Nature Inspired Algorithms for Prioritized Foraging

Jade Zoë Abbott

Abstract

Foraging is a major problem in swarm robotics, which has been applied to many areas such as agriculture, and search and rescue. This dissertation defines a variation of swarm robotics foraging, called prioritized foraging. Prioritized foraging differs from other foraging problems, in that there are two types of items: prioritized items and non-prioritized items. Furthermore, a novel honey bee inspired foraging algorithm is developed. An empirical analysis of three foraging algorithms (a naïve algorithm, and two nature inspired algorithms, namely a desert ant inspired algorithm and the novel honey bee inspired algorithm) is performed on the prioritized foraging problem. The analysis investigates each algorithm's performance on the prioritized foraging problem in terms of the major swarm robotics characteristics of efficiency, scalability, flexibility, and robustness as well as the behaviours that enable those characteristics. The work concludes that the honey bee algorithm is highly efficient, highly flexible, highly scalable in terms of problem density, and most robust in terms of redundancy due to the algorithm's division of labour between prioritized and non-prioritized items. The desert ant algorithm is almost as efficient as the honey bee algorithm. The naïve algorithm has very poor efficiency, compared to the other algorithms, but was the most scalable in terms of swarm size. Both the desert ant and naïve algorithm experience poor flexibility, scalability in terms of problem density, and robustness in terms of redundancy.

Keywords:
Swarm Robotics, Foraging, Prioritized Foraging, Desert Ant, Honey bee

1. Introduction

Consider a search and rescue mission after a natural disaster or mining accident. These search and rescue missions are usually extremely dangerous for the human rescuers who are deployed to search for survivors. The use of swarm robotics to perform such search and rescue mission has been proposed and explored as an alternative to using human rescuers [1, 2].

Swarm robotics is the co-coordination of large numbers of relatively simple robots to perform a single collaborative function. Swarm robotics is inspired from the observation of social insects such as ants, termites, and bees [3]. An important activity of all natural swarms is foraging for resources. Foraging is defined as the search and collection of resources from sources in an environment and returning the resources to a collection point [4]. These resources could be food, water, or building materials. Foraging is an abstraction of the search and rescue problem where the trapped humans are items that robots need to locate and remove to a safe location.

In a search and rescue mission, the location of trapped humans is usually unknown and humans can be potentially be blocked by rubble, which needs to be removed before the humans can be safely removed. A swarm of robots would need to search for the humans, as fast as possible, to avoid further danger, injury or loss of life. If robots cannot locate the humans, they will have to begin clearing debris in the hope of locating them, as well as clearing the route between the trapped humans and the safe zone. Locating and removing the humans is prioritized above removal of the debris, but often the debris must be removed, in order to locate the trapped humans. Thus there exists a resource prioritization from the perspective of the robot.

Resource prioritization is also relevant in mining problems where the metal ore is prioritized and the waste rock should be cleared to better access the valuable ore. In common gold mining techniques, the ore and waste rock are collected and transported to the surface and chemical techniques are used to separate them. The transport of the waste rock (which forms the majority of the load) to the surface is extremely expensive, so there is a cost benefit to separate the ore and the waste rock beneath the surface and transport only the valuable ore to the surface.

The above search and rescue problem and mining problem can be abstracted as a foraging problem where there exist resources with differing priorities. In the case of search and rescue, the humans are the prioritized resource and the debris is the non-prioritized resource. In mining, the ore is the prioritized resource and the waste rock is the non-prioritized resource.

In nature, in times of stress, the collection of one resource may be prioritized over others - such as water during a drought or food before winter. Individuals in a natural swarm often adapt behaviour appropriately to enable greater collection of the prioritized item.

This study defines the prioritized foraging problem, and proposes three swarm robotics foraging algorithms to be evaluated on the prioritized foraging problem. The study proposes metrics to evaluate each algorithm's performance on the prioritized foraging problem in terms of foraging efficiency, flexibility, scalability and robustness. The algorithms are developed in a simulated environment. Each algorithm's performance is evaluated on environments of a variety of complexities, with various swarm configurations.

2. Background

2.1. Foraging in Nature

As with many swarm robotics problems, the inspiration for foraging comes from nature, in particular, social insects such as ants and honey bees.

2.1.1. Ant Foraging Behaviour

As with many swarm robotics problems, the inspiration for foraging comes from nature, in particular, social insects such as ants and honey bees. This section describes the biological models that are used in the algorithms derived by this thesis.

The study of ants has revealed that impressive emergent activities can be achieved with simple interacting agents with only a few simple rules. Many ant species use a form of communication known as stigmergy [5]. Ants perform indirect communication between ants by depositing a substance known as pheromone. In foraging, pheromone is deposited on the paths between the nest and the food source. Other ants detect the deposited pheromone and will prefer to follow paths which have a greater amount of pheromone deposit. Ant pheromones have been modelled in numerous swarm intelligence algorithms such as ant systems [6, 7].

Although algorithms based on ant foraging behaviour are used to solve optimization problems, the algorithms are often difficult to replicate in a real-life robot environment, since most of the algorithms need to model pheromone dropping and detection. A robotics algorithm that uses pheromone requires robots to be equipped with a substance-distributors, beacon-deployers or complex communication that simulates pheromone deposition and trail-following [8].

The desert seed-harvester ant (*Pogonomyrmex ant*) does not make use of stigmergy to forage, because pheromone deposited on desert sand would be blown away by the wind [9, 10]. Instead, desert ants use a technique called path integration for navigation to relocate food sources that have already been found by random exploration [11, 12]. Fig 1 demonstrates path integration. The black solid lines represent the path of the ant and the blue dotted lines show how the direction to the nest is maintained as the ant explores. The ant is constantly monitoring the change in heading from the original heading such that, at the destination, the ant has a direction directly back to the sink. As a result, a shortcut back to the nest is calculated in order to minimize the heat stress caused by walking through the hot desert. The shortcut back to the nest is known as the home vector [13]. Path integration, more commonly known as dead reckoning, is a key evolutionary aspect gained from living in the hot barren desert. When path integration fails, the ant resorts to using landmarks, such as the sun, for navigation [11].

Due to the lack of pheromone, the desert seed-harvester ant was simpler to model for real-robot interaction. Desert seed-harvester ant foraging has been modelled in [14, 15]. Hecker *et al* [15] performed experiments to compare the performance of two foraging algorithms: The one algorithm was based on desert ant foraging which has no pheromone-like communication and the other algorithm included pheromone-like communication. It was shown that communication improved performance; however, the desert ant based algorithm still performed comparatively well.

2.1.2. Bee Foraging Behaviour

Bees have an impressive set of abilities considering the simplicity of a single individual. They have the ability to remember the colour and shape of flowers [16], and in terms of navigation, they are able to learn local features and routes due to well-developed learning and memorizing capacities [17]. Honey-bees are even time aware [18].

Honey bees have efficient division of labour between different functions in the hive, such as foraging and brood-care. Within the foraging activity itself, honey bees perform foraging-specific division of labours. Honey bee foraging is made up of three different roles namely employed foraging, unemployed foraging and scouting [19]. Scouts explore the environment to locate new food sources. Once a source has been found, scouts return to the hive to communicate information about the located food source. To communicate the foraging information, scout bees perform a waggle-dance at the hive. The waggle-dance communicates information about the distance and bearing of the resource. Recruitment [19] refers to the process of communicating high

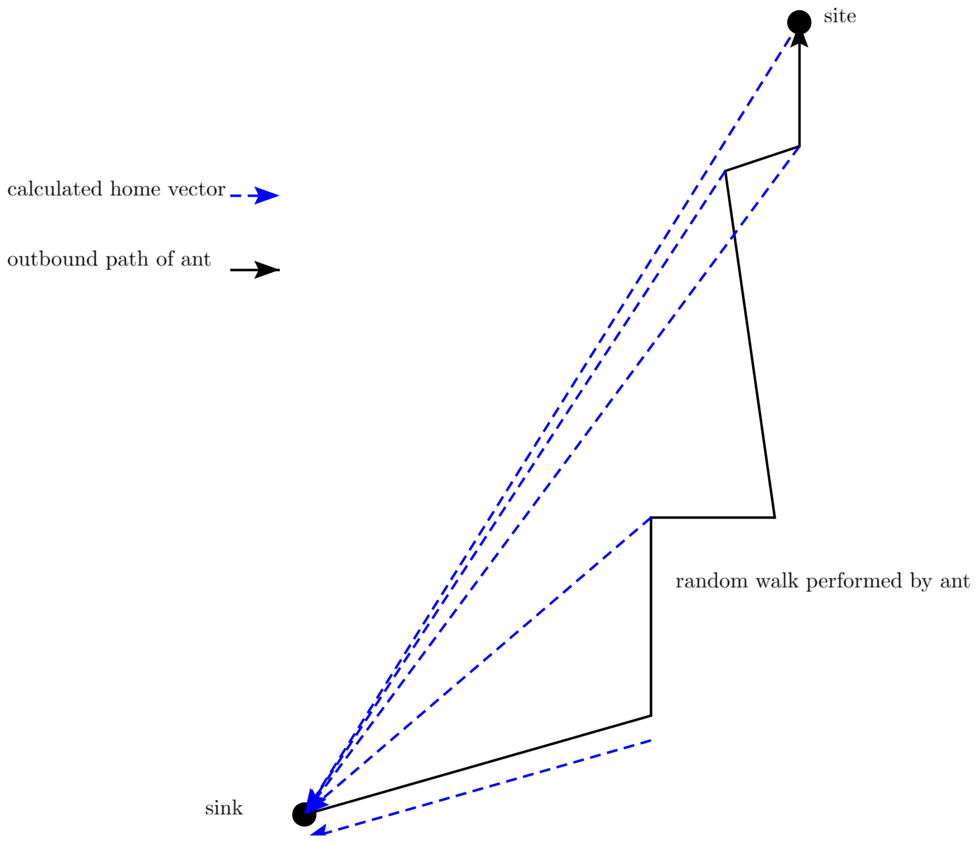


Figure 1: Path Integration

quality foraging sites to the swarm.

Unemployed foragers wait on the dance floor, and evaluate the dances of the scout bees. An unemployed bee selects a location described by the scout bees, and become an employed forager. Employed forager bees use the information about the resource, attempt to locate the source, load themselves with food, and return to the hive where unemployed foragers are ready to offload the food. Jansen [20] suggests that unemployed bees become exploring scouts when they do not detect any dancing scout bees.

2.2. Prioritized Foraging

The prioritized foraging problem, illustrated in Fig.2, is a modified version of the multi-foraging problem. In prioritized foraging, an environment has two types of items: prioritized items and non-prioritized items. The goal is to forage all the items of the prioritized type. The possibility exists that prioritized items become trapped among non-prioritized items and thus the non-prioritized items need to be removed from the environment to clear an access route to the prioritized items.

The prioritized foraging problem has increased difficulty compared to traditional multi-foraging problems, due the fact that foraging the non-prioritized item more than required will result in a waste of time and energy. The goal of research in prioritized foraging is to develop an algorithm to efficiently adapt the number of robots searching for prioritized items to those moving non-prioritized items out the way.

The prioritized foraging problem has similarities to the problem of search and rescue. For example, in the case of a collapsed building, robots will need to get to the survivors as quickly as possible. However, it is important that some robots move waste material in order to reach the trapped survivors. Prioritized foraging could be applied to the gold mining problem where gold needs to be foraged as a priority, and waste needs to be moved out of the way.

The prioritized foraging problem has the following characteristics: The environment has a bounded search space, with multiple source areas for items which can be placed in multiple sinks. Multiple object types exist in the environment - the prioritized items and the non-prioritized items. The primary performance measure for the prioritized foraging problem is time, in terms optimizing the time taken to forage each type of item.

3. Algorithms

3.1. Naïve foraging algorithm

Naïve foraging consists of the following tasks: searching for an item, grabbing an item, returning home with the item, and storing the item at the sink. The steps performed by the algorithm are illustrated in Figure 3, and described below:

1. Robots perform a random walk until they find an item.
2. On locating an item, the robot grips the item. If the item has been moved before the robot is able to pick it up, the robot will continue to explore; otherwise, the robot returns the item to the correct sink using a beacon-based homing algorithm.

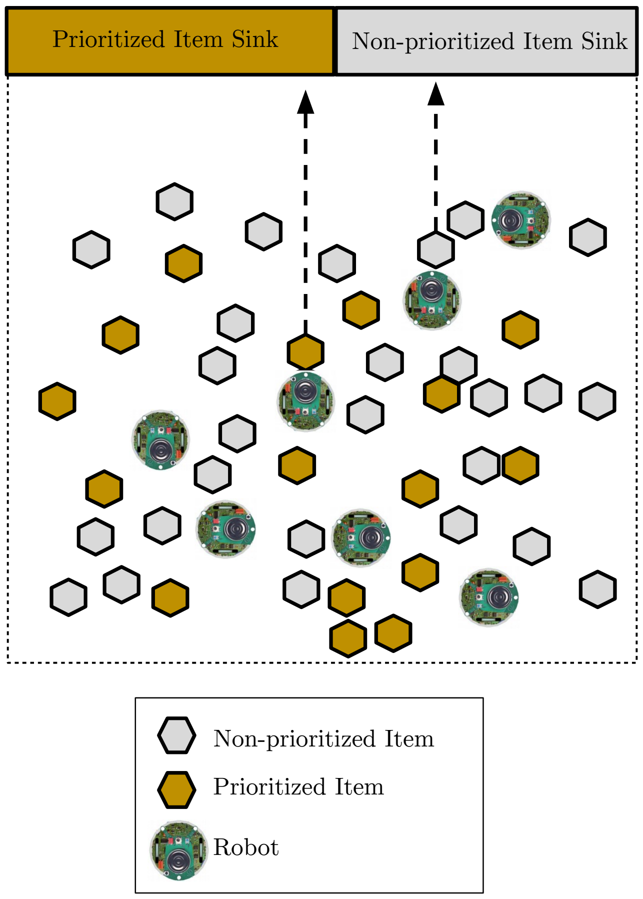


Figure 2: Prioritized Foraging Problem

Naïve foraging includes only the most minimal set of foraging actions and is included as a baseline for comparison to evaluate how novel techniques, such as the desert-ant foraging or honey-bee foraging, compare to a standard model [21, 8].

The following random walk is used: A robot chooses a random direction, σ, and a random distance, $m \in (0, M)$, where M is a chosen maximum path length. The robot walks in direction σ for distance m, or until the robot reaches a boundary. The robot then chooses new values for σ and m.

