo*: agent positioning and actions will depend on the concentration of pheromones on neighbouring patches and the heading of said patches. By default agents will rotate on the spot.
- *BeePCo*: agent positioning and actions will depend on the direction and distance of neighbouring agents and the total number of signals being received. If they are no longer within signal range of their neighbours, then they continue to move with their current heading until their energy fully decays.
- *Random Walk*: agent positioning and actions will be determined each time step. Agents will move at a constant velocity and rotate at a randomly generated angle.

4.3.2 Goal State

There are no strict “goals” within the environment that the agent are required to reach. The method in both *StiCo* and *BeePCo* for agents to determine whether they have reached a final state is when they are no longer receiving a pheromone from the surrounding patches or a signal from any neighbouring agents, respectively. When this occurs the agents will cease to perform any more actions. However, if convergence is met through the use of a set tick count, so too will the agents cease to perform any further actions.

4.3.3 Success

The overall success of the simulation will be based on the combined area all of the agents were able to cover. All appropriate measurements for this data will be collected when convergence is met. Convergence can be met in two ways:

1. All agents achieve a stable configuration (no more actions can be performed).
2. A set number of ticks has been met.

4.4 Environment

The environment in which the simulations take place is grid based: each cell or “patch” within the grid can have its individual properties altered depending on different scenarios. By changing these properties we can designate a “type” to each patch. There will be three possible patch types for each of my simulations:

- Boundary/Obstacle: agents are unable to pass through the patch
- Pheromone: an agent’s pheromone trail is present on the patch
- Covered: the patch is being covered by an agent’s area of effect
- Blank: the patch is not being covered by an agent’s area of effect

The patch properties can be specified randomly at runtime to generate non-specific obstacles each time the simulation is run. It is also possible to create distinct “maps” where the obstacles are already in place. This method can also be used to create a set of predetermined maps and have them selected at random. By testing all of these methods to generate different environments, it will provide a better, unbiased set of results.

Chapter 5: Implementation

This chapter will serve as an account to the implementation process of the project. It will provide a description of each step taken in the development of each model. Any relevant and major changes to the programs will be indicated by a version number, which are incremental so newer versions of the code will have higher version numbers. This series of steps is also presented in chronological order.

5.1 Early Tests

Prior to the implementation of the aforementioned area coverage algorithms, it was important for us to familiarize ourselves with the development environment. This meant learning the syntax and semantics of the coding language, as well as the features available from the interface, which was achieved through the use of public video tutorials, web forums and the NetLogo user manual [15].

Firstly, we began by writing generic agent models: creating standard and custom breed turtles, adjusting turtle and patch properties, enabling basic motion controls for agents, collision detection between agents and boundaries and creating links between turtles and tieing them together. Through the use of these initial programs, not only were we able to understand the foundations of the development environment but we also learnt how the models actually react during execution. For instance, the runtime would be slowed down significantly with more complex models, requiring the use of multiple turtles and links between turtles. This became more apparent when changing the execution speed manually through the use of a slider.

5.2 StiCo

V 1. The *StiCo* algorithm employs an indirect form of communication between agents. It achieves this through the use of pheromone trails which are propagated by each of the agents. Because of this, it was important to begin by implementing the fundamentals of pheromone propagation. In order to achieve this, we assigned a new property to all patches called “pher”. This property is an integer value used to indicate the pheromone level on each patch, with a higher value indicating a higher concentration of the pheromone, which were propagated onto patches by each agent. This was easily achieved with the use of the built in function “diffuse”. This function allows each agent to equally distribute pheromone deposits on its neighbouring patches. Consequently, when it came to implementing the procedure of detecting pheromones, agents would pick up the pheromone traces that they, themselves had left. In an attempt to counteract this, we set up a global variable called “thresh”. This too was an integer that had a predefined, constant value, which was meant to represent the pheromone level that each agent dropped itself. Detection of pheromones would only occur if the concentration on a patch that an agent is on is greater than this threshold. This was deemed inefficient and was removed.

V 2. With our second iteration of the model we decided to completely start anew. Feeling that our first version of the program began too ambitiously, in the second version, we reduced the code so as to fully understand what was happening at run time. We remained adamant about the fact that the pheromone propagation cycle was an important first task to implement. Bearing this in mind, we stripped away the process of pheromone detection to observe how the functions used to propagate pheromones actually worked. To aid with this, we set the patch property “color” to scale depending on the pheromone concentration. This provided a visual representation of what was being propagated by each of the agents.

Another important task was to implement a method for evaporation. As with all previous functions pheromone evaporation is time triggered and occurs at every tick. Evaporation happens by taking the current concentration and multiplying it by an integer > 0 and < 1. To achieve this, we incorporated a new variable called “evaporation-rate”. This, as well as two other variables called “population” and “diffusion-rate” were employed in the form of sliders on the front end of the model. These sliders allow the user to adjust the value of the variables manually, which makes it easier to refine and test the program.

V 3. While the behaviours for pheromone propagation and evaporation were now in place, detection was not. With the creation of a new agent owned variable ("detect"), we were able to store the current pheromone level each agent was receiving from the patches it came in contact with. This was updated by adding the pheromone dosage from the patch an agent is located on to the existing value. The method for evaporation was also updated to provoke the decay of this value. Similarly to previous versions of the program, turtles would detect their own trails. To counteract this, we assigned each turtle a new property called "threshold", which was an integer value that was associated to the natural pheromone level of an agent's own propagated pheromone. This way it ensures that an agent is not affected by their own pheromone trail. However, when they did detect higher concentrations of pheromones, they would rapidly alternate their movement patterns. This was due to the fact that they would remain in contact with the trails for a certain period of time before moving free or the propagator moved far enough away. An attempt to solve this was made by using wait statements but this was to no avail.

The introduction of two booleans, owned by agents were made. "direction?" and "interior?" which were used to indicate the direction of rotation and the location of the interior sensor respectively. Their default value were both set to *true* (CW). In reality, the value of "interior" did not truly represent anything and was used solely as a placeholder for future versions. Furthermore, a third integer called "magnitude" was created to control the magnitude of rotation performed by each agent. The value increases if the a pheromone is detected, while it decays in the same manner as other variables.

At this stage we also experimented by randomising the initial locations of each agent. However, we soon found this to become an issue when it came to agent interaction and came to the conclusion that it would make for poor and unreliable test results (this can be reverted at a later date for testing).

The organisation of the code came into question so we separated relevant sections into subclasses. Pheromone evaporation, detection and the main body of the *StiCo* algorithm now had their own methods.

We found that the value limits of the sliders needed to be changed as only a small range provides useful results. Minor changes were also made to the turtle shape.

V 4. The next step was to correct the issues with pheromone detection and interpret which angle the scent was being produced from. This meant updating the detection function significantly to allow for the differentiation between a hit from the interior sensor and other possible angles. The boolean “interior?”, (introduced in the previous version) began to play a role. To measure the concentration of pheromones on the patches ahead of a turtle, a *reporter* was used. This reporter is called before the main *StiCo* algorithm by the pheromone detection function, during execution. First, this function creates 3 local variables “scent-ahead”, “scent-right” and “scent-left” each given the corresponding angles 0, 45 and -45 to pass to the *reporter*. The values returned by the *reporter* are those of the pheromone concentrations on the patches at the given angles ahead of the agent. Then, the detection function makes an equality check between these values and checks which direction the agent is currently rotating towards. Depending on which is the highest value and the current direction of rotation, the position of the interior sensor is flipped. Finally, the dosage is added to the agent’s current pheromone level. With the addition of the *reporter* class the rapid switch in movement patterns was still occurring, albeit to a lesser degree.

At this stage in the development process we decided to test the order in which the functions are called to see if it played a role in the behaviour of the agents. The order was changed to *detection, StiCo, evaporation*.

V 5. The final version of the model set to stop the rapid alteration of agent motion when in contact with pheromone concentrations above the set threshold. To solve this problem a new breed of turtle called “trail” was employed. These turtles were created by each of the agents (using the *hatch* command) and were used to represent the pheromone trails. Then, we set the shape to “line” and a link was tied between themselves and the agent. Trails were created at every tick and the colour set to fade according to the fade-rate slider. They were then told to *die* once the colour intensity dropped below a certain level. The purpose of the trails is to propagate the pheromones instead of the agent themselves. Since the trails are called to *die* the problem of agents remaining stuck in pheromone deposits was solved.

Other changes to refine the model include:

- Delay to initial start.
- Convergence tick rate set to 10000.
- Patch colour changes to red when an agent visits it.
- Graph added (% covered by ticks).
- Monitors added (total patches & patches covered).
- Parameters refined.
- Constant magnitude was added

5.3 BeePCo

V 1. Before beginning to implement a model to represent the *BeePCo* algorithm we spent a considerable amount of time analysing each of the individual algorithms within it, as the *BeePCo* algorithm is much more complex compared to the methods discussed above.

The algorithm uses a vector-based system for locomotion so the first step we endeavoured to do was to create a method to initialise vectors and produce a way to update them accordingly. Because NetLogo does not explicitly allow the use of vectors, instead lists were used in their place. The same applies for vector addition. As the development environment does not have built in functions for this, a reporter was created to update the lists every step.

The use of while loops within the program produced incorrect results. As the case for most models is that they are executed continuously, until a set configuration is achieved or the user requests it to terminate. While loops within the pseudocode algorithms would have to be replaced with if statements.

As we were finished with the implementation of the *StiCo* model, we attempted to adapt the code to our new set of requirements. We were thus able to reuse certain aspects but due to the communication and movement methods involved, the majority of it had to be replaced.

To follow the design process of the algorithm separate classes were created to represent each of the individual algorithms. While strictly following this set routine we encountered several complications with some of the more complex commands, such as vector movement (stated earlier) and broadcasting signals directly.

V 2. The second step was to establish a method for direct communication between the agents. This implies that agents are able to transmit signals and share information to their neighbours without any external means. To achieve this we decided to use the built in agent called a "link", which are used to connect two agents together. The use of links enabled us to establish connections between agents directly. Two custom breed undirected links were specifically created. The first breed was a link that would simulate the broadcast of pheromone signals. This set of links are established on setup connecting all turtles together. Reporters were used to return the length of these links for the purpose of setting a distance threshold. When a set threshold has been reached, the link is told to *die*.

The second breed of link was a permanent link which was made hidden. Every turtle creates a permanent link to all other turtles, whose purpose was to reestablish pheromone connections between agents once they have been broken and come back into range. These links work in similar way to the pheromone links: they report their link length and when it is =< the set threshold, then agents will attempt to reconnect with pheromone links. The problem we encountered with this method however, is that when one pair of agents were in range to reestablish their pheromone link it would cause all pairs of agents to attempt to do the same, even if they were not close to one another. This functionality was thus removed.

