PSEUDO-HIERARCHICAL ANT-BASED CLUSTERING USING A HETEROGENEOUS AGENT HIERARCHY AND AUTOMATIC BOUNDARY FORMATION

by

JEREMY BERNARD BROWN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2009

ACKNOWLEDGEMENTS

I would first like to thank my supervising professor Dr. Manfred Huber. He has been ever patient and has spent countless hours advising me. I also wish to thank Dr.Kamangar and Dr.David Levine for serving on my committee. I would also like to thank my family, namely my parents, Jerry and Shirley Brown and grandmother, Ethel Clements who have heavily supported me while in graduate school through free rent and delicious food. I would also like to especially than my uncle, Kevin King, and grandfather, LeRoy King, for taking special interest in my thesis and making sure I "got it done". In addition, I would also like to thank Giles D'Silva and Jacob Heidenreich for all the support and help they have provided throughout the journey of creating this thesis.

April 20, 2009

ABSTRACT

PSEUDO-HIERARCHICAL ANT-BASED CLUSTERING USING
A HETEROGENEOUS AGENT HIERARCHY AND
AUTOMATIC BOUNDARY FORMATION

Jeremy Bernard Brown, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Manfred Huber

The behavior and self-organization of ant colonies has been widely studied and served as the inspiration and source of many swarm intelligence models and related clustering algorithms. Unfortunately, most models that directly mimic ants produce too many clusters and converge too slowly. A wide range of research has attempted to address this issue through various means, but a number of problems remain: 1) Ants must still physically move from one cluster to another through intermediate locations, 2) current methods for remote relocation of an item only consider one movement at time to a particular location and do not consider patterns in movement to that location, and 3) while current methods have included effective bulk item movement, they do not provide efficient movement while still maintaining the self-organizing nature of ant-based clustering which is essential for filtering out outliers and allowing effective splitting of clusters. This thesis addresses these problems by proposing a new algorithm for ant-based clustering. In this algorithm ants maintain a movement zone around each cluster, keeping ants from spending time in locations

where there is nothing to do. These movement zones around individual clusters are used to elect representatives that are responsible for all long distance movement. Each representative can, probabilistically, pass an object it has to any other representative. Since each cluster has approximately one representative at any given time, the search space for placing items over a long distance is reduce to the number of clusters. So instead of having all the ants use up time wandering around the map, one ant can be responsible for sampling all local clusters. This provides an infrastructure by which clusters can efficiently merge over long distances and better clusters for items in the wrong clusters can be found without having to travel to them. While this model does require a considerable overhead as compared with contemporary algorithms, a better convergence rate can be achieved because the efficient bulk movement of items and sampling at the cluster level.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The grouping of items is so ubiquitous throughout people's daily lives that many are unaware that they are doing it. Simply taking a trip to the local grocery store showcases the importance of clustering. One does not necessarily always need to go down every aisle in a grocery store to find a particular item. Simply looking at the items near the end of the aisle gives some indication of what can be found in that aisle. This improves the efficiency of the shoppers. Furthermore shoppers have become so accustomed to stores grouping items with similar traits that any store that did not use such a scheme would have difficulty surviving.

Grouping naturally occurs when cleaning out and organizing a garage or storage room whose contents vary greatly and may have been deposited over a span of years (sometimes decades). Once all the items are removed, similar items are grouped together. Another term for this is clustering. The items do not necessarily have to be exactly alike; one may put a screwdriver, screws, nails, wrenches and bolts together in one pile. While they all have distinct uses, most would agree that their purposes are similar in relation to everything else in the garage such as a rake. One important quality of clustering is that one does not necessarily need to have a specific classification in mind to create groups. Groups formed from clustering have similar properties from which classifications can be derived. These classifications can then later be used to efficiently sort and study the data. This same concept can be used to mine information from databases or detect anomalies in computer networks. In the garage example it could be derived that the owner of this garage could build

things. For a computer network this could be a detection of an attack, for a database this could be a detection of a pattern of data that could then be exploited. Uses for clustering extend far beyond the above examples and can involve extremely large sets of data.

One form of clustering, ant-based clustering has been shown to be remarkably robust and has qualities that set it apart from the rest of clustering algorithms. Ant-based clustering is, as its name implies, modeled after ants that display a remarkable ability to cluster items such as corpses (referred to as cemetery formation). The remarkable aspect of this is that ants achieve this task without any centralized control. In comparison to other methods, ant-based clustering has several qualities that set it apart from other clustering methods. 1) The number of clusters does not have to be defined beforehand. This proves useful when there is little knowledge about the data and when there is no need to have a specific number of clusters. The ant algorithm can even be used to estimate the correct number of clusters for another clustering algorithm that may be more efficient with that type of data but requires the number of clusters to be predefined. 2) The complexity of the algorithm scales with the number of items being clustered. This means that the main factor in determining how much time is needed to run the algorithm is primarily dependent upon only the number of items. While many other clustering algorithms are superior for small data sets, this ability of ant-based clustering to scale with only the number of items is promising for large data sets.

One main problem presents itself in ant-based clustering when scaling to large data sets. In ant-based clustering the ants operate on the data on a grid. Ants pick up items if they are surrounded by dissimilar items and then deposit them among similar items. Between the pick up and dropping operations the ant must travel on the grid. This transportation process however, can become increasingly

inefficient as the size of the map increases because the probability of randomly finding a desired location on the map decreases, especially as the number of distinct clusters is increased. Furthermore, the work of one ant can be undone by another. For example take two clusters, A and B, that are on opposite ends of a grid and should be merged together. An ant can take an item from cluster A and deposit it in B while another ant can take an item from B and deposit it in A. With no net movement of items, the clusters maintain approximately the same number of items, thus merging does not occur. Given enough time, the two clusters may merge, but only having two ants will significantly slow this down. In an effort to address these issues, this thesis proposes a new method of ant clustering.

This novel approach uses connections, or direct links, among clusters to facilitate long distance transport of items. Each end of a connection is held by an ant that represents the cluster, called the queen. Every queen has a connection to every other queen, and every queen represents a cluster, so when a queen selects a random connection to transfer an item through, this amounts to a random search among clusters. The advantage of this is that the ant never has to actually travel through intermediate steps on the grid to that location; it simply looks at the location of the queen at the other end. In addition, a connection can be changed to allow for a steady one-way transport of items (all other connections are then ignored) which allows for efficient cluster merging. An additional measure of the algorithm uses movement zones around items to restrict the movement of all ants (a queen and her workers) to only those locations near an item. With the queens using connections to form long distance transports, the ants do not need to wander around in free space, thus drastically reducing the length of time needed to accomplish clustering.

The remainder of this thesis has the following structure: Chapter 2 discusses background and related work that includes key papers in the development of ant-

based clustering and sorting algorithms. Also, a description of the type of ant-based algorithm that is used as a comparison for the novel algorithm proposed in this thesis is included. Chapter 3 describes the novel ant-based clustering algorithm. Chapter 4 outlines experiments and provides the analysis of the results of the experiments. Chapter 6 will conclude the thesis and propose future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Nature Inspired Algorithms

From the effective foraging behaviors of ants to the efficient v-formation flocking behavior of many migratory birds, nature is recognized as one of the greatest developers of efficient and effective algorithms. With an ever increasing amount of complex data generated, computer scientists are increasingly looking towards nature for inspiration in solving complex problems which in some cases require surprisingly little effort. Robotic control, data mining, neural networks, network security and routing are only a handful of applications which have benefited from methods inspired by nature. The subject area of nature inspired algorithms, particularly those based on swarm behaviors such as those of ants, bees, and termites has been well studied in [1][2][3]. The basic advantages of swarm based systems are that they provide robust solutions which can be computed in an asynchronous and parallel manner using little, if any, central control. With swarms, complex behaviors arise out of simple interactions among the members. While the rules that govern members are relatively simple, selecting rules and behaviors for members such that they achieve the desired emergent behavior has spurred much of the research in swarms and swarm dynamics.

2.2 Ant Colony Optimization

Ant colony optimization (ACO, [4]) is a metaheuristic that formally describes concepts that are based on the behaviors of ants in the real world. The inspiration for ACO originated with the foraging behavior of ants. ACO uses markers on a map,

called pheromones, to record the goodness of a particular trail or location. Using positive feedback mechanisms the pheromones are adjusted by the ants as they pass over them and at the same time the pheromones influence where the ants travel on the map. This creates a system of trail-building and trail-following to find solutions to problems. Since the ants use the environment as a communication medium, commonly referred to as stigmergy, ants do not have to communicate directly with each other, thereby allowing the process to be both distributed and asynchronous and often lending itself well to parallelization. For this reason ACO has been successful in addressing NP-hard problems such as the traveling salesman problem (TSP) [5][6], timetabling, and scheduling. An introduction to the area of ACO can be found in [7] and [8] provides a current online reference to many ACO related sources.

2.3 Ant Based Clustering and Sorting

While ACO and Ant-based clustering and sorting are highly related, they can be distinctively distinguished. Both are subcategories in a much broader category of *ant algorithms* which model some aspect or behavior of actual ants. Ant-based clustering and sorting has commonalities with ACO in that both use positive feedback, local information, and random and probabilistic elements. However, while ACO models are based on the ant foraging behavior and can be used in a wide variety of situations, ant-based clustering and sorting only deals with the clustering and sorting behavior of ants and does not use artificial pheromones. Also, at least for the basic algorithm, the synergetic effect is minimal, meaning that the performance of the algorithm is mostly independent of the population size. However, it is important to note that the population of ants in a clustering algorithm must be significantly smaller than the number of items. While this thesis proposes a variation on ant-based clustering, it does utilize a very abstract version of ACO where trails among clusters are reduced

down to a single link. The following sections will describe ant-based clustering and sorting in more detail.

2.4 Deneubourg's Clustering Model for Clustering

Deneubourg et al. [9] first proposed an ant based clustering and sorting algorithm in 1990. As implied by the name, the proposed model is based upon the clustering and sorting behaviors of actual ants. *Clustering* in this context refers to the collection of items into piles by the ants. The example ant species used for the sorting behavior is *Pheidole pallidula* which has been observed to do cemetery formation (the collection of corpses). *Sorting*, on the other hand, refers to the distinction of items by the ants and the specific placement of items according to the properties and attributes exhibited by the individual items. The example species used here is *Leptothorax unifasciatus* where larvae are spatially arranged according to their size. The aim of Deneubourg et al's work was to accurately model the behavior of the ant, not necessarily to design an efficient algorithm. In order to articulate these behaviors in a more formal sense, Deneubourg el al. proposed a continuous model and a Monte Carlo model, the latter of which was validated experimentally.

In the Monte Carlo experiments, the ants are represented by simple agents that randomly move about a square network of points. Ants cannot go past wall boundaries and only one agent and/or item can exist per location. Items of one or two types are initially scattered randomly on the grid. The agent has the ability to pick-up, carry, and drop items. The chance that an ant picks up or drops an item is determined by the following probabilities:

$$p_{pick}(i) = \left(\frac{k^+}{k^+ + f(i)}\right)^2 \tag{2.1}$$

and

$$p_{drop}(i) = \left(\frac{f(i)}{k^- + f(i)}\right)^2 \tag{2.2}$$

$f(i)$ is here the neighborhood function that estimates the fraction of items of a particular type in the local area of the agent. This was calculated by using memory to record the type of items seen by the agent within a particular time frame. This was primarily done to match the capabilities and constraints of robots and to somewhat replicate the mechanism by which ants would actually determine the local distribution of items. The parameters, k^+ and k^- are used to vary the influence of the neighborhood function, $f(i)$, on the ants' picking and dropping actions. These values were fixed at 0.1 and 0.3, respectively. The general idea of the algorithm is that if an ant detects an item that is dissimilar from its neighbors, it is likely to pick that item up off the map and move it away from that location. Conversely, if an item that the ant has is similar to the surrounding neighborhood, then it is likely to put that item down on the map. To the best of this author's knowledge, all ant-based clustering and sorting models share this basic principle in some form.

2.5 Lumer and Faieta's Sorting Extension

While Deneubourg el al.'s focus is primarily on the design for use in robots, Lumer and Faieta proposed several modifications that allowed the algorithm to be more applicable to data analysis [10]. While pseudo code and a detailed explanation of this basic algorithm can be found in [11], only the significant changes and modifications will be outlined in this thesis. The first major difference is in terms of the properties used to define the characteristics of the data that the ants used for sorting. In Deneubourg el al.'s experiments items were literally labeled as A or B, and were thus essentially pre-classified, and all the ants were assigned the trivial task of placing

all items for the classification together. Lumer and Faieta changed the description of a data item to a vector of numerical values. This allowed for a more general representation of data that permits the algorithm to be applied to all types of data as long as similarity (or dissimilarity) metric is defined. In this case the Euclidian distance of the vectors is used as the similarity metric. With this form of representation of data the new neighborhood function was redefined as:

$$f(i) = \begin{cases} \frac{1}{\sigma^2} \sum_j \left(1 - \frac{\delta(i,j)}{\alpha}\right) & \text{if } f(i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where, $\delta(i,j) \in [0,1]$ is the dissimilarity between two points, i and j in data space. $\alpha \in [0,1]$ is the data dependent scaling factor used to define the point at which two similar items become dissimilar. σ^2 is the size of the area of the local neighborhood for which the ant is usually the center. The radius of the ant's perception is then $\frac{\sigma-1}{2}$. $f(i)$ is described as the average similarity over the ant's local neighborhood, since item i is compared to item j in each location of the neighborhood and then divided by the size of the neighborhood σ^2. As the function sums over the neighborhood, empty locations are indirectly penalized since they provide no positive contribution, thereby creating a positive feedback mechanism that encourages a tight clustering as opposed to lose clustering.

Another novel idea put forth in [10] is the use of heterogeneous ants. The differences among the ants were limited to the parameter settings that affect the ant's *step size* (the number of cell locations on the map that an ant can cross in a single move). Larger step sizes allow for greater rates of movement or speed as the ant traverses the map. The neighborhood function is modified to account for the ant's speed with faster ants having small neighborhoods and slower ants having larger ones.

This allows faster, less discriminating ants to build coarse clusters, while slower, more discriminating ants can better organize established clusters.

To further reduce the number of statically equivalent clusters [10] (a set of separate clusters which, if merged, would produce a new single stable cluster) Lumer and Faieta put forth the idea of providing an ant with *short-term memory*. With this extension an ant can remember the last m items that were picked up and the item's respective dropping positions. An Ant would then travel in the direction of the location of the most similar item found in its memory. A variation of this idea is put forth later in this chapter.

The algorithm and all modifications mentioned thus far still suffer from convergence problems where small clusters of each data type are not merged together within a reasonable amount of time. To alleviate some of this problem, behavior switches were introduced so that when stagnation was detected, ants would begin to destroy clusters so that the elements of a destroyed cluster would hopefully be redeposited in another cluster.

2.6 Handls Sorting Extension

An additional basic algorithm with particular extensions, put forth by Handl in her thesis [12], is chosen for comparison with the new algorithm proposed in Chapter 3. This particular algorithm is chosen for several reasons: 1) The basic algorithm, while different from preceding versions is still fundamentally the same as that put forth by Lumer and Faieta, 2) the algorithm is well defined with complete pseudo-code (see Algorithm 1)) and 3)a relatively recent (2003) comparative study has already been provided with the algorithm.

This algorithm (see Algorithm 1) begins with an installation phase that is responsible for setting up the map and placing items on the map. First, data items

Algorithm 1: Basic Ant Clustering Used By Handl

```
1  begin
2      INITIALISATION PHASE;
3      Randomly scatter data items on the toridal grid;
4      for each j in 1 to #agents do
5          i := random_select(remaining_items);
6          pick_up(agent(i),i);
7          g := random_select(remaining_empty_grid_locations);
8          place_agent(agent(i),g);
9      end
10     MAIN LOOP  for each it_ctr in 1 to #iterations do
11         j := random_select(all_agents);
12         step(agent(j),stepsize);
13         i := carried_item(agent(j));
14         drop := drop_item?(f*(i));
15         if drop = TRUE then
16             while pick = FALSE do
17                 i := random_select(free_dat_items);
18                 pick := pick_item?(f*(i));
19             end
20         end
21     end
22 end
```

are scattered randomly on a toroidal grid. Then a fixed number of ants, referred to as agents in this algorithm, randomly select items on the map and pick them up. Then agents are placed on the map with the acquired items. After the installation, the sorting begins in the main loop. First an agent is randomly chosen. Next, the agent takes a step with a given *stepsize* on the grid. Then the agent probabilistically decides whether or not to drop the item on the map. If the agent decides to drop the item, the item is place in the grid position of the agent. If that location is already occupied by another data item then random search is done in the local neighborhood of the agent until an unoccupied location is found. After the item has been placed on the map the agent searches for a new item: an item is randomly selected from a

list of free items (items not held by an agent); the agent travels to the location of the item and evaluates the function based on the local neighborhood function $f^*(i)$, and then probabilistically decides to pick up that item; if this item is not picked up then a new item is chosen, and this process is repeated until an item is finally chosen.

In order to probabilistically determine to pick up or drop items Deneubourg et al.'s threshold functions (also see Equations 2.1 and 2.2) are used:

$$p_{pick}(i) = \left(\frac{k^+}{k^+ + f^*(i)}\right)^2 \tag{2.4}$$

and

$$p_{drop}(i) = \left(\frac{f^*(i)}{k^- + f^*(i)}\right)^2 \tag{2.5}$$

where $k^+ = 0.3$ and $k^- = 0.1$, and $f^*(i)$ is a modified version of Lumer and Faietas neighborhood function [10] (see Equation 2.3):

$$f^*(i) = \begin{cases} \frac{1}{\sigma^2}\sum_j \left(1 - \frac{\delta(i,j)}{\alpha}\right) & \text{if } \left(f^*(i) > 0 \bigwedge \forall j \left(1 - \frac{\delta(i,j)}{\alpha}\right) > 0\right) \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

This version still retains the important properties of the original such as penalizing empty locations by the division by σ^2 which promotes tightly packed clusters. In addition to this, the additional constraint of $\forall j \left(1 - \frac{\delta(i,j)}{\alpha}\right) > 0$ heavily penalizes high dissimilarities and improves separations of clusters.

The α parameter is determined by an extension of the basic algorithm which is referred to as the adaptive scaling parameter scheme. In this scheme the level of ant activity is tracked and any excessive or extremely low activity on the grid is detected and the α is adjust accordingly.[13]

In addition to the adaptive scaling parameter scheme, the overall algorithm adopts several other distinctions used for its comparative study: increasing the radius of an agent's perception, an interlude with a modified neighborhood function, short-term memory with look ahead capabilities and new probability thresholds.

Increasing the radius of perception is designed to reduce the initial complexity of calculating the neighborhood function at the beginning of the algorithm while improving sorting as the algorithm progresses. This is achieved by incremental increases in the radius, $\frac{\sigma-1}{2}$, to increase the size of the neighborhood, σ^2, which is used to measure the average similarity. The algorithm runs a fixed set of iterations determined by the number of items. The number of iterations is divided into four equal parts with any remaining parts making up the fifth and final section. After that particular section of iterations is executed the radius is incremented.

The interlude with a modified neighborhood function is an attempt to separate clusters after they have been well established. During an iteration interval from t_{start} to t_{stop} (both determined by the number of items) the modified neighborhood function replaces the scaling parameter $\frac{1}{\sigma^2}$ with $\frac{1}{N_{OCC}}$ in Equation 2.6. N_{OCC} is the actual number of items observed on the grid within the agent's neighborhood. This change causes the neighborhood function to temporarily ignore the item density, causing the clusters to become very lose and expand by pushing nearby clusters away. Once the scaling parameter is restored, the clusters re-tighten slightly further away from neighboring clusters.

Short-term memory with look-ahead extension is an effort to speed up convergence of the algorithm. This scheme is similar to the short-term memory extension used by Lumer and Faieta but there are several fundamental differences. Firstly, the last items dropped are not remembered, only their locations are. Instead of comparing the item that the agent has to the last items it carried, it goes to locations in

memory and evaluates the location using the neighborhood function $f^*(i)$. The location with the best match with the item currently held by the agent is chosen. Instead of biasing the agent's walk, this algorithm will probabilistically decide to jump the agent to that location using the drop probably function. If the jump is not made, then the memory is deactivated and the agent reverts to random steps on the map to find locations to drop an item.

For the final modification Handl used different threshold functions than those used by Deneubourg and Lumer and Faeita. The probability threshold for the picking operation is determined by:

$$p^*_{pick}(i) = \begin{cases} 1.0 & \text{if } f^*(i) \leq 1.0 \\ \frac{1}{f^*(i)^2} & \text{else} \end{cases} \tag{2.7}$$

and for the dropping probability:

$$p^*_{drop}(i) = \begin{cases} 1.0 & \text{if } f^*(i) \geq 1.0 \\ \frac{1}{f^*(i)^4} & \text{else} \end{cases} \tag{2.8}$$

These functions were experimentally derived for this algorithm and are not applicable to the basic ant algorithm. It is important to note however that the functions are completely deterministic for a given value produced by a given neighborhood function $f^*(i)$. More detail about the algorithm, extensions, and the reasoning behind it can be found in [12].