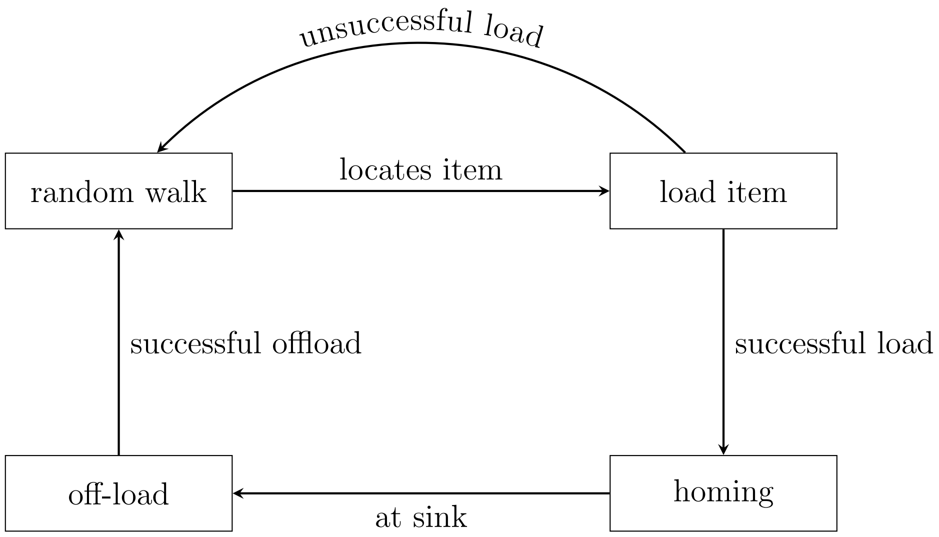


Figure 3: Naïve Foraging State Diagram

3.2. Desert ant foraging

As discussed in Section 2.1.1, desert ant foraging behaviour is a very suitable model for robot foraging, since no pheromone depositors or pheromone mimickry is required. Pheromone depositors or pheromone mimickry are quite complex to perform in robot environments. Instead of pheromone, desert ants use path integration (PI) to memorize the location of an existing food source and later to return to the memorized source to find more food, as discussed in Section 2.1.1. The notion of returning to a previously explored site is known as site fidelity [22]. The desert ant algorithm does not require communication between robots or the dispersal of beacons, and is thus simpler than many other swarm robotic foraging algorithms. The desert ant algorithm was defined by Hecker *et al* [15]. The desert ant foraging robots can be in the following states:

1. **Exploration State**: A robot in the exploration state performs a random walk with PI. The random walk used is the same random walk as discussed in Section 3.1. The purpose of the exploration state is to explore the environment to locate items.
2. **Loading State**: On finding an item, the robot switches into the loading state. In the loading state, the robot loads the item and memorizes the current PI vector. The PI vector is memorized so that the robot

can use it to return to the sink and then later to return to the site where the item was found. If loading of the item fails (perhaps due to another robot loading the item), then the robot returns to the exploration state; otherwise, the robot moves into the homing state.

3. **Homing State**: In the homing state, the robot uses the PI vector to move to the sink. The use of the PI vector will enable the robot to follow the most direct route back to the sink.
4. **Offloading State**: When the robot arrives at the sink, the robot switches into the offloading state, where the robot simply offloads the item into the sink.
5. **Locating State**: Once the robot has offloaded the item, the robot switches to the locating state. In the locating state, the robot follows the memorized PI vector to the location of the previous item. The premise of returning to the site where the previous item was found is that more items may exist where the previous item was located in order to locate more items. If another item is found, the robot moves into the loading state; otherwise, the robot returns to the exploration state in the search of new items.

All robots begin at random positions adjacent to the sink in the exploration state. The desert ant foraging states and transitions are illustrated in Figure 4.

3.3. Honey bee foraging

The honey bee foraging algorithm presented in this section is based on the foraging behaviour of honey bees as described in Section 2.1.2. The algorithm described requires robots to take on one of three roles, namely, scout robots, unemployed forager robots, or employed forager robots. The roles of the robots and the transitions to and from these roles are described in this section.

Figure 5 provides a simplified diagram for the honey bee prioritized foraging algorithm, illustrating the roles and the transitions between each role and the states within these roles. The dance and explore states of the scout robots are shown separately for clarity and the employed forager states are outlined separately in Figure 6.

To more succinctly define the honey bee prioritized foraging algorithm, the algorithms for each behavioural state, within each role, are provided below. For each algorithm, note that the current state of the robot is denoted

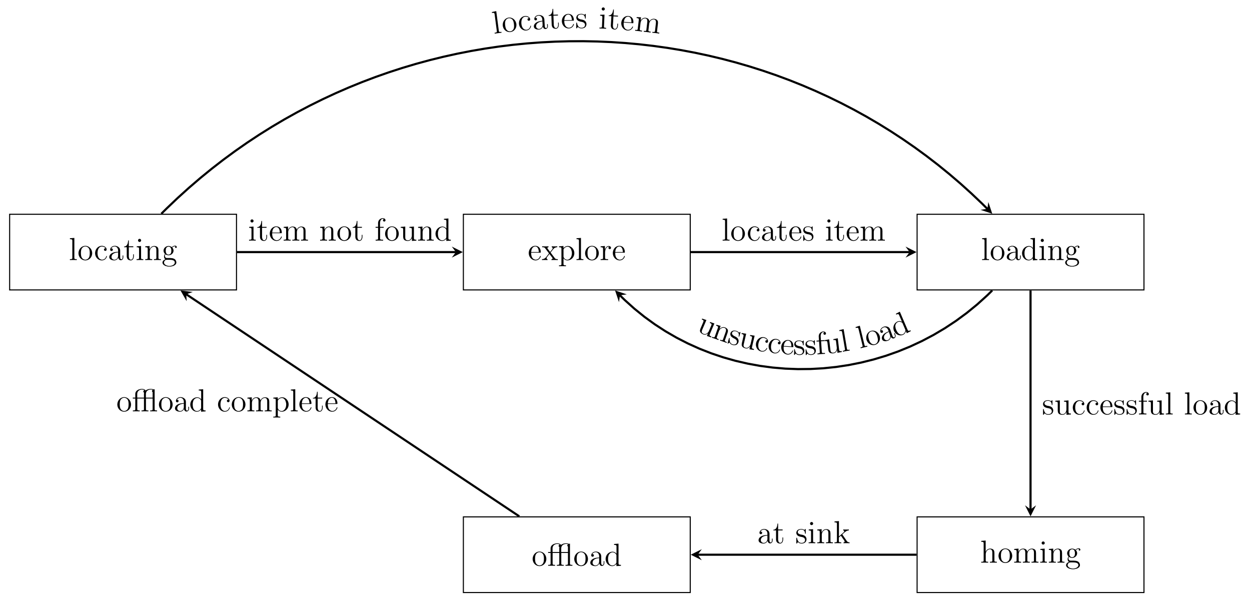


Figure 4: Desert Ant Foraging State Diagram

state, and the *state* is modified during the behaviour to denote the next state of the robot. The current time step is denoted as i. The algorithm for a single state is executed once per time step i.

Section 3.3.1 presents the behaviour of scout robots, while the behaviour of forager robots is described in Section 3.3.2. The initial state of the swarm is described in Section 3.3.3, while Section 3.3.4 describes the division of labour mechanism used by the honey bee algorithm.

3.3.1. Scout Robots

Scout robots mimic the scouting behaviour of the scout honeys bees as discussed in Section 2.1.2. The purpose of the scout honey bees is to locate high quality sites of resources. If the discovered site is of a high enough quality, then the scout will broadcast the location of the site to the unemployed forager robots at the sink.

Each scout robot begins in the explore state where the scout robot performs a random walk. Algorithm 1 provides a step-by-step explanation of the explore state of the scout robot. Variable ς saves the robot's item specialization as prioritized or non-prioritized. The random walk performed is the same random walk as discussed in Section 3.1. As the scout robot moves,

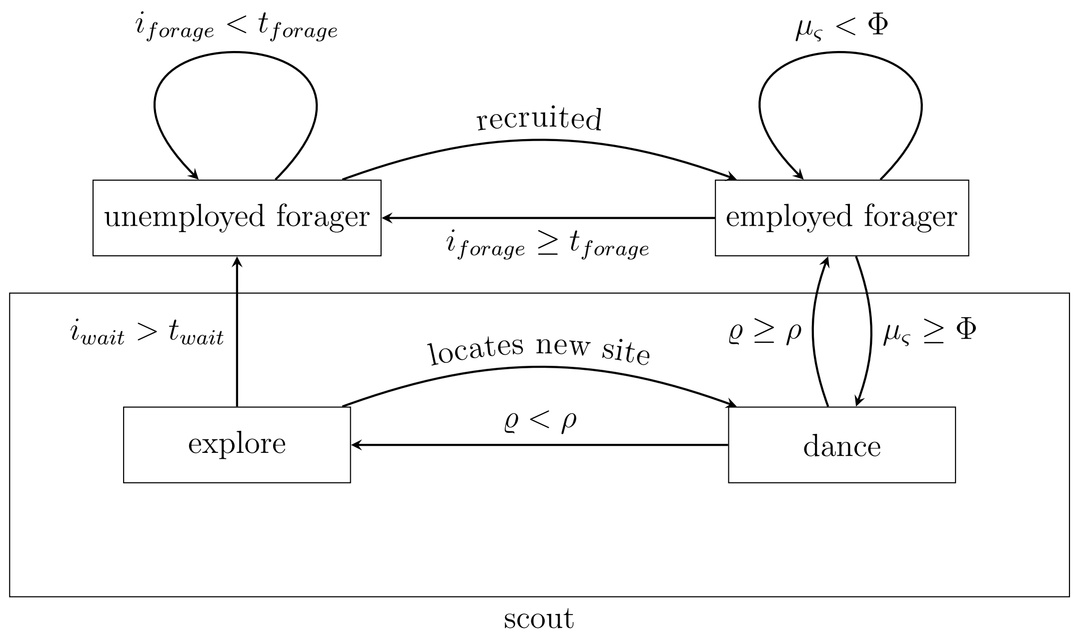


Figure 5: Honey Bee Foraging State Diagram

the robot maintains a PI vector v as explained in Section 2.1.1. Upon finding an item ϑ of type ς at site ξ, the robot loads the item and then performs an evaluation of the quality, μ_ς, of the site ξ for the item type ς.

The quality, μ_ς, of site ξ, for a robot scouting items of type ς is calculated as the estimated density of items of type ς in the local vicinity of the found item ϑ. The robot has distance sensor values $k_j \in [0,1]$ for $j = 1, ..., n$, where n is the number of distance sensors. A distance sensor reading of 0 means that nothing is detected in the sensor's range and a distance sensor reading of 1 indicates that the robot is touching an item. The sensor value k_j for item type ς, denoted k_{j_ς}, is calculated as

$$k_{j_\varsigma} = \begin{cases} k_j & \text{if detected item is type } \varsigma \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The site quality of type ς, μ_ς, is calculated using

$$\mu_\varsigma = \frac{1}{n} \sum_{x=1}^{n} k_{x_\varsigma} \tag{2}$$

Algorithm 1 Explore State of Scout Robot

```
1:  function EXPLORE($role, state, v, \varsigma, i$)
2:      perform a single random walk step from the current location
3:      update path integration vector $v$
4:      if item $\vartheta$ of priority $\varsigma$ is detected in vicinity then
5:          calculate quality $\mu_\varsigma$ of site $\xi$
6:          $\omega \leftarrow v$
7:          load item $\vartheta$
8:      else if $i_{explore} > f_{max}$ and $i_{explore} \leq t_{explore}$ then
9:          $\varsigma \leftarrow N$
10:     else if $i_{explore} > t_{explore}$ then
11:         $state \leftarrow homing$
12:     end if
13:     $i = i + 1$
14: end function
```

Before returning to the sink with the item, the scout memorizes the PI vector v in site location variable ω. The scout robot then switches to the homing state given in Algorithm 2. Using PI vector ω, the scout returns the item ϑ to the sink. When the scout robot has returned to the sink, S_ς, the scout robot offloads the item ϑ. If the quality μ_ς of the visited site ξ is larger than site quality threshold, Φ, the scout robot switches into the dance state, which is summarized in Algorithm 3. If the quality, μ_ς, of the visited site ξ is less than site quality threshold, Φ, the robot takes on the role of an employed forager and switches into the locate state, as outlined in Algorithm 4.

In the dance state, the scout robot communicates the location, ω, and quality, μ_ς, of the previous site, ξ, to the unemployed workers at the sink. The communication is akin to the waggle dance performed by honey bees in nature, as discussed in Section 2.1.2. A scout robot's "dance" takes the form of a localized broadcast communication between the scout and the unemployed forager robots near the sinks. The scout robot communicates the site quality and location for the unemployed foragers for t_{dance} time steps.

After the scout robot has completed broadcasting site details to the unemployed foragers, the scout robot must decide to either stay a scout robot and switch back to the explore state, or to become an employed forager robot and begin foraging the previous site. The scout robot becomes an employed forager robot with probability of ρ. If a random number $\varrho \in [0, 1]$ is selected

Algorithm 2 Homing State of Scout Robot

```
1: function SCOUT HOMING($role, state, v, \varsigma, i, \omega$)
2:     if robot is at sink $S_\varsigma$ and robot is loaded then
3:         robot offloads item $\vartheta$
4:         if $\mu_\varsigma > \Phi$ and $\varsigma$ = prioritized then
5:             $state \leftarrow dance$
6:         else
7:             $role \leftarrow$ employed forager
8:             $state \leftarrow locate$
9:         end if
10:     else
11:         calculate direction $\sigma$ to sink $S_\varsigma$ from current location
12:         if robot can move in direction $\sigma$ then
13:             robot moves a step in direction $\sigma$
14:         end if
15:     end if
16:     $i = i + 1$
17: end function
```

Algorithm 3 Dance State of Scout Robot

```
1: function DANCE($role, state, v, \varsigma, i, \omega, \mu_\varsigma$)
2:     if $i_{dance} < t_{dance}$ then
3:         broadcast $\omega$ and $\mu_\varsigma$ to robots at the sink
4:     else
5:         $\varrho = random(0, 1)$
6:         if $\varrho < \rho$ then
7:             $role \leftarrow$ employed forager
8:             $state \leftarrow locate$
9:         else
10:             $state \leftarrow explore$
11:         end if
12:     end if
13:     $i = i + 1$
14: end function
```

Algorithm 4 Locate State of Employed Forager

```
1: function FORAGE($role, state, v, \varsigma, i, \omega, \mu_\varsigma$)
2:     get direction $\sigma$ using path integration vector $\omega$
3:     robot moves a step in direction $\sigma$
4:     if (robot has finished path integration) OR (robot can see item of type $\varsigma$) then
5:         $state \leftarrow load$
6:     end if
7:     $i = i + 1$
8: end function
```

such that ϱ is less than ρ, then the scout robot remains a scout and begins to explore the environment. If ϱ is greater than or equal to ρ then the scout becomes an employed forager. Increasing the site quality threshold, ρ, will make the scout robots more selective about the sites they broadcast. Decreasing ρ will result in scout robots being less selective about the sites they broadcast.

3.3.2. Forager Robots

There are two types of forager robots: The unemployed foragers and employed foragers. The unemployed forager robots take on the role of unemployed foragers from foraging honey bees discussed in Section 2.1.2. Unemployed forager robots remain at the sink and await dance behaviour from a scout robot. This wait state is described in Algorithm 5.

A scout robot dances at the sink after locating an item ϑ at some site ξ. Each unemployed forager decides whether to listen to the scout robot with a probability of α. If an unemployed forager robot chooses to accept the details communicated by the scout, then the unemployed foraged robot has been recruited and becomes an employed forager robot. The unemployed forager takes on the same item specialization, ς, as the dancing scout robot. The unemployed forager will thus only search for items of type ς.

The employed forager robots are modelled based on the employed forager bees as discussed in Section 2.1.2. The purpose of the employed forager robot is to forage the sites communicated by scout robots. When an unemployed forager robot takes on the role of an employed forager robot after recruitment by a scout robot, the employed forager starts in the locate state, described in Algorithm 4. In the locate state, the employed forager robot uses the PI vec-

Algorithm 5 Wait State of Unemployed Forager

```
1: function WAIT($state, v, \varsigma, i, \omega, \mu_\varsigma$)
2:     if $i_{wait} \geq t_{wait}$ then
3:         $role \leftarrow scout$
4:         $\varsigma \leftarrow P$
5:         $state \leftarrow explore$
6:     else if scout robot is broadcasting at sink then
7:         receive site location $\omega$ and site quality $\mu_\varsigma$
8:         $role \leftarrow$ employed forager
9:         $state \leftarrow locate$
10:    end if
11:    $i = i + 1$
12: end function
```

tor ω to relocate the site ξ. Once the site ξ has been relocated, the employed forager robot switches into the load state, as described in Algorithm 6. If an item ϑ of type ς is detected in the vicinity of the located site, then the item is loaded. After successfully loading the item, the robot switches to the homing state. If the employed forager robot does not detect an object in the vicinity of the site ξ, then the employed robot switches to the local search state in order to perform a brief local search for items nearby.