V 3. To simplify the current code we took the decision to remove the reporters. Links would now be tasked to record their lengths within the main body of the program. This meant that less functions were needed and called each time step.

The forward magnitude of the agents was updated so that it was proportional to the number of pheromone connections a turtle was experiencing at any one time. Therefore, when a turtle is no longer receiving any neighbouring signals it ceases to move (magnitude = 0). When an agent reaches this stable configuration it displays its area of influence by changing the colour of the patches in a set radius, as stated in the requirements portion of this report.

V 4. The final step was to allow agents to detect the angle at which a signal is being received and from this information calculate the destination that it is required to move towards. Instead of following the method expressed in the paper, we decided to try and take advantage of the tools already in place that NetLogo offers. As direct link communications were already established between agents, this meant that the heading at which the link was pointing could be easily found by using the built in function "link-heading". A list was created for each agent with the purpose of storing all link headings that were connected to them. The mean of all headings within a list was then taken. Both the list and the mean heading was made to update every tick. An if statement was then employed to check whether the average heading was >= 180° or < 180°. If the average was >= then 180°, it would subtract 180° from it and store this as the destination angle. If the average was < then 180°, it would add 180° to it and store this as the destination angle instead. However, this method failed for both directed and undirected pheromone links which were tested. It was found that with undirected links, they only have one heading. This meant that both agents in a pair would be storing the same headings, thus move towards the same location. As for directed links, they were found to have two headings but when used in this manner each pair of agents stored the headings from either end of the link, causing the same issue. Unfortunately, this means that the model is incomplete and does not offer the full functionality of the proposed algorithm.

5.4 Random Walk

V 1. We took the decision to implement a *Random Walk* algorithm purely to serve as a method for comparison to the other algorithms.

The implementation of a *Random Walk* model was fairly simple: it began by repurposing one of the early collision test programs. From this, all that was required was to expand the movement features of the agents, required to perform a "random walk". This involved setting a forward speed to all agents that was equivalent to the previous models and applying a small random degree of rotation every step. Random floating point numbers can be generated through a built in function. New random numbers were also generated each tick and were used to change in angular movement. Furthermore, a slider was added to the front end of the model which allowed for the ease of changing the float range (min 0). The useful range was tested to produce comparable results.

Other additions include:

- Delay to initial start.
- Convergence tick rate set to 10000.
- Patch colour changes to red when an agent visits it.
- Graph added (% covered by ticks).
- Monitors added (total patches & patches covered).

5.5 Additional Implementation

An additional model that incorporated the fundamentals of goal based decision making was implemented. This model involved the use of setting goal states for each of the agents, which were assigned to each agent upon their creation. Agents were generated sequentially, meaning that a new one would only be created once the previous had reached its goal state. The initial location of each agent was generated randomly as well.

The goal state allocated to each agent was the patch with the greatest distance from its initial state. This was easily achieved with the use of the functions "max-one-of" and "distance". Once this patch was assigned the agent was asked to face it and begin moving towards it. When an agent was within range of its goal it would stop, display its area of effect and a boolean assigned to it called "static" was flipped to *true*. When this boolean is *true,* the model is told that a new agent can be created.

All new agents are asked to created undirected links with the static agents. The purpose of this is to stop two agents from occupying the same patch. This is done by the use of a reporter that records the length of the link. If the link length is below a set threshold then the agent will also be set to static and the links asked to *die*. The model would then loop until a population limit is reached.

However, we were faced with a number of problems during the completion and testing phases of this model: premature stopping by the initial and final agent and delay on link length reporting causing agents to come into contact. As a result, we were not able to successfully complete and implement this model. The source code for this model will be included in appendix C.

Chapter 6: Experimentation

The purpose of this chapter is to summarise the experiments performed using each of my final models. It will begin with a description of the testing standards that were adhered to for each of the models. As each of the models have different control parameters that need to be set prior to execution, the setup procedures somewhat vary. Tables that display the default control parameters that were used through each models experimentation will be included.

Next, the testing procedures will be explained. This will cover the steps taken to decide which metrics were important to cover, which measurements provided a representation of this and how the data returned will be organised for analysis and comparison.

The experimental results will then be presented with a brief explanation of each. Finally, it will summarise what the experimental results show in terms of functionality.

6.1 Testing Standards

It is important to ensure that the results are consistent. To achieve this certain elements of the models have been predefined. This will ensure that we are provided with more reliable and accurate test data. One must bear in mind that due to the manner in which each model executes these elements differ. Again, the parameters available for change (for all models) were made available through the use of sliders on the front-end of the program. The default values for each parameter are as follows:

Parameter	Value
Global Variables	
Convergence (tick rate)	10000
Constant Magnitude of Rotation	5
Patches Covered	0
Diffusion Rate	38
Evaporation Rate	80
Fade Rate	0.2

Patches Own	
Pheromone Level	0
Patch Visited?	False
Ants Own	
Size	1
Forward Speed	0.1
Rotation Direction (CW or CCW)	True (CW)
Magnitude	5
Threshold	2
Current Pheromone Level	0
Position of Interior Sensor	True (45° from Heading)

Table 6.1: Default parameter values for the *StiCo* model

Parameter	**Value**
Global Variables	
Convergence (tick count)	10000
Patches Covered	0
Agent’s Own	
Size	1
Forward Speed	0.1

Table 6.2: Default parameter values for the *BeePCo(V3)* model

Parameter	**Value**
Global Variables	
Convergence (tick count)	10000
Patches Covered	0
Angle of Rotation Range	0 - 5
Patches Own	

Patch Visited?	False
Agent's Own	
Size	1
Forward Speed	0.1

Table 6.3: Default parameters for the *Random Walk* model

The value for each parameter was decided after several tests of the models. These values significantly changed as new functionalities were added to each of the models throughout the implementation process. Indeed, for some of them, it was found that only a small range of values would produce reliable results, so it was important to examine this range to create balanced behaviour among agents. The default values served as a reference point which facilitated the comparison across the multiple experiments. The only parameter that was changed throughout the experimentation process of all models was the population size.

To provide a more reliable set of results for comparison, the agents were set to roughly have a 3 patch radius of effect in all directions. In the *StiCo* algorithm this is achieved through the use of a constant rotation magnitude for each agent. Although the magnitude is altered according to pheromone detection, over time it will decay back to the constant magnitude. In the *BeePCo* algorithm an agent's area of effect is made apparent once it has reached a stable configuration. This can either happen when an agent runs out of energy (link connections) or convergence is met through the tick limit. This is similarly the case for the *Random Walk* model. One important point is that in order to avoid bias, agents were set to the same size throughout all models.

The size of the environment was also an important variable that needed to be shared across all models. Due to the fact that the movement methods differ between the algorithms it was difficult to decide on an area that would allow each to produce a good spread for the agents. On the one hand, the environment needed to be large enough as to not restrict all agents range of motion (too many boundary collisions) but on the other, it needed to be small enough so that a good representation for percentage covered could be attained and compared. The size of the agents was also taken into account when deciding this. By gradually increasing population sizes, we tested the world size extensively. From this the extremums of the population sizes were decided (20 & 50). The decision was made to exclude the implementation of obstacles for the initial experiments to provide a general overview of how well each of the algorithms perform. The introduction of obstacles was skipped but should be tested for future work.

6.2 Testing Procedure

Agents do not explicitly have goal states which they need to reach and therefore the environment did not include any specific patches for this. For this reason, models will only conclude once convergence has been reached. As stated previously, convergence can happen in two ways: a set number of ticks has been met or all agents are in a stable configuration, receiving no more pheromones or signals, meaning no further actions can be made. The tick rate for convergence was chosen to provide all agents an ample amount of time to spread from their origin and experience a reasonable amount of interactions. It is important to acknowledge that not all agents would cease exploring. This was most common when testing the *StiCo* model. After a certain period of time agents were seen to stop making constructive movements and begin to retrace their steps.

As expressed previously, the only parameter that was changed in all models was the population size. Different population sizes were tested to provide a comparison not only to the other algorithms but to how well the models perform when more agents were involved in the coverage process. The population size was increased in increments of 10. Four population sizes were tested: 20, 30, 40 & 50.

The purpose of these set of experiments is to collect and compare three important metrics:

1. area covered
2. population size
3. time to converge

The area covered is defined by the total number of non-overlapping patches visited by all agents. Population size is the number of agents involved and time to converge is the total time taken for all agents to reach a stable configuration. The main variable returned by the program after execution is the total number of patches “visited” by all agents. This value is then divided by the total number of patches within the environment to create a percentage of the total coverage. This data is presented on the front end of the model in the form of a graph which shows time (ticks) on the x-axis ticks and percentage of area covered on the y-axis and which is updated every tick. The total number of patches and the total number of patches visited are also displayed on the front end.

To ensure accurate and reliable results, each model was tested for 10 iterations. Means of both the total number of patches covered and percentage of the environment covered were recorded for each model, at the different population sizes. Standard deviations of total number of patches covered was also taken.

6.3 Experiment Results

The data collected from running these experiments will now be presented. Included will be all the measurements expressed above. The results of each individual run have been omitted but they can be found in appendix B. The fact that all experimental data for the *BeePCo* model was collected from version 3 must be borne in mind.

6.3.1 StiCo

Total Patches: 1089		
Covered (mean)	Percentage (mean)	Standard Deviation
Population Size: 50		
624	0.57	37.42
Population Size: 40		
490	0.45	35.02
Population Size: 30		
403	0.37	35.33
Population Size: 20		
282	0.26	33.6

Table 6.4: *StiCo* results at different population sizes

The data above shows the average results from running the *StiCo* algorithm for 10 iterations. From this we can make a number of conclusions. Although the main form of communication between agents was to act as a deterrent, they were unable to reach a stable configuration to reach convergence. All tests run were completed when the tick rate reached the set limit.

As one can see from the data, the algorithm works very reliably. When the population size is increased so to is the efficiency for area coverage. This is almost proportional. Every time the population size is increased by 10, the total patches covered increases by approximately 114 (mean of differences). This is equally apparent in the percentage of the environment covered.

The standard deviation of each tested population size is also very similar, once again proving the reliability of the algorithm.

6.3.2 BeePCo

Total Patches: 1089				
Covered (mean)	Percentage (mean)	Standard Deviation	Convergence (mean)	Convergence (stdev)
Population Size: 50				
757	0.7	44.29	1714	3
Population Size: 40				
664	0.61	40.49	1709	6
Population Size: 30				
545	0.5	35.56	1537	359
Population Size: 20				
376	0.35	31.82	1335	484

Table 6.5: *BeePCo(V3)* results at different population sizes

As mentioned previously, the *BeePCo* algorithm tested is an incomplete version. It does not contain the full functionality of the original. Therefore, the data collected is unreliable and does not provide a satisfactory representation of what it could possibly achieve. The model was tested purely from a comparative perspective.