2.6.1 Other Work

Within the realm of data clustering, several other authors have put forth other extensions and modifications to the basic ant algorithm to improve its performance.

Some are hybrids with other algorithms and do use a grid for items or ants to exist on. Despite these differences they all share the quality of agents randomly moving in some space with an ability to move something from one location to another based on some probabilistic properties. Not all are directly comparable to the algorithm put forth in this thesis, but are presented to show the versatility of the basic concept put forth by Deneubourg et al. [9].

Many of the modifications include hybridization with other algorithms such as the fuzzy c-means and k-means algorithms. Hybridizations with fuzzy c-means, referred to as the fuzzy ants algorithm, involves the movement of centroids of clusters rather than moving the items themselves [14][15]. Instead of moving on a map the ants move in the feature space of the items. Each ant is responsible for a feature of a cluster centroid and randomly searches for better locations for its particular feature so that its centroid is at a location of feature space that optimizes the overall partitioning of the data. While some work has been done with selecting the best number of clusters [16], the fuzzy ants algorithm still requires prior knowledge of the number of clusters.

Monmarché [17] put fourth another hybridization with the k-means clustering algorithm called ANTCLASS. The algorithm uses one or more alternating sweeps of the two algorithms. This alteration is designed for each algorithm to compensate for some of the shortcoming of the other. The ant clustering algorithm provides the approximate number of clusters which is given to the k-means algorithm which provides clusters with less noise. Monmarché also fundamentally modified the algorithm to allow the ants to manipulated piles of data instead of individual data elements themselves. Entire piles can be transported, and items can be removed from and added to a pile.

Another algorithm with the ability to move whole piles of items at once is proposed by Li et al. [18] and is referred to as SICABA. This algorithm proposed a new structure called an attractor. The attractor uses a mixture of both local information and global information in order to attract or not attract ants to drop off or pick up items from its pile. Attractors represent data sets of objects that are similar items as a whole. Depending on an attractor's relative size, an ant can pick up an entire pile or only those items determined to be the furthest away.

Montes de Oca et al. [19] studies the effect of allowing ants to communicate and exchanged information. Several different types of information are shared both directly, when ants meet, and indirectly by dropping information packets on the map. Ants exchange information that effect there movement trajectories and information about environmental representations.

2.7 Points of Improvement

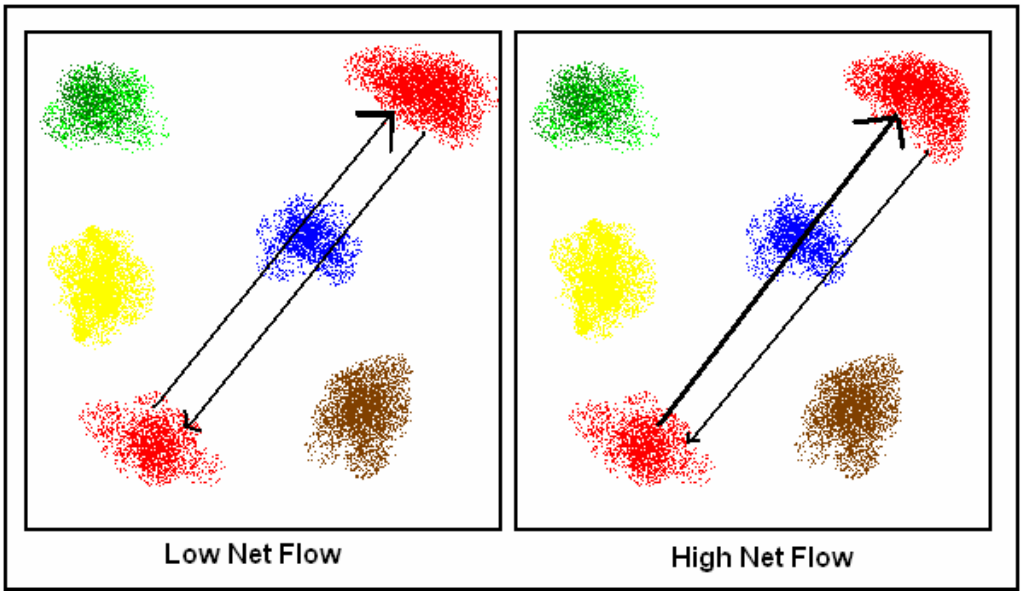

Figure 2.1. Net Flow of Grid Based Clustering.

All the algorithms in this chapter aim to improve upon the ant base clustering algorithm through alterations or extensions. However, room for improvement still exists. For the algorithms that still use a grid, effective travel on the grid is essential. As the number of items increases, so must the size of the map. As the size of the map grows in proportion to the size of a particular cluster it becomes increasingly harder to randomly find clusters to which items belong. Those algorithms that address this issue usually use some form of memory to address this. Memory allows the ant to jump to a particular location or provided a coordinate so the ant can travel there. Memory, however, represents a finite number of locations on the map; it does not represent how often ants travel between two locations, so even if several ants move items from one area to another repeatedly the ants will have no knowledge of it. In other words they have no record of the flow from one location to another (see Figure 2.1). There is a net flow of items in either direction between clusters, but one eventually dominates and clusters merge. Normally items in two clusters that should merge will have a new flow from the smaller cluster to the larger one because larger clusters are mores easily found by ants and they tend to have better organized neighborhoods than smaller ones; ants are more likely to leave a smaller cluster and more likely to stay in one with more items. The further merging clusters are apart and the greater the number of other clusters there are (which creates noise), the harder it is for ants to merge clusters. The level of net flow could be increased if there is some mechanism to focus the movement from one cluster specifically to another cluster while at the same time inhibiting flow in the reverse direction. This, in turn, would greatly speed up the merging process. Related to this problem is the amount of time ants spend in free space.

For algorithms that allow the ants to move on the map, one problem is the amount of time they spend in free space. Free space is the set of all the locations

on the map where no items are within the viewing radius of the ant. Since there are no items in these locations to compare with, the probability that ants drop an item is 0 since an item will always have 0 similarity to nothing. Clustering can only be done when ants compare similarity of items, but in free space there are no items to compare to, so those turns are wasted traveling between clusters. That is why many algorithms have jumping schemes that allow an ant to directly move from one location to another directly, but the number of jumps is usually small compared to the number of normal moves that an ant may make.

For clustering algorithms that move entire piles of items, there is the problem of how to judge similarity between two piles of items. Some algorithms just use properties of the entire piles in question. This forces a more stringent form of clustering where items must be somewhat similar to all the items in a pile. Some items, however, may be similar to a subset of data within a pile. Ants in algorithms that do not pile items simply look at the local neighborhood and drop the item in it (assuming it is similar enough). Judging whether or not to place an item based upon the characteristics of the entire cluster dissuades an ant from placing items in a pile even if the items would perfectly fit in a region of the pile.

While one of the most noted features of the ant-based clustering algorithm is that the number of clusters is not needed to be known beforehand, it is important to note that some of the modifications to the ant algorithms have reintroduced some dependence upon the need to know, at least approximately, the number of clusters. The novel algorithm proposed in Chapter 3 attempts to address these issues while still retaining the ability to locally organize clusters. While it may not address specific types of data sets as some modifications do, it does extend upon the basic ant-based clustering algorithm's ability to cluster competitively over a wide range of data.

CHAPTER 3

PSEUDO-HIERARCHICAL ANT-BASED CLUSTERING USING A HETEROGENEOUS AGENT HIERARCHY AND AUTOMATIC BOUNDARY FORMATION

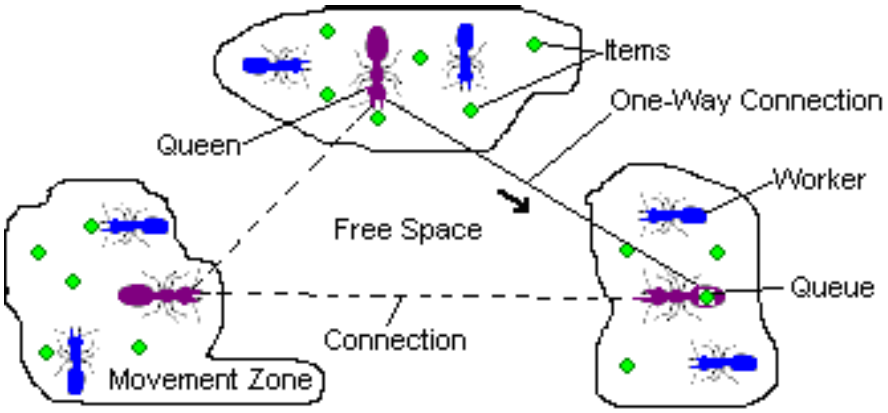

Figure 3.1. Pseudo-Hierarchical Ant-Based Clustering.

The novel approach for ant-based clustering as represented in Figure 3.1 introduces several new concepts that, to the best of the writer's knowledge are unique in the wide verity of already existing ant-based clustering and sorting algorithms. One of the most notable differences is the use of a hierarchy of heterogeneous ants. Other algorithms have used heterogeneous ants, but the difference among the ants was limited to variations in a few parameters and not completely different roles. The hierarchy consists of a queen and her workers. The queens act like cluster representatives, and at any given time there is approximately one queen per cluster. The queen's workers are primarily charged with the task of organizing the cluster but they also aid the queen in picking up and putting down her items. In order to keep

the workers in the same cluster as their queen, workers maintain movement zones around clusters in which they may walk. All other spaces are considered free space to which the ant may not move since it contains no items to manipulate. The queens, being the representatives of their cluster have the additional task of passing items long distances to other queens through connections. When a queen is successful in putting items through a connection with better than average success rate, the connection is turned into a one-way connection and the queen on the entrance end of the connection will only attempt to pass items to the queen at the exit end and to no other queen. This allows efficient transport of a large number of items from one cluster to another while still allowing for rejection of particular items that should not be put through the connection. With mass movements of items from on particular cluster to other clusters, similar clusters can effectively and efficiently merge from any arbitrary distance away on the map.

The algorithm proposed in this thesis contains a larger number of components compared to the algorithms in the previous chapter from which it is derived and is discussed over several modules to improve understandability. Each module has a corresponding figure in the text and complete pseudo-code in Appendix A. In the flow charts, all rectangles indicate that that section is discussed further in another module. Due to its length, before describing the algorithm in detail, this section provides an overview of the algorithm and describes the terminology used to describe the algorithm.

The ants in this algorithm consist of three casts of ants: workers, queens, and zombies. Additionally, the term colony will refer to a queen and the workers that she creates. Workers serve almost the same role as the ants in the original algorithms; their primary purpose is to pick up items from a neighborhood of dissimilar items, move in a random fashion, and then drop items in a neighborhood of similar items.

Workers also serve as an extension of their queen by helping pick up and put down objects within the colony's region. Unlike the other algorithms where the number of ants is constant, the workers are created dynamically by a queen. Queens are periodically created on top of items that relatively few ants have passed over. While the queen also does the job of a worker, its role serves three other purposes: to create workers, to serve as a representative of a cluster, and to create connections with other queens. In fact the queens use workers in order to establish dominance over a cluster. It is important to note that a queen is considered its own queen, so if an ant's queen is referenced and that ant is a queen, then the reference is to the ant itself. Queens can send items through the connections and give them to other queens. Items placed through the connection are place on the recieving queen's queue. The queue acts as a buffer of items that the queen's colony collectively handles. The third and final cast, the zombie ant, has the sole purpose of finding a place on the map for the item that it is holding. Since ants are continuously created throughout the span of the simulation, ants must also be periodically destroyed. In fact, each worker ant is given a finite lifespan at the moment of creation which dictates how many iterations it is allowed to operate. A queen has no set lifespan, but the queen does have a strength, which is used to determine the worker's lifespan at the time of creation of the worker. Both workers and queens can die, however, if they become stagnant for too long (e.g. when an ant gets caught outside a cluster).This presents a problem when an ant dies or is killed while it has an item and is on top of an item. Since no stacking is allowed, the ant cannot simply drop the item. Instead, the ant changes into a zombie, alive in that it is allowed to move around the map like any other ant, but dead in that its sole duty is to find an empty location to put the item in that location and then be destroyed upon doing so.

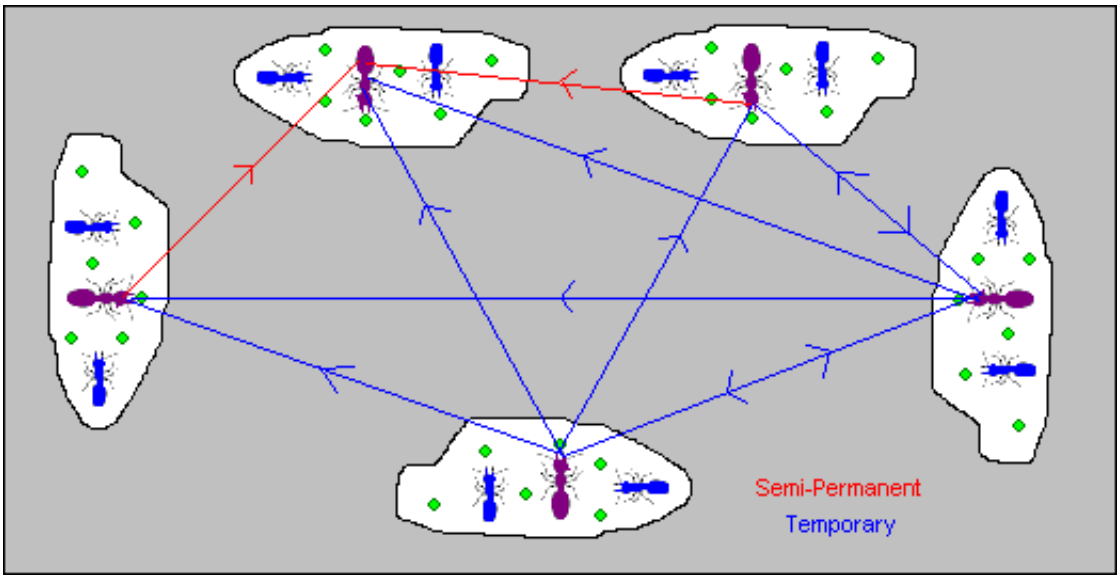

Figure 3.2. Connection's Influence Over Item Passing Among Queens.

The reason for the increased number of components of the algorithm as compared to its predecessors mentioned in Chapter 2, is to support efficient use of the connections among queens. During every step a queen takes, a queen attempts to establish a connection with another queen. The greater the chance of an item being passed through a connection the higher utility it has, thus providing a positive feedback mechanism. Once the utility rises above the average of the utility of all connections, a semi-permanent one-way connection is established, and, aided by the queen's workers, objects are passed though the connection. It is important to note that even though a connection is established, an item's suitability to be put through the connection is recalculated for each attempt, so just because there is a connection does not necessarily mean that an item will be put through. Also, queens at the entrance end of a connection can share the same ant at the exit end, but not vice versa. As a result, each queen can only have one additional destination and thus the number of actions it must consider while holding an object is strictly limited. For as long as

the connection remains one-way the queen at the entrance will continue to attempt to give items to the queen at the exit end; once the connection is downgraded to normal, the queen will go back to randomly selecting queens to pass items to. Figure 3.2 shows an abstract view of the relationship between the types of connections and how items can be passed among the queens. Any queen can receive from any other queen. Queens at the entrances of semi-permanent connections can only pass to the queen at the exit end. Queens at the exit end of a semi-permanent connection can still receive from any other queen, but may not pass items to other queens.

The connection is intended to take the place of short term memory and other measures used in both Lumer and Faieta's and Handl's algorithms to aid in the speed of convergence. Unlike short term memory, the unidirectional nature of the connection makes it less likely that two ants will cancel out each other's work when merging a cluster. For example, ants from cluster A taking an item and depositing it across the map to cluster B and another ant taking one from B and deposits it in A. The net flow of items is zero, and the number of items in the two will remain the same. While one cluster will eventually collapse, for large, relatively stable clusters the time needed to merge the clusters can become intractable due to time constraints. In small maps with few items this may not be a concern but with larger maps, larger numbers of items, and a larger number of clusters, this effect grows in importance.

One other important distinction of the algorithm proposed here is the use of a movement zone around the objects that ants are allowed to travel in. With the overlapping of movement zones around items, a contiguous movement zone is created around a cluster. This restricts workers and queens to the cluster that they were created in. While it is possible for ants to end up in different clusters and for clusters to split, ants that are in the wrong cluster will eventually die out and new ones that are unique to that cluster will take over.

3.1 The Algorithm

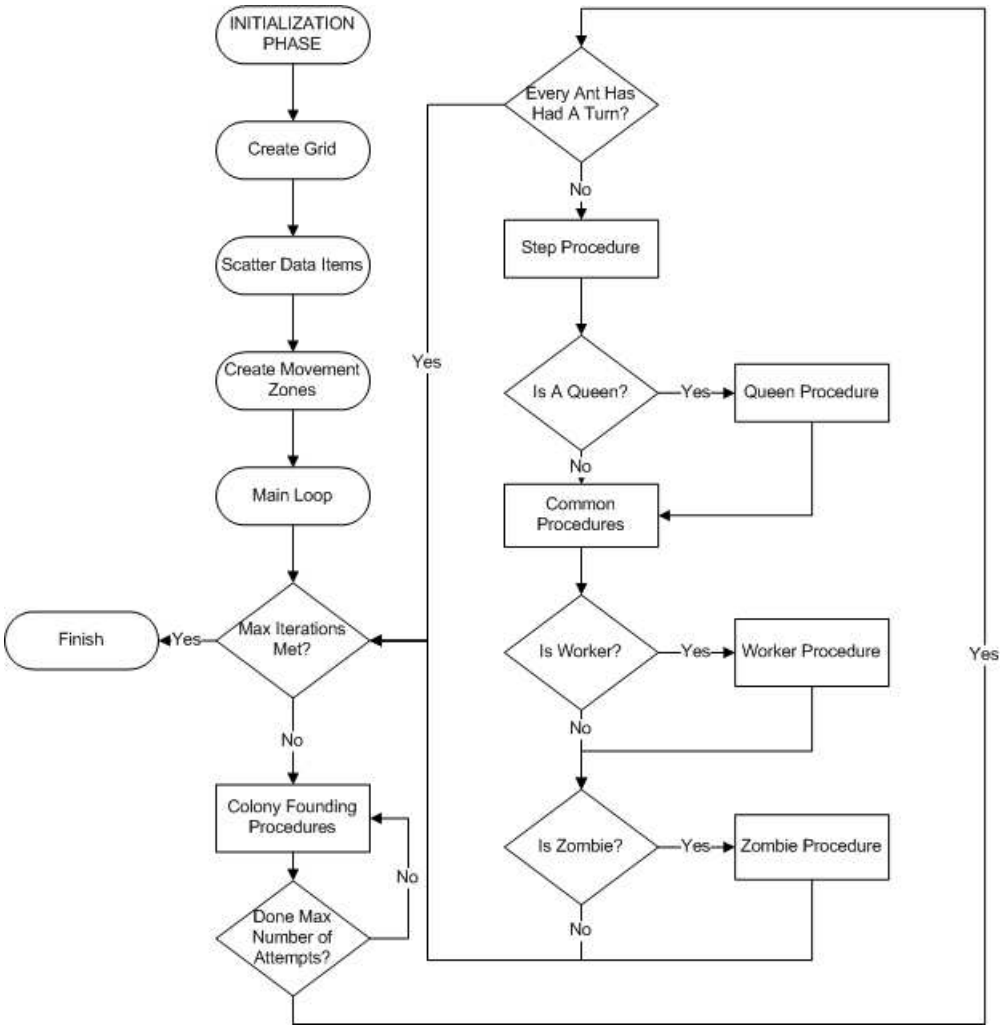

Figure 3.3. Pseudo-Hierarchical Ant-Based Clustering Algorithm.

The algorithm (Figure 3.3 or in Appendix A) starts with an installation phase where a toroidal grid is created with all ant presence values set to zero. The ant presence is the approximate density or level of presence of ants that have been in a particular location on the grid. This is used later to help determine where new queens should be placed and when new workers should be created. The next step

in the algorithm is randomly placing data items on the grid. Zones are then placed around each item with zone radius, r_z. r_z is the maximum number of grid locations away from the location of the item that the grid locations will be marked as inside the movement zone.

Then the main loop begins with the establishment of new colonies through the placement of new queens. In order to ensure that all the items have at least one ant that can move them, the *colony founding procedure* samples the map to find areas in which few, if any, ants have visited in order to ensure that every continuous moment zone has at least one ant. This is achieved by checking the location's presence value which approximates the ant density in a given area. The exact mechanisms are discussed in the Section 3.1.1.

After new queens have been placed, each ant is given a turn to take a step or movement and perform its operations according to its particular role. While the order of the procedures is not mandatory for the algorithm to work, the algorithm is designed with the particular order in mind, so the effects of changing the order are unknown. The following should provide the general logic to understand why this order is chosen. The step procedure allows the ant to move to a new location so that it can do its work in the new location. Next, the queen procedure is executed if the ant is a queen. The procedure maintains all the connections among the queens and is responsible for the movement of items through the connections. The queen procedure is executed before the common procedures in order to allow items a chance to be put through a connection before they are used to reorganize the current cluster in the common procedures. The common procedures are executed by all ants and are based on the basic algorithms in Chapter 2. They are primarily responsible for picking up and dropping of items on the map, but have been augmented to provide efficiency and support for the connections of the queens. Zombies only use this time to attempt to

drop objects. The worker procedure and zombie procedure then deal with conditions where worker and zombie ants, respectively, have live past their time allotments.