Algorithm 6 Load State of Employed Forager

```
1: function LOADING($role, state, v, \varsigma, i, \omega, \mu_\varsigma$)
2:     if item $\vartheta$ of priority $\varsigma$ is detected in vicinity then
3:         LOAD($\vartheta$)
4:         $state \leftarrow homing$
5:     else
6:         $state \leftarrow local\_search$
7:     end if
8: end function
```

The local search is performed for a limited number of time steps, t_{ls}. At each time step, i_{ls}, the robot checks if an item of priority ς is nearby and if the item can be loaded. If an item ς can be loaded, then the robot loads the item and switches to the homing state. If the employed forager robot can see an item ς, but it is not close enough to be loaded, then the robot moves in the direction of the item. If the employed forager can not see an item in the vicinity, then the robot moves in a random direction, σ. The local search state is summarized in Algorithm 7. If i_{ls} is greater than t_{ls}, then the foraging site is depleted and so the employed forager robot must return to the sink without an item.

Algorithm 7 Local Search State of Employed Forager

```
function LOCAL_SEARCH($role, state, v, \varsigma, i, \omega, \mu_\varsigma$)
    if $i_{ls} < t_{ls}$ then
        if item $\vartheta$ of priority $\varsigma$ is nearby and can be loaded then
            LOAD($\vartheta$)
            $state \leftarrow homing$
        else if item $\vartheta$ of priority $\varsigma$ can be seen but is not close enough then
            select direction $\sigma$ to move towards item $\vartheta$
            robot moves a step in direction $\sigma$
        else
            select a random direction $\sigma$
            robot moves a step in direction $\sigma$
        end if
    else
        $state \leftarrow homing$
    end if
    $i = i + 1$
end function
```

When in the homing state, the employed forager robot moves towards the appropriate sink, S_ς, in order to off-load item ϑ. Once at sink S_ς, if the employed forager robot is loaded, it offloads the item and switches back to the locate state and thus repeats the steps of foraging the site ξ. If the employed forager robot is not loaded, the employed forager robot takes on the role of an unemployed forager and switches into the wait state. The reason for switching from an employed forager to an unemployed forager is because an item could not be found at the site ξ, and the site has likely been depleted. Thus the employed forager switches to an unemployed forager in order to await recruitment by a scout robot.

The states and transitions of the employed forager are shown in more detail in Figure 6.

Algorithm 8 Homing State of Employed Forager Robot

```
1:  function EMPLOYED FORAGER HOMING($role, state, v, \varsigma, i, \omega$)
2:      if robot is at sink $S_\varsigma$ then
3:          if robot is loaded then
4:              robot offloads item $\vartheta$
5:              $state \leftarrow locate$
6:          else if robot is not loaded then
7:              $role \leftarrow$ unemployed forager
8:              $state \leftarrow wait$
9:          end if
10:     else
11:         calculate direction $\sigma$ to sink $S_\varsigma$ from current location
12:         if robot can move in direction $\sigma$ then
13:             robot moves a step in direction $\sigma$
14:         end if
15:     end if
16:     $i = i + 1$
17: end function
```

The unemployed forager robot role allows the number of active robots in the environment to be regulated so that there are not too many robots attempting to forage or explore at once. An environment with too many employed foragers would result in more collisions between robots, which would mean that the employed foragers take longer to forage items. Also, if there are too many employed foragers in an environment which is sparse in items, then the employed foragers are not only causing collisions but they are also wasting energy by exploring areas unnecessarily.

Unemployed forager robots become scout robots if no scout robot broadcasts are detected for t_{wait} time steps. The control parameter t_{wait} is the maximum waiting time an unemployed forager can spend in the waiting state before switching to a scout robot, and i_{wait} is the time spent by a robot in the waiting state. Decreasing t_{wait} results in more scout robots exploring the environment and less unemployed foragers that a scout robot, who may have found quality sites, can recruit. Increasing t_{wait} results in a greater number of unemployed forager robots waiting to be recruited. The greater number of unemployed foragers can form a large work force for a recruiting scout. However, too many unemployed foragers result in a smaller work force in the

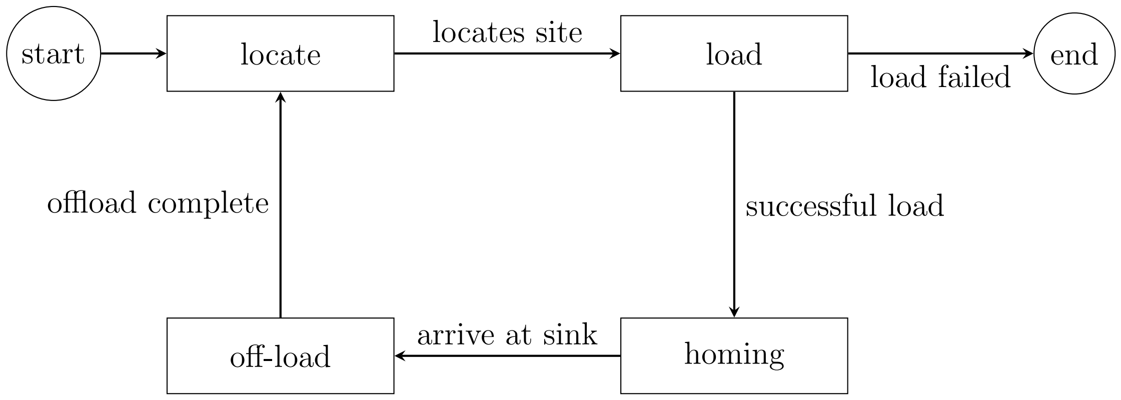


Figure 6: State Diagram of an Employed Forager Robot

foraging environment.

3.3.3. Initial States

A portion of the robots are initialized as scout robots in the explore state and the rest as unemployed forager robots. All robots are initialized adjacent to the sink. The percentage of robots initialized as scout robots is $X \in (0, 100)$. Unemployed forager robots require a scout robot to recruit them to become employed forager robots. At initialization, the scout robots do not have site location details available, and therefore robots can not be initialized as employed forager robots.

3.3.4. Division of Labour

The honey bee algorithm has two levels of division of labour. The first level is the division of labour between the scout, employed forager, and unemployed forager roles.

Another level of division of labour exists to deal with division of labour between foraging items with different priorities. Item-type division of labour is defined in this thesis as the division of labour between foraging prioritized item types and non-prioritized item types.

In nature, in times of drought, bees prioritize water over nectar or pollen. Honey bees would be sent out to forage for water but may encounter pollen while searching for water. If the foraging honey bee happens to encounter pollen, it will forage the pollen but will not communicate the discovery of the pollen site to the unemployed foragers [19]. Using the bee's prioritization of resources as inspiration, the following rules for item-type division of labour are defined:

1. A scout robot that is set to search for the prioritized type (i.e. ς is set to prioritized items) will forage a non-prioritized type only if a prioritized item can not be located for the maximum time period, f_{max}. If $i_{explore} > f_{max}$ then ς will switch to foraging the non-prioritized type ς = non-prioritized. The described rule is given explicitly in Algorithm 1.
2. An employed robot foraging the non-prioritized item type will forage the non-prioritized item type until the robot fails to relocate a previously located non-prioritized item type site, or until the robot locates a prioritized item. For both of these cases, the robot will switch to foraging the prioritized item type ς = non-prioritized.
3. A robot foraging a non-prioritized item will not communicate the location of the non-prioritized item site by dancing.

For the purposes of this study, each robot of each algorithm is assigned an initial item type to forage. The robots of the desert ant foraging algorithm and the naïve algorithm do not have item-type level division of labour and thus will continue to forage the same item-type that they were assigned throughout the experiment. The robots in the honey bee algorithm may switch what item-types they forage during the experiment, due to the item-type division of labour.

4. Methodology

4.1. Robots

Foraging robots occur in all shapes, sizes and capabilities. Some robots have powerful GPS capabilities and advanced long distance sensors, while others are much simpler. This chapter defines the capabilities of the simulated robots to be used in this study. The robots are described in Section 4.1.1, while Section **??** outlines the navigational capabilities of the robots.

4.1.1. Robot Description

The artificial robots modelled in this study are based on e-puck robots [23], with additional grippers. Each simulated robot is equipped with a 360 degree camera to identify objects around the robot, as well as eight local distance sensors equally spaced around the circular perimeter of the robot. Both camera and distance sensors have a depth of view of five times the robot's size. Robots use local communication which can occur in a radius of five times the robot's size. The sensor and communication range is sufficiently localized with respect to the size of the environment. Each simulated robot can forage a single item at a time. The robots do not have a global positioning system (GPS) capability to locate items to position themselves in the environment. A robot can not see occluded items. As a result, the simulated robots have to explore the environment to find the prioritized items.

4.2. Simulator

A spatially discrete 2-dimensional grid world simulator has been developed and used in this thesis in order to accelerate computation. Discrete 2-dimensional grid world simulators are also used in [24, 10]. In real robot experiments, algorithm performance is sensitive to the amount of time taken to load items and manoeuvre the robots [21]. The 2-dimensional grid world simulator allows for movement and loading time to be standardized across all algorithms for effective comparison.

The simulated robots function as follows:

- Each robot fits into one grid block and each item takes up one grid block.
- Only a single object can occupy a grid block at a time. An object is either a robot or an item. Since only one object can occupy a grid block at a time, collisions and congestion can occur.
- Each robot can move to an adjacent cell in any direction.
- Robots can load, transport and offload a single item at a time.
- If a robot cannot pick up an item, the item is an obstacle that a robot may have to navigate around in order to reach its destination.

The prioritized and non-prioritized sinks were placed next to each other, on a single side of the environment. The sinks were marked by beacons that all robots can detect and navigate towards. The reason why the sinks were not placed in the centre of the environment, as is commonly found in swarm robotics research [25], is because the prioritized foraging problem is inspired from using a swarm of robots to rescue trapped miners in mining tunnels, discussed in Section **??**. A mining tunnel has a single entrance where the gold and waste must be moved to, in order to be transported to the surface [26]. Since there is only a single entrance at the beginning of a tunnel, the sinks need to occur at the beginning of the tunnel so that items can be easily exported.

4.3. Environments

In order to test flexiblity on different environment types, many types of environments should be examined. This section defines the various environments and presents how they were generated. Section 4.3.1 discusses the parameters for the environments, while different environment distributions, and the algorithms to generate those distributions are presented in Section 4.3.2. Finally, the accessible environment is defined in Section 4.3.3.

4.3.1. Environmental Parameters

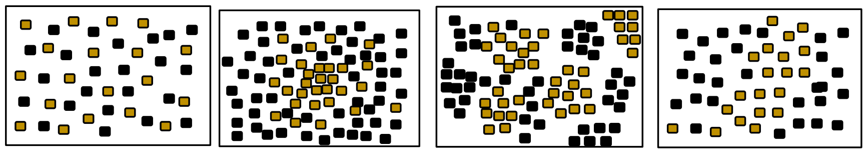
The experiments were run on different environments where each environment has different item distributions, environment sizes, item densities and different ratios of prioritized to non–prioritized items. All the environments are square. The length and width of the square environment grid is denoted as Λ, where $\Lambda \in \{50, 100, 200, 300, 500\}$.

Different values for the density, p, of the items on the grid were chosen, such that if $p = 0.9$, then 90% of the grid cells are occupied by items, where $p \in \{0.05, 0.2, 0.5, 0.7, 0.9\}$. Environments with a larger item density are more complex to forage, because there exists a higher probability that items will occlude each other. Thus, a robot may not be able to access items of the type that it is specialized to forage, since its access to those items is occluded by items of a different type. For this reason, we refer to the density of items on the grid as the problem complexity.

The environment type ratio, r, is defined as the ratio of prioritized to non-prioritized items in an environment, where $r \in \{0, 0.2, 0.25, 0.33, 0.5, 0.67, 0.75, 0.8, 1\}$. When $r = 0$, an environment contains no prioritized items, and when $r = 1$, only prioritized items exist in the environment. Environments with a small r value have an abundance of non-prioritized items which increases the likelihood of non-prioritized items preventing access to prioritized items.

4.3.2. Environment Distributions

The environment distribution, ϵ, is defined as the distribution of prioritized and non-prioritized items in an environment, such that $\epsilon \in \{uniform, clustered, gaussian, vei$ Each environment distribution was chosen in order to examine different characteristics of the algorithms. The item distributions are illustrated in Figure 7, where each lighter shaded square is a prioritized item and a non-prioritized item is shown by a darker shaded square. The four environment distributions were generated as follows:



(a) Uniform (b) Gaussian (c) Clustered (d) Vein

Figure 7: Environment Classes

1. The position of each item in a uniformly distributed environment is selected from a uniform distribution (refer to Figure 7a). The uniformly distributed environment has uniform concentrations of prioritized and non-prioritized items across the environment and thus is used as a control environment. Pseudo-code for generation of uniform environments is provided in Algorithm 9.
2. For the Gaussian environments, the positions of the prioritized items are sampled from a Gaussian distribution. The mean of the Gaussian disitribution is the centre of the grid. The deviation of the Gaussian distribution was selected as $deviation = S * E_p/2$ to ensure that generation took a reasonable amount of time (that the same positions are not reselected regularly), and that the non-prioritized items are densely concentrated (refer to Figure 7b). Pseudo-code for generation of Gaussian environments is provided in Algorithm 10. The positions of the non-prioritized items are selected from a uniform distribution, after placing the prioritized items.
   In Gaussian distributed environments, prioritized items occur in high concentration towards the center of the environment. More non-prioritized items occur on the outskirts of the environment, surrounding the prioritized items in the centre.

Algorithm 9 Uniform Distributed Environments

```
function UNIFORM($numberItems, r, S$)
    nonPrioritizedItemsLeft = floor((1-$r$)*numberItems)
    prioritizedItemsLeft = floor( $r$*numberItems)
    while $prioritizedItemsLeft > 0$ do
        x ← uniform(0, S)
        y ← uniform(0, S)
        if gridCell (x,y) is empty then
            Place prioritized item at (x,y)
            Decrement $prioritizedItemsLeft$
        end if
    end while
    while $nonPrioritizedItemsLeft > 0$ do
        x ← uniform(0, S)
        y ← uniform(0, S)
        if gridCell (x,y) is empty then
            Place prioritized item at (x,y)
            Decrement $nonPrioritizedItemsLeft$
        end if
    end while
end function
```

The Gaussian environments are used to examine whether each algorithm will enable the robot swarm to forage or navigate past the non-prioritized items to reach the high concentration of prioritized items in the environment's centre.

3. Environments with a vein distribution resemble the patterns observed in naturally occurring gold reefs [27] (refer to Figure 7d). In a gold reef, molten gold fills planar fractures between rock resulting in a vein of gold. Inspired by gold reefs, vein distributed environments have a long thin vein of prioritized items running from one side of the environment to another. The vein of prioritized items was surrounded by non-prioritized items.

   The vein environments aimed to test whether a swarm of robots could forage the continuous length of the vein of prioritized items, before foraging non-prioritized items. A swarm that is able to detect the location of the vein and return to the vein's location after foraging an item should forage more prioritized items initially than an algorithm that cannot detect and remember the location of the vein. Pseudo-code for generation of vein environments is provided in Algorithm 11.
4. Environments with a clustered item distribution have a random number of clusters of items of the same type (refer to Figure 7c). After clusters have been generated, each cluster is labelled randomly (with a Bernoulli disitribution with probability equal to the item ratio, r, required for that environment), as either a cluster of prioritized items or a cluster of non-prioritized items.