Due to the nature of the algorithm, that is, each agent moves with a magnitude that is proportional to the number of signals it is receiving. The time taken to converge was a lot shorter than expected, converging on average a lot faster than the other models. Throughout all the experiments that were run on the algorithm, all agents were able to reach a stable configuration in under 1800 ticks (figure B.2). The standard deviation for convergence with the lower populations was much higher compared to the higher populations. It is hard to predict why exactly this was, but with further experiments on the lower population sizes it may alleviate itself.

Overall, the model proved to produce fairly reliable results in terms of total patches and percentage covered. It showed a similar increase in coverage (as the population size was increased) to the *StiCo* model.

6.3.3 Random Walk

Total Patches: 1089		
Covered (mean)	Percentage (mean)	Standard Deviation
Population Size: 50		
513	0.47	84.71
Population Size: 40		
499	0.46	57.38
Population Size: 30		
417	0.38	49.88
Population Size: 20		
365	0.33	37.14

Table 6.6: *Random Walk* results at different population sizes

Because the *Random Walk* algorithm does not employ any form of communication between agents, it was decided that the method of convergence would be to reach a set tick count only. This tick count being 10000. This was to provide better comparable results to the *StiCo* algorithm.

In terms of results, the *Random Walk* was very surprising. With the lower value of the population sizes, it produced reasonably good area coverage. However, as the number of agents was increased, the level of efficiency did not. This is extremely apparent when we compare the results from population sizes of 50 & 40. The means for total patches covered and percentage covered only showed a minimal increase (14 & 0.01 respectively). This, compared to the lower population sizes, proves it to be an unreliable algorithm to use for area coverage applications.

With the increase in population size we also saw an increase in the sensitivity of the algorithm. Looking at the standard deviations we notice a considerable increase.

6.3.3 Comparison

The collected data was then compiled into four graphs. Each graph depicts the iteration number and the percentage of area coverage for each of the algorithms, over the different population sizes. The data for each iteration can be seen in appendix B. The graphs also illustrate the mean percentage covered by each of the models by the dotted lines in the corresponding colour.

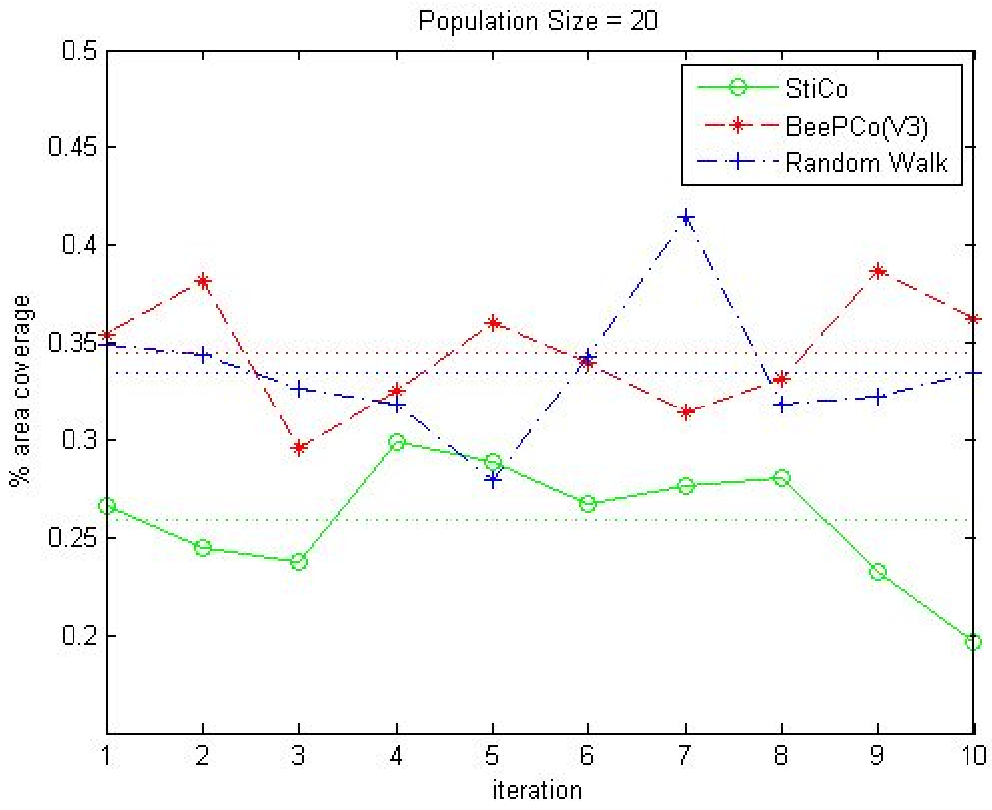


Figure 6.1: The percentage of area coverage with a population size of 20, with all models

The first population size tested was 20. From the graph above we can see that both the *BeePCo(V3)* and *Random Walk* models did rather well in terms of percentage covered at this low population size, with the *StiCo* algorithm trailing behind.

We can already see telltale signs of the sporadicness from the *Random Walk* model with the high peak at iteration number 7 and it having a slightly higher standard deviation than the other two.

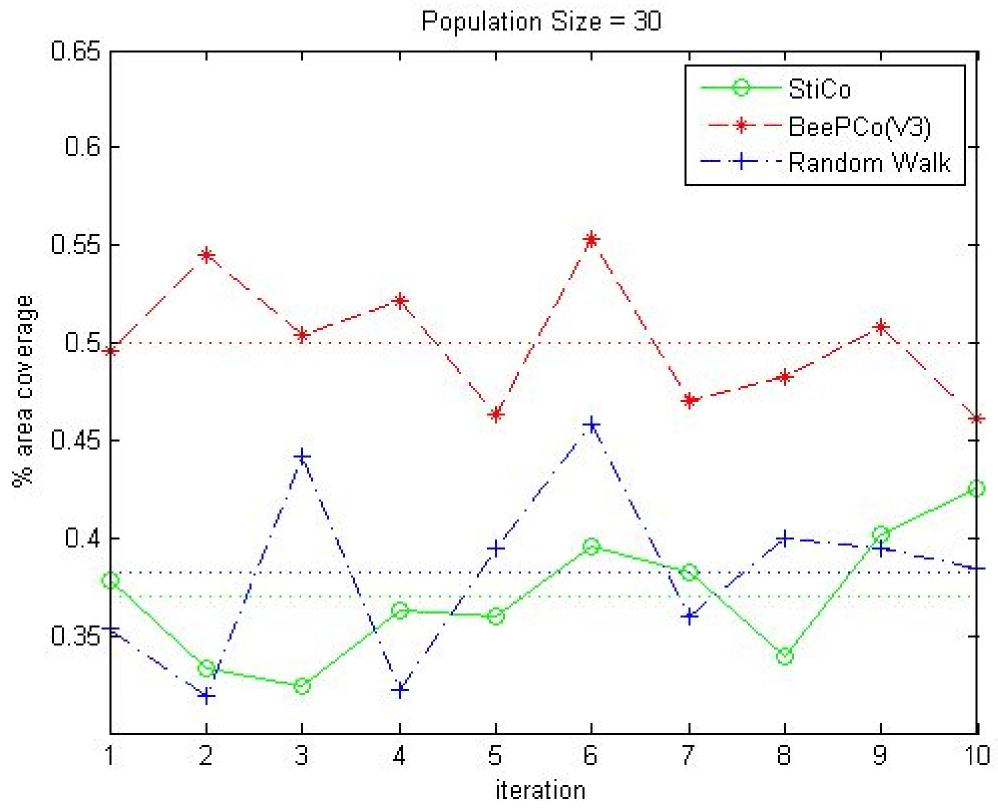


Figure 6.2: The percentage of area coverage with a population size of 30, with all models

The second population size tested was 20. From the graph above we can clearly see the effectiveness of the *Random Walk* model begins to diminish with the average increase only being 0.05, compared to *StiCo* and *BeePCo(V3)* being 0.11 and 0.15 respectively. We can also see this with the difference in means between *Random Walk* and *StiCo* now only being 0.01.

We also see the standard deviation for the *Random Walk* model begins to increase fairly significantly, whereas the other two's remain fairly stable. This can be seen from the large peaks in the graph.

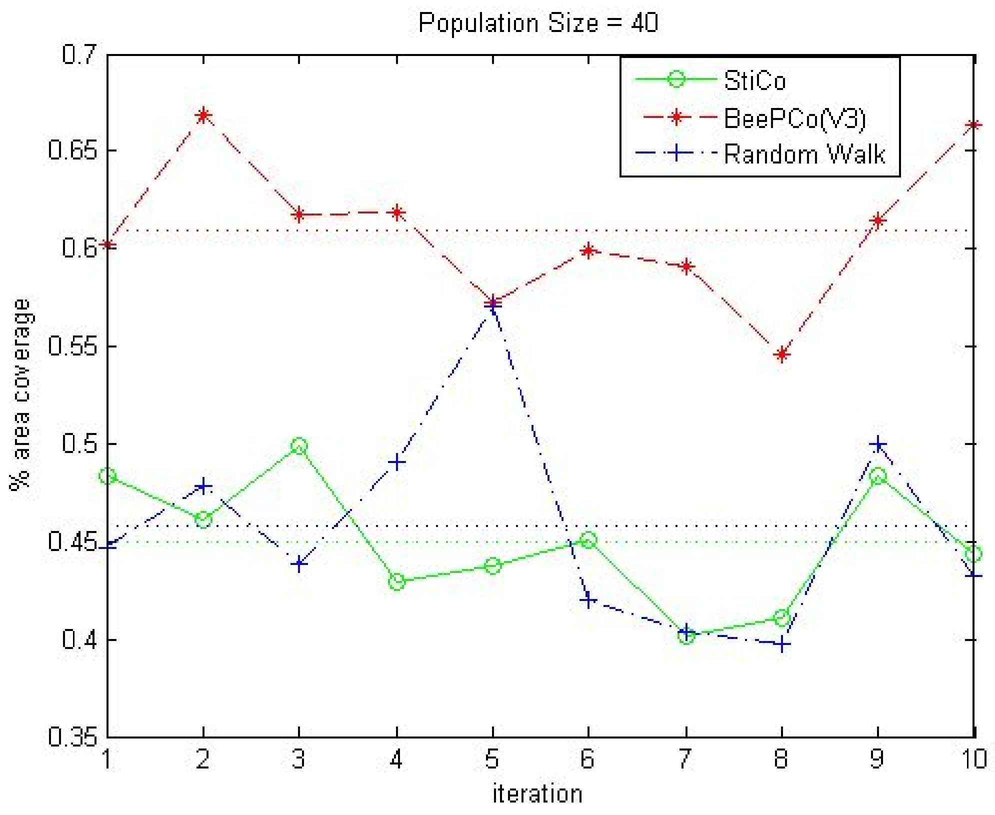


Figure 6.3: The percentage of area coverage with a population size of 40, with all models

The next population size test was 40. The data for each iteration can be seen in the graph above. Although with a slightly higher average increase than the previous population size, we can still see the trend of the *Random Walk's* effectiveness decreasing. Despite the average increase between this and the *StiCo* model both being 0.8, it is only saved due to its higher peak which met the lower end of *BeePCo(V3)'s* range. Similarly to the previous population size, the difference in their average percentage covered was only 0.01. *BeePCo(V3)* is still clearly ahead in terms of percentage covered but *StiCo* shows slightly more reliability with its lower standard deviation of 35.02.