3.1.1 Colony Founding Procedure

The *colony founding procedure*'s (see Figure 3.4 or Algorithm 3 in Appendix A) purpose is to ensure that all items have access to ants that can cluster them. In general it does this by finding items on the map whose ant presence is low and then placing a queen on them. The number of attempts to add a queen, which is determined by *Max_Attemps*, (refer to Figure 3.3) is dynamically determined by the number of ants, and is capped by the number of objects needed to be sampled in order to minimize the probability of missing an item with little or no presence around it. See Equations 3.1 and 3.2

$$Max\_Attemps = min\left(A_n, O_m\right) \tag{3.1}$$

$$O_m = \frac{log\left(O_n - 1\right) - log\left(O_n\right)}{log\left(P_n\right)} \tag{3.2}$$

$$P_n = \left(1 - \frac{1}{O_n}\right)^{O_m} \tag{3.3}$$

where A_n is the current total number ants, O_n is the total number of items, and O_m is the number of items that have to be sampled in order to ensure that the probability of missing an item per iteration is P_m. Equation 3.3 is used to derive Equation 3.2 and shows that P_m represents the probability of missing one item out of O_n items, O_m times in a row. So the items must be sampled O_m times in a row in order to ensure that the probably of not sampling all the items is P_m. Since the map begins with no ant presence, a queen ant will be very likely to be placed within the first few attempts by the colony founding procedure. So the use of A_n is used to keep

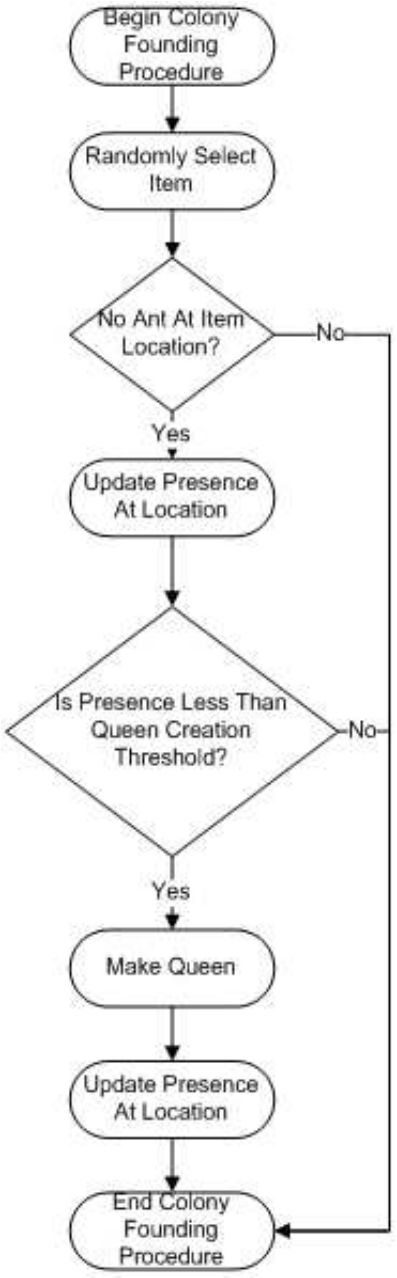

Figure 3.4. Colony Founding Procedure.

the number of attempts directly proportional to the number of ants for the initial iterations. This allows queens a chance to establish a presence in their areas before more queens are added, thereby lessening the chance of a cluster having too many queens initially. Figure 3.5 shows how the queen updates presence values as it moves around which inhibits the placement of a new queen and shows how new queens are placed in areas that have low presence values.

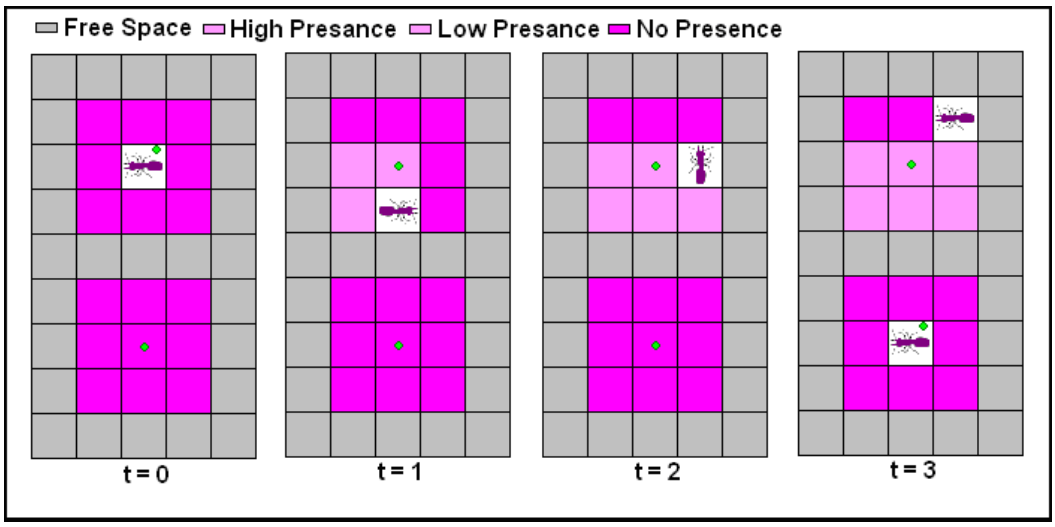

Figure 3.5. Update of Presence Values by an Ant.

First, the algorithm will select a random item from all the items and if there is no ant in that location then the presence of that location will be updated in accordance with the following equation:

$$p_{(x,y)}(t_i) = h_{(x,y)}(t_i) + (\gamma_p)^k p_{(x,y)}(t_{i-1}) \tag{3.4}$$

where $p_{(x,y)}(t_i)$ is the given presence value of the location (x, y) at a given iteration t_i. t_{i-1} denotes the last iteration in which the presence for that location was updated. k is the number of iterations since the last update where $k = t_i - t_{i-1}$. γ_p is the

given discount factor or rate of decay. Function h is the amount that the location's presence value is modified by and is given by the function:

$$h_{(x,y)}(t_i) = \begin{cases} \frac{1}{p} & \text{if an ant is present at location } (x,y) \text{ at iteration } t_i \\ 0 & \text{if location } (x,y) \text{ is empty at iteration } t_i \end{cases} \tag{3.5}$$

where $\frac{1}{p}$ is the normalized contribution an ant would make if it were present in that location. p then represents the maximum presence value obtained if an ant would be present in each iteration and serves to normalize the presence value to a value between 0 and 1. p is calculated by the following equation:

$$p = \sum_{t=0}^{\infty} \gamma_p^t(1) = \frac{1}{1-\gamma_p} \tag{3.6}$$

so

$$\frac{1}{p} = 1 - \gamma_p \tag{3.7}$$

The presence value of a location is analogous to ant pheromone trails used by real ants, but in this case is not used as a trail for an ant to follow; rather it is used as a record of the approximate density of ants around a particular location.

After the presence value of a location is updated, it is compared with the queen creation threshold. The queen creation threshold is determined by the equation:

$$queen\_creation\_threshold = \frac{1}{p}\gamma_{p^{E_Q}} \tag{3.8}$$

where E_Q is the maximum number of iterations that a location can go without an ant being present before a new queen should be placed in it. If the presence value is less than this threshold then a new queen will be placed in that location and the

presence value will be updated again to denote that a new ant has been placed in that location.

3.1.2 Step Procedure

The *step procedure* (see Figure 3.6 or Algorithm 4 in Appendix A) deals with the various obstacles that can get in the way of an ant as it moves, but the actual movement of the ant is handled by the movement procedure that will be addressed in Section 3.1.2.1. First, a random location is chosen exactly one unit away from the ant in either the horizontal, vertical, or diagonal directions. If the proposed location is within the movement zone then the ant will attempt to move to that location. If a location is unoccupied then the ant will move into that location, otherwise the ant must battle the ant in that location to determine who gets to have that location. In cases of a stalemate the ant will remain in its current location and its stagnation count will not be reset because the ant has not moved. The stagnation count will be incremented before it is checked.

If the ant attempts to move into a location that is not in a movement zone the ant will update the border of the movement zone. When items are picked up, the movement zones around them are not automatically collapsed. Instead, the zones are collapsed when the ant bumps into the boundaries. When the ant attempts to move into a location that is not in the movement zone (see Figure 3.7), it checks to see if there is an item within a radius, r_z, to support a movement zone in the ant's location. If the ant cannot find an item then it will mark the location as outside the movement zone, otherwise it will update the presence value of the location to record that it was at that location.

Ants occasionally fail to move several times in a row. This usually happens when an ant gets trapped outside a movement zone. s is the maximum number of

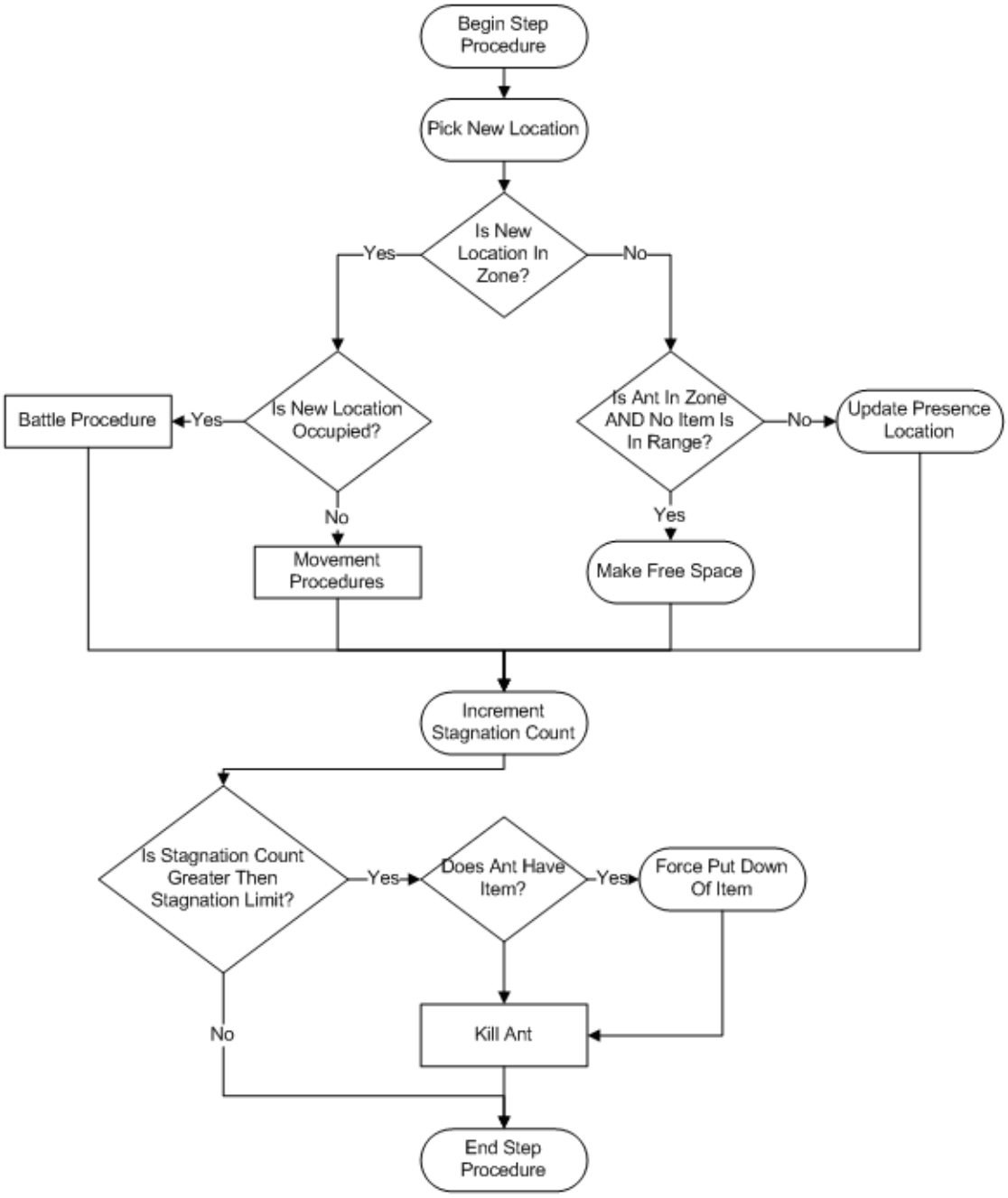

Figure 3.6. Step Procedure.

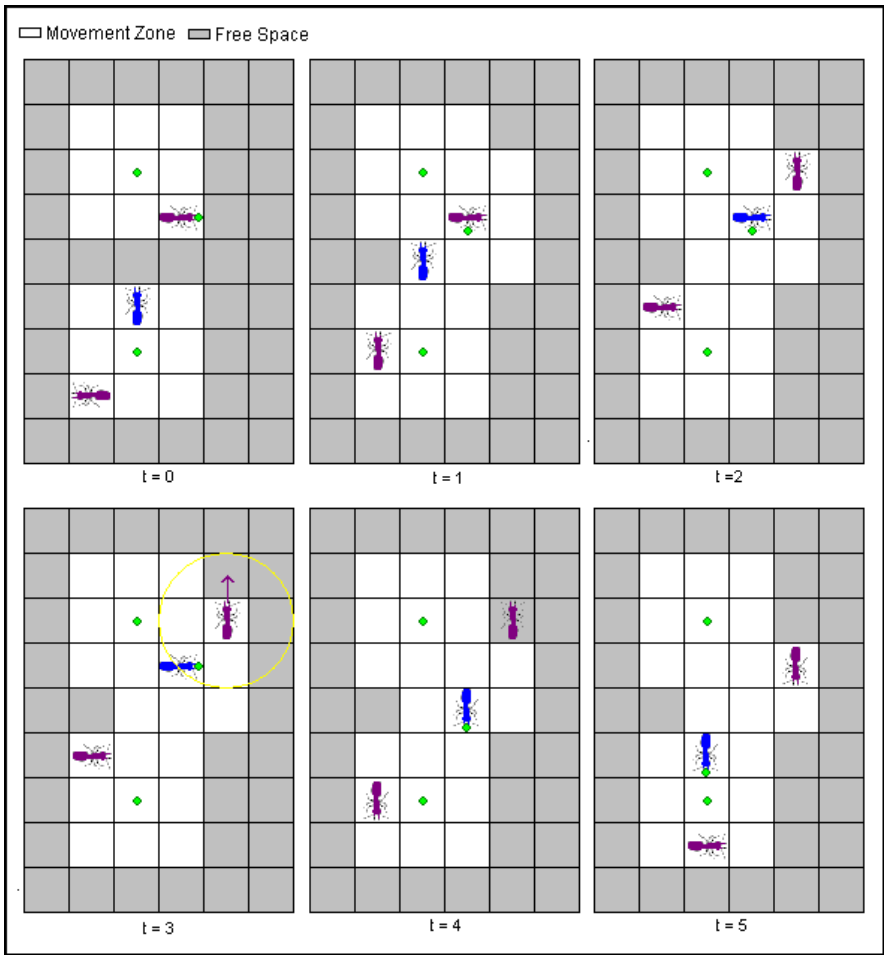

Figure 3.7. Updating Movment Zone Boundaries.

times an ant will be allowed to remain stagnant (to remain in the same location on the grid). If s is too low, ants will be removed for bumping into other ants or the edge a few times in a row, and if s is too large, then processing time is wasted on an ant that does nothing. If the ant has an item when it reaches its maximum stagnation limit, it is forced to put it down by placing it in the nearest available location. This does not include items on a queue. Since this usually happens when the ant is stuck outside the zone, the item is usually placed where the ant is. Then the ant is killed by the kill ant procedure.

3.1.2.1 Movement Procedure

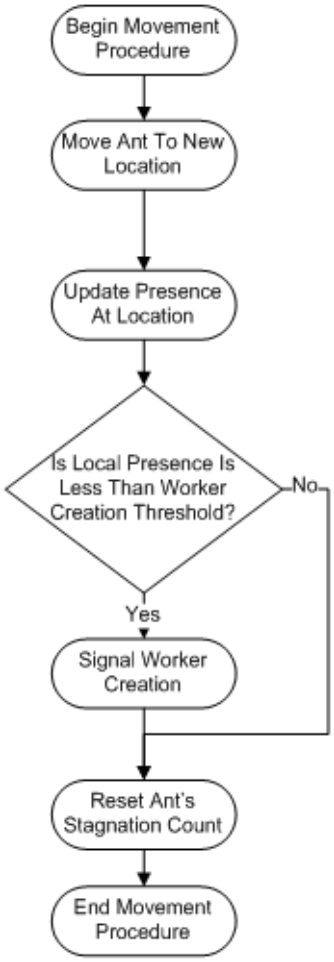

Figure 3.8. Movement Procedure.

While the *movement procedure* (See Figure 3.8 or Algorithm 5 in Appendix A) deals with actual movement of an ant from one location to the next, it also serves the other vital function of regulating the number of ants. After the ant has moved, the

presence value is updated and the ant checks to see if the ant location is below the new worker threshold. The new worker threshold is calculated by:

$$new\_worker\_threshold = \frac{1}{p}\sum_{k=0}^{\infty}\gamma_p^{E_w k} = \frac{1-\gamma_p}{1-\gamma_p^{E_w}} \tag{3.9}$$

where E_w is the minimum number iterations that an ant must not be in a location before the density of ants is considered to be too sparse. If the presence value is less than the new worker threshold, the ant's queen is signaled to make more workers. Figure 3.9 illustrates both the worker and queen discovering locations with low presence and the queen creating new workers in response. Next, the ant's stagnation count is reset in order to indicate that the ant has moved.

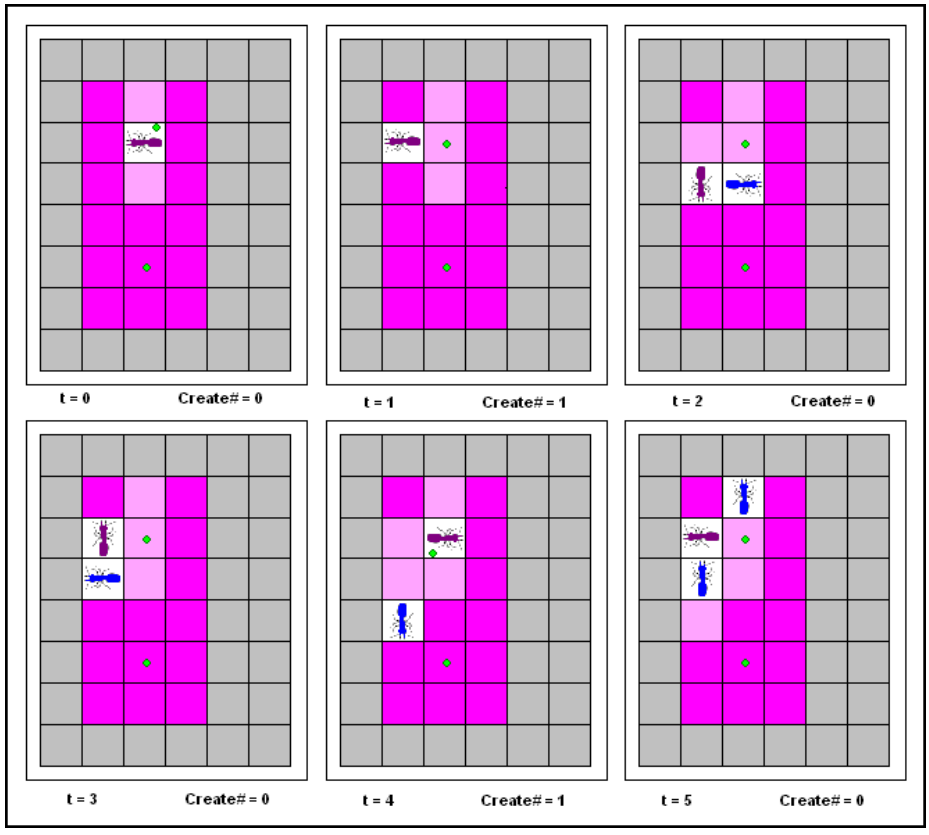

Figure 3.9. Creating New Workers in Response to Low Presence Detection.

3.1.2.2 Battle Procedure

The *battle procedure* (see Figure 3.10 or Algorithm 6 in Appendix A) outlines a protocol for ants to follow when they meet, or, more accurately, when one ant attempts to move into another ant's location. The reason ants battle is to help establish one colony over another in a given movement area. For ease of explanation, the procedure is written from the aggressor's point of view, so for the remainder of the section "attacker" will refer to the ant attempting to take the location, and "defender" will refer to the one whose location may be taken over.