   The goal of performing experiments in a clustered environment was to test an algorithm's ability to exploit areas which are rich in prioritized items. Clustered environments are also used to test whether an algorithm can either navigate around or forage non-prioritized items. A clustered environment can also test an algorithm's ability to remember locations of areas which have a high density of prioritized items in order to aid more efficient access to prioritized items. Pseudo-code for generation of clustered environments is provided in Algorithm 12 and Algorithm 13.

To summarize, the challenges introduced by the more complex distributions (the Gaussian, clustered and vein environment) aim to test a swarm's ability to:

- navigate past obstacles efficiently, or alternatively, to forage obstacles efficiently, and to
- return to areas rich in prioritized items to forage these areas.

Algorithm 10 Gaussian Distributed Environments

```
function GAUSSIAN($numberItems, r, S$)
    Calculate environment centre point $(x_c, y_c)$
    prioritizedItemsLeft = floor($r$*numberItems)
    nonPrioritizedItemsLeft = floor((1-$r$)*numberItems)
    deviation = S*$r/2$
    while $prioritizedItemsLeft > 0$ do
        $x \leftarrow floor(gaussian(x_c, deviation))$
        $y \leftarrow floor(gaussian(y_c, deviation))$
        if gridCell (x,y) is empty and is valid then
            Place prioritized item at (x,y)
            Decrement $prioritizedItemsLeft$
        end if
    end while
    while $nonPrioritizedItemsLeft > 0$ do
        x ← uniform(0, S)
        y ← uniform(0, S)
        if gridCell (x,y) is empty then
            Place non prioritized item at (x,y)
            Decrement $nonPrioritizedItemsLeft$
        end if
    end while
end function
```

4.3.3. Accessible Environment

To aid discussions about the algorithms' performance on different types of environments, the concept of an **accessible environment** should be introduced. For the purposes of this thesis, an accessible environment is the parts of the environment that can be foraged by the swarm and are not blocked by other items. Items that are hidden behind other items are not accessible to the robot swarm and are thus not part of the accessible environment. The accessible environment is much larger in a less densely populated

Algorithm 11 Vein Distributed Environments

```
function VEIN($numberItems, r, S$)
    Select two random sides from the grid
    Select a random points on each sides, $(x_0, y_0)$ and $(x_1, y_1)$
    Calculate gradient of vein, $m = (y_0 - y_1)/(x_0 - x_1)$
    Calculate $c$ of equation for line vein, $c = (y_0 - m * x_0)$
    prioritizedItemsLeft = floor($r$*numberItems)
    nonPrioritizedItemsLeft = floor((1-$r$)*numberItems)
    while $prioritizedItemsLeft > 0$ do
        $x \leftarrow uniform(min(x_0, x_1), max(x_0, x_1))$
        $y \leftarrow m * x + c$
        if gridCell (x,y) is valid and gridCell (x,y) is empty then
            Place prioritized item at (x,y)
            Decrement $prioritizedItemsLeft$
        end if
    end while
    while $nonPrioritizedItemsLeft > 0$ do
        x ← uniform(0, S)
        y ← uniform(0, S)
        if gridCell (x,y) is empty then
            Place nonprioritized item at (x,y)
            Decrement $nonPrioritizedItemsLeft$
        end if
    end while
end function
```

Algorithm 12 Clustered Distributed Environments (Part 1)

```
function CLUSTERED($numitems, r, S$)
    Generate number of clusters, $total = uniform(3, 15)$
    Calculate number of prioritized clusters, $clusters_p = floor(r * total)$
    Calculate number of non-prioritized clusters, $clusters_{np} = floor((1 - r) * total)$
    Calculate number of prioritized items, $total_p = floor(r * numitems)$
    Calculate number of non-prioritized items, $total_{np} = floor((1 - r) * numitems)$
    $ave_p = total_p / clusters_p$
    $ave_n p = total_{np} / clusters_{np}$
    $clustersToGenerate_p = clusters_p$
    $clustersToGenerate_{np} = clusters_{np}$
    while $clustersToGenerate_p > 0$ do
        sample centroid for cluster $C$ (x,y) uniformly from grid
        $num_p = uniform(ave_p/2,\ 2 * ave_p)$
        Calculus radius, $r$, of cluster $C$ as $r = \sqrt{num_p/\pi}$
        if cluster $C$ does not collide with existing clusters then
            Save centroid and radius of cluster $C$ to list
            Decrement $clustersToGenerate_p$
            $itemsToGenerate_p = num_p$
            while $itemsToGenerate_p > 0$ do
                $x_p = gaussian(x, r)$
                $y_p = gaussian(y, r)$
                if position $(x_p, y_p)$ is valid then
                    Place prioritized item at $(x_p, y_p)$
                    Decrement $itemsToGenerate_{np}$
                end if
            end while
        end if
    end while
```

Algorithm 13 Clustered Distributed Environments (Part 2)

```
29:     while $clustersToGenerate_{np} > 0$ do
30:         sample centroid for cluster $C$ (x,y) uniformly from grid
31:         $num_{np} = uniform(ave_{np}/2, 2ave_{np})$
32:         Calculus radius, $r$, of cluster $C$ as $r = \sqrt{num_{np}/\pi}$
33:         if cluster $C$ does not collide with existing clusters then
34:             Save centroid and radius of cluster $C$ to list
35:             Decrement $clustersToGenerate_{np}$
36:             $itemsToGenerate_{np} = num_{np}$
37:             while $itemsToGenerate_{np} > 0$ do
38:                 $x_p = gaussian(x, deviation = r)$
39:                 $y_p = gaussian(y, deviation = r)$
40:                 if position $(x_{np}, y_{np})$ is valid then
41:                     Place non-prioritized item at $(x_{np}, y_{np})$
42:                     Decrement $itemsToGenerate_{np}$
43:                 end if
44:             end while
45:         end if
46:     end while
47: end function
```

environment. This also introduces the concept of the environment ratio of a given accessible environment. For example, an environment may have a high ratio of prioritized items overall, but if the prioritized items are surrounded completely by non-prioritized items, then the accessible environment consists only of non-prioritized items and has an environmental item type ratio of 0. The environment ratio of the accessible environment changes differently per environment disitribution type. In high density Gaussian environments, the accessible environment will initially consist of mostly non-prioritized items, but as robots forage towards the centre, the accessible environment will consist largely of prioritized items. For uniform environments, all accessible environments would have a similar or same environment item ratio as the entire environment. In clustered environments, depending on the configuration of the clusters, the environment ratio of the accessible environments would vary. For vein environments, at the beginning of foraging, the accessible environment ratio would be low in high density environments if the vein is perpendicular to the sink since only the "entrance" of the prioritized vein would be exposed to the robots and the rest of the prioritized items would be stacked behind those items.

4.4. Swarm parameters

For all algorithms, each robot swarm was initialized with a swarm specialization ratio, $\tau \in \{0, 0.2, 0.25, 0.333, 0.5, 0.667, 0.75, 0.8, 1\}$. The swarm specialization ratio is the ratio of robots foraging prioritized items to non-prioritized items. When no robots are set to initially forage prioritized items, then $\tau = 0$ and when all robots are set to initially forage prioritized items, then $\tau = 1$.

The ability of an algorithm to adapt the value of τ appropriately for a given r indicates the flexibility of the algorithm.

The swarm density, c, is defined as the number of cells occupied by a robot, as a percentage of the grid size Λ (the length of the side of an environment), where values of $c \in \{0.1, 0.3, 0.5, 0.7, 1\}$ were evaluated. The swarm density is varied in order to test the scalability of the algorithm. The swarm density is also varied to test each algorithms' ability to adjust the number of actively foraging robots to the density of the items in the environment, c, as well as to adjust the number of robots actively foraging an item type, τ, to the item type density, r.

The honey bee algorithm specific parameters were selected based on [19], where $t_{wait} = 200$ time steps, $f_{max} = 100$ time steps, $\Phi = 0.8$ and $\rho =$

0.1. Testing the effect of each of the parameters specific to the honey bee algorithm was not in the scope of this thesis.

4.5. Experimentation

The initial position of each robot was randomly selected, adjacent to the sink. For the naïve algorithm and desert ant algorithm, all robots began in the exploration state. For the honey bee algorithm, all unemployed foraging robots were initialized in the waiting state, and the scout robots were initialized in the explore state. An experiment is defined as the 30 independant runs for each algorithm, with a specific set of swarm parameters, and on an environment which has a specific set of environmental parameters. An experiment is run for every possible combination of swarm and environmental parameters.

4.6. Performance measures

The performance of swarm robotics algorithms can be measured by examining the algorithm's ability to perform a task in terms of efficiency, scalability, flexibility, and robustness.

4.7. Foraging Efficiency

The foraging efficiency, E_P, of an algorithm is defined as the ratio of the number of prioritized items collected by all robots in a fixed time period on a specific environment to the total number of prioritized items that exist in the environment. The foraging efficiency metric is similar to the efficiency metric used by Hecker *et al* [10]. The metrics used in experiments performed by Hecker *et al* evaluate the performance of foraging algorithms and are thus appropriate for use in the experiments of this study. The foraging efficiency measure, E_P, only considers the prioritized items for the prioritized foraging problem.

4.7.1. Flexibility

As stated in Section **??**, flexibility refers to the effect of variations in the environment and tasks on the co-ordination mechanisms of swarm robotics algorithms. An algorithm that is highly flexible is one that has been optimized for a specific environment, but is equally efficient on a different environment [10].

The flexibility performance measures used in this study are based on the flexibility performance measures used in [10], which have been adapted for the prioritized foraging problem by only taking into account the prioritized item. The flexibility study addresses two aspects: Flexibility with respect to different prioritized item ratios, r, and flexibility with respect to different environment distributions.

In order to more succinctly define the flexibility performance measures, the following notation is defined:

Let $\overline{r}$ be the list containing all considered elements for environment item type ratio, r, and let n_r be the number of elements in the list. Let $\overline{\tau}$ be the list containing all considered elements for initial swarm specialization ratio, $\tau(0)$, and let the number of elements in the list be n_t. Similarly, let $\overline{\epsilon}$ be a list of each environment distribution and let n_ϵ be the number of elements in the list.

Let Z be a matrix of dimensions $n_t \times n_r$. Define each entry of the matrix Z_{ij} as the average foraging efficiency, E_P, over all experiments, for a specific environment ratio r_j, where r_j is the jth element of $\overline{r}$, and for a specific initial swarm specialization ratio $\tau_i(0)$, such that $\tau_i(0)$ is the ith element of $\overline{\tau}$.

Similarly, define matrix D with dimensions $n_t \times n_\epsilon$, such that each entry of the matrix D_{ij} is the average foraging efficiency, over all experiments with environment distribution, ϵ_j, and with the initial swarm initialization ratio, $\tau_i(0)$.

The goal of examining flexibility over different prioritized item ratios is to determine how well the optimum value for initial swarm specialization ratio, $\tau(0)$, for a specific environment item ratio, r_u, generalizes across all other environment item ratios, r_v, where $u, v = 1, ..., n_r$ and $u \neq v$. The optimum value for $\tau(0)$ for environments of a specific environment item ratio is the value that yields the greatest average efficiency for those environments.

To do so, a list I of size n_r is defined, where each entry I_j is set to the index k that yields the maximum value for Z_{ij}, for $i = 1, ..., n_\tau$, for environments with a specific environment type ratio r_j. More succinctly:

$$I_j = (k | Z_{kj} = \max_{i=1,...,n_\tau} \{Z_{ij}\}) \tag{3}$$

The moment, F_{r_u}, around the maximum foraging efficiency for a specific environment item type ratio, r_u, is defined as follows:

$$F_{r_u} = \sum_{v=1}^{n_r} \frac{|Z_{I_u u} - Z_{I_u v}|}{Z_{I_u u}} \tag{4}$$

The final performance measure is the normalized sum of the above moments around the maximum foraging efficiency for every environment item type ratio in $\overline{r}$:

$$F_r = \frac{1}{n_r}\sum_{u=1}^{n_r} F_{r_u} \tag{5}$$

The performance measure, F_r, measures how well swarm parameters, which are optimized to yield the best efficiency for environments with a specific prioritized item ratio generalize across environments with different prioritized item ratios. The less the foraging efficiency varies when an algorithm that is configured with swarm parameters optimized for a specific environment ratio, is run on all other environment ratios, the smaller F_r will be. Therefore, the smaller F_r is, the more flexible an algorithm is considered to be. F_r is a macro performance indicator which is run on the results of all experiments.

This performance measure evaluates flexibility by analysing how well swarm parameters, which are optimized to yield the best efficiency for environments with a specific environment distribution generalize across environments with different environment distributions. The performance measure, F_ϵ, is defined similar to F_r, except over environment distributions, rather than environmental item ratios. A list Y of size n_ϵ is defined as follows:

$$Y_j = (k | D_{kj} = \max_{i=1,\dots,n_\tau}\{D_{ij}\}) \tag{6}$$

The performance measure, F_ϵ, is defined as follows:

$$F_\epsilon = \frac{1}{n_\epsilon}\sum_{u=1}^{n_\epsilon}\sum_{v=1}^{n_\epsilon} \frac{|D_{Y_u u} - D_{Y_u v}|}{D_{Y_u u}} \tag{7}$$

F_ϵ is interpreted in a similar manner to F_r, where the smaller the value for F_ϵ is, the more flexible an algorithm is considered to be over environment distributions.

4.7.2. Scalability

Scalability is described in Section **??**. The scalability performance measures used in this thesis are based on the measures used in [10]. This study analyses the scalability of each of the algorithms with respect to two performance measures, namely, swarm density scalability and problem complexity scalability. Swarm density scalability and problem complexity scalability are defined in the following sections. Both swarm density scalability and problem complexity scalability are evaluated on the largest environment size (i.e. $\Lambda = 500$) in order to prevent the possibility that an environment runs out of items to forage.

To simplify the explanation of the scalability performance measures, the following notation is defined:

Let $\overline{c}$ be the list of all considered values for swarm density, c, with length n_c. Let c_{min} be the minimum value considered for c and c_i be the ith value of $\overline{c}$. Let E_{c_i} be the average foraging efficiency, E_P, over all experiments, with swarm density c_i.

Similarly, define the list of all considered values for problem complexity, p, as $\overline{p}$ with length n_p. The smallest problem complexity is p_{min} and E_{p_i} is the average foraging efficiency over all experiments, with problem complexity p_i.

The efficiency of a robot swarm should be relatively undisturbed by changes in group sizes. This property is known as swarm density scalability. Ideally, foraging efficiency should increase linearly as swarm density increases, which is known as linear scalability. However, due to increased inter-robot interference in larger swarm densities, scalability is often sublinear (i.e. logarithmic) [28]. Many swarm robotics algorithms focus on the use of division of labour, navigation or communication in order to decrease inter-robot interference [28, 29].

In order to analyse swarm scalability, one could simply examine the average efficiency of each algorithm, over all experiments, at each swarm density. However, to ensure that each algorithm is compared on scalability alone, the individual efficiencies of each algorithm must be eliminated. To eliminate the individual efficiencies of each algorithm, the average efficiency, E_{c_i}, for each swarm density, c_i, is normalized by the average efficiency, , $E_{c_{min}}$, for the smallest sized swarm for that algorithm, as follows:

$$S_{c_i} = \frac{E_{c_i} - E_{c_{min}}}{E_{c_{min}}} \tag{8}$$

To analyse how each algorithm scales, the normalized average foraging efficiency, S_{c_i}, for each value in $\overline{c}$ can be plotted for each algorithm to determine whether an algorithm scales linearly, sublinearly, or superlinearly.