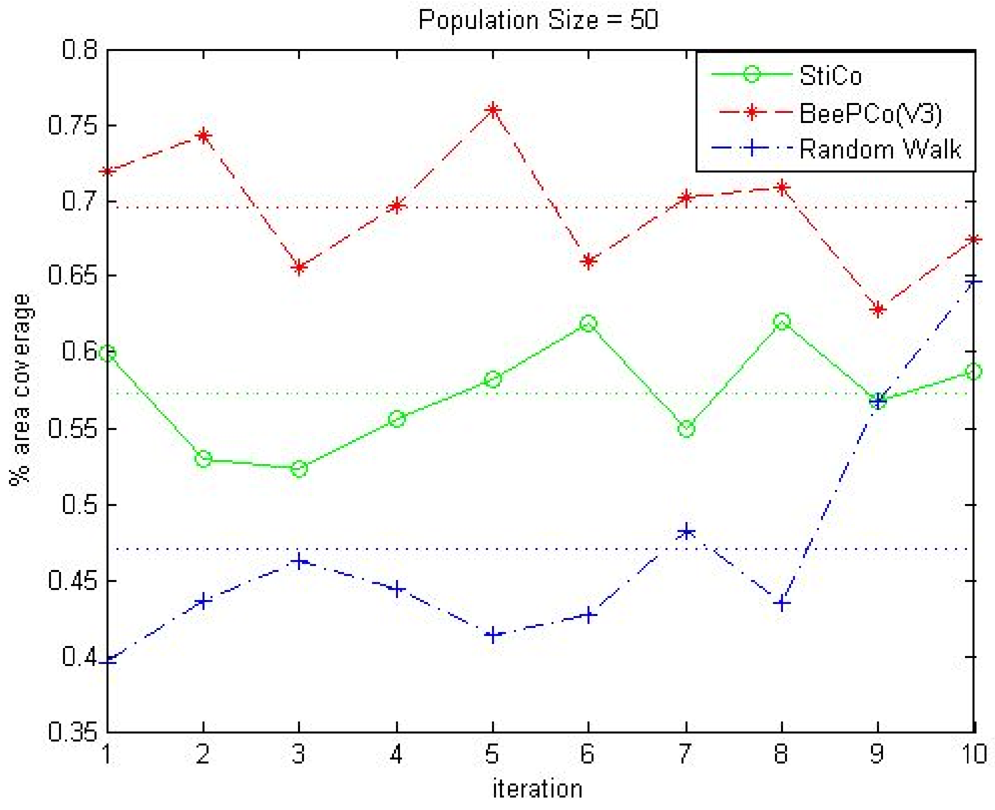


Figure 6.4: The percentage of area coverage with a population size of 50, with all models

Finally, the last population size tested was 50. With this increase, we can see a clear distinction within the results. The mean values for each of the models are almost an equal distance apart. The *BeePCo(V3)* model is the clear winner when it comes to percentage covered. The *StiCo* model was seen to rise above the *Random Walk* model for the first time and by a clear margin. Although the *Random Walk* model had a higher maximum, on average the *StiCo* model performed much better and to a much more reliable degree. The higher maximum of the *Random Walk* may be the result of an anomalous result, thus causing the standard deviation of the model to be much greater than the others.

6.4 Experiment Conclusion

The data collected from these experiments is but a small amount of the possible outcomes these algorithms have. There are many more possibilities that need to be tested to fully exhaust the capabilities and limitations of each. Furthermore, the experiments show a small amount of the total functionality of each algorithm. However, the results that have been collected from these experiments are promising. They show a similar outcome to those expressed in the paper [6]. For future work, the algorithms can be explored in further depth.

Chapter 7: Evaluation

This chapter will evaluate the different facets of the project and suggest how they may be improved in the future.

7.1 Software

To evaluate the implementation of each model it makes sense to compare them to the project requirements that were conveyed earlier in the paper. These were split into several categories.

7.1.1 Front End

Despite the fact that the front end requirements did not end up playing a significant role on the computational performance and the main purpose of the models is for research, creating a user friendly interface was not a large priority for this project. Although, having said this the front end on all of the models is easy enough of access to enable anyone to use them in the future.

The interface of each model allows the user to initialise the environment and agents, setting all parameters to their default values. It also allows for the execution, interruption and termination of the model through the use of a continuous button. As requested, appropriate data is extracted and displayed in the forms a graph and monitors. Moreover, each model also includes relevant sliders for the configuration of certain parameters of the agents.

7.1.2 Environment

The requirements for the environment were not adhered to for the most part. Due to time restrictions and in order to provide results that express an overview of the performance of each algorithm without additional hindrance, the environment used in each of the models is the default workspace (plain). However, despite the definition of different patch types, no obstacles were implemented (bar the boundaries). The patch types included blank, visited patches and in the case of the *StiCo* algorithm, presence of pheromone. The models did not offer to randomly generate environments nor were any predetermined maps created.

7.1.3 Agents

The main focal point of this project revolved around agent behaviour to simulate that of a real robot. The algorithms that were implemented and tested throughout this project are designed to work both in simulation and in real world scenarios. However, the agents within the simulation environment are not representative of how a robot in the real world would be defined. Indeed, the shape and size of the agents within the simulation are abstract and only serve to provide relevance to the model in question and a reasonable scale to the size of the environment they are in.

Ae vast part of the project, was spent developing the agent behaviours, which was a lengthy and incremental processes. Once features were added, they were tested before having new functionalities implemented. At times this caused conflicts between new and old behaviours and backtracking was required to solve these issues.

The desired functionalities of the agents, as stated in the requirements were not all met. This was both due to programming difficulties and a change in decision making as the project progressed from the requirements stage to implementation. The agents within the *StiCo* and *Random Walk* models move in accordance to their algorithms. However, achieving the full functionality that was required for the *BeePCo* algorithm was not met. As a result the experimental data collected from this is from a previous version that was implemented.These results do not provide a reasonable representation of the true functionality of the algorithm.

In all models, agents have collision avoidance procedures to the boundaries of the environment but we chose to omit the detection of agent to agent collisions. This was partly to simplify the behaviours of the agents as it was not necessarily required in the a simulation environment. It was also noticed that because of the nature of the biologically inspired algorithms, collisions between agents was usually avoided, as communication methods employed send signals that automatically repel other agents.

As for the methods of communication between agents, they were implemented as required. These include direct communication, through the use of communication links and indirect communication, through the process of depositing artificial pheromone trails. Although the movement procedures for the *BeePCo* algorithm were not fully implemented, the method for communication was.

Once actions have been taken an agent updates its properties and it is then able to make further actions in accordance to the algorithm. Agents also update the properties of the patches they have come into contact with.

7.2 Experimentation

The experimentation process of this project and the similarly named chapter in this report only provide a demonstration of the performance of these algorithms as used for robot area coverage in a simulated environment. The data collected is by at best indicative of how they would perform in a real world scenario. This is because the simulated environment is non stochastic and static whereas, in real life, the environments these algorithms would usually be asked to perform in are the opposite. However, this was beyond the scope of our project, which merely attempted to show how the basic principle of each algorithm performed within a plain environment.

The results obtained in this project compared to those that were assessed in the paper written by Broecker, Caliskanelli, Tuyls, Sklar and Hennes also analysing these algorithms show similar trends in coverage and time taken to converge, for each of the algorithms. Although the results attained for the *BeePCo* show similar trends this cannot be taken as factual evidence as the model tested did not have the full set of functionalities, as that of the one in the paper.

The experimentation is far from comprehensive: many further steps are required to fully exhaust the effectiveness of each of the algorithms. For instance, the parameters need to be refined and a level of constraint needs to be applied to provide optimal results. Then, the environments of the models also need to redefined to show how the algorithms perform with obstacles in place and other dynamic instances that occur in real-world scenarios.

Chapter 8: Professional Issues

The British Computing Society have released both a Code of Conduct & Code of Good Practice. They stand to deliver a set of guidelines to consider when working in computing. All parties involved with the development of this project were aware of these codes. We followed them and applied the principles, where appropriate. However, only a few of these rules were relevant to the project, as they were written with the intent to be upheld by a multi-person development environment.

Chapter 9: Conclusion and Future Work

8.1 Conclusion

This project exhibits the potential that exists for these algorithms but also the subject topic in general. Suitable and comprehensive methods for robot area coverage still need to be defined, therefore further research, implementation and testing such as this is required before we have any form of standardized method.

In terms of agent-based modelling, the project provided a means to explore the capabilities of the algorithms in question with a different approach. It also displayed how well they performed in a new development environment and primarily expressed the potential that indirect communication methods have. On a larger scale it showed that methods for robot area coverage can easily be defined with an agent-based modelling approach.

Although the algorithms that were implemented could have been endowed with more computational complexity, it demonstrates the extent of their performance under such circumstances.

8.2 Future Work

The possibilities for future work on this project are almost limitless, as the subject of the project itself was robot area coverage there are many more directions the project could take. The topic itself is become increasingly relevant and the field of agent-based modelling is a compelling way to implement and experiment with area coverage algorithms.

In terms of the models that were created, many improvements and additions can be made. Most notably is the necessity of improving the *BeePCo* algorithm.. The addition of the movement techniques proposed in the original paper need to be implemented before a true representation of the algorithm's capabilities can be extracted. After further experiments the variables could be refined to provide optimal results. It is also important that these are constrained to a degree as to not exaggerate the functionality of the models.

Another main aspect of the models that can see improvement is the environment. Indeed, future work could see the implementation of the remaining environmental requirements that were not accomplished during the programming segment of the project, such as the introduction of obstacles. It would prove useful to know how well the algorithms perform when faced with the challenge of objects of varying shapes and sizes. New patches types and agent procedures would need to be included to allow for this. To better simulate the effects of a real world environment, it would be relevant to add dynamic events that change the properties of the environments during execution. This could be as simple as having obstacles being generated within the environment at random intervals. Overall, this would better simulate how the algorithms would fare when it came to the hazards that are present in a physical space.