In order for two ants to battle neither can be a zombie. Ants are zombies because they have died in battle or have exceeded their lifespan, so they no longer represent a cluster. There is little reason to have a battle if either one is a zombie because they do not aid in establishing one colony over another. Furthermore if two ants have the same queen, they are in the same colony, therefore they must have different queens in order to battle. In order to decide which ant wins, their respective queen's strengths (number of iterations new workers live) are compared. If the attacker's queen's strength is less than the defender's queen's strength then the attacker loses and the defender wins; otherwise the attacker wins. The strictly less than condition gives the advantage to the attacker in case of a tie. If the attacker loses, it is killed and the defender's queen's lifespan is increased. Likewise, if the defender loses, it is killed and the attacker's queen's strength is increased. The attacker would then be moved into the defender's former location if the defender is not turned into a zombie in which case the location is still occupied. Note that increasing a queen's strength increases a new worker's lifespan which allows them to travel farther from the queen and thereby increasing the area of influence of the colony.

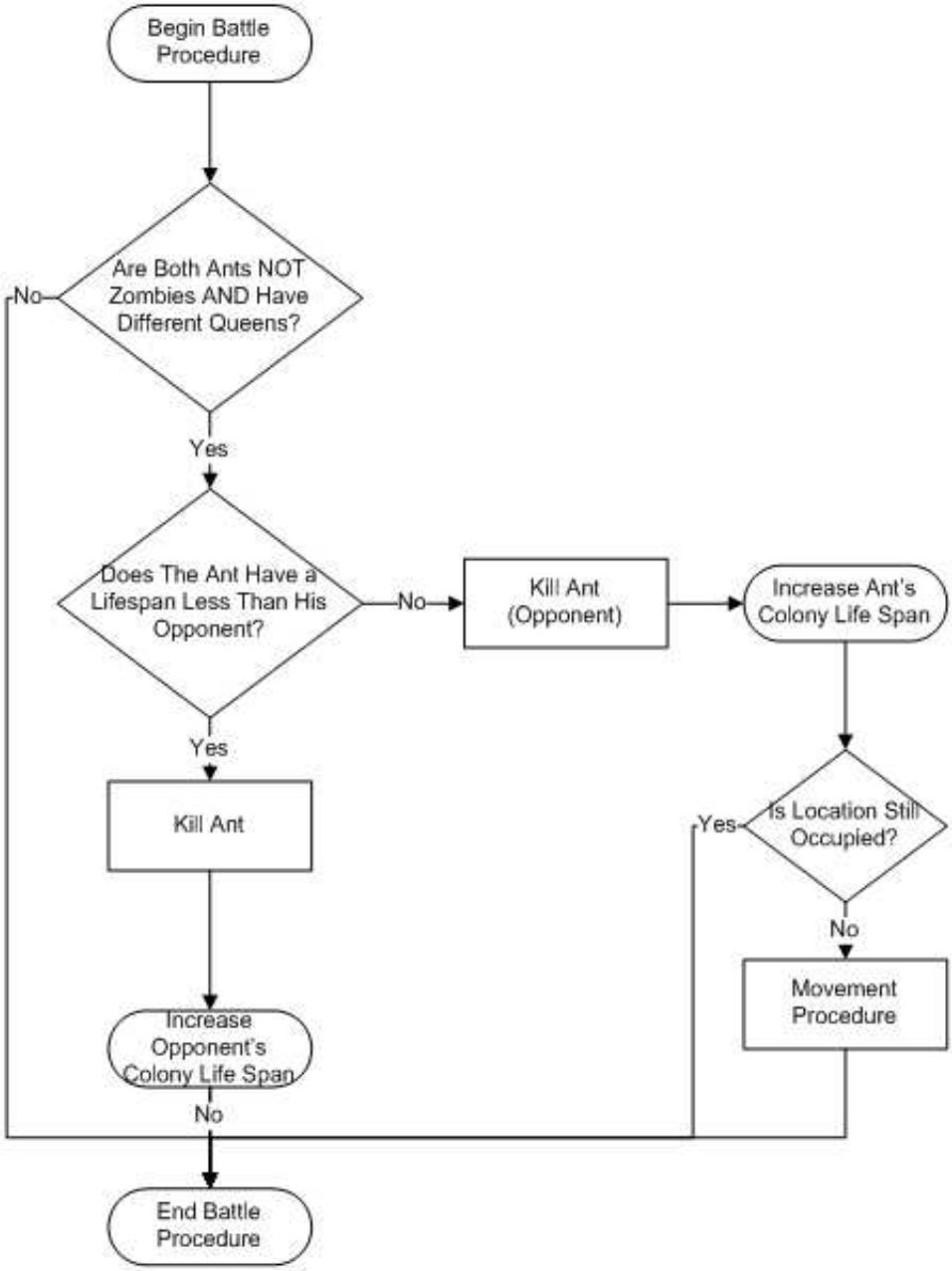

Figure 3.10. Battle Procedure.

3.1.2.3 Kill Ant Procedure

The kill ant procedure (see Figure 3.11 or Algorithm 7 in Appendix A) deals with all the conditions related to removing an ant from the map. Ants with items, however, cannot simply be removed from the map, especially if the ant is a queen with several items on her queue, so the first step of the procedure is to see whether or not the ant is a queen. Then, if the queen does not carry an item, one is taken from the queue if one exists. If a queen was killed in battle, then it had to have been killed by a member of a stronger (having a longer lifespan) colony. So if the queen of the rival colony is still alive the queue is transferred to the rival queen's queue. Since the rival queen has influence in the same cluster, the items were taken from the same cluster and they should then fit equally into the giving queen's queue. After this, if the ant is at the entrance of a connection, its connection is broken. The queens at the entrance end check to see if their counterpart is still alive before attempting to move items through, so there is no need for an ant at the exit to break a connection.

After any queen related issues are dealt with, there remains one key question: does the ant have an item or not? If the ant does not, the ant is removed from the map no matter what cast it belongs to. If the ant still has an item then its cast is changed to zombie. This change in designation has the effect of stripping the ant of every ability except to put an item down, and for the act of putting items down it will always have a probability of one. A zombie is not removed from the map for any reason until it has put the items it has down.

3.1.3 Queen Procedure

The *queen procedure* (see Figure 3.11 or Algorithm 8 from Appendix A) deals with the long distance transport of items among queen ants. The first act of a

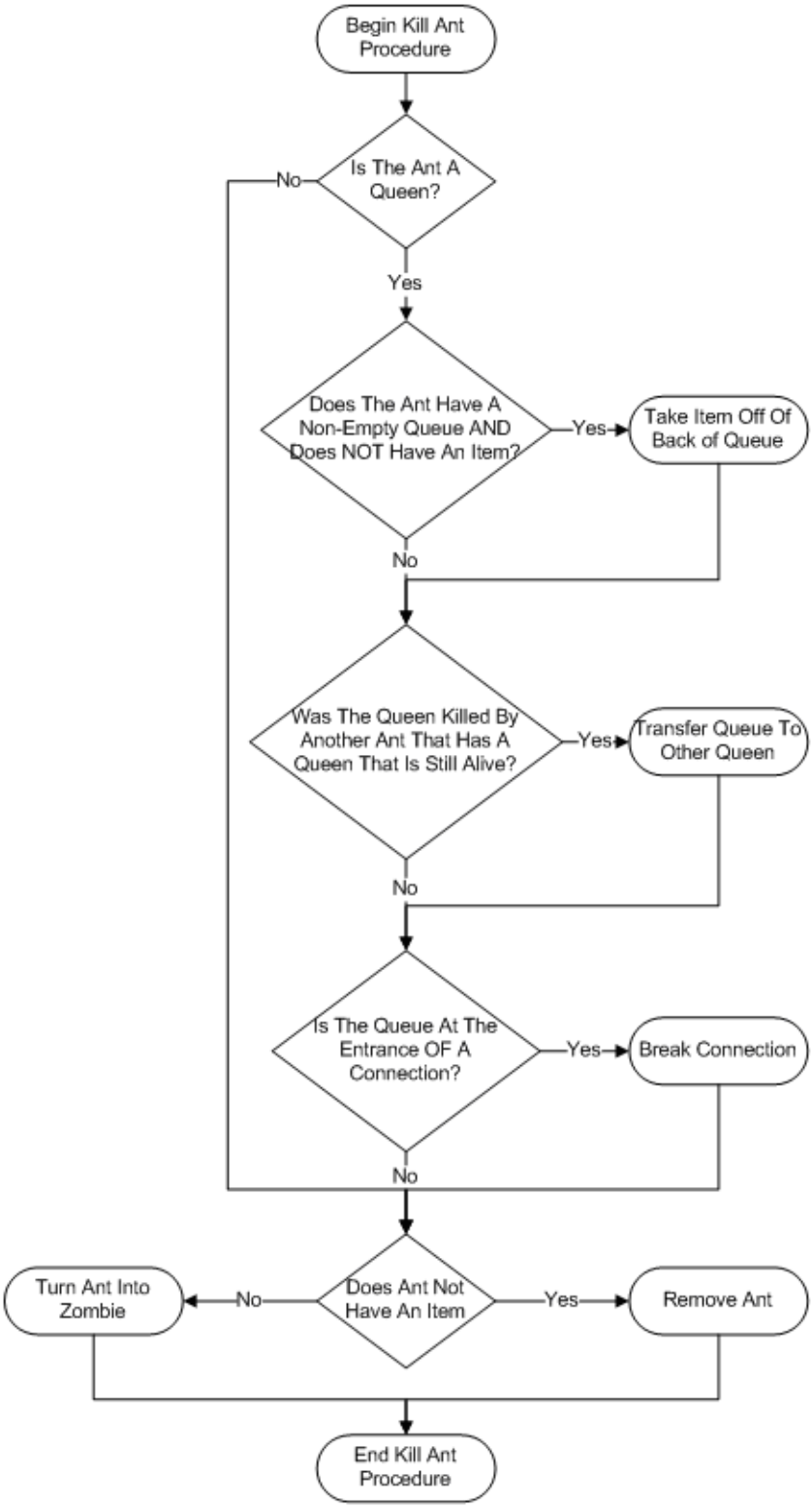

Figure 3.11. Kill Ant Procedure.

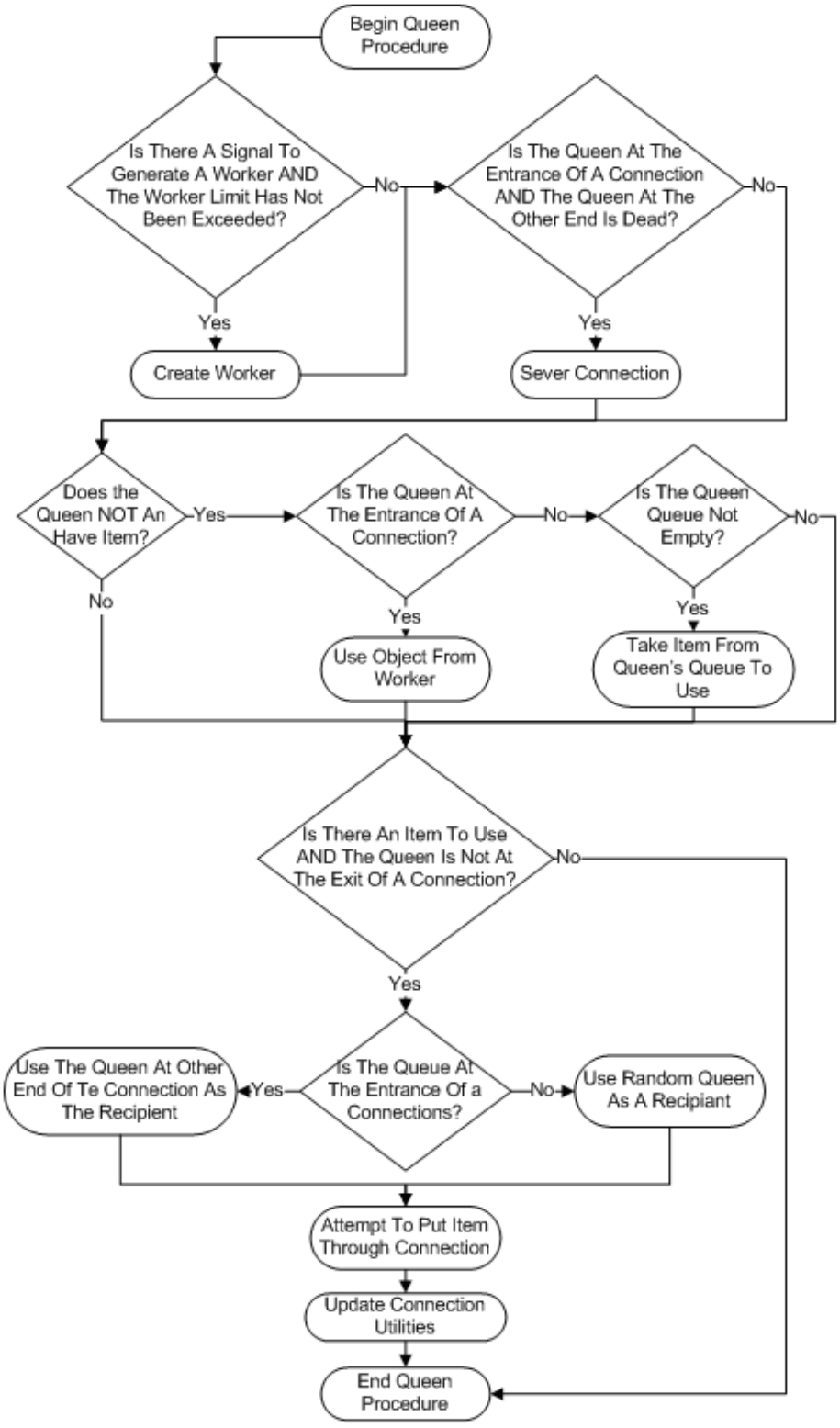

Figure 3.12. Queen Procedure.

queen, however, is to create a worker if it has been signaled to do so (see Section 3.1.2). For the implementation used to calculate the results in Chapter 4 the signal is actually a counter that keeps track of how many ants the queen needs to create and is decremented every time an ant is created. No more than one ant is created per queen per iteration.

After workers are created, queens that are at the entrance end of a connection check to see if the queen at the exit end is still alive. If not, the connection is severed and the queen goes back to randomly choosing another queen to establish a connection to. The next task for the queen is to find an item to be considered for passage through the connection. If the queen already has one then that item is use. If it does not and the queen is at the entrance of a connection then a worker is selected, and if it has an item, then that item is the one used. Note that the item is not taken from the worker unless it is accepted to be passed through the connection. In the implementation used for the experiments, each worker notifies the queen when it has picked up an item (as seen in the Section 3.1.4), and the worker selected is simply the last worker to notify the queen that it has picked up an item, and, therefore, the most likely to still carry one.

It is important to note that a queen at the entrance of a connection never takes an item off its own queue. As seen in Section 3.1.4 workers periodically take items off the queue and only put items dissimilar to the cluster on it. Over time this has the net effect of leaving only items on the queue that are dissimilar to the queen's cluster. Workers who take items off the queue do not notify the queen so it is unlikely that they will be the last worker to have checked in with the queen. When a semi-permanent connection has been established, a large number of the cluster's items are likely to be put through. Eventually, if the connection lasts long enough, all the

similar items will be exhausted, leaving the queen with the task of creating a cluster out of the remaining items or finding new clusters for them.

If the queen does not carry an item and is not at the entrance of a connection, it takes an item off the back of its queue if there is one (a queen's queue is separate from the item that a queen holds). The items at the back of a queue are the most recent additions to the queue. Items added to the back are closer to the spot where the queen acquired them and therefore more probable that they belong in the spot where their queen is, so queens will always take items from the back of the queue. Since items on the queue are likely to be dissimilar to the cluster and the queen would be randomly picking a new queen to establish a connection to, this creates a random search mechanism that is analogous to an ant wandering around the map in other ant algorithms except the attempts to place items in different clusters are taking place in cluster space[1], thus saving the ant from having to cross empty space to try to deposit in a new cluster. If a queen is at the exit end of a connection, or if no item is found to pass through a connection, then the queen's duties as queen are done. If there is an item to pass through a connection, then an attempt is made to pass it through. If a semi-permanent connection exists, the attempt is to pass the item to the queen at the other end of it. If not, then the queen attempts to establish a connection to a random queen and tries to pass the item to the corresponding queen. The decision of passing an item through a connection is the same as determining whether or not to drop it at the location at the other end of the connection. Therefore the probabilities that are used to calculate the probability of dropping (see Equation 3.14) are the same as those that calculate the probability of an item passing through a connection to other queens (see Equation 3.10). The only difference is that the neighborhood used for the

[1]Clusters space is a space where every location represents a cluster and any locations can be reached from any other location in a single move.

calculation is that of the queen that the item is being passed to, and not of the ant actually holding the item. In particular, the probability of an item to be dropped is:

$$p^{\#}_{drop}(i) = \left(\frac{f(i)^{\#}}{k^{-} + f(i)^{\#}} \right)^{2} \tag{3.10}$$

where $f(i)^{\#}$ is the similarity of the neighborhood of the ant at the exit end of a connection to item i which is being held by the ant at the entrance end of the same connection.

The probably that is calculated to determine whether to pass an item through are used to determine its utility. Equation 3.11 below is used to stochastically calculate the utility of a particular connection, c:

$$u_c(c_i) = p^{\#}_{pick}(i) + \gamma_u u_c(c_{i-1}) \tag{3.11}$$

where c_i is the i_{th} update of the connection, c,where $u_c(c_0) = 0$ and γ_u is the utility decay constant. After a particular connection's utility is updated, the average utility of all connections is updated according to Equation (3.12):

$$u_g\left(u_c(c_i)_j\right) = \begin{cases} \frac{u_c(c_i)_j - u_c(c_{i-1})_j}{c_n} + u_g\left(u_c(c_i)_{j-1}\right) & \text{if no new connection is made} \\ \frac{u_c(c_i)_j - u_c(c_{i-1})_j + u_g\left(u_c(c_i)_{j-1}\right) \cdot c_n}{c_{n+1}} & \text{if a new connection is made} \end{cases} \tag{3.12}$$

where c_n is the current number of connections, $u_c(c_i)_j$ is the most recently updated connection's utility, $u_c(c_i)_{j-1}$ is the utility of the pervious connection that was updated, and $c_{n+1} = c_n + 1$. To keep c_n from constantly increasing, connections in the implementation are recycled. This means that if a queen dies all its connections (semi-permanent or not) are returned to a pool of connections. If any connections

are still in the pool, then the utility is reset, meaning $u_c(c_i) = u_c(c_1)$ while $u_c(c_{i-1})$ is the utility of the connection when it was put into the pool.

As previously mentioned queens have access to all other queens on the map, and if a connection between two queens reaches a high enough utility then it becomes a semi-permanent one-way connection among the queens, and likewise is downgraded to a regular connection if the utility drops. For a queen at the entrance end of a semi-permanent connection this means it will always attempt to pass an object to the queen at the exit end of the connection instead of randomly choosing a queen. For a queen at the exit end of a semi-permanent connection this means that it will not try to pass objects to any other queen. The queen at the exit end of a connection only accepts items. All incoming items from other queens are placed on the back of the queen's queue as illustrated by Figure 3.13. Note that in an effort to simplify the algorithm, the item a queen carries is separate from those that are on her queue. All ants in a colony have access to the queen's queue.

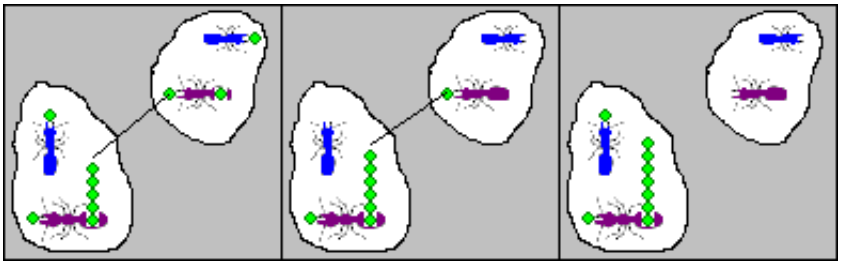

Figure 3.13. Placement of Items as They Are Passed Through a Connection.