The total swarm scalability, S_c, provides a single value per algorithm, which can be used to compare overall swarm scalability:

$$S_c = \sum_{i=1}^{n_c} S_{c_i} \tag{9}$$

The larger the value of S_c, the better foraging efficiency scales with respect to swarm density.

Ideally, foraging efficiency should be indirectly proportional to problem complexity. As the problem complexity increases, the foraging efficiency should decrease linearly. An algorithm is scalable when performance decreases linearly or sublinearly as efficiency increases. Similar to swarm density scalability (described in Section **??**) problem complexity scalability is often sub-linear, due to increased environmental interference.

In order to evaluate the problem scalability of each of the algorithms presented, the efficiency of each algorithm is examined over a variety of environment item densities, p, defined in Section 4.3. Similar to swarm density scalability performance measures S_{c_i} and S_c, the problem complexity scalability measures, S_{p_i} and S_p, are defined as follows:

$$S_{p_i} = \frac{E_{p_i} - E_{p_{min}}}{E_{p_{min}}} \tag{10}$$

$$S_p = \sum_{i=1}^{n_p} S_{p_i} \tag{11}$$

4.7.3. Behavioural Performance Measures

Despite not being one of the more typical performance measures for swarm robotics, a number of measures are included to enable further understanding of the obtained quantitive performance measures.

4.7.4. Items foraged over time

To give insight into the effects of environmental and swarm parameters on overall performance, and to characterise the effects of certain behaviours, the following metrics have been recorded:

- The total number of prioritized items foraged over time, E^t_P. The total number of prioritized items foraged are recorded every 200 time steps and are normalized with the total number of prioritized items that exists in the specific environment.

- The total number of non-prioritized items foraged over time, E^t_{NP}. The total number of non-prioritized items foraged are recorded every 200 time steps and are normalized with the total number of non-prioritized items that exist in the specific environment.

4.7.5. Swarm specialization ratio over time

Swarm specialization ratio stays constant for the desert ant algorithm and the naïve algorithm, but the swarm specialization ratio changes for the honey bee algorithm, due to the division of labour mechanisms. In order to gain a better understanding of how the division of labour mechanism performs, the swarm specialization ratio over time, $\tau(t)$, is tracked. The swarm specialization ratio is recorded every 200 time steps.

4.7.6. Time spent performing recruitment

Unlike the desert ant algorithm and the naïve algorithm, the honey bee algorithm robots perform behaviours that are not directly involved in retrieving items - in particular, the recruitment activities form part of the division of labour mechanism. In order to better understand the influence of the division of labour mechanisms, the average percentage of the time that each robot spents on each division of labour activity is recorded as follows:

- t_{wait}, the average percentage of the total number of time steps spent by each robot in the waiting state.

- $t_{recruitment}$, the average percentage of the total time number of steps spent by each robot in the recruitment state.

5. Results

The analysis of the results of the experiments is organised with respect to four major characteristics of swarm robotics algorithms, namely efficiency, flexibility, robustness, and scalability. Section 5.1 addresses foraging, while the flexibility of the algorithms over different environment distributions and item type ratios is analysed in Section 5.2. Section 5.3 evaluates the scalability of each algorithm, and robustness is analysed in Section 5.4. Section **??** summarizes the findings of the analysis.

5.1. Foraging efficiency

Despite the fact that determining the most efficient algorithm is not a main focus of this study, this section does a brief comparison of the foraging efficiency, E_P (defined in Section 4.7), of each algorithm.

This is done by comparing the performance of the three algorithms over all environments and all swarm parameters giving a total at 121500 unique experiment configurations each run for 30 independant samples. A Wilcoxin test, at a confidence level of 0.05, was applied over the 30 independant samples, for each pair of algorithms, for each experiment, to determine if a statistically significant difference in foraging efficiency of the pair of algorithms exists. If no statistically significant difference is indicated between the two algorithm experiment samples, then nothing is done. If a statistically significant difference is indicated between the two algorithm experiment samples, then two one-tailed Mann-Whitney U tests were performed at a significance level of 0.05 to determine which algorithm had a greater foraging efficiency on a particular experiment. Suppose A and B are two of the selected algorithms. If the first one-tailed Mann-Whitney U test indicates that A is significantly more efficient than B, then a "win" is counted for algorithm A. If the second one-tailed Mann-Whitney U test indicates that the E_P for A is significantly less efficient than B, then a "loss" is counted for algorithm A.

To determine an algorithm's foraging efficiency in comparison to each other algorithm, the wins and losses for each algorithm are summed per experiment. This statistical analysis approach has been used in [30]. The percentage of wins per algorithm and the percentage of losses per algorithm is calculated, because percentages are more informative than counts. The percentage of wins per algorithm and the percentage of losses per algorithm are summarized in Table 1. The desert ant foraging performed better than the naïve algorithm. The honey bee algorithm out-performed both the naïve algorithm and the desert ant algorithm. The following sections analyse the swarm robot properties of each algorithm, and explain why the foraging efficiency of the algorithms is as shown in Table 1.

5.2. Flexibility

This section analyses the flexibility of each of the considered algorthms in terms of the prioritized item ratio and the environment distribution types, over all experiments. Section 5.2.1 analyses the flexibility of each algorithm in terms of the prioritized item ratio of the environment. Section **??** discusses the flexibility of each algorithm in terms of environment distribution types.

Table 1: Pairwise one-tailed Mann Whitney U wins and loss counts, for the foraging efficiency, E_P, for each algorithm, over all environments and swarm parameter values

Algorithms	**Wins**	**Losses**
Naïve	1.3%	66.5%
Desert ant	47.4%	16.4%
Honey bee	51.2%	17.1%

5.2.1. Flexibility in terms of prioritized item ratio

The performance measure, F_r (defined in Section **??**), measures how well the swarm parameters, which yield the best efficiency for environments with a specific prioritized item ratio, generalize across environments with different prioritized item ratios. Therefore, the smaller F_r is, the more flexible an algorithm is considered to be. F is a macro performance indicator which is run on the results of all experiments.

Table 2 summarizes the flexibility of each foraging algorithm in terms of environment type ratio, F_r. According to Table 2, the honey bee algorithm is the most flexible in terms of F_r, followed by the naïve algorithm, with the desert ant algorithm exhibiting the worst flexibility.

Table 2: Flexibility in terms of prioritized item ratio, F_r, and flexibility in terms of environment distribution, F_ϵ, for each algorithm

	Naïve	Desert ant	Honey bee
F_r	1.186	3.124	0.828
F_ϵ	1.112	0.507	0.458

The honey bee algorithm is the most flexible in terms of prioritized item ratio. To understand why this is the case, the mechanisms used specifically by the honey bee algorithm have to be examined. In particular, the honey bee algorithm employs a division of labour mechanism that allows robots to switch from foraging prioritized items to non-prioritized items (and vice versa). The honey bee algorithm's division of labour mechanism suggests the following hypothesis: The honey bee algorithm's superior flexibility is due to the fact that the robots can switch their item type specialization. In that way, the honey bee algorithm was able to adapt the swarm specialization ratio to more efficiently forage a given environment item type ratio.

To provide evidence for the above hypothesis the average swarm specialization ratio over time, $\tau(t)$, for the honey bee algorithm on a uniformly distributed environment with a high ratio of prioritized items (i.e. $r = 0.75$) is examined. The initial swarm specialization ratio, $\tau(0)$, was set to 0, which means that all robots are set to forage non-prioritized items. The swarm specialization ratio over time, $\tau(t)$, was averaged over 30 independant runs. The purpose behind examining the described scenario is to determine if the honey bee algorithm is able to adapt the swarm's specialization ratio (which initially can only forage non-prioritized items), to more efficiently forage the high ratio of prioritized items. The above scenario is illustrated in Figure 8.

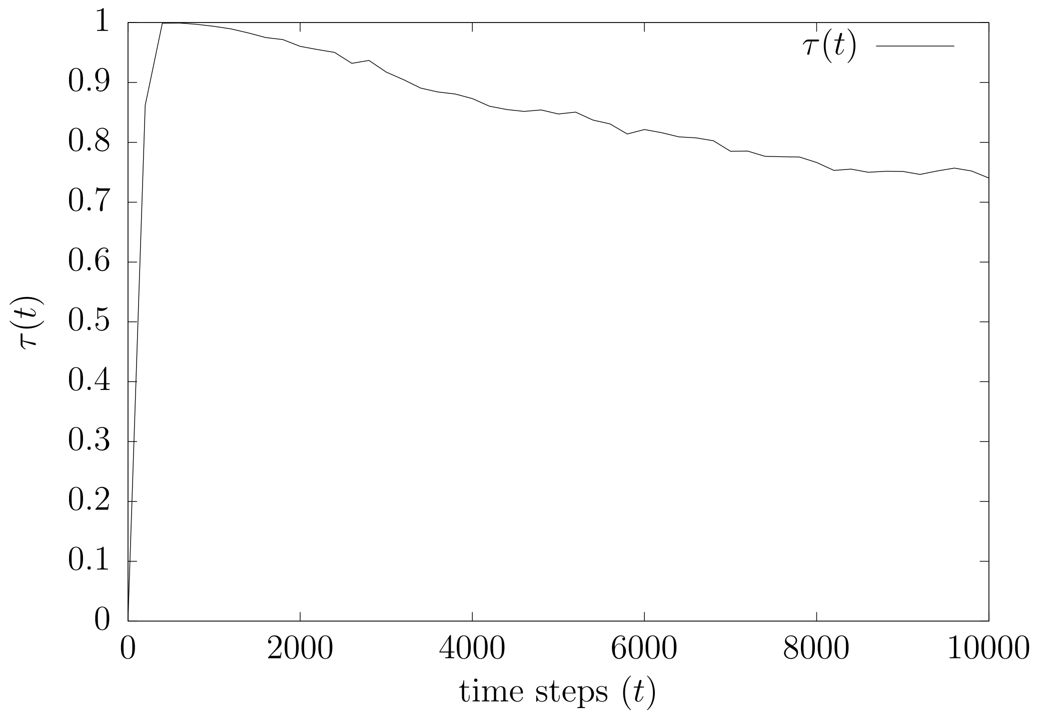


Figure 8: Specialization ratio over time, $\tau(t)$, of the honey bee algorithm, on a uniform environment, with environment item type ratio, $r = 0.75$, and initial swarm specialization ratio, $\tau(0) = 0$

Figure 8 shows that τ increased from 0 to 1 in 200 iterations, which suggests that the honey bee scout robots switched their specializations to search for prioritized items when the honey bee scout robots could not find any non-prioritized items. After 200 iterations, the specialization ratio steadily declined from 1 to 0.73. The final swarm specialization ratio, $\tau(10000) = 0.73$, is nearly the same as the environment ratio, $r = 0.75$, suggesting that the honey bee algorithm eventually adapted the specialization ratio to effectively forage the environment item type ratio of the given environment.

The above analysis of the behaviour of the honey bee algorithm supports the hypothesis that the honey bee algorithm's superior flexibility can be attributed to the algorithm's ability to adapt the swarm's specialization ratio (via the division of labour mechanism described above) to more efficiently forage an environment of a given environment item ratio.

Both the naïve algorithm and desert ant algorithm do not have the ability to adapt the initial swarm specialization to forage any environment ratio, which means that both algorithms are less flexible than the honey bee algorithm.

The desert ant algorithm is the least flexible, which indicates that the algorithm is sensitive to the choice in $\tau(0)$ for each environment type ratio r. Unlike the naïve algorithm, which has the worst efficiency over all experiments, a good choice of τ will allow the algorithm to forage the environment significantly better for a specific environment item ratio. Despite the desert ant algorithm outperforming the naïve algorithm in efficiency (refer to Section 5.1), the naïve algorithm is substantially more flexible than the desert ant algorithm. F_r reveals that the naïve algorithm's performance is invariant under the variation of $\tau(0)$ on different environment item type ratios, resulting in a high level of flexibility. However, the naïve algorithm's flexibility is practically irrelevant given how poorly the algorithm performs on all environments.

5.2.2. Flexibility in terms of environment distributions

Similar to F_r, the performance measure, F_ϵ (defined in Section **??**), measures how well the swarm parameters yielding the best efficiency for environments with a specific environment distribution generalize across environments with different environment distributions, for a specific algorithm. Similar to F_r, the smaller F_ϵ is, the more flexible an algorithm is considered to be, and F_ϵ is also calculated based on the results of all experiments. The results summarized in Table 2 indicate that the honey bee algorithm was the most flexible in terms of environment distribution, followed by the desert ant algorithm. The naïve algorithm was the least flexible.

The honey bee algorithm's superior flexibility suggests the following hypothesis: The honey bee algorithm's ability to adapt τ (described in Section 3.3) enables the swarm to more efficiently forage the differing environment ratios of the accessible environment (refer to Section 4.3.3) at any point during the foraging process.

To provide evidence for this hypothesis, an in-depth examination of the honey bee algorithm is performed on a Gaussian environment with a high density of items. The environment has a low ratio of prioritized items and a swarm which has a high initial swarm specialization ratio (i.e. most robots are initially set to forage prioritized items). Since the non-prioritized items surround the prioritized items, the swarm should first forage the non-prioritized items in order to get access to the prioritized items. Once the swarm can reach the prioritized items, the swarm should switch to foraging the prioritized items. The example Gaussian environment is illustrated in Figure 9. The white cells represent an empty cell, the dark cells represent the cells containing a non-prioritized item and the shaded cells represent the cells containing a prioritized item.

The environment in Figure 9 has the following properties:

- A Gaussian environment item distribution,
- a low environmental item ratio of $r = 0.2$,
- a high density of items with $p = 90$, and
- an environment size, $\Lambda = 100$.

Also, consider a swarm with the initial specialization ratio set to mostly forage prioritized items (i.e. $\tau(0) = 0.8$) and with swarm density set to 0.5. The items nearest to the sink are all non-prioritized items. Robots will have to forage a large portion of the non-prioritized items between the sink and the centre of the environment in order to reach the high density of prioritized items at the centre of the environment.

Figure 10 shows the efficiency of foraging prioritized items, E_P, the efficiency of foraging non-prioritized items, E_{NP}, and specialization ratio over time, $\tau(t)$, for the honey bee algorithm on the environment described above. The performance measures were averaged over 30 independant runs.