Additional experimentation to develop a full apprehension of the functionalities of the algorithms can also be undertaken. For example, heat maps could be implemented to show effectiveness of coverage, the overlapping coverage and level of redundancy. This information can be used to gauge the efficiency of each algorithm and possibly provide a noticeable way to improve it. Supplementary models can be implemented and compared such as the ant-bee hybrid algorithm (*HybaCo)*. This model could be combined with the others mentioned above into a single application that allows the user to choose between which model they wish to run. Such organisation would make conducting further research on each of them much more accessible and practical. It may also be advantageous to attempt to improve upon these methods. Finally, an implementation of the methods in a different simulation environment may provide a new understanding of them but primarily it is would better to test them in the real world.

Bibliography

1 Tokyo Electric Power Company Holdings, Inc. (TEPCO). (2016). *Application of Robot Technology*. [online] Available at: http://www.tepco.co.jp/en/decommision/principles/robot/index-e.html [Accessed 24 April. 2016].

2 SPREAD Co Ltd. (2016). *RESEARCH & DEVELOPMENT*. [online] Available at: http://spread.co.jp/en/development/ [Accessed 24 April. 2016].

3 Google.com. (2015). *Loon for All – Project Loon – Google*. [online] Available at: http://www.google.com/loon/ [Accessed 24 April. 2016].

4 Center for Robot-Assisted Search and Rescue (CRASAR), Texas A&M University. (2015). http://crasar.org/.

5 Balch, T. and Arkin, R. C. (1999). Behavior-based formation control for multi-robot teams. *IEEE transactions on robotics and automation, Vol. XX, No. Y.*

6 Broecker, B., Caliskanelli, I., Tuyls, K., Sklar, E. I. and Hennes, D. (2015). Hybrid Insect-Inspired Multi-Robot Coverage in Complex Environments.

7 Lam, H. K. (2016). An Introduction to Optimisation. *Biologically Inspired Methods, King's College London.*

8 Lam, H. K. (2016). The Binary Genetic Algorithm. *Biologically Inspired Methods, King's College London.*

9 Lam, H. K. (2016). The Continuous Genetic Algorithm. *Biologically Inspired Methods, King's College London.*

10 Lam, H. K. (2016). Evolution Strategies (ES). *Biologically Inspired Methods, King's College London.*

11 Lam, H. K. (2016). Ant Colony Optimisation. *Biologically Inspired Methods, King's College London.*

12 Lam, H. K. (2016). Particle Swarm Optimisation. *Biologically Inspired Methods, King's College London.*

13 Althoefer, K. (2015). From Perception to Maps. *Robotics Systems, King's College London.*

14 Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

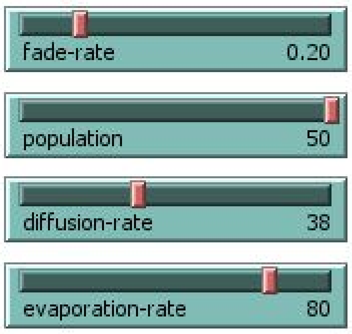
15 Wilensky, U. (2016). NetLogo Dictionary. https://ccl.northwestern.edu/netlogo/docs/dictionary.html. *NetLogo 5.3.1 User Manual.*

Appendix A: User Guide

1. Install NetLogo from https://ccl.northwestern.edu/netlogo/.
2. Either double click the model file in explorer or open NetLogo and click “File” then “Open” and locate the model file in your directory.

Model 1: StiCo

1. Set the parameter requirements with the use of the sliders.
    - Fade-rate: rate at which trails *die*.
    - Population: the amount of agents.
    - Diffusion-rate: rate at which pheromones are propagated.
    - Evaporation-rate: rate at which pheromones, current agent pheromone level and magnitude of rotation evaporate.



fade-rate 0.20
population 50
diffusion-rate 38
evaporation-rate 80

2. Initialise the environment and create the agents by pressing the “setup” button.

3. Start the simulation by pressing the “go” button (the simulation can be paused and reset at anytime by pressing the “go” and “start” buttons respectively).

4. Once the simulation has terminated all relevant data will be displayed.

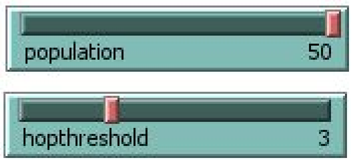
Option 1: fade-rate, diffusion-rate and evaporation-rate can be altered whilst the simulation is running.

Option 2: the simulation speed can be altered by using the slider at the top of the screen.

Option 3: further parameters of the model can be altered by opening the code tab. Parameters include: constant-mag (default=5)(line 12), direction? and interior? (default=true)(lines 23 & 27), tick rate convergence (default=10000) (line 42) and forward speed (default=0.1) (line 88). WARNING may have adverse effects on the execution of the model and/or produce errors.

Model 2: BeePCo (V3)

1. Set the parameter requirements with the use of the sliders.
   - Hop-threshold: maximum distance links exist.
   - Population: the amount of agents.



2. Initialise the environment and the create agents by the pressing "setup" button.

3. Start the simulation by pressing the "go" button (the simulation can be paused and reset at anytime by pressing the "go" and "start" buttons respectively).

4. Once the simulation has terminated all relevant data will be displayed.

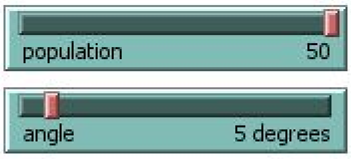
Option 1: hop-thresh can be altered whilst the simulation is running.

Option 2: the simulation speed can be altered by using the slider at the top of the screen.

Option 3: further parameters of the model can be altered by opening the code tab. Parameters include: forward speed (default=0.1) (line 84) and tick rate convergence (default=10000) (line 30). WARNING may have adverse effects on the execution of the model and/or produce errors.

Model 3: Random Walk

1. Set the parameter requirements with the use of the sliders.
   - Angle: range of degrees of rotation (min=0).
   - Population: the amount of agents.



2. Initialise the environment and the create agents by pressing “setup” button.

3. Start the simulation by pressing the “go” button (the simulation can be paused and reset at anytime by pressing the “go” and “start” buttons respectively).

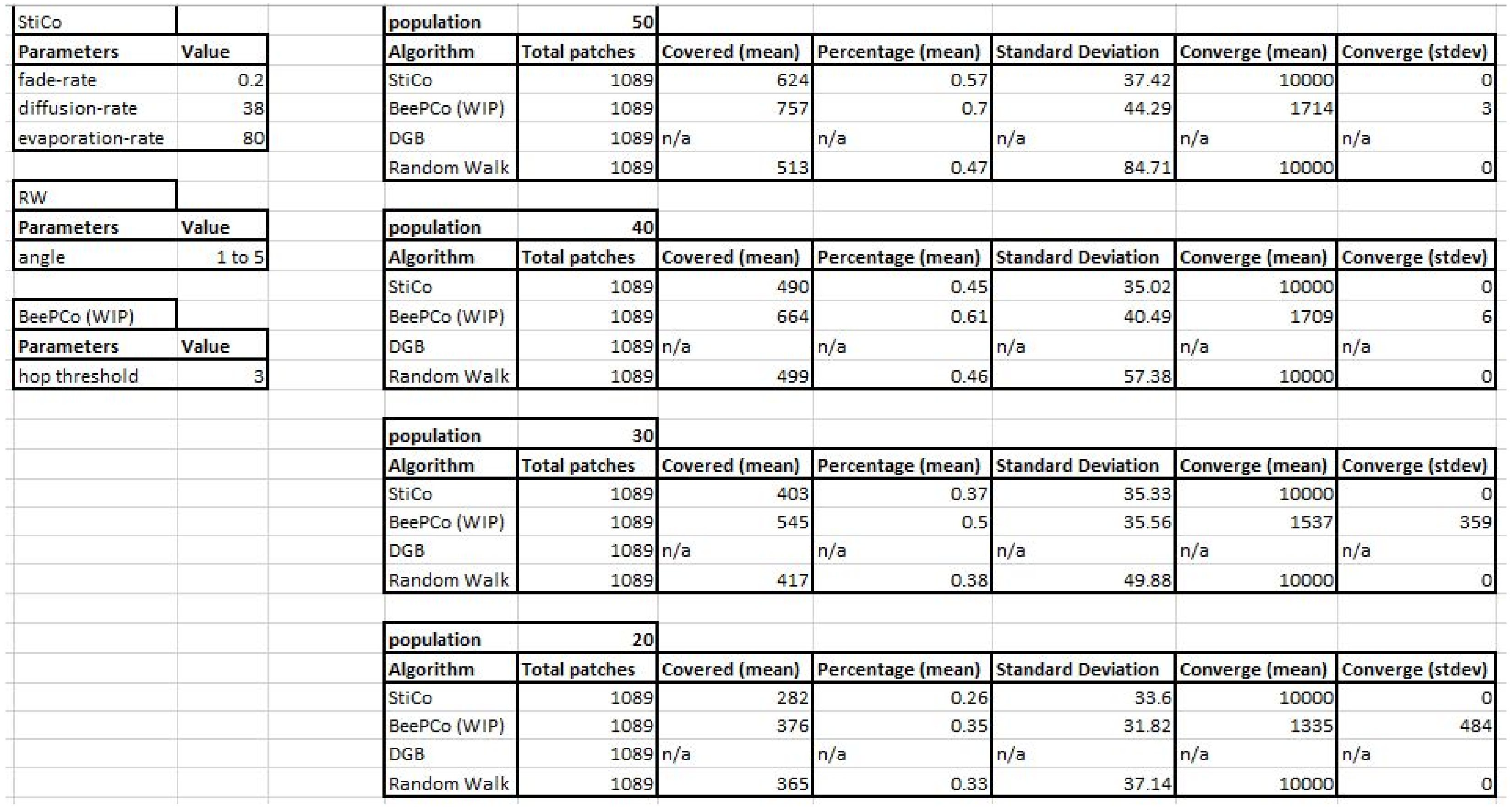
4. Once the simulation has terminated all relevant data will be displayed.

Option 1: angle can be altered whilst the simulation is running.

Option 2: the simulation speed can be altered by using the slider at the top of the screen.

Option 3: further parameters of the model can be altered by opening the code tab. Parameters include: forward speed (default=0.1) (line 34), tick rate convergence (default=10000) (line 29). WARNING may have adverse effects on the execution of the model and/or produce errors.