3.1.4 Common Procedure

The common procedure (see Figure 3.14 or Algorithm 9 in Appendix A) is responsible for maintaining the cluster itself. This section is closely related to the

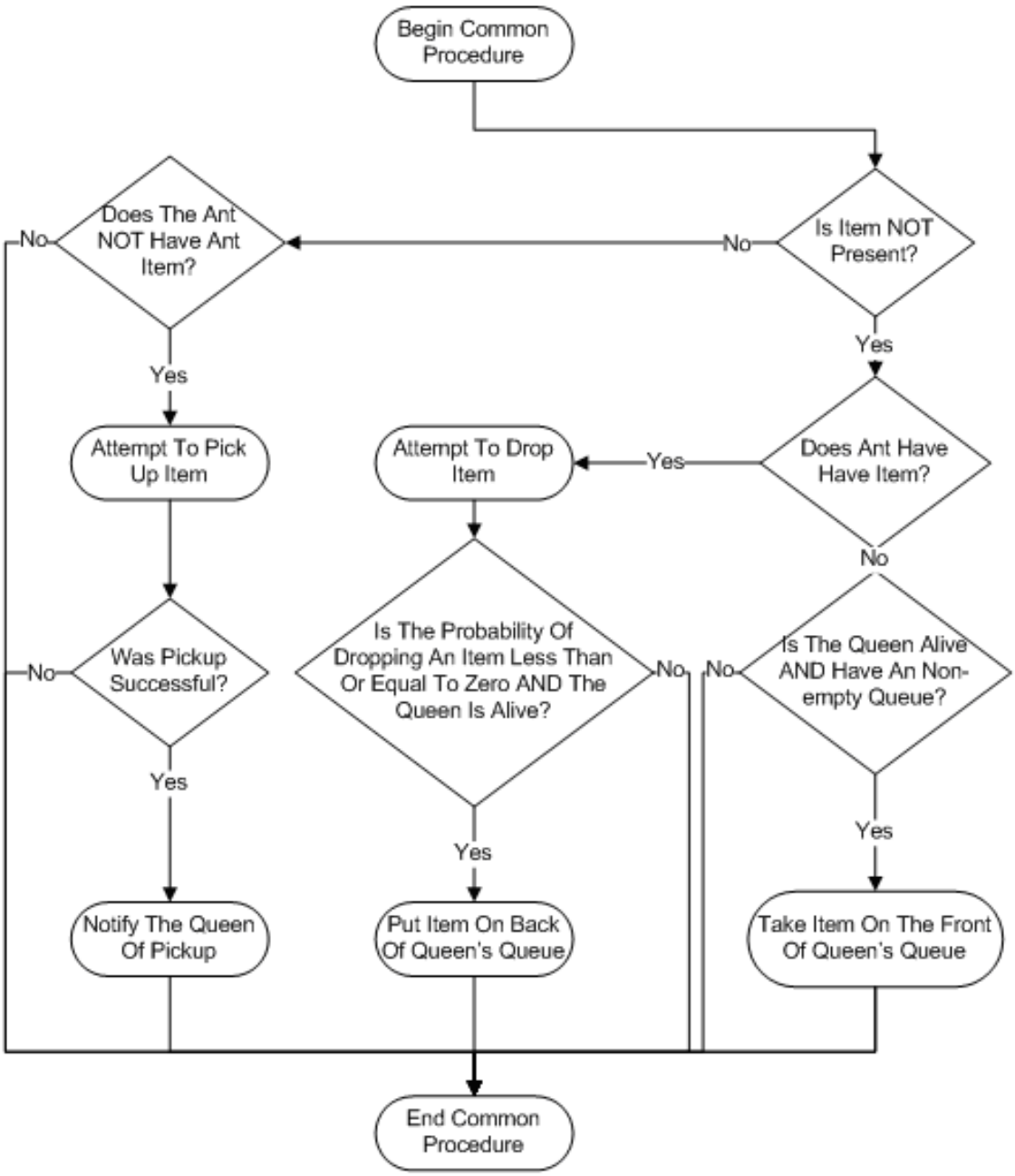

Figure 3.14. Common Procedure.

basic algorithms for ant clustering found in Chapter 2 but has been augmented to provide efficiency for the connections among the queens. The significance of the portions that relate to the connections is commented upon in Section 3.1.3. It is important to note that these procedures still retain the ability to split clusters if multiple homogenous groups with a significant number of members are in a single cluster. Also, the other algorithms used their basic algorithm to organize the whole grid while this algorithm only uses it to organize the cluster and to weed out items that are dissimilar to the cluster so that they can be removed by means of a connection among the queens.

The first action an ant takes in the common procedures is to determine if an item is not present in the location of the ant. If an item is present and the ant does not have an item then it will attempt to pick up the item off the map. The probability of picking up an item is determined by Deneuborg et al.s [9] threshold formula (also see Equation 2.1 in Chapter 2):

$$p_{pick}(i) = \left(\frac{k^+}{k^+ + f(i)}\right)^2 \tag{3.13}$$

where $k^+ = .3$ and $f(i)$ is the neighborhood function (see Equation 2.3 in Chapter 2). If the ant is successful then it will notify the queen of the pickup.

If an item is not present in a location and the ant has an item then it will attempt to drop it based upon Deneuborg et al.s threshold formula (also see Equation 2.2 in Chapter 2):

$$p_{drop}(i) = \begin{cases} \left(\frac{f(i)}{k^- + f(i)}\right)^2 & \text{if the ant is not a zombie} \\ 1 & \text{if the ant is a zombie} \end{cases} \tag{3.14}$$

where $k^+ = .1$ and $f(i)$ is the neighborhood function (see Equation 2.3 in Chapter 2). Zombies only try to find locations for items so their drop probability is always 1. If it is found that the portability to drop an item is zero, $p_{drop}(i) = 0$, then the item is placed onto the back of the queen's queue. $p_{drop}(i) = 0$ occurs only when the neighborhood in which it is found has no similarity to the item, $f(i)$, and is therefore very unlikely to belong in the cluster. As a result it is put onto the queen's queue so that the item can possibly be place in another cluster by the queen.

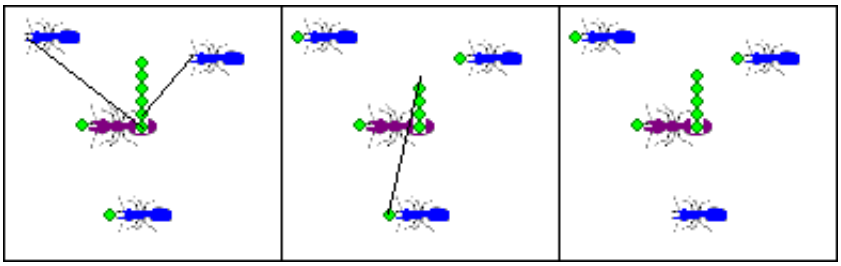

Figure 3.15. Workers Moving Items Off and On the Queue.

If an item does not occupy the location of the ant and the ant does not have an item, then the ant will take an item off the front of its queen's queue if the queen is still alive and there is an item on the queue (see Figure 3.15). Items in the front of a queue are the least recently added, so items at the front end of a queue are probabilistically the furthest from where the queen acquired them. Since they are the least associated with the queen's current location, worker ants take items from the front of the queue as opposed to the back of the queue. By taking an item off the queue instead of only using its turn to simply move as done in Lumer et al.'s basic algorithm [10], the ant helps to disperse items that its queen has on its queue, especially if the queen is at the exit end of a connection. If the queen is not at the exit end of a connection then the ant acts as a filter for the queue as it takes items

off the queue, puts them back on if they are dissimilar, and tries to find a location for them if there is any relative similarity. Thus similar items to the cluster tend to stay off the queue and dissimilar ones tend to remain on the queue.

3.1.5 Worker Procedure

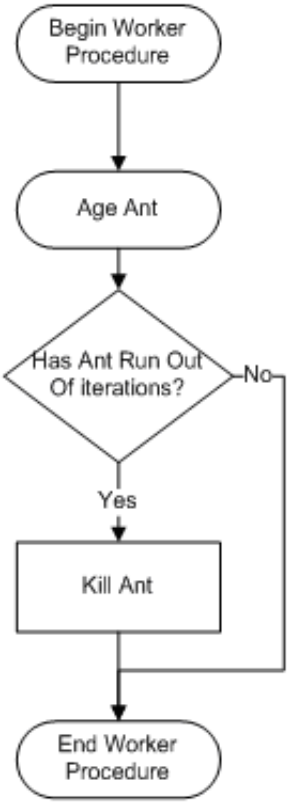

Figure 3.16. Worker Procedure.

The *worker procedure* (see Figure 3.16 or algorithm 10 in Appendix A) is responsible for removing workers that have used up all of their allotted iterations. First the ant is "aged," meaning that the number of iterations that it is still to be run has to be decremented. Next the ant is checked to see if it has run out of iterations, and if it has run out of iterations the ant is killed.

The main reason for killing workers is to keep the colony's influence in the same movement zone as the queen. Workers could wander and become trapped in a different movement zone than the queen. That would cause the queen to have workers in more than one movement zone at a time. In order for the queen's queue to work properly, the work must be in the same movement zone, or cluster, so workers must not be allow to exist indefinitely outside the cluster that the queen is in.

3.1.6 Zombie Procedure

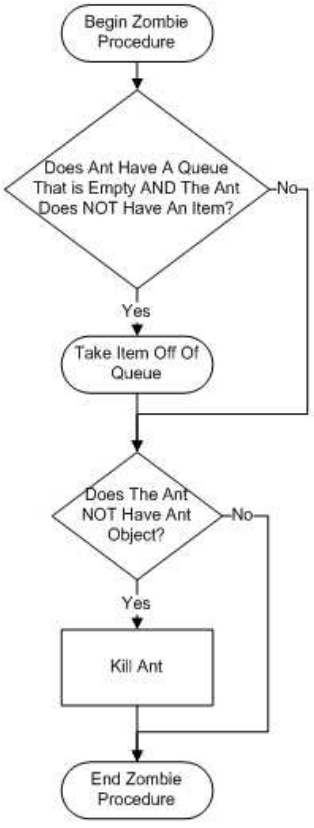

Figure 3.17. Zombie Procedure.

The *zombie procedure* (see Figure 3.17 or Algorithm 11 in Appendix A) deals with the discarding of zombies after they have put down all of their items. The sole

purpose for a zombie is to find an empty location to place an item in, so there is no longer a need for the zombie to continue on the map after it has deposited its item. There are two types of zombies: ones with a queue and ones without a queue. Zombies with queues are queens that were killed with non-empty queues that were unable to transfer them to another queen. Zombies without queues are killed workers or killed queens that died with empty queues.

If a zombie does not have an item but does have one in a queue it will take an item off the queue and find a location for it. If, after that, the zombie ant does not have an item, it is killed (see Section 3.11) once and for all.

3.2 Queen Dominance

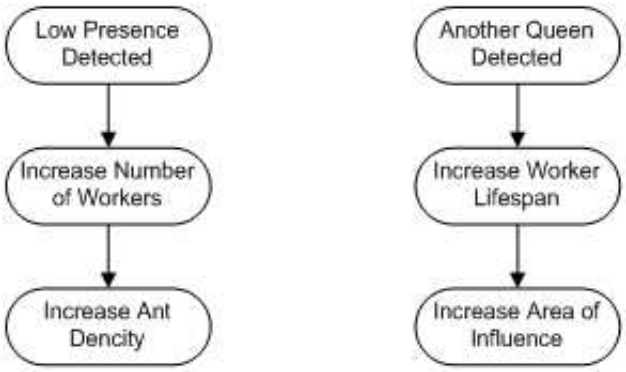

Figure 3.18. Queen Dominance.

One of the key aspects to making the algorithm work is the election of a queen for every movement zone. Since each movement zone with an item in it roughly corresponds to a cluster, once a queen has established dominance, meaning its colony controls the majority of a zone and any member of another colony would be instantly killed, then that queen would act as a representative of that cluster.

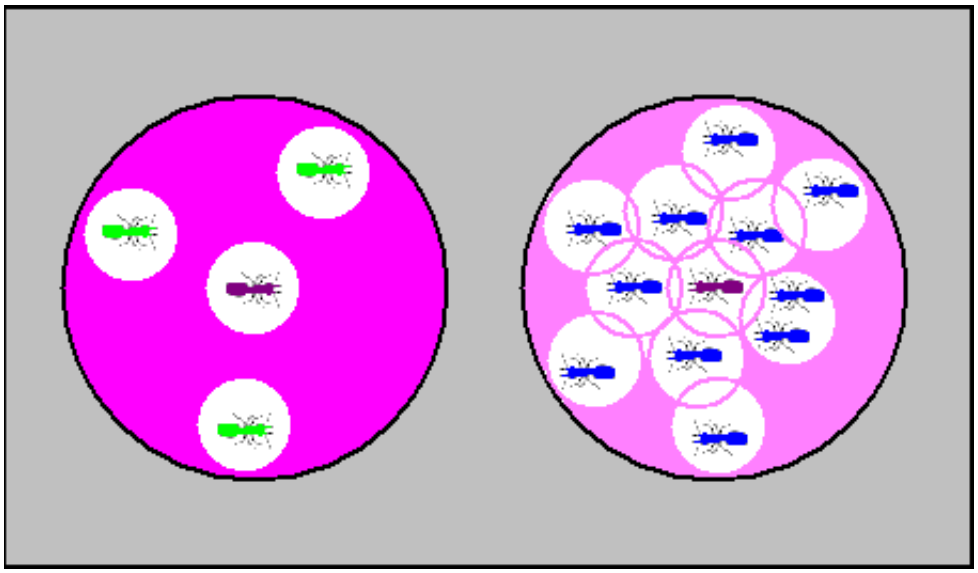

Figure 3.19. Controlling Ant Density.

The queen establishes dominance over a zone by sending out workers to help control and organize a zone. Two parameters help to control how effectively a queen controls a zone. First, if a low presence value is detected (Section 3.1.2.1) then the queen is stimulated to create more workers (Section 3.1.3). If a low presence value is detected, then that means the workers are not dense enough to discourage a new queen from being placed (see the left queen in Figure 3.19) (Section 3.1.1) so more workers need to be created so that the density can be increased (see the right queen in Figure 3.19). The second parameter determines the size of the zone a queen can control. The lifespan of a worker is the number of iterations a worker can live, and since a worker can take only one step at a time, a worker can move no more steps from were it started than a number equal to its lifespan. Since the queen moves at the same time, the total separation of the queen and a given worker can be at most twice the lifespan that the worker was given at the moment of creation (which is the strength of the queen represented by the red circles in Figure 3.20). This would only occur if both travel in separate directions at every step, which would be extremely rare for

larger life spans. Since larger lifespan also allows an ant to backtrack more often it requires an increasing amount of lifespan to increase a queen's radius of influence.

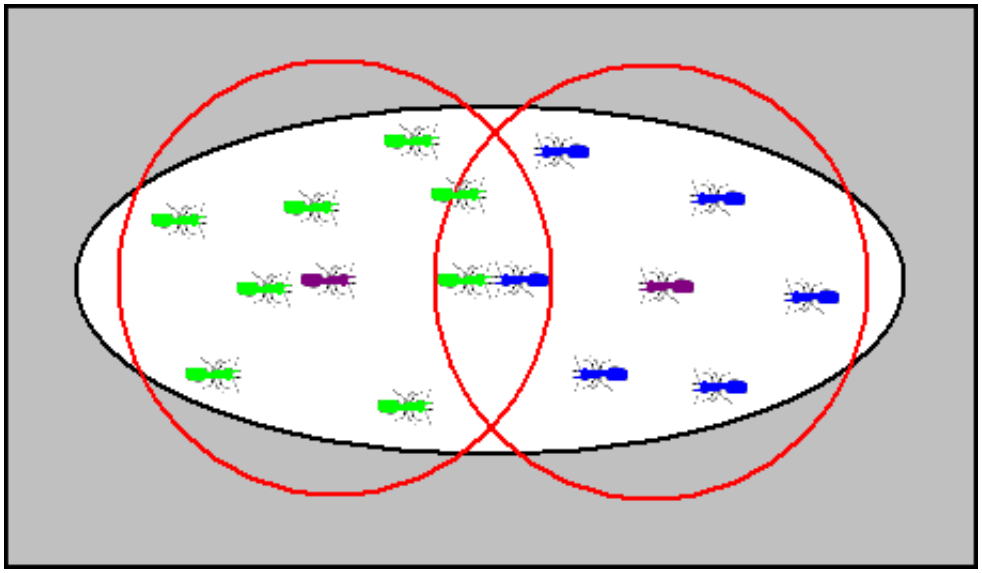

Figure 3.20. Increasing a Queen's Area of Influence.

CHAPTER 4

EXPERIMENTS

To show the increased efficiency of the novel algorithm proposed in Chapter 3, both the new Pseudo-Hierarchical Clustering (PHC) algorithm and a variation of Handle's algorith in Chapter 2 were implemented. Synthetic data is created to test the effects of clustering of different data characteristics including: number of clusters, total number of data elements, and data quality. This chapter covers various aspects pertinent to the experiments and the results of the experiments.

4.1 Implementation

The setup of the implementation of the algorithms consists of three main parts: the data set generator, the simulator, and the interface. All code is written in C++ with Microsoft Visual Studio 2005[1] Standard Edition.

4.1.1 Setup

The data set generator generated all synthetic data with specific properties. All generated data is saved to an external file so that both simulations can cluster over the same data for all clustering runs. More on generating data will be discussed in Section 4.3.

The simulator is responsible for the actual execution of the two algorithms. It maintains all aspects of the algorithms including a toroidal map, ants, data items, and acts upon them as dictated by the algorithms. Due to the radical differences in

[1]Microsoft Visual Studio 2005 Version 8.0.50727.42

the nature of the two algorithms much of the code written for them is only used by its respective algorithm, but whenever possible, sections that are common to both were shared. This includes the toroidal map, data item index, and all functions pertaining to the pick up and drop probabilities. The simulator also has the ability to stop the execution of the algorithms at regular intervals and calculate statistics, including execution time (not including the time it takes to calculate the statistics themselves).

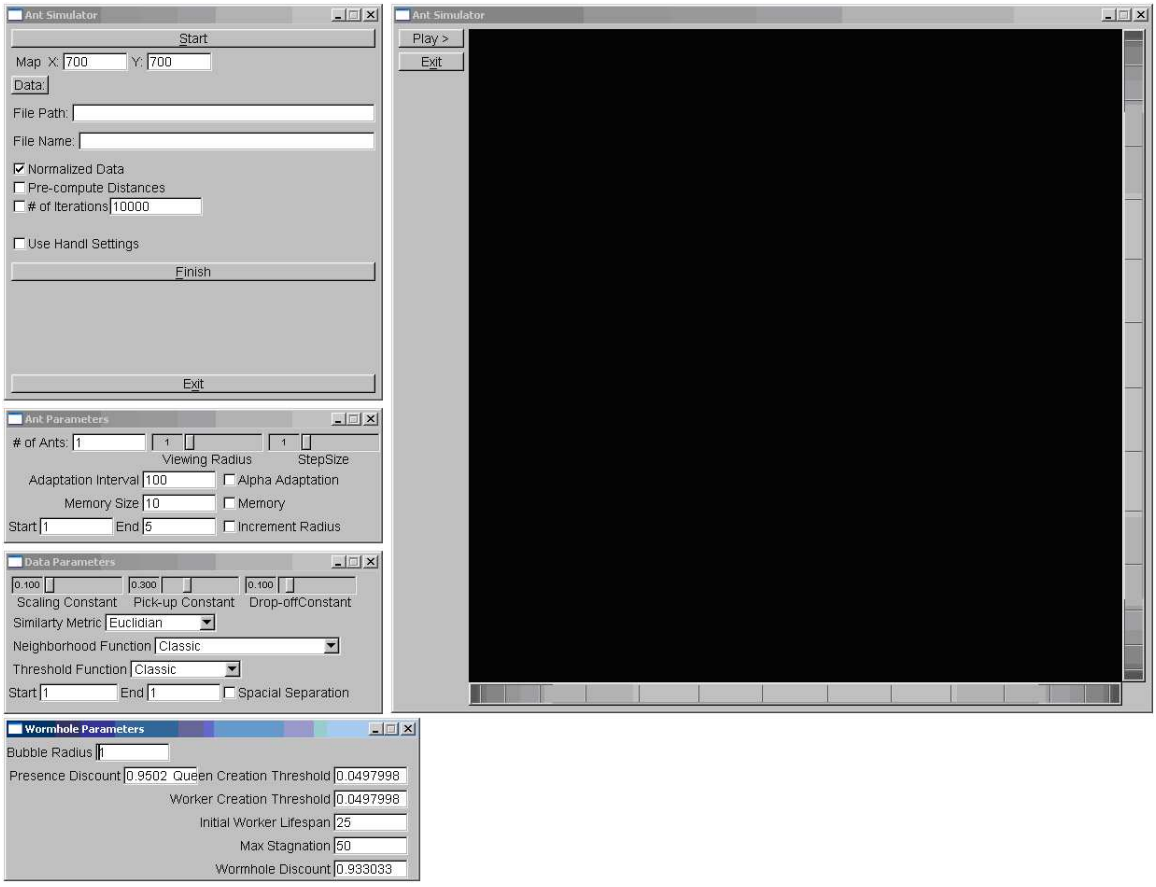

Figure 4.1. GUI Interface.

The GUI (see Figure 4.1) was written with the Fltk[2] graphic libraries so that the interface would be as lightweight as possible to increase the speed of execution for

[2]Fltk version 1.1.7

both testing and demonstration. It allows for easy modification to all the simulator's variables, and for some variables, such at the pick up and drop off constants, the values could be changed dynamically during execution. To lessen its impact on the execution time of simulation, the interface is executed as its own thread separate of the simulator which also allows the simulator to execute while the interface is turned off.

4.1.2 Variation of Handl's Algorithm

While the basic algorithm that Handle used (see Algorithm 1) was used for clustering, not all of the modifications were used in the implementation used for comparison. The selection of the modifications used were an effort to compare the new algorithm against a relatively basic algorithm, while at the same time extending it with the capabilities that are comparable to the proposed algorithm.