In the first 200 time steps, the efficiency of foraging nonprioritized items, E_{NP}, increased very slowly. This was because there were very few robots that could forage non-prioritized items because most of the robots were initialized to forage prioritized items. The efficiency of foraging prioritized items, E_P,

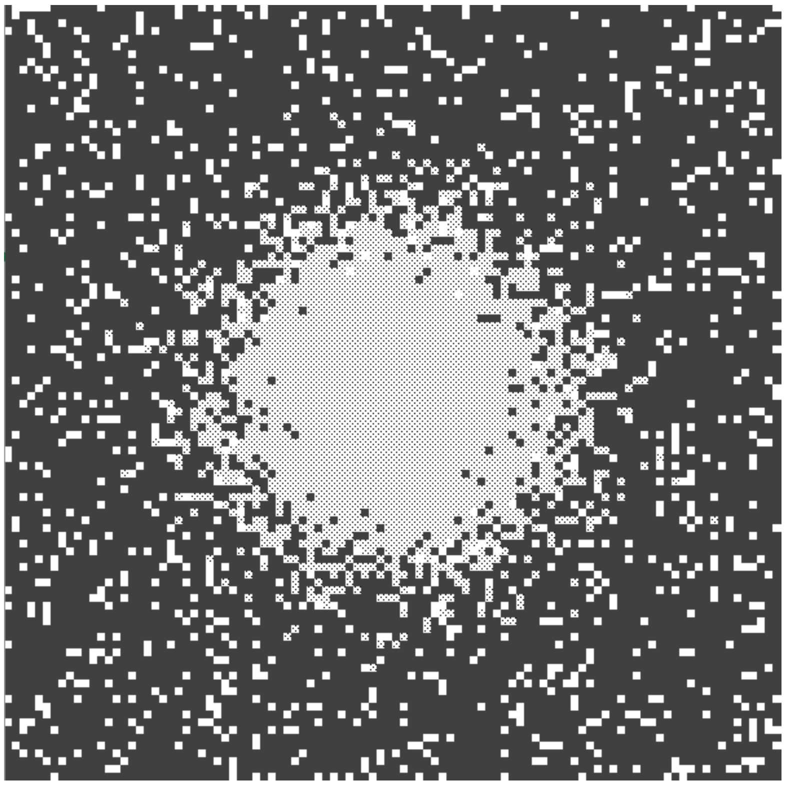


Figure 9: Gaussian environment with environment item type ratio, $r = 0.2$, environment item density of $p = 0.9$, and environment size $\Lambda = 100$

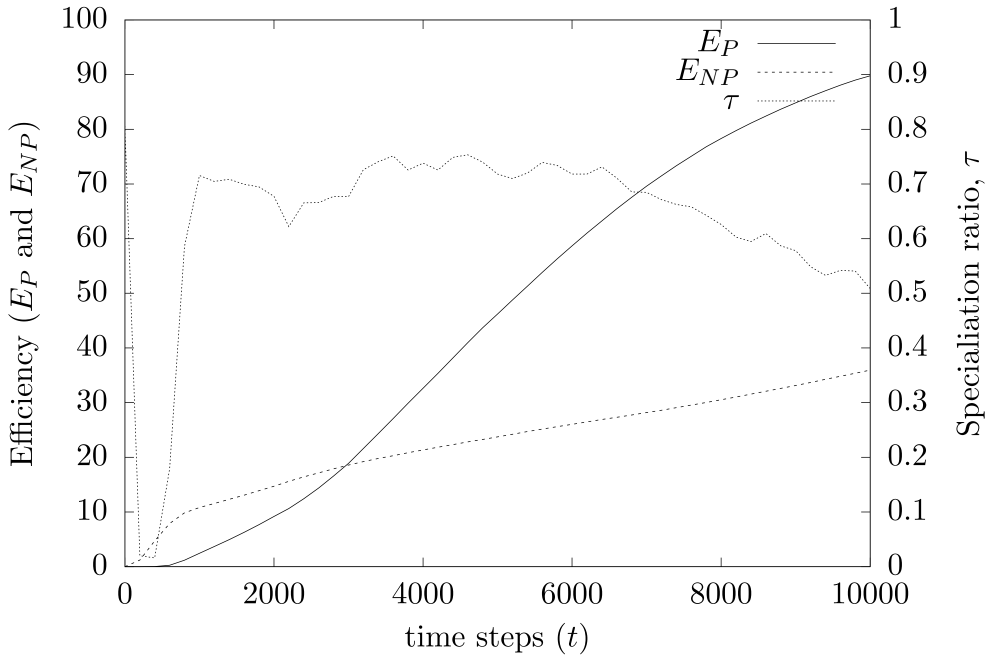


Figure 10: Efficiency of prioritized item foraging, E_P, efficiency of non-prioritized item foraging, E_{NP}, and specialization ratio over time, $\tau(t)$, for the honey bee algorithm on an example Gaussian environment, averaged over 30 independant runs

did not increase in the first 200 time steps, because the robots had no access to the prioritized items as they were blocked by the non-prioritized items. The initial specialization ratio, $\tau(0)$, that was initialized at 0.8, decreased very quickly to close to 0 in the first 200 time steps. The decrease in specialization ratio is good, because then most robots switched to forage non-prioritized items in the accessible environment which consisted entirely of non-prioritized items. Between 200 and 800 time steps, the E_{NP} increased very quickly, due to the decrease in the swarm specialization ratio. The E_P remained low between 200 and 800 time steps, because the prioritized items were not yet accessible to the swarm. After 800 time steps, τ increased from near 0 to just above 0.7, suggesting that the non-prioritized items have been cleared and the concentration of prioritized items in the centre have been reached. Thus, most of the robots in the swarm switched from foraging non-prioritized items to foraging prioritized items. The increase in swarm specialization ratio after

800 time steps was followed by a sharp increase in E_P. This demonstrated that the honey bee algorithm could adapt τ in order to initially forage the large number of non-prioritized items, and then switched to foraging mostly prioritized items when the prioritized items became accessible.

In constrast, the naïve algorithm's performance suffered because it lacks the ability to adapt swarm specialization. Figure 11 illustrates the same performance measures, on the same Gaussian environment and swarm parameters for the naïve algorithm.

Because the naïve algorithm does not enable robots to switch their item specialization, the swarm specialization ratio, τ, remains constant. Because so few robots could clear non-prioritized items, due to the large initial swarm specialization ratio, E_{NP} increased slowly throughout the experiment. Because the swarm was slow to clear the non-prioritized items, E_P also remained small throughout the experiment, since the swarm struggled to make the prioritized items easily accessible. This suggests that prioritized foraging robots experienced environmental interference while attempting to navigate around the unforaged non-prioritized items. The rate of increase in E_P increased over the duration of the experiment, as the swarm cleared more of the non-prioritized items. The behaviour for the desert ant algorithm was similar to that of the naïve algorithm, except that the final efficiency was greater for the desert ant algorithm.

Figure 10 shows that the final E_P for the honey bee algorithm, averaged over 30 independant runs, was 0.9, while Figure 11 shows that the final E_P for the naïve algorithm was only 0.3. Since the honey bee algorithm adapts τ, the honey bee algorithm shows an increased ability to forage obstacles (the non-prioritized items) in order to more easily access hard to reach prioritized items. As a result, the honey bee algorithm is more flexible to different environment distributions than the naïve or desert ant algorithms.

To provide further evidence for the above hypothesis, consider other environment distributions to show that the honey bee algorithm can adapt the swarm's swarm specialization ratio appropriately for different distributions. Thus, consider a uniform environment with the same environment parameters as above. Because the environment distribution is uniform, the environment ratio of the accessible environment will fluctuate erratically as the swarm forages the environment.

Figure 12 shows the same performance measures for an experiment with the same swarm parameters as above on the uniform environment for the honey bee algorithm. The specialization ratio has a downward erratic trend,

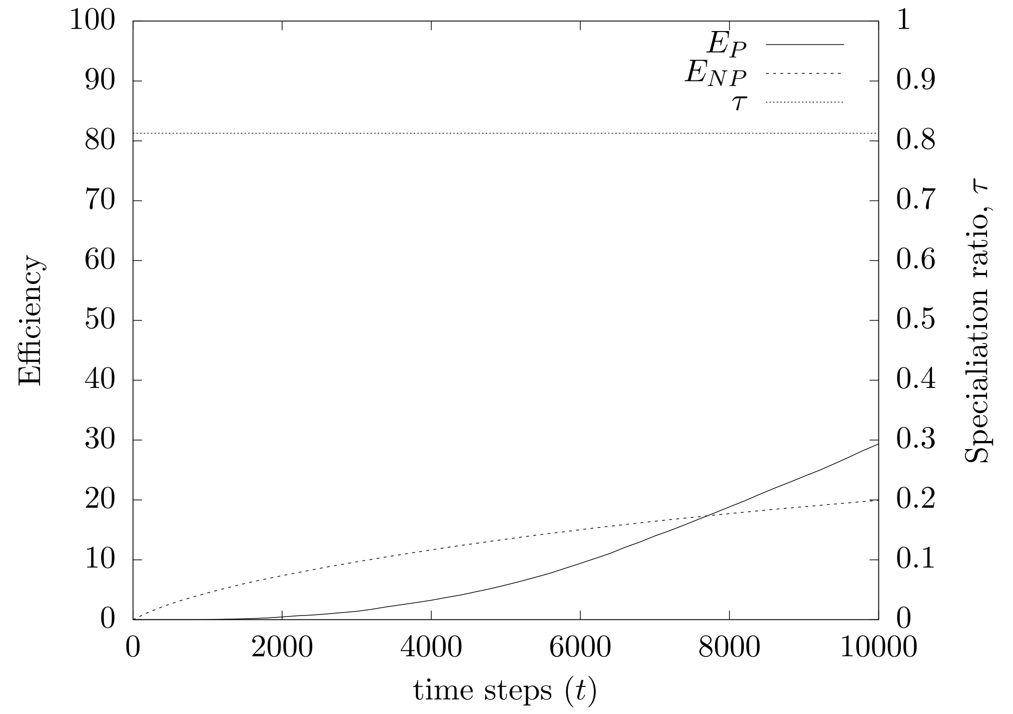


Figure 11: Efficiency of prioritized item foraging, E_P, efficiency of non-prioritized item foraging, E_{NP}, and the specialization ratio over time, $\tau(t)$, for the naïve algorithm on a Gaussian evironment, averaged over 30 independant runs

from over 0.9 to approximately 0.4. The erratic fluctuation of τ is due to minor and short-lived erratic differences in the environment ratio of the accessible environment which would feature in uniform environments. The rates of foraging efficiency, E_P and E_{NP}, increase constantly throughout the experiment.

The above analysis provides evidence for the hypothesis that suggests that the honey bee algorithm is capable of adapting the swarm to forage the accessible environment. Thus the honey bee algorithm has higher flexibility on different environment distributions compared to the desert ant and naïve algorithms. The desert ant and naïve algorithms have more difficulty foraging certain distributions as they cannot adapt τ to more effectively forage the accessible environment.

Table 2 indicates that the desert ant algorithm is more flexible over environment distributions than the the naïve algorithm. Considering the fact

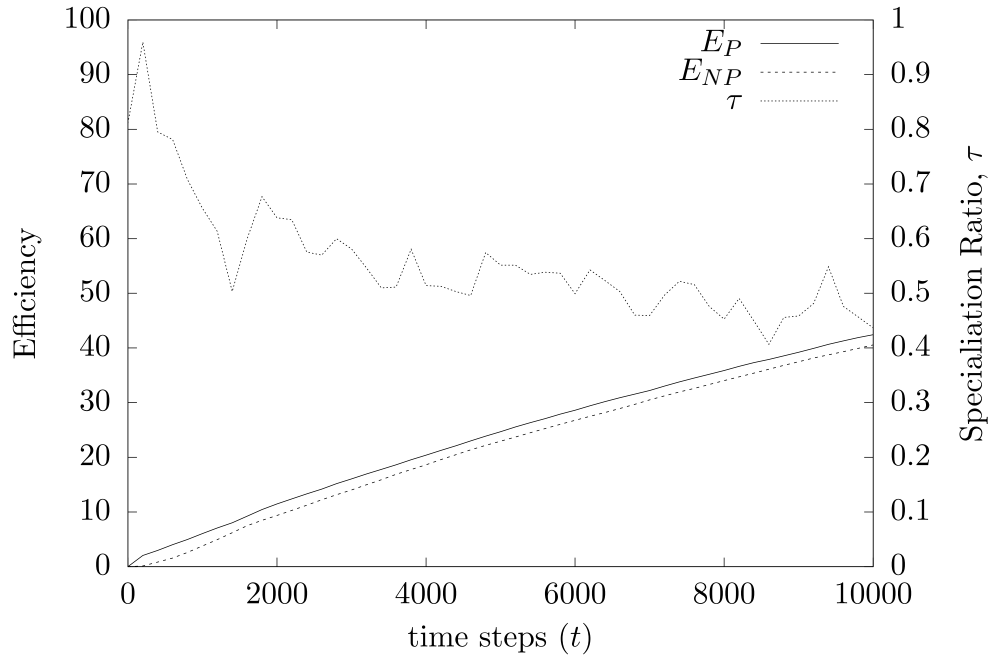


Figure 12: Efficiency of prioritized item foraging, E_P, over time t, efficiency of non-prioritized item foraging, E_{NP}, over time t, and specialization ratio, $\tau(t)$, for the honey bee algorithm on a uniform environment, averaged over 30 independant runs

that the only difference between the naïve algorithm and the desert ant algorithm is the fact that the desert ant algorithm uses site fidelity while the naïve algorithm does not, the difference in performance must be attributed to the desert ant's use of site fidelity.

5.3. Scalability

This section discusses the scalability of each algorithm in terms of the macro performance measures: swarm density and problem complexity. Section 5.3.1 discusses the scalability of each algorithm in terms of swarm density, while Section 5.3.2 evaluates the scalability of each algorithm in terms of the complexity of the problem.

5.3.1. Swarm scalability

Table 3 presents the swarm density scalability results, S_c. Figure 13 shows the swarm density scalability for each density, S_{c_i}, for each algorithm (as defined Section **??**). The naïve algorithm was the most scalable while the desert ant and the honey bee algorithms show similar scalability. The honey bee algorithm is more scalable than the desert ant algorithm for $c < 0.9$, but at $c \geq 0.9$, the swarm scalability, S_{c_i}, of the desert ant algorithm was greater than the honey bee algorithm.

Table 3: Swarm scalability, S_{c_i}, for each swarm density in $\bar{c}$, for each algorithm.

c	**0.1**	**0.3**	**0.5**	**0.7**	**1**
naïve	1	2.148	2.951	3.568	4.255
desert ant	1	1.973	2.605	3.058	3.556
honey bee	1	2.067	2.706	3.119	3.532

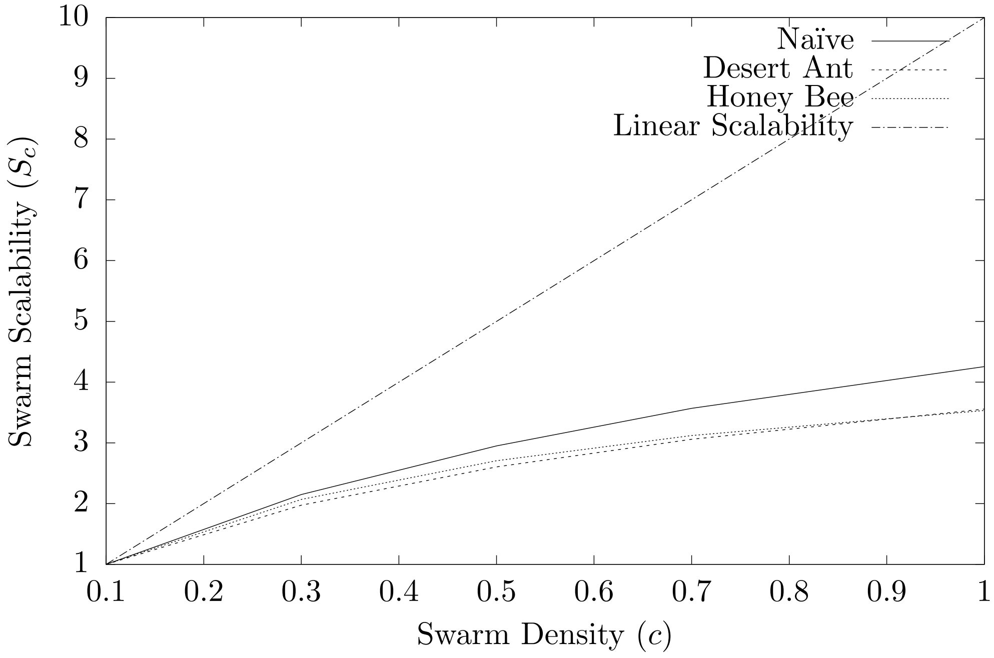


Figure 13: Swarm scalability, S_c, for each swarm density c_i in $\bar{c}$ for each algorithm.