Appendix B: Experimental Data



StiCo	
Parameters	Value
fade-rate	0.2
diffusion-rate	38
evaporation-rate	80

RW	
Parameters	Value
angle	1 to 5

BeePCo (WIP)	
Parameters	Value
hop threshold	3

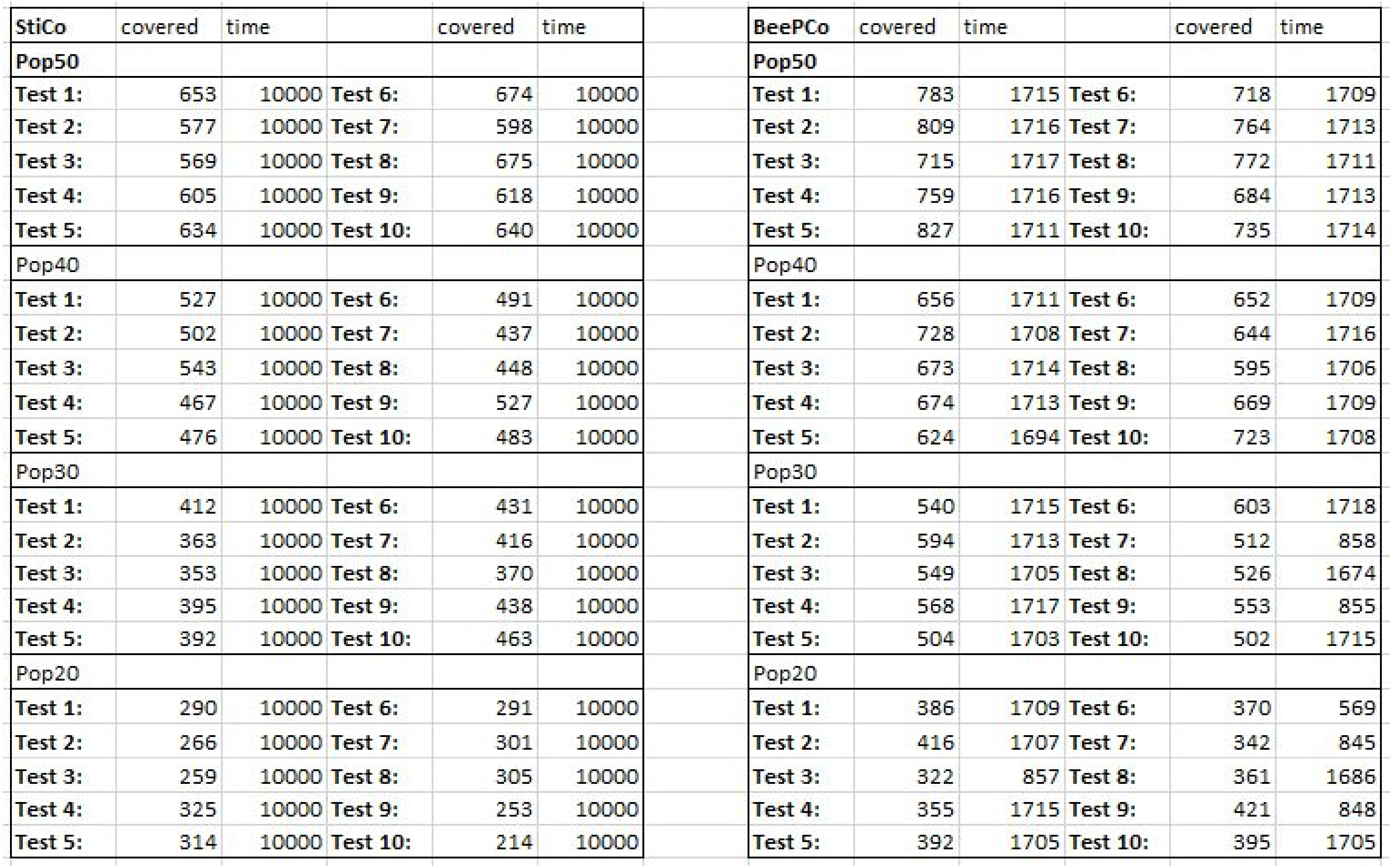
population	50					
Algorithm	Total patches	Covered (mean)	Percentage (mean)	Standard Deviation	Converge (mean)	Converge (stdev)
StiCo	1089	624	0.57	37.42	10000	0
BeePCo (WIP)	1089	757	0.7	44.29	1714	3
DGB	1089	n/a	n/a	n/a	n/a	n/a
Random Walk	1089	513	0.47	84.71	10000	0

population	40					
Algorithm	Total patches	Covered (mean)	Percentage (mean)	Standard Deviation	Converge (mean)	Converge (stdev)
StiCo	1089	490	0.45	35.02	10000	0
BeePCo (WIP)	1089	664	0.61	40.49	1709	6
DGB	1089	n/a	n/a	n/a	n/a	n/a
Random Walk	1089	499	0.46	57.38	10000	0

population	30					
Algorithm	Total patches	Covered (mean)	Percentage (mean)	Standard Deviation	Converge (mean)	Converge (stdev)
StiCo	1089	403	0.37	35.33	10000	0
BeePCo (WIP)	1089	545	0.5	35.56	1537	359
DGB	1089	n/a	n/a	n/a	n/a	n/a
Random Walk	1089	417	0.38	49.88	10000	0

population	20					
Algorithm	Total patches	Covered (mean)	Percentage (mean)	Standard Deviation	Converge (mean)	Converge (stdev)
StiCo	1089	282	0.26	33.6	10000	0
BeePCo (WIP)	1089	376	0.35	31.82	1335	484
DGB	1089	n/a	n/a	n/a	n/a	n/a
Random Walk	1089	365	0.33	37.14	10000	0

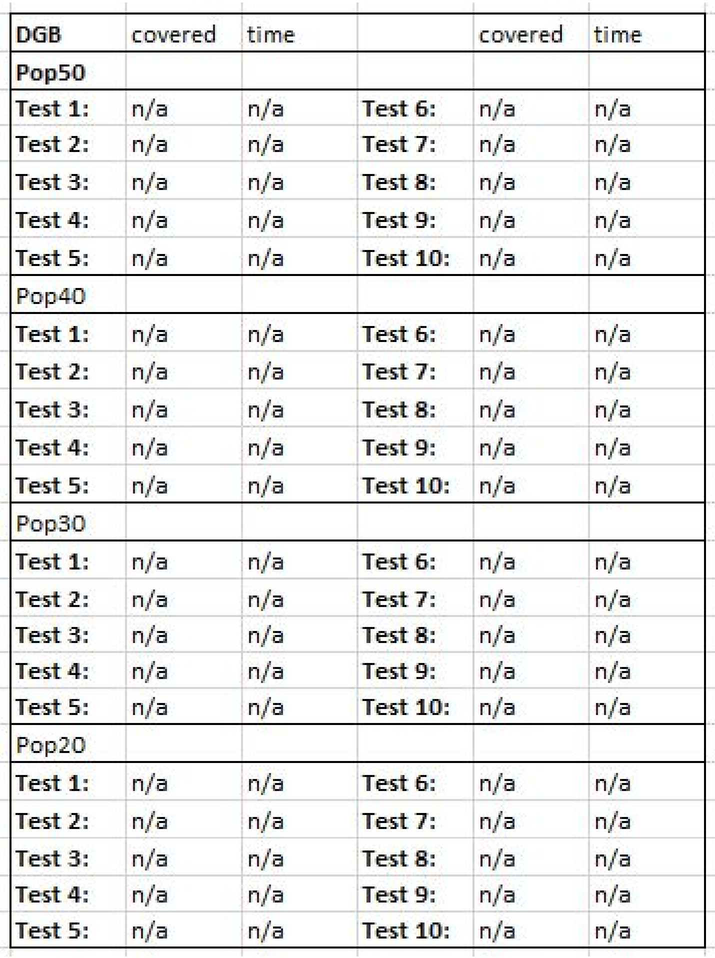
Figure B.1: default parameters, means and standard deviations for each model at different population sizes.



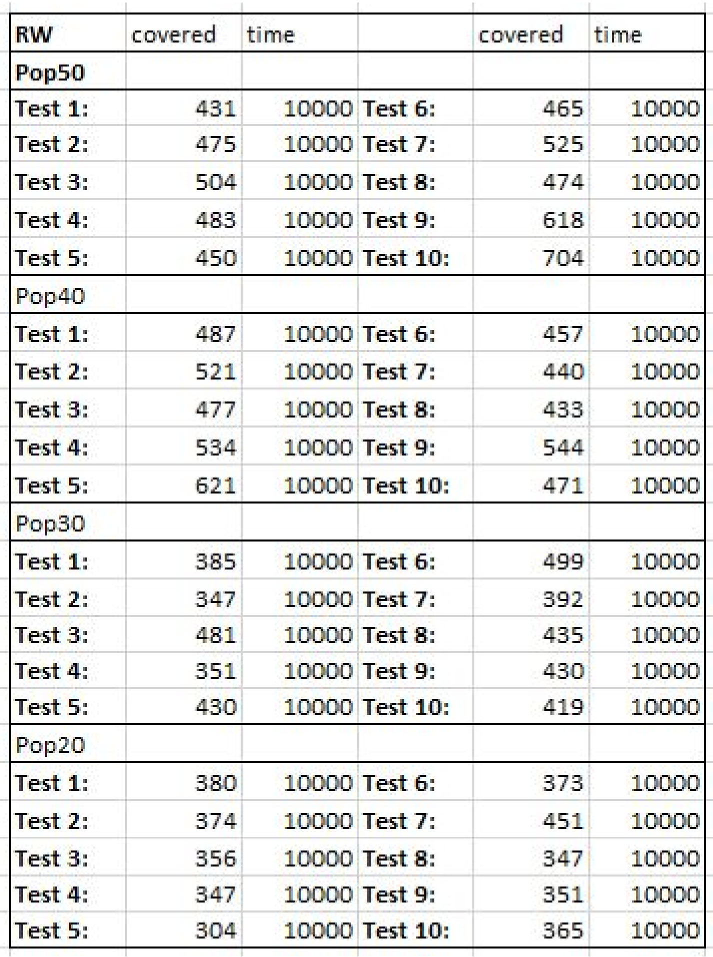
StiCo	covered	time		covered	time
Pop50					
Test 1:	653	10000	Test 6:	674	10000
Test 2:	577	10000	Test 7:	598	10000
Test 3:	569	10000	Test 8:	675	10000
Test 4:	605	10000	Test 9:	618	10000
Test 5:	634	10000	Test 10:	640	10000
Pop40					
Test 1:	527	10000	Test 6:	491	10000
Test 2:	502	10000	Test 7:	437	10000
Test 3:	543	10000	Test 8:	448	10000
Test 4:	467	10000	Test 9:	527	10000
Test 5:	476	10000	Test 10:	483	10000
Pop30					
Test 1:	412	10000	Test 6:	431	10000
Test 2:	363	10000	Test 7:	416	10000
Test 3:	353	10000	Test 8:	370	10000
Test 4:	395	10000	Test 9:	438	10000
Test 5:	392	10000	Test 10:	463	10000
Pop20					
Test 1:	290	10000	Test 6:	291	10000
Test 2:	266	10000	Test 7:	301	10000
Test 3:	259	10000	Test 8:	305	10000
Test 4:	325	10000	Test 9:	253	10000
Test 5:	314	10000	Test 10:	214	10000

BeePCo	covered	time		covered	time
Pop50					
Test 1:	783	1715	Test 6:	718	1709
Test 2:	809	1716	Test 7:	764	1713
Test 3:	715	1717	Test 8:	772	1711
Test 4:	759	1716	Test 9:	684	1713
Test 5:	827	1711	Test 10:	735	1714
Pop40					
Test 1:	656	1711	Test 6:	652	1709
Test 2:	728	1708	Test 7:	644	1716
Test 3:	673	1714	Test 8:	595	1706
Test 4:	674	1713	Test 9:	669	1709
Test 5:	624	1694	Test 10:	723	1708
Pop30					
Test 1:	540	1715	Test 6:	603	1718
Test 2:	594	1713	Test 7:	512	858
Test 3:	549	1705	Test 8:	526	1674
Test 4:	568	1717	Test 9:	553	855
Test 5:	504	1703	Test 10:	502	1715
Pop20					
Test 1:	386	1709	Test 6:	370	569
Test 2:	416	1707	Test 7:	342	845
Test 3:	322	857	Test 8:	361	1686
Test 4:	355	1715	Test 9:	421	848
Test 5:	392	1705	Test 10:	395	1705

Figure B.2: data from each iteration for the *StiCo* and *BeePCo(V3)* at different population sizes.