One of the most noted omissions from the version of Handl's implementation is the lack of the *alpha-adaptation* extension (discussed in Section 2.6). The alpha-adaptation extension estimates the correct alpha value by monitoring the level of activity of the ants in terms of the number of failed drops. Since ants in the basic algorithm are allowed to move in free space where there are no items, and therefore have a zero probability of dropping the item, the number of failed drops is significantly higher in the basic algorithm than in the proposed PHC. So the rule experimentally derived by Handl to update the alpha would have to be derived again. Furthermore, while alpha-adaptation alleviates the need for selecting alpha, several other factors are introduced including: how to detect a needed change in alpha, the thresholds to increase it and decrease it, and the rate at which to increase and decrease the alpha. So for simplicity and comparability the same alphas where used in both clustering algorithms.

Another omission is that instead of using Handl's derived pickup and drop off probabilities (see Equations 2.7 and 2.8 respectively), the pick up and drop off probabilities used by both algorithms are the same as those used in Handl's description of the basic algorithm, which are the same as those used by Deneubourg in Equations 2.1 and 2.2 respectively. Since the experimentally derived equations were developed for use with all the extensions, removing or modifying any of them would drastically affect the performance, so the classic probably threshold functions were used. Not only the threshold functions are the same in both algorithms but also the neighborhood function is the same (Equation 2.3). The neighborhood function used in Handl's basic algorithm (see Equation 2.6) heavily penalizes dissimilar items which is intended to increase separation of clusters. This was found to be true for relatively small data sets (less than 500 items), but seemed to slow the convergence of the algorithm as the data sets got larger, so the unmodified version is used.

Another modification that was not used was the *interlude with a modified neighborhood function* which is an effort to better separate clusters before final cluster extraction. Experiments with this setting showed that this method proved less effective on larger clusters because of the overhead of the ants having to spread out a large number of items and then having to condense them back into tight clusters. The interlude time proposed by Handl proved to be too short to allow the ants to accomplish this task. Also, in the latter stages of clustering the interlude would sometimes begin before the clusters had converged which dramatically slowed the final convergence. Since this process only marginally aided in the separation of clusters and sometimes slowed convergence this modification is not used.

Another modification that was omitted was the increasing radius of perception. The amount of increase given by Handl often increases the radius too quickly which caused the clustering to stagnate very early.

One of the modifications that is kept is the *short-term memory with 'look ahead.'* The jumping action used in the look-ahead process allows the ants to jump long distances on the map into locations where it is highly likely to drop an item. The long distance aspect of this modification is of key interest of comparison as it is a characteristic feature of the PHC algorithm.

4.1.3 Parameter Settings

All the parameter settings used for both algorithms are displayed in Table B.1 in Appendix B. All parameters were experimentally derived or defied by Handl [12]. It is important to note that map size for the PHC algorithm was not based upon the number of items for the experiments. Since the ants in the PHC algorithm never travel in free space, once the clusters are far enough apart that they are unlikely to collide when drifting, any distances beyond this point have little impact on the performance of the algorithm. Colliding clusters slow the clustering process, so it is best to have an extremely large map with h a ratio of map area to item number of being at least 1000:1. A 700 x 700 map size proves sufficient for all experiments.

4.2 Measures

To evaluate and compare the performance of the proposed ant clustering approach performance was measured in terms of clustering time, the number of clusters found, and the quality of the resulting clusters.

4.2.1 DB measure of Quality

Several measures of clustering found in the literature measure the *quality* of clustering results, each based on different conceptual ideas of what quality is. Good surveys can be found in [20] and [21]. The measure chosen for the comparison of

quality for this thesis is referred to as "DB" measure [22][21]. The "DB" measure is defined by the following equation:

$$DB\left(n\right)=\frac{1}{n}\sum_{i=1}^{n}\left(\max_{j=1,\cdots,nj\neq i}R_{ij}\right) \tag{4.1}$$

where n is the number of clusters,$R_{ij}\frac{S_i+S_j}{d_{ij}}$,$S_i=\frac{1}{n}\sum_{x\in C}d\left(x,c_i\right)$, and $d_{ij}=d\left(c_i,c_j\right)$ where d is the distance in feature space between items. S_i measures the average distance items have from the cluster mean which reflects the spread of the items in the cluster. R_{ij} measures the ratio between the two clusters' spread and the distance between the two clusters' means. So if the distance between two particular clusters is so close that the spreads of the two clusters coincide then the clusters overlap and should be merged into a single cluster. Therefore, $R_{ij}\geq 1$ means that there is a high probability that there are too many clusters on the map and it then follows that $DB\left(n\right)\geq 1$ indicates that it is highly likely that there are clusters that still need to merge on the map.

For the data sets in Section 4.3 produced with specific DB values, the following is a brief explanation of how the DB values effect the data given knowledge of the correct clustering, and the fact that all elements of clusters were generated from a normal distribution and standard deviation of 1. It is important to note that the amount of overlap measured by the DB quality reflects the degree of overlap of clusters, so for high DB values the number of clusters found by the clustering algorithms will be significantly smaller than the number of clusters used to generate the data since the clusters overlap so much that the clustering algorithms interpret them as one cluster.

Clusters of data with a normal distribution will have $S_i = 0.8$[3]. Since about 97% of all elements of a cluster with a normal distribution occur within 2 standard deviations of the mean, two well separated clusters i and j will have a $d_{ij} = 4$. Therefore, values for $R_{ij} \leq 0.4$ are considered well separated clusters. Values $R_{ij} > 0.4$ represent clusters that are increasingly more likely to overlap and merge into a single cluster. Since $DB(n)$ is the maximimum value of R_{ij} among all pairs of clusters ij, values $DB(n) > 0.4$ in test data indicate that the clusters generated have an increasing amount of overlap and are increasingly likely to be clustered as a single cluster.

4.2.2 Clustering Time

While the time of execution of an algorithm is an important measure of an algorithm's performance, the actual time needed is heavily influenced by the implementation of the algorithms, set up of the system under which it is executed, and the efficiency of the compiler with different optimization settings. All experiments here are run on a system with an AMD Atholon 64 X2 Dual Core Processor 4200+ 2.21 GHz with 2.93 GB of RAM. Programs are complied with Microsoft Visual Studio 2005 with optimization set for speed.

4.2.3 Ant Actions

In order to compare relative efficiency of the ants in the two algorithms the number ant actions is counted. This is done in place of counting iterations since the numbers of ants in the PHC algorithm vary in every iteration. An ant action is counted in both algorithms when *i*) an ant takes a turn, *ii*) an ant calculates a drop

[3]0.8 was experimentally derived by averaging over the average distances from the mean of 10 distributions with 1000 elements each. Each distribution has a standard deviation of 1.

probity, and *iii*) an ant calculates a pickup probability. The action counted when the ant takes a turn includes all movements and any other non pick up or drop off related actions. Since queens calculate drop probability when putting items though a wormhole this counts as a separate action.

4.3 Data sets

Table 4.1. Parameter Settings for the Basic and PHC Algorithms

Experiment	Data Set	Number of Clusters	Number of Items	DB Measure within 0.05
1	1	4	360	0.1
	2	4	360	0.2
	3	4	360	0.3
	4	4	360	0.4
	5	4	360	0.5
2	6	4	120	0.2
	7	4	240	0.2
	8	4	360	0.2
	9	4	480	0.2
	10	4	600	0.2
3	11	2	360	0.2
	12	3	360	0.2
	13	4	360	0.2
	14	5	360	0.2
	15	6	360	0.2

The data sets derived for testing are designed to test the algorithm's ability to handle varying cluster sizes, numbers of elements, and qualities of data. To create the data sets, means of clusters were selected according to a normal distribution with a standard deviation starting at 4. The individual cluster elements are then randomly selected around each mean according to a normal distribution with a standard devi-

ation of 1. For all experiments, the number of elements per cluster is the same. The data set is then tested for quality according to the DB measure. If the quality of the set falls within ε of the desired quality value then the set of clusters is accepted and stored in a file so that both algorithms will be tested on the exact same data.

Table 4.1 summarizes the data sets synthesized for the experiments. To measure the performance of each algorithm, two data parameters are held to a representative constant while the third varies over a range around that constant. Five sets of data are produces for each particular setting of data parameters.

4.4 Cluster Retrieval

Extracting clusters from the map during and after clustering is not a trivial matter. While clusters are visually obvious displayed on a two dimensional grid, systemically deriving the number of clusters and which data items belong in each section proves to be a relatively time consuming task.

For this thesis a single link hierarchical clustering algorithm is used to extract clusters over the 2D map [12]. Each element begins as its own cluster and for each iteration of the algorithm the two clusters that are closest together in terms of location on the grid are merged together. The process repeats until the stopping criterion is met.

For a given pair of clusters, C_1 and C_2 on the grid and, without loss of generality, $|C_1| \leq |C_2|$ the spatial distance is defined on the grid as:

$$weighted\_singlelink\,(C_1, C_2) = singlelink\,(C_1, C_2) \cdot weight\,(C_1, C_2) \tag{4.2}$$

where $singlelink\,(C_1, C_2)$ is the standard linkage metric defined by the single link algorithm. This is equal to the minimum distance between all parings of i and j

where $i \in C_1$ and $j \in C_2$. Distance in this case is the Euclidian distance between the grid locations on the map of the respective clusters. An additional scaling term is introduced to take into account the relative sizes of clusters:

$$weight\,(C_1, C_2) = 1.0 + \log_{10}\left(1.0 + 9.0 \cdot \frac{|C_1|}{|C_2|}\right) \tag{4.3}$$

which is restricted to $(0, 2]$. This factor helps to bridge gaps between clusters and causes smaller clusters to be absorbed into larger core clusters.

The stopping criterion used for the basic algorithm is four times that of the ants' viewing radius at the time of clustering. Since clusters for the PHC are much looser, but at the same time the clusters are extremely far apart due to the significantly larger map size, a stopping criterion of 25 times the ant radius is used for the PHC. The reason for such a large radius is the very loose cluster created because of the lack of pressure for other clusters to force them to be tighter. The single-link clustering algorithm will stop when all clusters are further apart than this stopping criterion.

After the clustering process is done, all clusters with 2 or few items in it are merged into the nearest significant cluster consisting of 3 or more items. This is done in order to decrease the effects of outliers as it is unclear that two elements together or a single element by itself can be considered a cluster. So for large maps with initially well separated elements, the number of clusters will appear to be zero. As the elements condense into significant clusters, the number of clusters will rise and then, as the significant clusters merge, the number of clusters will fall and converge to a stable number of clusters.

4.5 Test Results

The tests for the algorithms were designed to test the algorithms' ability to scale with qualities of data, numbers of elements, and varying cluster sizes. Five different sample datasets were created for the datasets outlined in Section 4.3. Each of these samples was then tested with both algorithms five times each for a total of twenty five tests for each data set. All the statistics collected for these runs were averaged and plotted and the standard deviations were also calculated and the confidence intervals of one standard deviation in each direction where graphed. Each data set was run for a specific number of ant steps that was experimentally derived by visually watching the basic algorithm (since it take more steps) to see when it appeared to converge. Each dataset was run for a different number ant steps, but the number of ant steps for a particular data set for both algorithms was the same. Samples were taken at fifty equal intervals over the number of ant steps it was run.

The statistics collected are the number of ant actions, the number of clusters, the DB measure, and the time in seconds. The time does not include the time it took to extract the statistics. Unlike the tests used by Handl [12] the distances between the data was not pre-calculated so ants recalculate the distances between two items each time the neighborhood function is calculated. This greatly increases the total execution time. For the complexity and scaling comparisons that each experiment is designed to test, the point of convergence for each of the data sets in a particular experiment were determined. These were identified by manually looking at each of the data sets to determine when both the number of clusters and the DB measure had converged.

4.5.1 Experiment 1: The Amount of Overlap

The quality of the data when it was created was determined by the DB measure described in Section 4.2.1 as applied to the generating, normally distributed clusters. Five different levels of quality were tested for Experiment 1 as outlined in Section 4.3. Values for the DB measures tested ranged from 0.1 to 0.5 where lower values indicate better separated clusters (higher quality) and higher values indicate greater levels of overlap among the clusters (lower quality). It is important to note that data sets that were created with a DB measure higher than a 0.4 have a significant level of overlap among clusters.

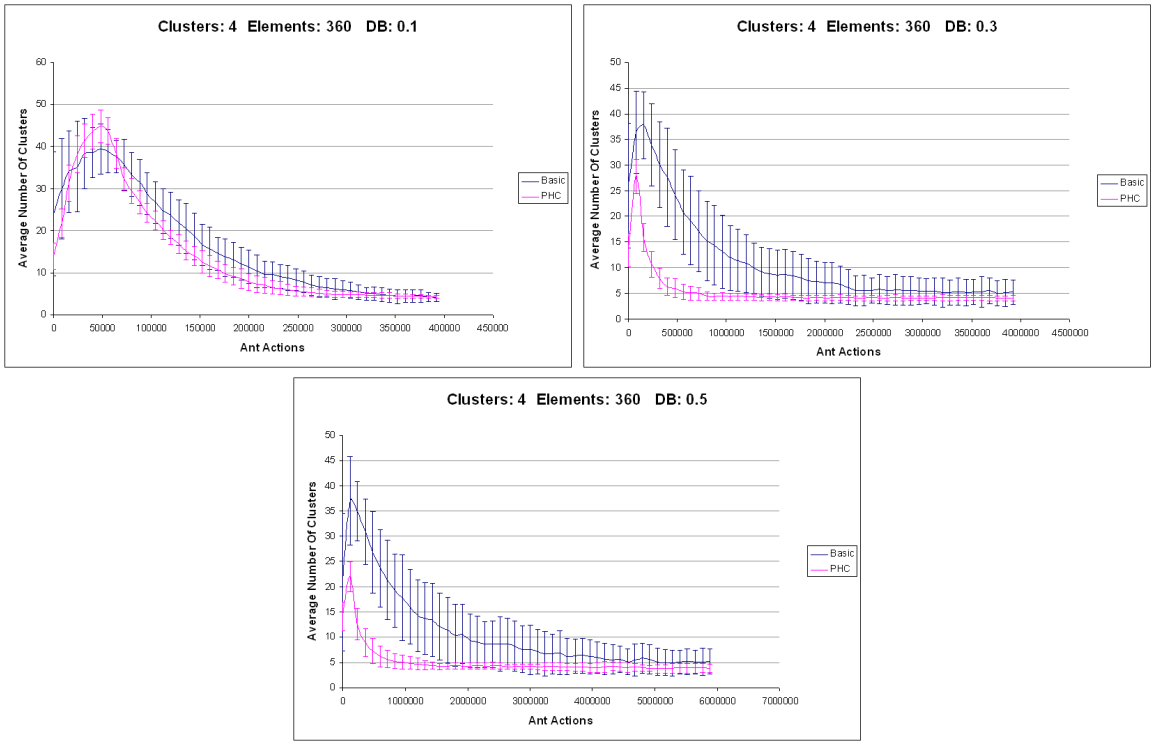

Figure 4.2. The Effect of Cluster Overlap on Cluster Convergence.

Figure 4.2 shows how the number of clusters changes throughout the executions of the different algorithms for data sets that had varying amounts of overlap. The

correct number of clusters in all cases is 4 which both algorithms did converge to. The results show that with highly separated data with a DB measure of 0.1 the basic algorithm initially performs better, but the PHC quickly begins to outperform the basic algorithm. As the quality of the clusters goes down (i.e. the DB measure goes up) the differences in convergence speed are significantly more pronounced between the two algorithms. This is attributed to the PHC algorithm's ability to transport items in a single direction between clusters so ants do not continue to place items in clusters that are dissolving.

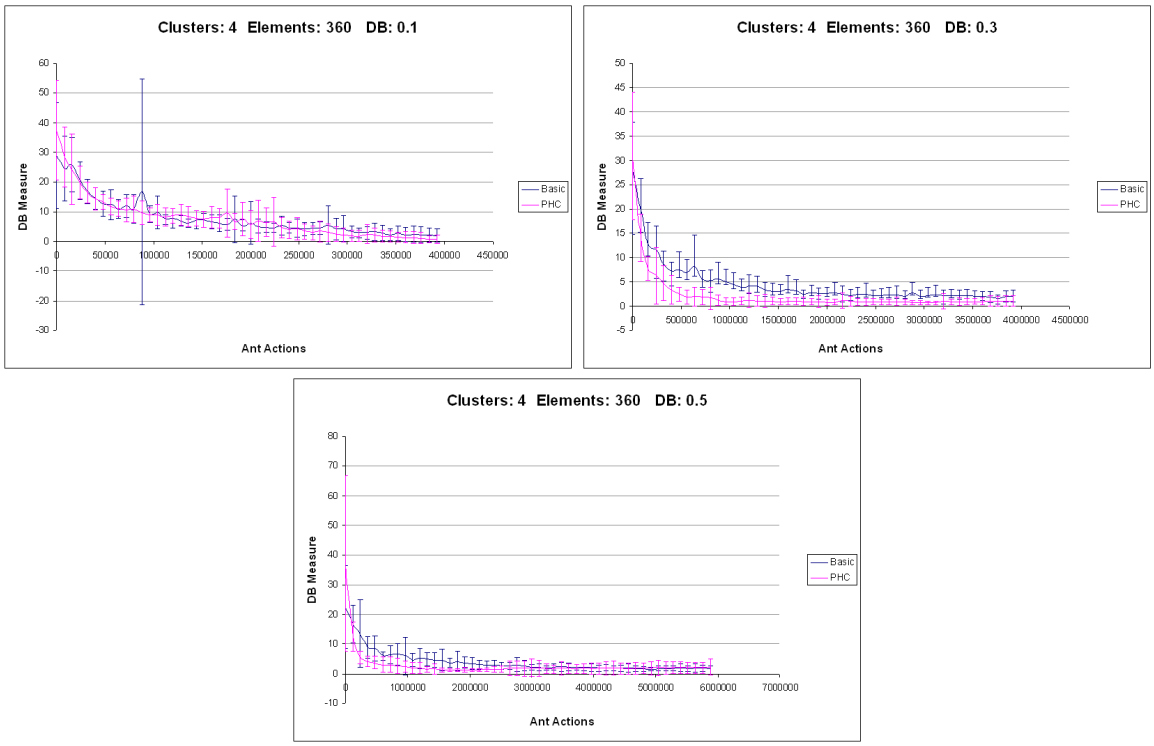

Figure 4.3. The Effect of Cluster Overlap on Convergance to the DB Measure.

Figure 4.3 shows how the overall quality of the extracted clusters changes throughout the clustering process. The differences between the two algorithms were not pronounced with the well separated data, resulting in graphs that are virtually the

same. The reason that they are the same with the high quality data is that to the ants there is little ambiguity as to whether an item belongs to a cluster or not. So while there may be too many clusters on the grid, each cluster is relatively homogeneous in terms of the items that it contains. The better the quality of the data, the easier it is to choose a cluster an item belongs in and, conversely, the less likely it is to be put into a suboptimal cluster. The use of connections by the PHC algorithm slows the movement of items that may be on the fringes of clusters until it is determined that the two clusters should merge. This may slightly improve the intermediate cluster qualities since items are less likely to be moved into new clusters until they have a good match. Since the PCH algorithm is faster in converging to a specific number of clusters, the ants have more time to spend on improving the quality of the clusters. While the majority of the data was relatively similar, resulting in copact confidence intervals, single outliers from time to time caused large confidence intervals. For the DB measure this usually happens when the cluster extractor determines an incorrect number of clusters. Such inaccuracies are very infrequent and are recovered from quickly.

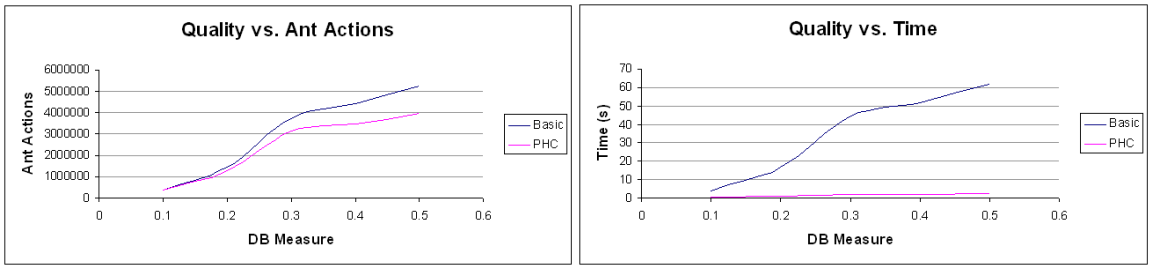

Figure 4.4. Performance with Respect to Cluster Overlap.

Figure 4.4 shows how the clustering time of the two algorithms scales with respect to the quality of separation of the data sets as denoted by the DB measure.

In terms of the number of ant steps needed to cluster items, both algorithms display relatively similar trends, but they clearly begin to diverge as the amount of overlap begins to increase among the clusters. While in depth discussion of the differences in execution time for each ant action is discussed in Section 4.5.4, it is clear that the PHC algorithm outperforms the basic algorithm in terms of scalability.

4.5.2 Experiment 2: The Number of Items

For this particular experiment the number of clusters was kept at 4 for all trials and the level of overlap was kept at 0.2in terms of the DB measure while the total number of items varied. In particular, the total number of items per cluster is varied from 120 to 600 at increments of 120 as outlined in Section 4.3.