Figure 13 also plots the ideal scalability. Ideal scalability is when the increase in swarm density is directly proportional to the increase in efficiency.

The ideal values for S_{c_i} were calculated and plotted. All algorithms fall below the ideal scalability line and thus have sublinear (logarithmic) scalability. Section **??** highlighted that increased inter-robot interference is a major factor that impacts on swarm scalability. It can be concluded that the reason for sub-linear performance in all algorithms was because all algorithms suffer from increased inter-robot interference as swarm size increases.

To understand why the desert ant and the honey bee algorithms were less scalable than the naïve algorithm, consider the following: The naïve algorithm, being the most simple foraging algorithm, has no mechanism to enable the swarm to exploit high quality sites of prioritized items and can only locate items by randomly exploring the environment. Both the desert ant algorithm and the honey bee algorithm have mechanisms to exploit high quality areas: The desert ant algorithm uses site fidelity and the honey bee algorithm uses recruitment to exploit high quality sites.

Suppose that there exists a single high quality site in an environment. Using site-fidelity and recruitment, the desert ant and the honey bee algorithms can exploit that high quality area. In exploiting that high quality site, the path between the sink and the high-quality site becomes more congested as more robots share the path between the sink and the high quality area. The increased congestion on that path will cause more inter-robot interference. However, due to the focused efforts of the swarm on a high quality area of the environment, the exploitation will still improve foraging efficiency overall, compared to the naïve algorithm at low swarm densities. However, as swarm density increases, the congestion on the route between the high quality site and the sink will increase, hindering foraging efficiency due to increased inter-robot interference. It follows that the desert ant algorithm and the honey bee algorithm are less scalable in terms of swarm density than the naïve algorithm, due to the increased inter-robot interference as a result of exploitation of high quality sites.

The recruitment mechanism used by the honey bee algorithm is a more extreme form of exploitation than site fidelity used by the desert ant algorithm, because scouts recruit groups of robots to forage single areas. On the other hand, when using site fidelity, the high quality site is only foraged by the robot that found it. Considering that the recruitment mechanism is more exploitative than the site fidelity mechanism, one would expect the honey bee algorithm to be much less scalable than the desert ant algorithm, but that is not demonstrated in Table 3. Instead, the honey bee algorithm performed slightly better than the desert ant algorithm for all $c < 0.8$.

To explain the honey bee algorithm's ability to outperform the desert ant algorithm at most swarm densities, consider the honey bee algorithm's division of labour mechanism as described in Section 3.3. The division of labour mechanism will cause an active forager robot to return to the sink if that robot can not find an item for a maximum amount of time. The inactive forager robot waits at the sink, until the inactive forager robot is recruited by a scout robot. Section **??** discussed that a mechanism of division of labour which could adjust the number of robots actively foraging to an appropriate number, would result in decreased levels of inter-robot interference and increased swarm scalability.

In order to determine whether the slight improvement in scalability of the honey bee algorithm over the desert ant algorithm can be attributed to the discussed division of labour mechanism, the average time spent by robots in the waiting state is plotted for different values of c in Figure 14. Figure 14 illustrates that the robots spend a significant portion of time in the waiting state for $c < 0.5$, which provides evidence that the honey bee algorithm's swarm scalability is greater than the desert ant algorithm's swarm scalability for low values of c.

However, the time spent by honey bee robots in the waiting state decreases as swarm density increases. This result is counter-intuitive, because an appropriately functioning division of labour mechanism would lead to an increase in average time in the waiting state as swarm density increases, due to increases in inter-robot interference. Further analysis is required in order to understand why the division of labour mechanism of the honey bee algorithm did not behave as expected. That said, the problem in the honey bee algorithm's division of labour mechanism does explain why the desert ant algorithm overtakes the honey bee algorithm for higher values of c: The honey bee algorithm is unable to regulate the number of active foragers for high values of c. The honey bee algorithm's recruitment mechanism is more exploitative than the site fidelity mechanism of the desert ant algorithm, which results in increased inter-robot interference when the honey bee algorithm fails to regulate the number of active foragers. The increased inter-robot interference for the honey bee algorithm for high values of c results in a lower scalability than that of the desert ant algorithm.

5.3.2. Problem scalability

Figure 15 plots the problem scalability performance measure, S_p (described in Section **??**), for each algorithm using the values given in Table 4.

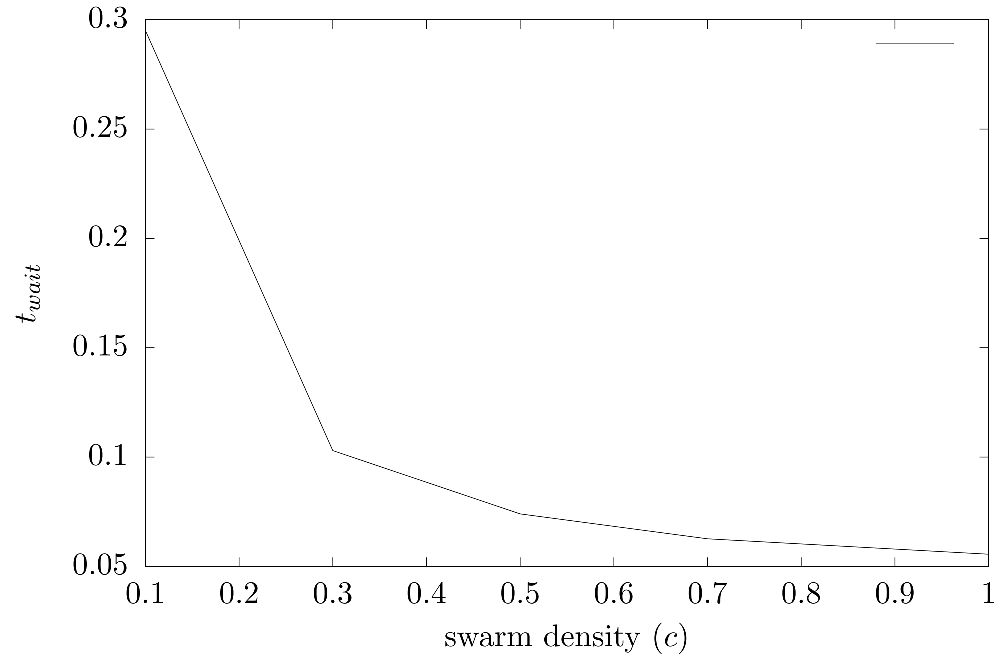


Figure 14: Average time in waiting state, t_{wait}, for each swarm density c, for the honey bee algorithm

The results present the macro performance measure, S_p, which is calculated over all experiments, for each value of p. The figure includes a plot of "Expected scalability". The values for "Expected scalability" were calculated based on the assumption that foraging efficiency would degrade directly proportional to the increase in problem complexity. The expected values for foraging efficiency at each problem complexity were used to calculate S_p for "Expected scalability". All algorithms outperformed the expected scalability and thus problem scalability for all algorithms is better than the expected scalability which is good.

According to Figure 15, the honey bee algorithm is the most scalable in terms of problem scalability, followed by the naïve algorithm, with the desert ant algorithm exhibiting the worst scalability.

In order to explain the reason for this, consider the following rational argument: The desert ant algorithm differs from the naïve algorithm in only one aspect: The desert ant algorithm employs site fidelity and the naïve

Table 4: Problem scalability, S_{p_i}, for each environment density, p_i, for each algorithm

p	**0.05**	**0.2**	**0.5**	**0.7**	**0.9**
desert ant	1	0.488	0.231	0.182	0.169
honey bee	1	0.602	0.283	0.217	0.196
naïve	1	0.502	0.257	0.195	0.175
Expected	1	0.25	0.1	0.071	0.056

algorithm does not. The site fidelity (discussed in Section 3.2) enables a robot to return to a previously foraged site. The desert ant algorithm's use of site fidelity must, directly or indirectly, be the reason that the naïve algorithm is more scalable than the desert ant algorithm. To understand why the site fidelity mechanism is negatively influencing scalability, consider the following scenario: A desert ant robot employs a random walk through a complex environment, as shown by the dashed line in Figure 16. The random walk leads the desert ant robot to a hard-to-reach source of prioritized items. The desert ant robot stores the PI vector, shown by the solid line on the figure, which is needed to return back to the site after offloading the loaded item at the sink. The next time the desert ant wants to return to the previous site, the robot will still need to perform the same obstacle avoidance in order to navigate around the obstacles to return to the previously foraged site, since the PI vector only consists of an overall heading and distance to the site. As is evident in Figure 16, there may exist an easier to reach site for prioritized items, but the desert ant robot will waste time by continuing to forage the hard to locate source of items. The naïve algorithm will not try to return to the hard to find site and is more likely to randomly find the prioritized resource site that is nearer to the sink (indicated by the dotted line), and thus the time taken to forage a single item will be faster, and so foraging efficiency will be increased. The time spent on relocating hard to access sites in more complex environments can slow the desert ant algorithm down. The naïve algorithm benefits from a more complex environment, because there will be lots of items to find nearer to the sink that do not require any complex search to relocate.

Additionally, in environments that are more dense, all desert ant robots are more likely to locate the same high density area. The more dense the area, the more robots will try to forage the same area and the more inter-robot and environmental interference will occur. The desert ant robots try to

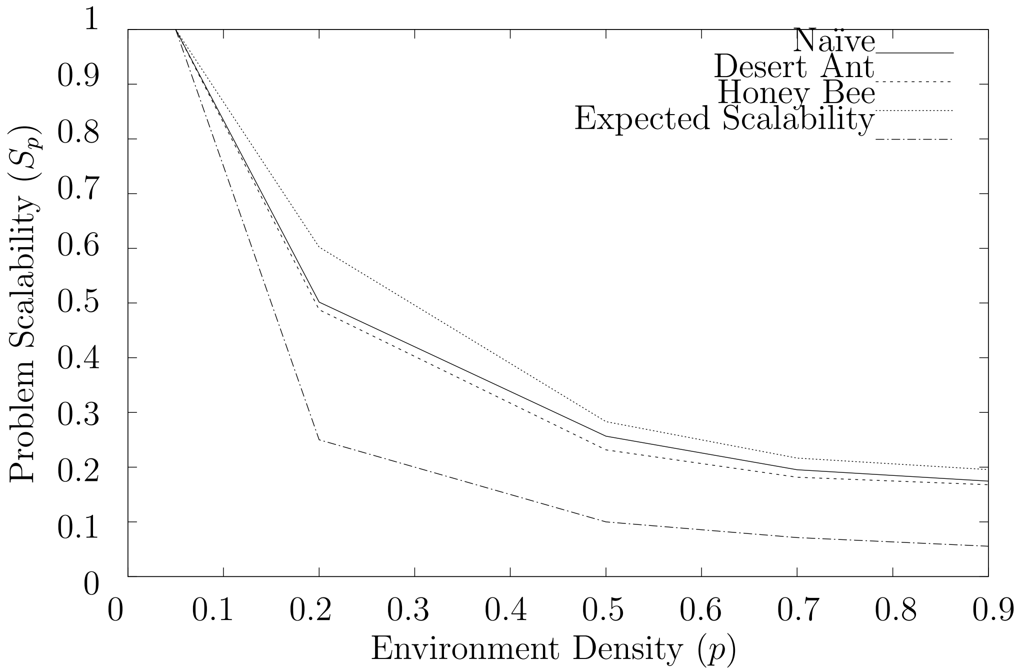


Figure 15: Problem Scalability, S_p, for each environment density p_i, for each algorithm

exploit the same high density areas. On the other hand, the naïve algorithm will explore more. Thus the naïve algorithm will experience less inter-robot and environmental interference as a result.

The honey bee algorithm uses site fidelity, and also communicates those sites to others. Despite the use of a site fidelity mechanism, similar to the desert ant algorithm, the honey bee algorithm is the most scalable. The reason why the site fidelity of the honey bee algorithm does not impact on the scalability can be attributed the ability of the honey bee algorithm to adapt the swarm specialization ratio, τ, to focus on foraging dense areas of obstacles, rather than having to expensively navigate around the obstacles. The ability of the honey bee algorithm to concentrate on clearing obstacles in highly dense environments will increase overall ease of access to the high quality sites, as discovered in Section **??**. Thus efficiency of the honey bee algorithm in highly dense environments is improved. Therefore, the honey bee algorithm is the most scalable in terms of the problem complexity.

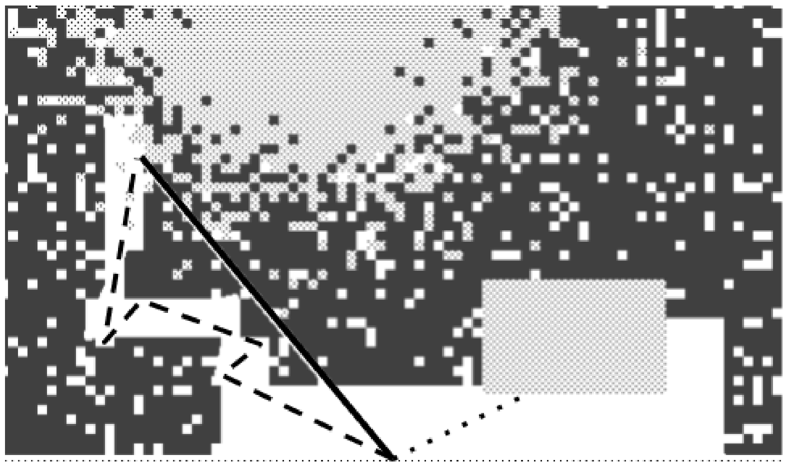


Figure 16: Illustration of the inefficiencies of the desert ant algorithm's site fidelity in dense environments. Solid areas are non-prioritized items, white areas are free cells, and shaded areas are prioritized items. The dashed path is a hypothetical random walk performed by a robot, the solid path is the PI vector for the dashed random walk, and the dotted path is an alternative random walk.

5.4. Robustness

As discussed in Section **??**, robot swarms achieve robustness by exhibiting the properties of redundancy, multiplicity of sensing, and decentralized coordination. This study does not perform an empirical robustness study to specifically address how fault tolerant each algorithm is. Instead, this section provides rational arguments supported by empirical evidence to discuss each algorithm's robustness. Section 5.4.1 discusses robustness in terms of the resulting redundancy of each of the algorithms, and Section 5.4.2 discusses robustness in terms of decentralized coordination.

5.4.1. Redundancy

As discussed in Section **??**, redundancy in a swarm is achieved by giving all, or a portion of the robots in the swarm the same capabilities. In this way, if a percentage of the swarm malfunctions, the other robots have the capability to take the malfunctioning robots' place. A swarm demonstrates the highest level of redundancy when any robot can take on the role of any other robot that malfunctions in a swarm.

For each of our experiments, each swarm is configured with an initial swarm specialization ratio (as discussed in Chapter **??**) to enable a portion of robots to forage prioritized items and the another portion to forage non-prioritized items.

If a portion of robots in a swarm malfunctions or are destroyed, then the swarm specialization ratio will be affected. As shown in Section 5.2, the foraging efficiency of the naïve and desert ant algorithms were effected by the initial swarm specialization ratio τ. It follows that, if the swarm specialization ratio is unexpectedly changed during the swarm experiment, a change in the foraging efficiency of the swarm would persist for the experiment. In particular, consider the following extreme scenarios: Suppose that all the robots which were foraging for prioritized items (for some $\tau > 0$) are destroyed or malfunction (thus $\tau = 0$). Foraging efficiency would drop to 0, because no robots exist to forage prioritized items. Similarly, consider an environment where all prioritized items are blocked by non-prioritized items and that $\tau < 1$. If all non-prioritized robots suffer a malfunction or are destroyed, then $\tau = 1$ and no robots will be available to forage non-prioritized items. Foraging efficiency, E_P, will then drop to 0.