DGB	covered	time		covered	time
Pop50					
Test 1:	n/a	n/a	**Test 6:**	n/a	n/a
Test 2:	n/a	n/a	**Test 7:**	n/a	n/a
Test 3:	n/a	n/a	**Test 8:**	n/a	n/a
Test 4:	n/a	n/a	**Test 9:**	n/a	n/a
Test 5:	n/a	n/a	**Test 10:**	n/a	n/a
Pop40					
Test 1:	n/a	n/a	**Test 6:**	n/a	n/a
Test 2:	n/a	n/a	**Test 7:**	n/a	n/a
Test 3:	n/a	n/a	**Test 8:**	n/a	n/a
Test 4:	n/a	n/a	**Test 9:**	n/a	n/a
Test 5:	n/a	n/a	**Test 10:**	n/a	n/a
Pop30					
Test 1:	n/a	n/a	**Test 6:**	n/a	n/a
Test 2:	n/a	n/a	**Test 7:**	n/a	n/a
Test 3:	n/a	n/a	**Test 8:**	n/a	n/a
Test 4:	n/a	n/a	**Test 9:**	n/a	n/a
Test 5:	n/a	n/a	**Test 10:**	n/a	n/a
Pop20					
Test 1:	n/a	n/a	**Test 6:**	n/a	n/a
Test 2:	n/a	n/a	**Test 7:**	n/a	n/a
Test 3:	n/a	n/a	**Test 8:**	n/a	n/a
Test 4:	n/a	n/a	**Test 9:**	n/a	n/a
Test 5:	n/a	n/a	**Test 10:**	n/a	n/a



RW	covered	time		covered	time
Pop50					
Test 1:	431	10000	**Test 6:**	465	10000
Test 2:	475	10000	**Test 7:**	525	10000
Test 3:	504	10000	**Test 8:**	474	10000
Test 4:	483	10000	**Test 9:**	618	10000
Test 5:	450	10000	**Test 10:**	704	10000
Pop40					
Test 1:	487	10000	**Test 6:**	457	10000
Test 2:	521	10000	**Test 7:**	440	10000
Test 3:	477	10000	**Test 8:**	433	10000
Test 4:	534	10000	**Test 9:**	544	10000
Test 5:	621	10000	**Test 10:**	471	10000
Pop30					
Test 1:	385	10000	**Test 6:**	499	10000
Test 2:	347	10000	**Test 7:**	392	10000
Test 3:	481	10000	**Test 8:**	435	10000
Test 4:	351	10000	**Test 9:**	430	10000
Test 5:	430	10000	**Test 10:**	419	10000
Pop20					
Test 1:	380	10000	**Test 6:**	373	10000
Test 2:	374	10000	**Test 7:**	451	10000
Test 3:	356	10000	**Test 8:**	347	10000
Test 4:	347	10000	**Test 9:**	351	10000
Test 5:	304	10000	**Test 10:**	365	10000

Figure B.3: data from each iteration for the *Random Walk* at different population sizes.

Appendix C: Source Code

C.1 Avowal

I, Benjamin Edward Buckley, hereby certify that I understand the nature of plagiarism and collusion, and that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary. (25/04/2016)

C.2 Contents

C.3 Source

C.3.1 StiCo (code for the StiCo algorithm)

```
breed [ants ant]
breed [trails trail]
patches-own [pheromone visited?]
ants-own [direction? magnitude threshold detect interior?]
globals [constant-mag covered]

; ; ; ; ; Setup Procedure ; ; ; ; ;

to setup
  clear-all ;; clear screen

  set constant-mag 5 ;; default magnitude of angular rotation
  set covered 0 ;; total patches covered

  set-default-shape trails "line" ;; set turtle shapes
  set-default-shape ants "bug"
  create-ants population ;; create turtles

  ask ants [ ;; set turtle properties
    ;setxy random-pxcor random-pycor
    set color green
    set size 1
    set direction? true
    set magnitude 5
    set threshold 2
    set detect 0
    set interior? true
    ]

  ask patches [ ;; set patch properties
    set pheromone 0
    set visited? false
    ]

  reset-ticks
end

; ; ; ; ; Go Procedures ; ; ; ; ;

to go
  ;; set convergance
  if ticks >= 10000 [ stop ]

  ask ants [
    if who >= ticks [ stop ] ;; delay start
    hatch-trails 1 ;; hatch trails for each turtle
    [set size 0.3
```

```
      __set-line-thickness 0.1]

    detectpher
    stico

    if visited? = false ;; check if visited
    [ set visited? true ;; set visited
      set covered covered + 1 ;; increase total
      set pcolor red ] ;; set patch colour
  ]

  ask trails [
    set color color - fade-rate ;; make tail color darker
     if color mod 10 < 1  [ die ]      ;; die if we are almost at black
  ]

;; old code
;  ask patches [
;    if visited? = true [
;      set covered covered + 1
;         set pcolor red ;; set patch colour to red if visited by agent
;    ]
;  ]

  evaporate

  tick
end

to stico
 ifelse detect < threshold ;; no pheromone detected
   [ ifelse direction? = true ;; CW rotation
        [
        rt magnitude
        fd 0.1
          ask patch-here [set pheromone pheromone + 2] ;; propagate pheromone
        ]

        [ ;; CCW rotation
        lt magnitude
        fd 0.1
          ask patch-here [set pheromone pheromone + 2] ;; propagate pheromone
        ]
   ]

    [ ifelse interior? = true [ ;; interior sensor detects pheromone
```

```
          ifelse direction? = true ;; change rotation
            [ set direction? false ]
            [ set direction? true ]
        ] ;; pheromone detected
            [ set magnitude magnitude + 10 ] ;; increase magnitude
      ]
end

; ; ; ; ; Pheromone Detection Procedures ; ; ; ; ;
; Based on the model "Ants".
; Wilensky, U. (1997). NetLogo Ants model.
; http://ccl.northwestern.edu/netlogo/models/Ants.
; Center for Connected Learning and Computer-Based Modeling,
; Northwestern University, Evanston, IL.

to detectpher  ;; turtle procedure
   let pheromone-ahead pheromone-at-angle   0 ;; ahead angle
   let pheromone-right pheromone-at-angle  45 ;; right angle
  let pheromone-left  pheromone-at-angle -45 ;; left angle

         if (pheromone-right > pheromone-ahead) or (pheromone-left > pheromone-ahead) ;; check angle values
       [ ifelse direction? = true [ ;; check direction of rotation
          ifelse pheromone-right > pheromone-left ;; compare left and right pheromones
            [ set interior? true ] ;; set interior sensor to right
            [ set interior? false ] ] ;; set interior sensor to left
        [ifelse pheromone-right > pheromone-left
           [ set interior? false ]
           [ set interior? true ] ]
      set detect detect + pheromone] ;; increase detected pheromone
end

to-report pheromone-at-angle [angle] ;; return pheromones on patches at angles ahead
  let p patch-right-and-ahead angle 1
  if p = nobody[ report 0 ] ;; check for no patch
  report [pheromone] of p ;; return pheromone
end

; ; ; ; ; Pheromone Decay Procedure ; ; ; ; ;

to evaporate
```

```
    diffuse pheromone (diffusion-rate / 100) ;; share pheromone

    ask patches [
       set pheromone pheromone * (100 - evaporation-rate) / 100  ;; slowly evaporate pheromone
      ;set pcolor scale-color blue pheromone 0.1 10
    ]

    ask ants [
       set detect detect * (100 - evaporation-rate) / 100 ;; evaporate stored pheromone
       set magnitude (magnitude * (100 - evaporation-rate) / 100) + constant-mag ;; decay magnitude
      ;show magnitude
    ]
end

; ; ; ; ; Unused ; ; ; ; ;

;to bounce ;; move away from boundaries
;  if abs [pxcor] of patch-ahead 0.1 = max-pxcor
;    ;[ set heading (- heading) ]
;           [ifelse direction? = true ;; change rotation
;         [ set direction? false ]
;         [ set direction? true ]]
;  if abs [pycor] of patch-ahead 0.1 = max-pycor
;    ;[ set heading (180 - heading) ]
;          [ ifelse direction? = true ;; change rotation
;         [ set direction? false ]
;         [ set direction? true ]]
;end
```