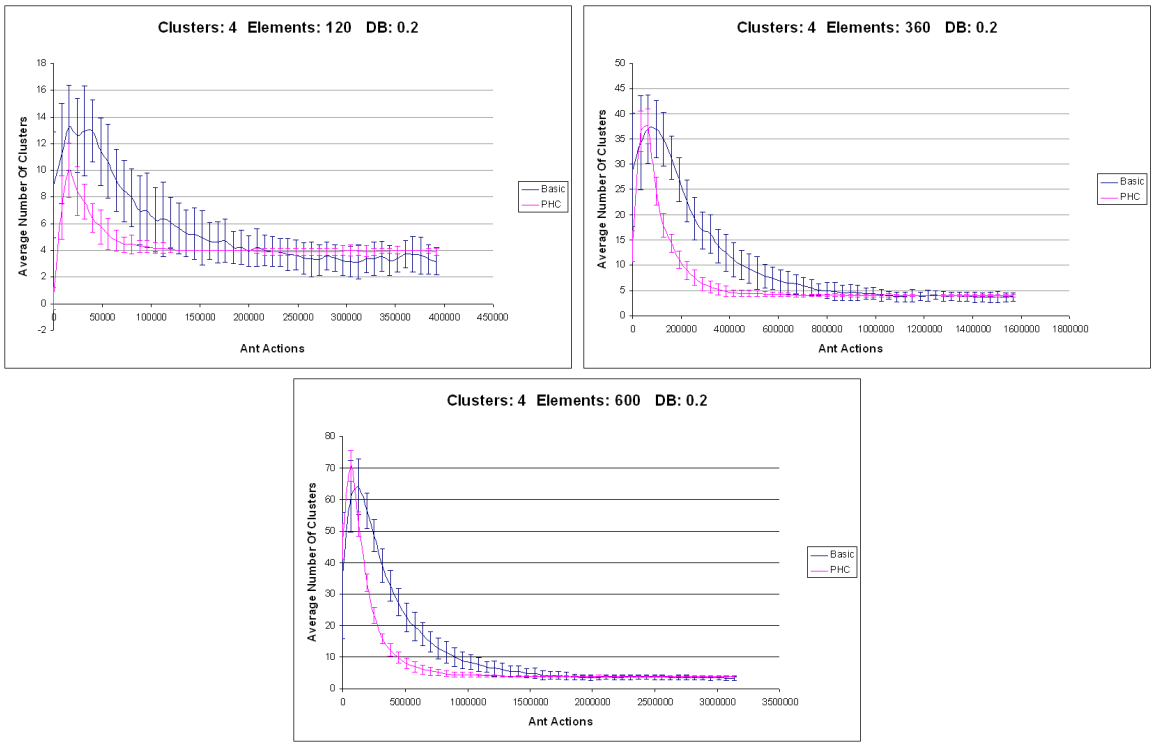

Figure 4.5. The Effect of Item Numbers on Cluster Convergence.

Figure 4.5 shows how the total number of items affects the performance of each algorithm. In all cases both algorithms both converged to four clusters. In the experiment with 120 items the basic algorithm slightly underestimates the number, but this is more likely due to the threshold used to extract the clusters being slightly too high for this small a data set. In general, for all but initial iterations the PHC algorithm clearly outperforms the basic algorithm.

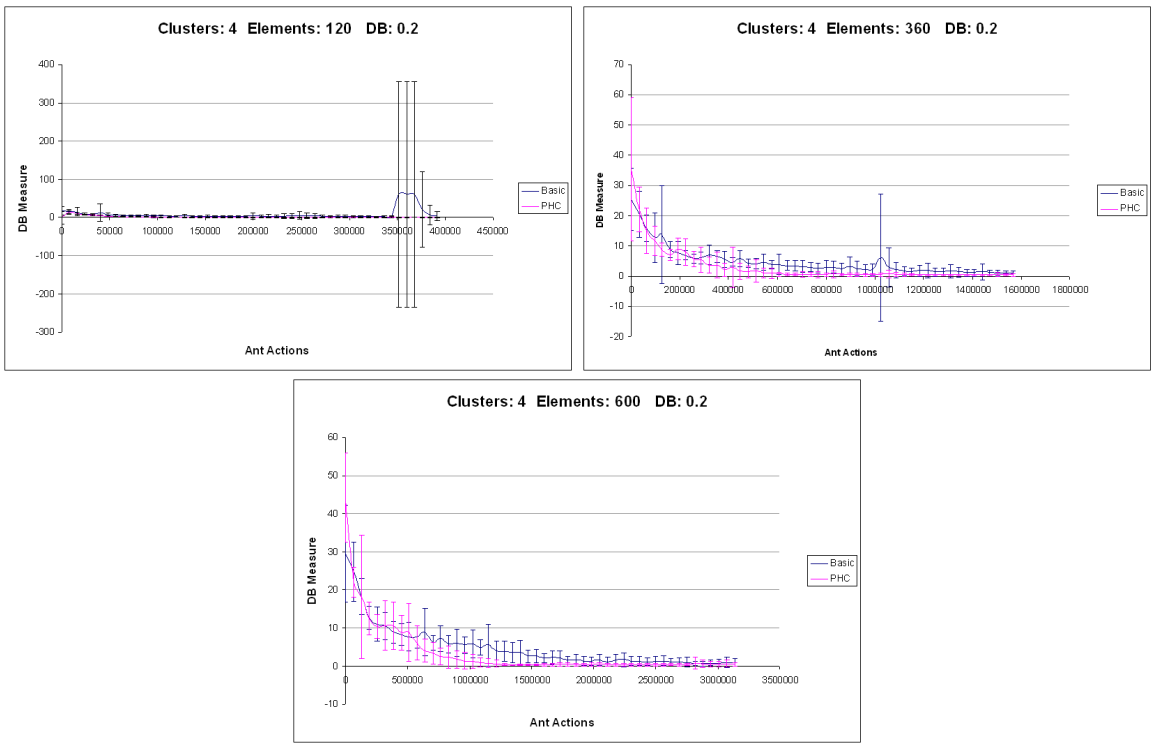

Figure 4.6. The Effects of Item Numbers on Convergance to the DB measure.

Figure 4.6 shows how the overall quality, or level of overlap, of the extracted clusters changes while the clustering process is occurring. For the most part, the two algorithms have very similar results. This is probably due to the fact that the proportions of each type of data are statistically equivalent over the different numbers of items. The spike at the end of the 120 element experiments is due to a single

outlier. In this case the number of clusters extracted was vastly below the actual number, thus causing the DB measure to rapidly rise. The effect is short lived and the basic algorithm recovers to the correct number of clusters.

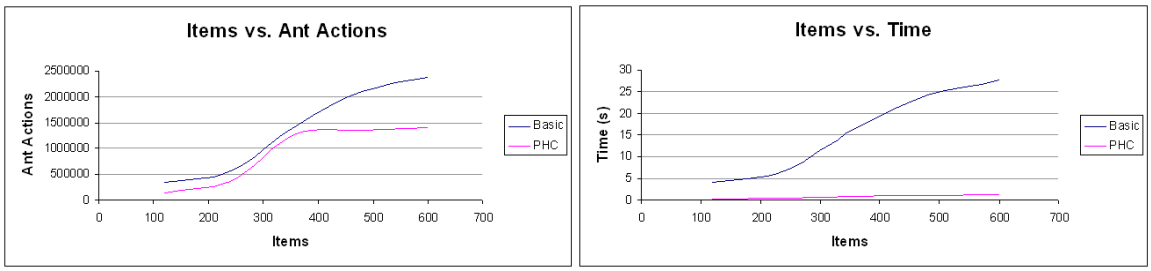

Figure 4.7. Performance with Respect to Number of Items.

Figure 4.7 shows how the clustering time of the two algorithms sales with respect to the number of items. Initially both algorithms have the same trend with respect to the number of ant steps but as the number of items continues to rise the number of ant actions needed by the PHC algorithm increases less compared to the basic algorithm. This is attributed to the more efficient bulk movement of the PHC algorithm. As the clusters grow larger the one-way connections efficiently merge clusters together, and can merge two clusters no matter what there relative sizes are. The basic algorithm still hast to merge one cluster with another by slowly dissolving one and having the elements of the dissolved cluster be deposited in the other. Larger clusters, however, have greater stability and therefore are harder to dissolve, thus lengthening the amount of time required for overall convergence. The PHC also outperforms the basic algorithm in terms of execution time, (further discussion of this is in Section 4.5.4)

4.5.3 Experiment 3: The Number of Clusters

For this experiment the number of items is kept at 360 and the level of overlap was kept at 0.2 in terms of the DB measure while the number of clusters varied from 2 to 6 as outlined in Section 4.3. As the number of clusters grows, the number of items per cluster shrinks.

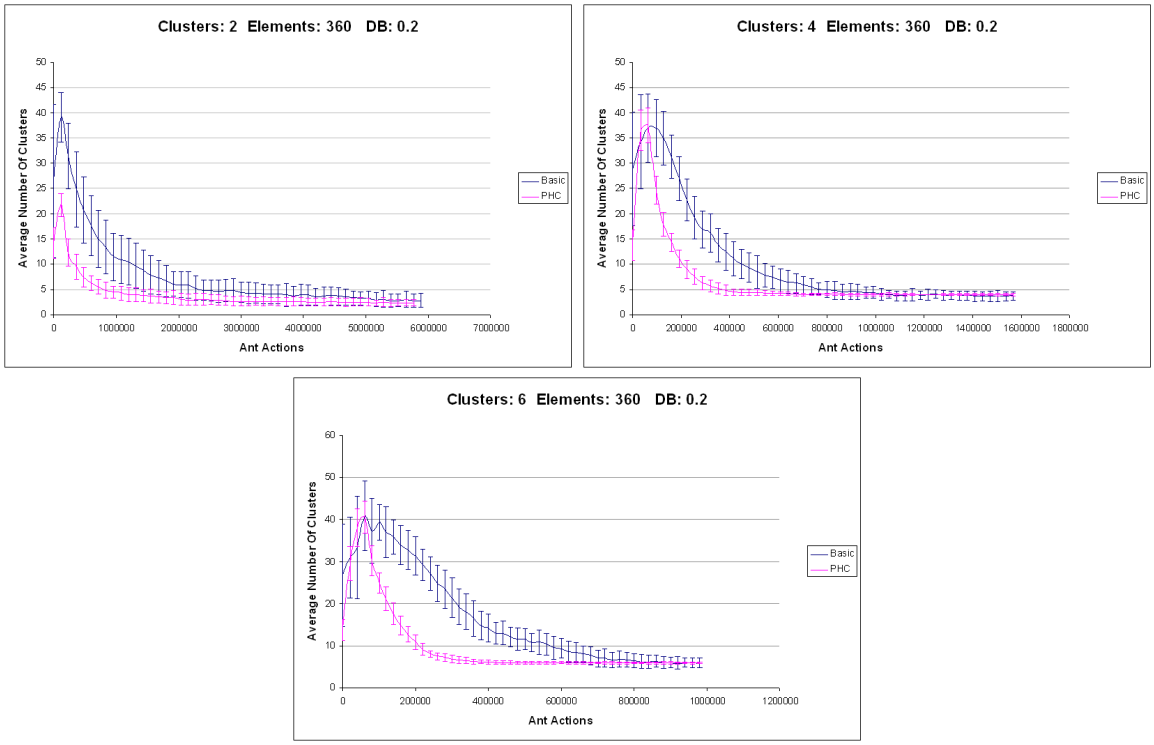

Figure 4.8. The Effect of Cluster Numbers on Cluster Convergence.

Figure 4.8 shows how the number of clusters with the same total number of items affects the performance of each algorithm. For all instances the PHC outperformed the basic algorithm. With fewer types of data the PHC algorithm does much better initially since there is an extremely high probability that the cluster at the other end of the connection is one that that cluster should merge with. Conversely, as the number of types of data grows it becomes the less probable that it will find a similar

cluster with a connection, so the indicial development of clusters is slowed. After the initial clusters are established the PHC algorithm quickly picks up and consistently outperforms the basic algorithm.

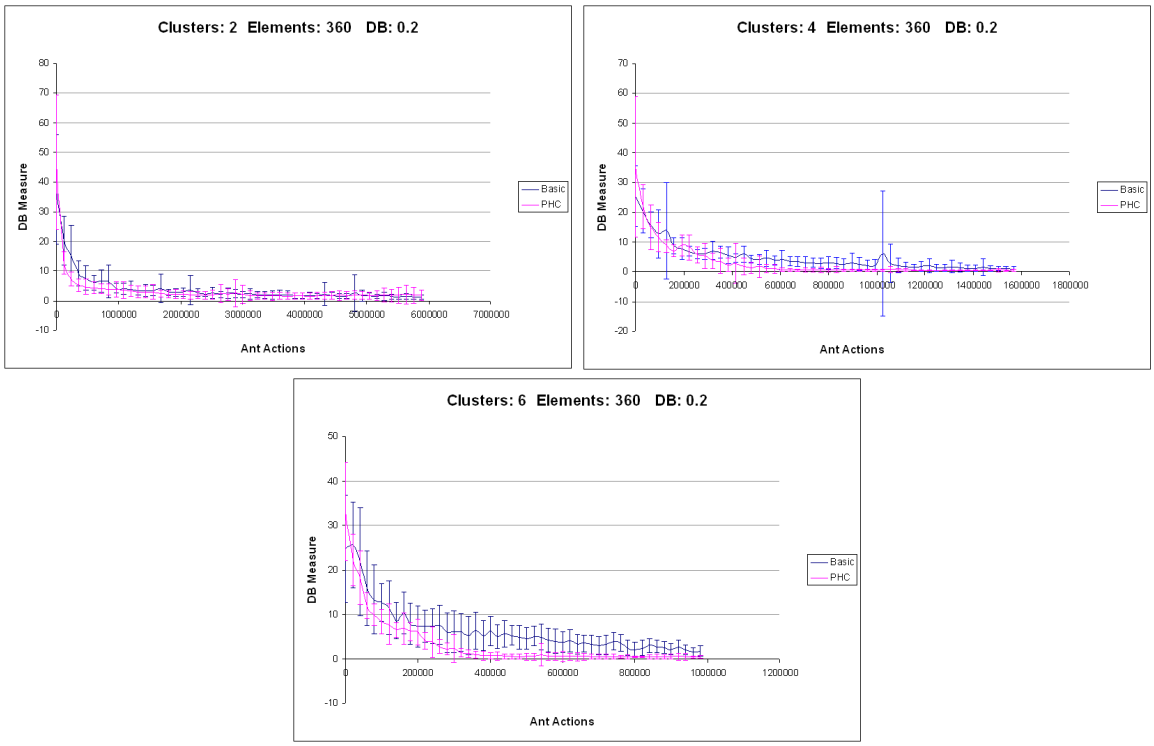

Figure 4.9. The Effects of Cluster Numbers on Convergance to the DB measure.

Figure 4.9 shows how the overall quality or level of overlap, of the extracted clusters changes while the clustering process is occurring. The trends of both algorithms remain roughly the same for both algorithms. The basic algorithm experiences a larger variance due to the increased noise of the wider variety of data points. Since the items move less freely in the PHC algorithm, it experiences much less noise from the increased types of data.

Figure 4.10 shows how the clustering time of the two algorithms scales with respect to the number of clusters. In terms of ant steps the trends of the two algo-

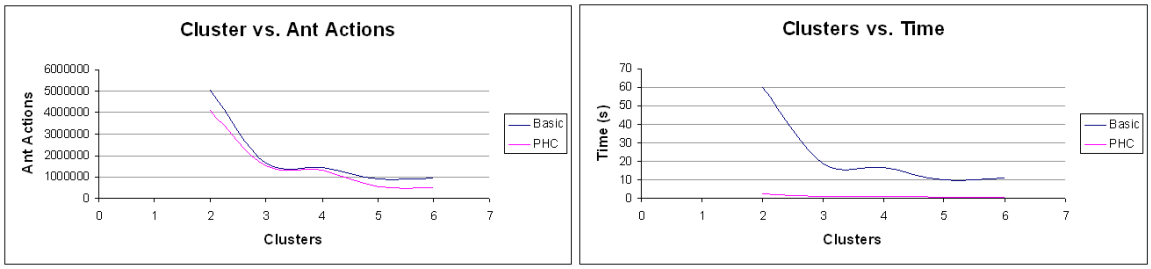

Figure 4.10. Performance with Respect to the Number of Clusters.

rithms very closely match each other with the PHC being slightly more efficient as the number of clusters grow. With respect to actual time, both algorithms become faster as the number of clusters increases. This is because the number of items per cluster is decreasing. The PCH algorithm proves most superior in the situation where there are two clusters. While the basic algorithm's time reduces as the number of clusters increases, it is important to note that the PCH algorithm remains relatively constant as the number of clusters increases.

4.5.4 Time Performance

Figure 4.11 shows the relationship between the number of ant actions and the execution time. This relationship is consistent throughout all the experiments (see Appendix C). The data set for which the resuls are shown is used in all experiments so it is chosen as a representative here. One reason for the difference in the time that each ant action requires can be found in the kind of actions that are taken by the ants. As mentioned in Section 4.2.3, ant actions are counted when the ant has a turn or when it calculates a drop off or a pick up probability. In the basic algorithm, when the ant attempts to find a new item, it will repeatedly attempt to pick random items until it has found one. Between attempts the ant does not move (take a turn) as it does in the PHC algorithm. So the basic algorithm has a higher number of pick up

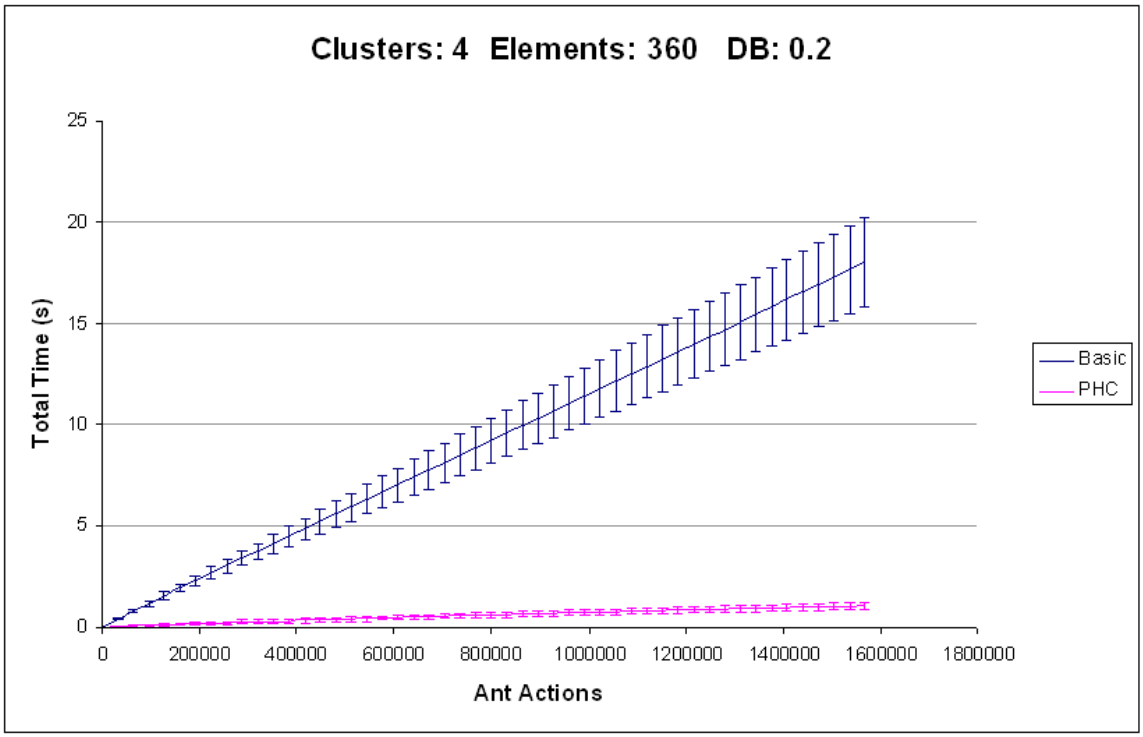

Figure 4.11. Ant Actions vs. Time.

and drop off calculations per turn than the PCH algorithm. In addition, in the PHC algorithm it is possible for an ant to take a turn without ever doing a drop off or pick up calculation when the ant does not have an item to drop or pick up. This further increases the difference between the two algorithms. Since all parts of a turn that are not part of a pick up or drop off calculation require significantly less execution time, the relatively fewer number of pick up and drop off calculations per turn greatly improve the overall execution time of the PCH algorithm over the basic algorithm, even thought it requires only moderately fewer steps to cluster data.