As shown in Section 5.2, the honey bee algorithm experiences little change to efficiency when $\tau(0)$ is varied. The honey bee algorithm adapts the swarm specialization ratio over time. It follows that, for the above scenarios, the honey bee algorithm will be able to re-adapt the swarm specialization ratio in order to replenish the robots that were destroyed.

The robots controlled by the honey bee algorithm are homogeneous, in that every robot in the swarm has the ability to take on any of the roles required for the algorithm to function (i.e. scout, unemployed forager, employed forager), as well as to adapt to forage either prioritized or non-prioritized items. It follows that the swarms running the honey bee algorithm are more redundant than swarms running the desert ant algorithm and the naïve algorithm.

5.4.2. Decentralized Coordination

Decentralized coordination can be attained by creating algorithms that do not depend on decisions, actions, or sensor readings of any single individual or a few individuals of the swarm. This section provides rational arguments to compare the decentralized coordination mechanisms for each algorithm, supported by evidence.

The robots of both the desert ant algorithm and naïve algorithm do not depend on each other and there is no explicit coordination mechanism. Despite no explicit coordination mechanism existing between robots in the desert ant and naïve swarms, the robots coordinate implicitly by using obstacle avoidance to avoid obstructing each other's movement. In that way, the robots in the swarm attempt to avoid inter-robot interference. The mechanism for obstacle avoidance is entirely decentralized, because each robot only depends on its own internal sensors and thus coordination is entirely decentralized.

On the other hand, the honey bee algorithm uses explicit communication as a coordination mechanism in order to recruit foragers to areas with a high density of prioritized items (as described in Section 3.3). If the scout robots experience a fault in evaluating the quality of a site (for example, a robot detects that a site is of high quality when it is actually a site of poor quality), then foragers will be incorrectly recruited to forage sites of low quality. Only a few individuals (the scouts) can coordinate the swarm and therefore any faults in the scout robots may negatively influence the efficiency of the entire swarm. The coordination mechanism of the honey bee algorithm is not entirely decentralized and the honey bee algorithm is less robust than both the desert ant algorithm and the naïve algorithm in terms of decentralized coordination.

To provide further evidence that the scout robots can mislead the swarm to incorrectly forage areas that have low quality resources, an examination of the average time spent in the recruitment state, per item foraged, per robot has been done. Table 5 and Figure 17 illustrate the average time steps, $t_{recruitment}$, spent performing recruitment per robot, per (both prioritized and non-prioritized) item foraged for the honey bee algorithm, for each environment distribution. The results were averaged over each environment item type ratio, r.

Table 5 illustrates that, on average, each robot spent at least 11.96% of its total foraging time recruiting other robots to forage higher quality areas for all values of $r > 0$. This is particularly relevant for uniformly distributed environments with a low ratio of prioritized items ($r = 0.2$), because it is unlikely that sites of high quality do not exist, because the prioritized items will be scarce and scattered. This means that the scouts were recruiting other robots to forage areas that were not of a high quality. The fact that scouts can mislead the swarm to foraging sites with low quality, demonstrates that the honey bee algorithm is less robust due to less decentralized coordination.

Table 5: Average time steps, per robot, per prioritized item foraged, that were spent performing recruitment for the honey bee algorithm, in each environment distribution, for each environment item type ratio r.

r	**Clustered**	**Gaussian**	**Uniform**	**Vein**
0	0	0	0	0
0.2	0.119	0.152	0.120	0.161
0.25	0.127	0.144	0.128	0.159
0.33	0.140	0.135	0.139	0.152
0.5	0.162	0.157	0.163	0.169
0.67	0.182	0.175	0.183	0.152
0.75	0.179	0.177	0.184	0.172
0.8	0.183	0.183	0.186	0.191
1	0.167	0.173	0.163	0.219

6. Conclusions

The first contribution of this research was the introduction of a novel variation of the swarm robotics multi-foraging problem, denoted as the prioritized foraging problem. The prioritized foraging problem differs from other multi-foraging swarm robotics problems in that two types of items which have different priorities exist. The items with higher priorities should be foraged as fast as possible, while the non-prioritized items should only be foraged to enable the prioritized items to be foraged at a faster rate. The prioritized foraging problem is a novel way of modelling the real-world problem of search and rescue.

Existing naïve and desert ant algorithms were reviewed and selected for evaluation on the prioritized foraging problem. Furthermore, a novel honey bee inspired foraging algorithm was developed, which specifically modelled the recruitment mechanism and division of labour mechanism of honey bee swarms.

Emperical experiments were defined in Chapter **??**, and an emperical analysis was performed in Chapter **??**. The purpose of the emperical analysis was to investigate each algorithm's performance on the prioritized foraging problem in terms of the major swarm robotics characteristics of efficiency, scalability, flexibility, and robustness as well as the behaviours that enabled those characteristics.

Each algorithm's efficiency was compared to each other algorithm's ef-

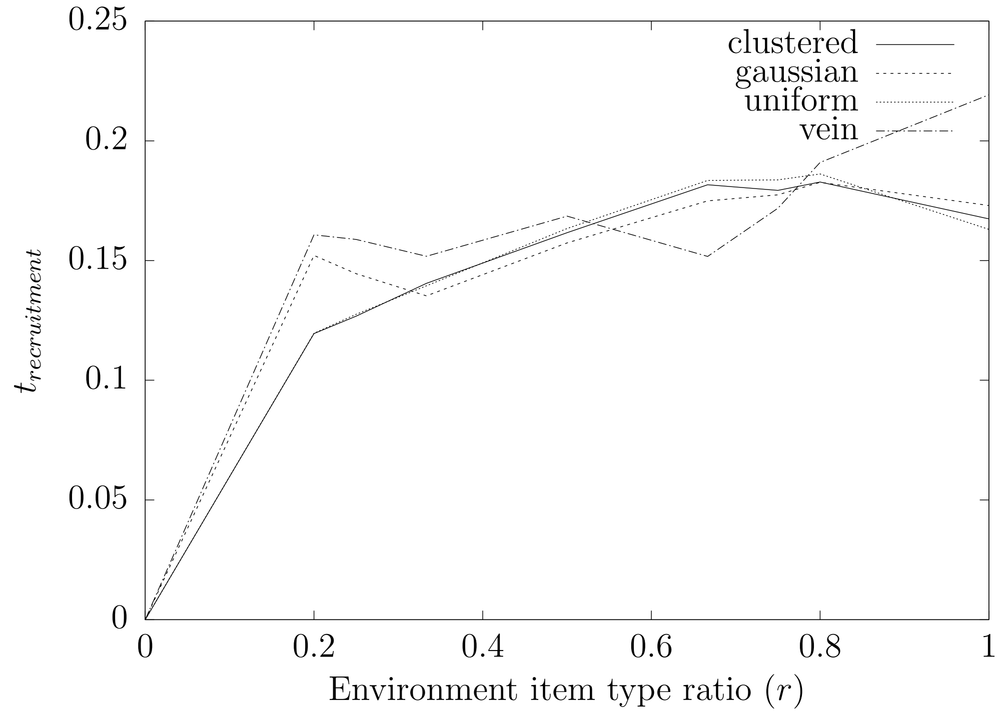


Figure 17: Average time steps spent performing recruitment activities by robots of the swarm for each item foraged, $t_{recruitment}$, for the honey bee algorithm, per item distribution, for each environment item type ratio, r.

ficency using a wins and losses approach. Results showed that the honey bee algorithm was the most efficient over all environments and swarm configurations, while the desert ant was the next most efficient and the naïve algorithm was the least efficient.

The flexibility of the algorithms was analysed in terms of the flexibility over environmental item ratio, and the flexibility over different environment distribution types. Results indicated that the honey bee algorithm could adapt the swarm's specialization ratio to effectively forage a given environment item ratio, and was therefore the most flexible in terms of environmental item ratio. The naïve algorithm only appeared more flexible than the desert ant algorithm because the naïve algorithm performed equally poorly across all environment item distributions, where as the desert algorithm favoured particular item distributions.

The honey bee algorithm was the most flexible over different types of environmental distributions, followed by the desert ant algorithm, and lastly the naïve algorithm. The honey bee algorithm's high flexibility was attributed to its adaptation of the specialization ratio to better suit the environment ratio of the accessible environment. The honey bee algorithm's adaptation of specialization ratio allowed the swarm to focus on foraging non-prioritized items when only non-prioritized items were accessible, and then to adjust the ratio to focus on foraging prioritized items when more prioritized items became accessible, and vice versa. The desert ant and naïve algorithms were less flexible than the honey bee algorithm, due to their inability to adapt the initial swarm specialization ratio.

The scalability of each algorithm was analyzed in terms of swarm scalability and problem scalability. Swarm scalability was sub-linear for all algorithms, however the naïve algorithm was the most scalable. The desert ant algorithm had poor swarm scalability when compared to the naïve algorithm. The desert ant algorithm's site fidelity increased inter-robot interference, which resulted in decreased efficiency when swarm density was high. The honey bee algorithm was shown to be slightly more scalable than the desert ant algorithm, due to the honey bee algorithm's attempt to regulate the number of active foragers by division of labour. Future investigation is necessary to determine why the honey bee algorithm did not regulate the number of foragers at high swarm densities very well.

The honey bee algorithm was the most scalable in terms of the problem density, followed by the naïve algorithm, and lastly the desert ant algorithm. The desert ant algorithm performed comparatively better than the naïve algorithm in terms of problem scalability. It was theorized that the desert ant algorithm could potentially exploit hard to reach sites in complex environments. The problem scalability of the honey bee algorithm was attributed the honey bee algorithm's ability to adapt the swarm specialization ratio to help clear non-prioritized items. By clearing the non-prioritized items, the honey bee algorithm decreased inter-robot and environmental interference, which in turn increased the foraging efficiency in problems of high complexity.

Lastly, the robustness of each algorithm was evaluated in terms of redundancy and decentralized coordination. A rational argument motivated that the honey bee algorithm was the most redundant, because the honey bee algorithm swarm is completely homogeneous. The robots of the desert ant algorithm and naïve algorithm are heterogeneous, and thus the swarm is less redudant. The coordination between robots of the desert ant and naïve algorithms was determined to be more decentralized than the coordination of robots in the honey bee algorithm. Evidence showed that invalid information was communicated to the rest of swarm by individuals with faulty information, impacting efficiency and showing that communication is less decentralized.

In summation, this dissertation primarily contributed the prioritized foraging problem and the novel honey bee inspired foraging swarm robotic algorithm. The emperical analysis provided a detailed view of how well each algorithm embodied each major characteristic of swarm robotics on the prioritized foraging problem.

[1] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, A. M. Erkmen, Search and rescue robotics, in: Springer Handbook of Robotics, Springer, 2008, pp. 1151–1173.

[2] A. M. Naghsh, J. Gancet, A. Tanoto, C. Roast, Analysis and design of human-robot swarm interaction in firefighting, in: Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication, IEEE, 2008, pp. 255–260.

[3] M. Dorigo, E. Sahin, Swarm robotics, Autonomous Robots 17 (2004) 111–113.

[4] A. F. T. Winfield, Foraging Robots, in: Encyclopedia of Complexity and Systems Science, 2009, pp. 3682–3700.

[5] M. Dorigo, E. Bonabeau, G. Theraulaz, Ant algorithms and stigmergy, Future Generation Computer Systems 16 (8) (2000) 851–871.

[6] M. Dorigo, Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings, Vol. 4150, Springer-Verlag New York Incorporated, 2006.

[7] M. Dorigo, M. Birattari, Ant colony optimization, in: Encyclopedia of Machine Learning, Springer, 2010, pp. 36–39.

[8] N. R. Hoff, A. Sagoff, R. J. Wood, R. Nagpal, Two foraging algorithms for robot swarms using only local communication, in: Proceedings of the 2010 IEEE International Conference on Robotics and Biomimetics, IEEE, 2010, pp. 123–130.

[9] T. S. Collett, E. Dillmann, A. Giger, R. Wehner, Visual landmarks and route following in desert ants, Journal of Comparative Physiology A 170 (4) (1992) 435–442.

[10] J. P. Hecker, M. E. Moses, Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms, Swarm Intelligence 9 (1) (2015) 43–70.

[11] M. Collett, T. S. Collett, S. Bisch, R. Wehner, Local and global vectors in desert ant navigation, Nature 394 (6690) (1998) 269–272.

[12] R. Wehner, Desert ant navigation: How miniature brains solve complex tasks, Journal of Comparative Physiology A 189 (8) (2003) 579–588.

[13] M. Müller, R. Wehner, Path integration in desert ants, Cataglyphis fortis, Proceedings of the National Academy of Sciences 85 (14) (1988) 5287–5290.

[14] R. Möller, D. Lambrinos, R. Pfeifer, T. Labhart, R. Wehner, Modeling ant navigation with an autonomous agent, From Animals to Animats 5 (1998) 185–194.

[15] J. P. Hecker, K. Letendre, K. Stolleis, D. Washington, M. E. Moses, Formica ex machina: Ant swarm foraging from physical to virtual and back again, in: Swarm Intelligence, Springer, 2012, pp. 252–259.

[16] S. Zhang, S. Schwarz, M. Pahl, H. Zhu, J. Tautz, Honeybee memory: A honeybee knows what to do and when, Journal of Experimental Biology 209 (22) (2006) 4420–4428.

[17] R. Menzel, M. Giurfa, Cognitive architecture of a mini-brain: The honeybee, Trends in Cognitive Sciences 5 (2) (2001) 62–71.

[18] D. Moore, D. Siegfried, R. Wilson, M. A. Rankin, The influence of time of day on the foraging behavior of the honeybee, Apis mellifera, Journal of Biological Rhythms 4 (3) (1989) 305–325.

[19] T. D. Seeley, The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies, Harvard University Press, 2009.

[20] S. Janson, M. Middendorf, M. Beekman, Searching for a new home–scouting behavior of honeybee swarms, Behavioral Ecology 18 (2) (2007) 384–392.

[21] E. H. Østergaard, G. S. Sukhatme, M. J. Mataric, Emergent bucket brigading-a simple mechanism for improving performance in multi-robot constrained-space foraging tasks, in: In Autonomous Agents, Citeseer, 2001.

[22] P. V. Switzer, Site fidelity in predictable and unpredictable habitats, Evolutionary Ecology 7 (6) (1993) 533–555.

[23] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in: Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Vol. 1, 2009, pp. 59–65.

[24] K. Sugawara, T. Watanabe, Swarming robots-foraging behavior of simple multirobot system, in: Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3, IEEE, 2002, pp. 2702–2707.

[25] T. H. Labella, M. Dorigo, J.-L. Deneubourg, Division of labor in a group of robots inspired by ants' foraging behavior, ACM Transactions on Autonomous and Adaptive Systems 1 (1) (2006) 4–25.

[26] J. F. Brune, Extracting the Science: A Century of Mining Research, SME, 2010.

[27] H. E. Frimmel, W. Minter, Recent developments concerning the geological history and genesis of the Witwatersrand gold deposits, South Africa, Special Publication-Society of Economic Geologists 9 (2002) 17–46.

[28] K. Lerman, A. Galstyan, Mathematical model of foraging in a group of robots: Effect of interference, Autonomous Robots 13 (2) (2002) 127–141.

[29] M. Schneider-Fontan, M. J. Mataric, Territorial multi-robot task division, IEEE Transactions on Robotics and Automation 14 (5) (1998) 815–822.

[30] M. Helbig, A. P. Engelbrecht, Performance measures for dynamic multi-objective optimisation algorithms, Information Sciences 250 (2013) 61–81.