C.3.2 BeePCo (V3) (code for the BeePCo (V3) algorithm)

```
breed [bees bee]
undirected-link-breed[perms perm]
undirected-link-breed[phers pher]
;directed-link-breed[phers pher]
bees-own[angle magnitude head-list]
globals[covered]


to setup
  clear-all ;; clear screen

  set covered 0

  create-bees population [ set shape "bee" fd random-float 1];; create turtles

  ask bees [ ;; set turtle properties
    set head-list []
    set size 1
    ;create-phers-to other bees
    ;create-phers-from other bees
    create-phers-with other bees
    create-perms-with other bees [hide-link]
    ]

  reset-ticks
end

to go

  if ticks >= 10000 [stop]

  ask bees [
    ifelse any? pher-neighbors = true
    [
     let hy p
     set head-list lput hy head-list
      let headav mean head-list ;(sum head-list) / length head-list
       ifelse headav > 180 [set angle headav - 180] [set angle headav + 180]

;     ask pher-neighbors [
;       set heading angle
;     ]

     ;set angle headav
       ;ifelse headav > 180 [set angle headav - 180] [set angle headav + 180]
     ;show heading
```

```
       ;set angle headav + 180
       ;show angle
       ;show headav
       ;show head-list
  ;     ask phers [
  ;       show link-heading
  ;     ]




  ;old
  ;      ask pher-neighbors [
  ;         set head-list [link-heading]
  ;           ;set angle theta self myself;sum link-heading / count (pher-neighbors)
  ;      ]
        ;set head-list [pher-neighbors [link-heading]]

        ;ask pher-neighbors [
          ;set head-list lput h head-list
        ;]

              ;set heading (180 - sum head-list / count (pher-neighbors))

        ;set magnitude count (pher-neighbors)
        ;rt angle
        ;set heading angle
        ;;;;;;;;



  ;      if ticks mod 20 = 0 [
  ;         set heading mean-heading [ heading ] of turtles in-radius 3 + 180]
          ;ifelse headi > 180 [set angle headi - 180] [set angle headi + 180]

        ;set heading angle

        fd 0.1
          if length head-list >= population [set head-list []]]
      [ ask patches in-radius 3 [
          set pcolor red
          set covered covered + 1
        ]
      ]
    ]

  ask phers [
    ;show link-heading
    if link-length > hopthreshold [
      die
    ]
  ]


;; reestablish pheromone connection
;  ask perms [
;    if link-length < 10 [
;      ask bees with perm link-length < 10[create-phers-with other bees]
;    ]
;  ]

if any? phers = false [stop]

tick

;if ticks >= 2500 [setup]
end

to-report p
  let d 0
  ask phers [
    set d link-heading
  ]
  ;ifelse d = 0[report 0]
  report d
end

;to-report mean-heading [ headings ]
;  let mean-x mean map sin headings
;  let mean-y mean map cos headings
;  report atan mean-x mean-y
;end


;to-report h
;  let b 0
;  ask pher-neighbors [ set b link-heading ]
;   ;ask phers [ ask phers with [self > myself] [ set b link-heading ] ]
;  ;ask pher-neighbors [set b link-heading]
;  ifelse b = 0 [ report 0 ]
;  [ report b ]
;end

;to-report theta [x y]
;  let m [tan (90 - link-heading)] of x
;  let n [tan (90 - link-heading)] of y
;
;  report list x y
;end

; ; ; ; ; UNUSED CODE ; ; ; ; ;
; ; ; ; ; Link Length Reporters ; ; ; ; ;

;to-report pherLength
;  let l 0
;  ask phers [ set l link-length ]
;  report l
;end
;
;to-report permLength
;  let d 0
;  ask perms [ set d link-length ]
;  report d
;end
```

C.3.3 Random Walk (code for the random walk algorithm)

```
breed [points point]
breed [radii radius]
patches-own[visited?]
globals[covered]

to setup
  clear-all
  set-default-shape points "circle"
  set-default-shape radii "circle"
    create-points population
    set covered 0

    ask points [
      set color green
      set size 1
    ]

    ask patches [
      set visited? false
    ]

    reset-ticks
  end

  ; ; ; ; GO PROCEDURES ; ; ; ; ;

  to go
    ;; set convergance
    if ticks >= 10000 [ stop ]

    ask points [
      if who >= ticks [ stop ] ;; delay start
      bounce
      fd 0.1
      rt random-float angle

      if visited? = false ;; check if visited
      [ set visited? true ;; set visited
        set covered covered + 1 ;; increase total
        set pcolor red ] ;; set patch colour
    ]
    tick
  end

  to bounce
    if abs [pxcor] of patch-ahead 0.1 = max-pxcor
      [ set heading (- heading) ]
    if abs [pycor] of patch-ahead 0.1 = max-pycor
      [ set heading (180 - heading) ]
  end
```

```

; ; ; ; ; OPTIONAL ; ; ; ; ;

to draw-walls
  ask patches with [abs pxcor = max-pxcor]
  [set pcolor grey]
  ask patches with [abs pycor = max-pycor]
  [set pcolor grey]
end
to hide-walls
  ask patches with [abs pxcor = max-pxcor]
  [set pcolor black]
  ask patches with [abs pycor = max-pycor]
  [set pcolor black]
end

to make-radius
  hatch-radii 1
  [ set size 5
    set color lput 64 extract-rgb color
    __set-line-thickness 0.2
    create-link-from myself
    [ tie
      hide-link ] ]
end
```

C.3.4 Additional Implementation (DGB: distance goal based model)

```
breed [points point]
breed [radii radius]
undirected-link-breed[laps lap]
directed-link-breed[r rs]
;breed [walls wall]
points-own [goal static?]
patches-own [visited?]
globals [pcurrent rcurrent far free li]

  to setup
    clear-all

    set-default-shape points "circle"
    set-default-shape radii "circle"
    set pcurrent 0
    set rcurrent 1

    create-points 1

    ask points [
      setxy random-pxcor random-pycor
      set color green
      set goal max-one-of patches [distance myself]
      show max-one-of patches [distance myself]
      set static? false
      make-radius
    ]

    ask patches [
      set visited? false
      ;set free patch-set self
      ;map patch
      ;[set free [pxcor pycor]]
    ]

    reset-ticks
  end

  to make-radius

      hatch-radii 1
    [ set size 5
      set color lput 64 extract-rgb color
      __set-line-thickness 0.2
      create-rs-from myself
      [ tie
        hide-link ] ]
  end

  to draw-walls
  ask patches with [abs pxcor = max-pxcor]
  [set pcolor grey]
  ask patches with [abs pycor = max-pycor]
  [set pcolor grey]
end
to hide-walls
  ask patches with [abs pxcor = max-pxcor]
  [set pcolor black]
  ask patches with [abs pycor = max-pycor]
  [set pcolor black]
end

;##############################################################

to go
  ;if (count (points with [goal != []]) = 0) [stop]
  if (count points >= population) [stop]

  ask point pcurrent [
    set li lapLength
    ;show li

    ifelse (static? = true)
    [ spawn ]

    [let d (distance goal)
    face goal
    ;show d
    ifelse d >= 1 ;or li > 3

    [if any? lap-neighbors = false or li > 5[fd 0.1]

      ifelse li > 5 [fd 0.1][
        set static? true
              ask laps [die]
    ask radii [
      ask patches in-radius 3[
      set visited? true
      set pcolor red
      ;set vis patch-set self
      ]]]]

    [ set static? true
      ask laps [die]
    ask radii [
      ask patches in-radius 3[
      set visited? true
      set pcolor red
      ;set vis patch-set self
      ]]]
    bounce]
    ]

  ;;if patch visited don't allow goal

  tick
end

to spawn
  set pcurrent pcurrent + 2
  ;set free free - vis

   hatch 1 [
      setxy random-pxcor random-pycor
      set color green
      set static? false
      ask points with [self < myself] [create-lap-with myself]
      ;set far max-one-of patches [distance myself]

      ;ifelse is-patch? far visited? = false
      ;[set goal far]
      ;[set goal max-one-of patches [distance myself + 3]]

      set goal max-one-of patches [distance myself]
      ;show max-one-of patches [distance myself]
      make-radius
   ]

end

to bounce
  if abs [pxcor] of patch-ahead 0.1 = max-pxcor
    [ set heading (- heading) ]
  if abs [pycor] of patch-ahead 0.1 = max-pycor
    [ set heading (180 - heading) ]
end

to-report lapLength
  let l 0
  ask laps [ set l link-length ]
  report l
end
```

C.3.5 Matlab Code (code used to make graphs)

```
st50 = [653,577,569,605,634,674,598,675,618,640]; st40 = [527,502,543,467,476,491,437,448,527,483];
st30 = [412,363,353,395,392,431,416,370,438,463]; st20 = [290,266,259,325,314,291,301,305,253,214];
be50 = [783,809,715,759,827,718,764,772,684,735]; be40 = [656,728,673,674,624,652,644,595,669,723];
be30 = [540,594,549,568,504,603,512,526,553,502,]; be20 = [386,416,322,355,392,370,342,361,421,395];
rw50 = [431,475,504,483,450,465,525,474,618,704]; rw40 = [487,521,477,534,621,457,440,433,544,471];
rw30 = [385,347,481,351,430,499,392,435,430,419]; rw20 = [380,374,356,347,304,373,451,347,351,365];

st50p = st50 / 1089; st40p = st40 / 1089;
st30p = st30 / 1089; st20p = st20 / 1089;
be50p = be50 / 1089; be40p = be40 / 1089;
be30p = be30 / 1089; be20p = be20 / 1089;
rw50p = rw50 / 1089; rw40p = rw40 / 1089;
rw30p = rw30 / 1089; rw20p = rw20 / 1089;

st50pm = mean(st50p); st40pm = mean(st40p);
st30pm = mean(st30p); st20pm = mean(st20p);
be50pm = mean(be50p); be40pm = mean(be40p);
be30pm = mean(be30p); be20pm = mean(be20p);
rw50pm = mean(rw50p); rw40pm = mean(rw40p);
rw30pm = mean(rw30p); rw20pm = mean(rw20p);

x = 1:10;

figure(1);
plot(x,st50p,'g-o');
title('Population Size = 50');
xlabel('iteration');
ylabel('% area coverage');
hold all;
plot(x,be50p,'r--*');
plot(x,rw50p,'b-.+');
s = refline([0 st50pm]);
b = refline([0 be50pm]);
r = refline([0 rw50pm]);
legend('StiCo','BeePCo(V3)','Random Walk');

figure(2);
plot(x,st40p,'g-o');
title('Population Size = 40');
xlabel('iteration');
ylabel('% area coverage');
hold all;
plot(x,be40p,'r--*');
plot(x,rw40p,'b-.+');
```

```
s2 = refline([0 st40pm]);
b2 = refline([0 be40pm]);
r2 = refline([0 rw40pm]);
legend('StiCo','BeePCo(V3)','Random Walk');

figure(3);
plot(x,st30p,'g-o');
title('Population Size = 30');
xlabel('iteration');
ylabel('% area coverage');
hold all;
plot(x,be30p,'r--*');
plot(x,rw30p,'b-.+');
s3 = refline([0 st30pm]);
b3 = refline([0 be30pm]);
r3 = refline([0 rw30pm]);
legend('StiCo','BeePCo(V3)','Random Walk');

figure(4);
plot(x,st20p,'g-o');
title('Population Size = 20');
xlabel('iteration');
ylabel('% area coverage');
hold all;
plot(x,be20p,'r--*');
plot(x,rw20p,'b-.+');
s4 = refline([0 st20pm]);
b4 = refline([0 be20pm]);
r4 = refline([0 rw20pm]);
legend('StiCo','BeePCo(V3)','Random Walk');
```