A comparison of the clustering times shows that the PHC algorithm drastically outperforms the basic algorithm. Figure 4.12 shows the convergance times for all three experiments. As previously mentioned, this is partially a result of the smaller number of drop off and pick up calculations per turn. Another reason is the more efficient

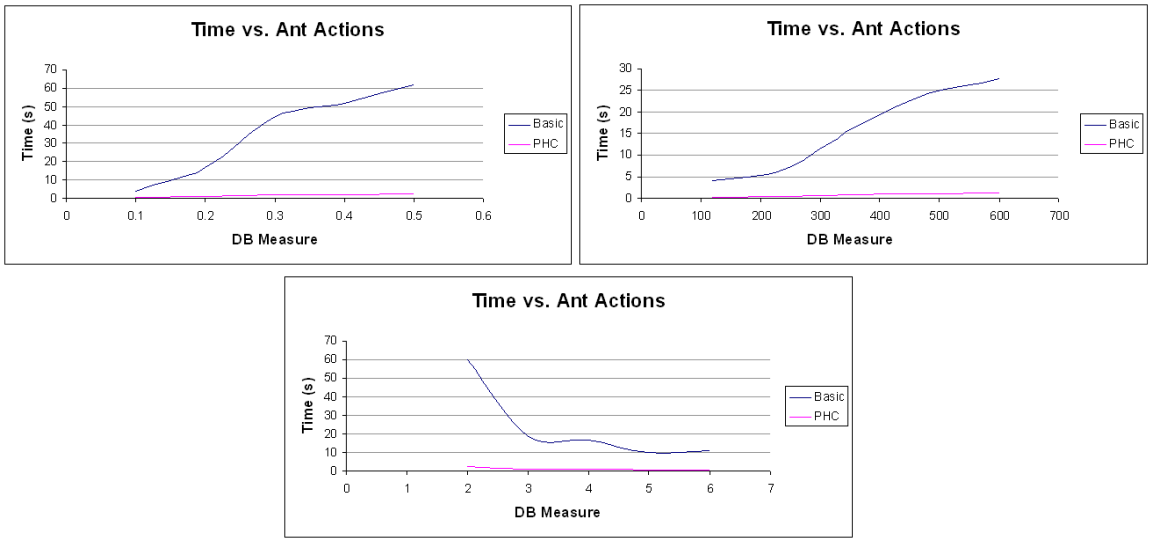

Figure 4.12. Scaling Parameters with Respect to Time.

use of actions. Two main reasons for this increased efficiently stand out. First, ants in the basic algorithm spend more time in free space. Since ants are allowed to move in free space they use both turns and drop attempts in locations that they will never drop items in and therefore do not contribute to the clustering process. In the PHC algorithm, ants are kept within movement zones that keep items within the viewing radius of ants. Thus they spend a higher percentage of time actively clustering items. Furthermore, the use of connections in the PHC algorithm eliminates the need for ants to move in free space and allows them to sample other clusters without ever having to move through free space to get to them.

The second reason is the use of the one-way connections to direct the movement of items in a particular direction. This cuts down on the wasted ant actions that are used when the items are added to a cluster that may be dissolving and merging with another cluster. Items can be passed back and forth between two similar clusters several times before the clusters actually merge in the basic algorithm which slows the merging of two clusters. While some of this does occur in the PHC algorithm, the

one-way connections significantly lessen this effect and thus reduce the overall need for more movement among clusters.

CHAPTER 5

CONCLUSION

Inspiration from simple ant behaviors has lead to a robust clustering and sorting approach. Beginning as an algorithm designed for robots to cluster pucks, ant based clustering algorithms have evolved into algorithms that can organize data in time proportional to the size of the data without having to define the number of desired clusters. Despite these unique properties, basic ant-based clustering algorithms do not solve the problem of having ants wasting time in free space while allowing them to organize clustered items.

The proposed new Pseudo Hierarchical Clustering (PHC) algorithm addresses many of the problems associated traditional ant algorithms. It solves the undoing of work among fellow ants by providing one-way connections. Having the connections span long distances keeps the ants from having to waste time in free space. Since the ants no longer must travel trough free space the performance of the algorithm is only negligibly affected by large grid sizes. Furthermore, with the ability of ants to move a large amount of items in a single direction over large distances, ants can easily merge clusters with the same effort no matter how far apart the clusters may be on the grid. Also, by the use of movement zones, ants can elect a single representative, a queen, to be responsible for all movement of items between clusters. With a single representative per zone, the movement in and out of clusters can be controlled so that ants do not cancel out each other's work.

Several comparative tests were performed against a contemporary solution that also allows for long distant transport, but for only one item at a time, and with little

guarantee that its work would not be canceled out by the actions of another ant. While the PHC algorithm incurs significantly more computational overhead than its basic counterpart, its aforementioned abilities more than compensated for additional computation.

5.1 Future Direction

While the new Pseudo Hierarchical algorithm in this thesis has shown promising results there are several points of possible improvement. One observation is that the queens would occasionally get stranded outside a movement zone and die before two clusters could complete a merge. While the ants would eventually reestablish connection, a significant overhead is incurred in its reestablishment. Possible solution that will be investigated are 1) a routine to detect such a situation and returning a queen, 2) methods to keep the queen away from the edges of a clusters and even preferably in the center of the cluster, and 3) possibly passing to queen's title to one of her workers. For any solution great care must be taken not to dramatically increase the complexity of the algorithm.

Another area worth studying is the ability for the moment zones to accurately extract clusters in linear time. Instead of having to compare the location of every item to the location of every other algorithm as done in the single link clustering algorithm, using the movement zones would only require a single pass over the map.

Beyond this, other challenges include extending the algorithm beyond simple data clustering to other domains that may benefit from an algorithm that explicitly deals with connections over long distances in combination with local sorting and clustering. Such applications could include the balancing of resources of a cluster of computers with unique characteristics over long distances with other clusters of computers. Sending information over long distances can be expensive, and a central

coordinator is likely to be intractable, so having an algorithm that requires no central control would prove advantageous in such a setting.

APPENDIX A

PSEUDO-CODE FOR THE PSEUDO-HIERARCHICAL BASED CLUSTERING ALGORITHM

This Appendix contains all of the pseudo code for the novel algorithm proposed in this thesis. Each function in this pseudo code has a flow chart representation and a corresponding section in Chapter 3 that discusses each procedure. While not necessary for the understanding of the algorithm, pseudo code is provided to facilitate accurate reimplementation of the algorithm.

Algorithm 2: Main Algorithm

```
1  begin
2      INITIALIZATION PHASE;
3      Create a toroidal grid with no ant presence;
4      Randomly scatter data items on the toroidal grid;
5      for each j in 1 to #objects do
6          zone_item(item(j),zone_radius)
7      end
8      MAIN LOOP;
9      for each it_ctr in 1 to #iterations do
10         ESTABLISHED NEW COLONIES;
11         repeat
12             colony_founding_procedure(void);// Alg.  3
13             increment attempt_count;
14         until attempt_count is less than Max_Attemps ;
15         MAINTAIN EXISTING COLONIES;
16         for each agent_ctr in 1 to #active_agents do
17             ant:= agent(agent_ctr);
18             step_procedures(ant);// Alg.  4
19             if is_queen(ant) then
20                 queen_procedures(ant);// Alg.  8
21             end
22             common_procedures(ant);// Alg.  9
23             if is_worker(ant) then
24                 worker_procedures(ant);// Alg.  10
25             end
26             if is_zombie(ant) then
27                 zombie_procedures(ant);// Alg.  11
28             end
29         end
30     end
31 end
```

Algorithm 3: colony_founding_procedure(void)

```
1  begin
2      i:= random_select(all_items);
3      if no_ant(location(i)) then
4          p:= update_presence(location(i),discount_factor,no_ant);
5          if p is less than queen_creation_threshold then
6              make_new_queen(location(i));
7              update_presence(location(i),discount_factor,ant_present);
8          end
9      end
10 end
```

Algorithm 4: step_procedure(*ant*)

```
1  begin
2      l:= random_one_step_location(location(ant));
3      if in_zone(l) then
4          if unoccupied(l) then
5              movement_procedures(ant,l);// Alg.  5
6          else
7              battle_procedures(ant,l);// Alg.  6
8          end
9      else
10         if in_zone(location(ant) AND no_object_in_range(zone_radius) then
11             make_free_space(location(ant);
12         else
13             update_presence(location(i),discount_factor,ant_present);
14         end
15     end
16     increment_stagnation_count(ant);
17     if stagnation_count(ant) is greater than stagnation_limit then
18         if has_object(ant) then
19             forced_putdown(ant);
20         end
21         kill_ant(ant,no_killer);// Alg.  7
22     end
23 end
```

Algorithm 5: movement_procedure(*ant*,*l*)

```
1  begin
2      move_agent(location(ant),l);
3      update_presence(l,ant_present);
4      if presence(l) is less than new_worker_threshold then
5          signal_worker_creation(queen(ant));
6      end
7      reset_stagnation(ant);
8  end
```

Algorithm 6: battle_procedure(*ant*,*l*)

```
1  begin
2      attacker:= ant;
3      defender:= get_ant(l);
4      if NOT same_queen(attacker,defender)AND NOT
       is_zombie(attacker) AND NOT is_zombie(defender) then
5          if worker_lifespan(get_queen(attacker)) is less than
           worker_lifespan(get_queen(defender)) then
6              kill_ant(attacker,defender);// Alg.  7
7              increase_worker_lifespan(get_queen(defender));
8          else
9              kill_ant(defender,attacker);// Alg.  7
10             increase_worker_lifespan(get_queen(attacker));
11             if is_occupied(l) then
12                 movement_procedures(attacker,l);// Alg.  5
13             end
14         end
15     end
16 end
```

Algorithm 7: kill_procedure(*ant*,*killer*)

```
1  begin
2      if is_queen(ant) then
3          if NOT is_queue_empty(get_queue(ant)) AND NOT
           has_object(ant) then
4              take_object_from_queue(ant,get_queue(ant),back)
5          end
6          if is_killer(killer) AND is_alive(get_queen(killer)) then
7              transfer_queue(get_queen(killer),ant)
8          end
9          if is_entrance(ant) then
10             break_connection(ant,get_exit(ant))
11         end
12     end
13     if NOT has_object(ant) then
14         remove_ant(ant)
15     else
16         make_zombie(ant)
17     end
18 end
```

Algorithm 8: queen_procedure(*ant*,*killer*)

```
1  begin
2      if signaled_worker_creation(ant) AND num_workers is less than
       MAX_NUM_WORKER then
3          create_worker(location(ant),lifespan(ant));
4      end
5      if is_entrance(ant) AND is_dead(get_exit_ant(ant)) then
6          sever_connection(ant);
7      end
8      w:= agent(ant);
9      if NOT has_item(w) then
10         if is_entrance(w) then
11             w:= get_worker_with_object(ant);
12         else
13             if NOT queue_empty(w) then
14                 take_object_from_queue(w,get_queue(queen),back);
15             end
16         end
17     end
18     if has_item(w) AND not_exit(ant) then
19         if is_entrance(ant) then
20             r:= exit_agent(ant);
21         else
22             r:= select_random(all_queen_agents);
23         end
24         utility:= attempt_put_item_through_connection(w,r);
25         update_connection_utilities(w,r,utility)
26     end
27 end
```

Algorithm 9: common_procedure(*ant*)

```
1  begin
2      queen:= get_queen(ant);
3      if item_not_present(location(ant)) then
4          if has_object(ant) then
5              drop_prob:= attempt_drop_item(ant);
6              if drop_prob is less than or equal to 0 AND is_alive(queen)
               then
7                  put_object_on_queue(get_object(ant),get_queue(queen),back);
8              end
9          else
10             if is_alive(queen) AND NOT is_queue_empty(queen) then
11                 take_object_from_queue(ant,get_queue(queen),front);
12             end
13         end
14     else
15         if NOT has_object(ant) then
16             if attempt_pickup_object?(ant) then
17                 notify_queen_of_pickup(queen);
18             end
19         end
20     end
21 end
```

Algorithm 10: worker_procedure(*ant*)

```
1  begin
2      age_worker(ant);
3      if get_age(ant)is NOT positive then
4          kill_worker(ant,no_killer);// Alg.  7
5      end
6  end
```

Algorithm 11: zombie_procedure(*ant*)

```
1 begin
2     if has_queue(ant) AND NOT is_queue_empty(get_queue(ant)) NOT
      has_item(ant) then
3         take_object_from_queue(ant,get_queue(ant),back);
4     end
5     if NOT has_object(ant) then
6         kill_ant(ant,no_killer);// Alg.  7
7     end
8 end
```

APPENDIX B

PARAMETER SETTINGS

Table B.1. Parameter Settings For The Basic And PHC Algorithms

Parameter	Value	Algorithm	Equ.
K^-	0.1	Both	(2.2)(3.14)
K^+	0.3	Both	(2.1)(3.13)
Alpha	0.1	Both	(2.3)
Look Ahead Memory Size	10	Basic	
Ant Viewing Radius	2	PHC	(2.3)
Step Size	1 to $\sqrt{20N_{items}}$	Basic	
Step Size	1	PHC	
Map Size	$\sqrt{10N_{items}}$ x $\sqrt{10N_{items}}$	Basic	
Map Size	700 x 700	PHC	
Presence Discount	0.497998	PHC	(3.4)
New Worker Threshold	0.497998	PHC	
Queen Creation Threshold	0.497998	PHC	(3.8)
Initial Worker Lifespan (Queen strength)	25	PHC	(3.9)
Max Stagnation	50	PHC	

APPENDIX C

CHARTS OF ANT ACTIONS VS. TIME

This appendix contains the charts of Ant Actions vs. Time for representative data sets. Discussion of these charts can be found in Section 4.5.4.

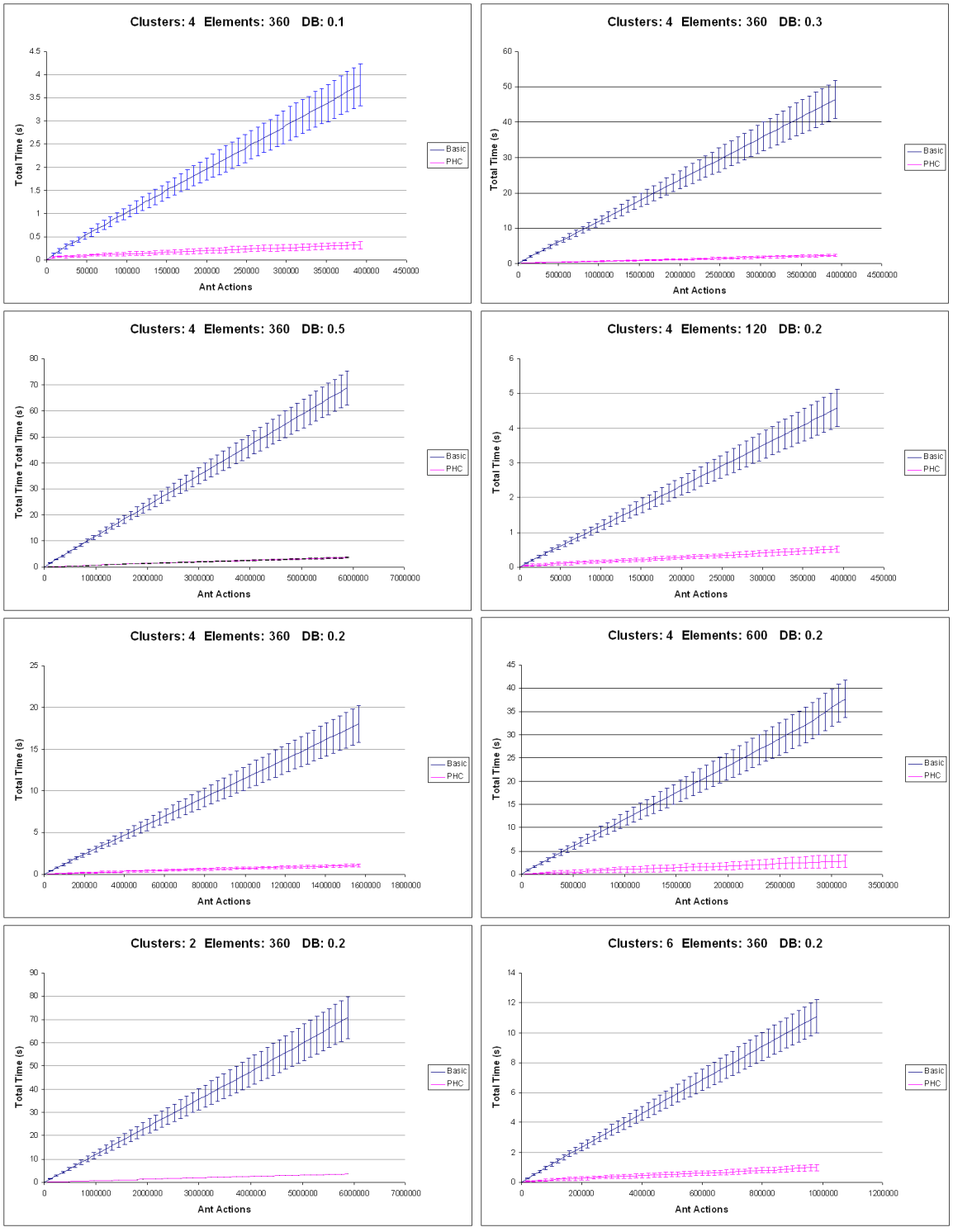

Figure C.1. Charts of Ant Actions vs. Total Time.

REFERENCES

[1] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Wiley, 2005.

[2] R. C. Eberhart, *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*. Academic Press, 2001.

[3] N. R. F. J. S. G. T. E. B. Scott Camazine, Jean-Louis Deneubourg, Ed., *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[4] G. D. C. Marco Dorigo, "Ant colony optimization: A new meta-heuristic," Université Libre de Bruxelles, Tech. Rep., 1999.

[5] M. Dorigo, "Ant colonies for the traveling salesman problem," *BioSystems*, vol. 43, no. 2, pp. 73–81, July 1997.

[6] Z.-J. L. Kuo-Sheng Hung, Shun-Feng Su, "Improving ant colony optimization algorithms for solving traveling salesman problems," *Journal of Advanced Computational Intelligence ant Intelligenct Informatics*, vol. 11, no. 4, pp. 433–442, 2007.

[7] M. Dorigo; and T. Stützle, *Ant Colony Optimization*. MIT Press, 2004.

[8] M. Dorigo, "Ant colony optimization," [Online], March 2009, http://iridia.ulb.ac.be/ mdorigo/ACO/publications.html [Accessed: Apr. 7 2009].

[9] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, and L. C. C. Detrain, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proceedings of the First Enternational Conference on Simulation of Adaptive*

Behavior on From Animals to Animats, S. W. W. Jean-Arcady Meyer, Ed., 1991, pp. 356–363.

[10] E. D. Lumer and B. Faieta, "Diversity and adaptation in populations of clustering ants," in *From Animals to Animates 3:Proceedings of the Third International Conferance on Simulation of Adaptive Behavior*. MIT Press, Cambridge, MA, 1994, pp. 501–508.

[11] J. Handl and B. Meyer, "Ant-based and swarm-based clustering," *Swarm Intelligence*, pp. 95–113, November 2007.

[12] J. Handl, "Ant-based methods for tasks of clustering and topographic mappin: imporvments, evalutation and comparison with alternative methods," Ph.D. dissertation, Friedrich-Alexander-Universität, Erlangen-Nürnburg, 2003.

[13] J. Handl and B. Meyer, "Improved ant-based clustering and sorting in a document retrieval interface," in *Parallel Problem Solving from Nature PPSN VII*, vol. 2439. Springer-Verlag, 2002, pp. 913–923.

[14] Y. Gu and L. Hall, "Kernel based fuzzy ant clustering with partition validity," 0-0 2006, pp. 61–65.

[15] P. Kanade and L. Hall, "Fuzzy ant clustering by centroid positioning," vol. 1, July 2004, pp. 371–376 vol.1.

[16] L. Hall and P. Kanade, "Swarm based fuzzy clustering with partition validity," May 2005, pp. 991–995.

[17] N. Monmarché, "Algorithmes de fourmis artificielles: applications á la classification et á l'optimisation," Ph.D. dissertation, Université de Tours, December 2000.

[18] Q. Li, Z. Shi, J. Shi, and Z. Shi, "Swarm intelligence clustering algorithm based on attractor," in *Advances in Natural Computation*, ser. Lecture Notes in Computer Science, 2005, vol. 3612.

[19] M. A. M. de Oca, L. Garrido, and J. L. Aguirre, *Effects of Inter-agent Communication in Ant-Based Clustering Algorithms: A case study on communication policies in swarm systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, vol. 3789.

[20] M. R. Rezaee, B. B. Lelieveldt, and J. H. Reiber, "A new cluster validity index for the fuzzy c-mean," *Pattern Recogn. Lett.*, vol. 19, no. 3-4, pp. 237–246, 1998.

[21] M. Kim and R. Ramakrishna, "New indices for cluster validity assessment," *Pattern Recognition Letters*, vol. 26, no. 15, pp. 2353 – 2363, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6V15-4GFV5BT-4/2/7d38dda54356555ca55946e069de0a14

[22] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-1, no. 2, pp. 224–227, 1979. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.1979.4766909

BIOGRAPHICAL STATEMENT

Jeremy Brown earned his degree of Bachelor of Science with the dual majors of Computer Science and Mathematics from McMurry University in Abilene, Texas in December of 2004, and his Master of Science in Computer Science from The University of Texas at Arlington in Arlington, Texas in 2009. He is pursuing a career in the defense industry and will later return to continue his education in a Doctoral program.