Ant-inspired strategies for opportunistic load balancing in the distributed computation of solutions to embarrassingly parallel problems

by

Ronald Klazar

Submitted in partial fulfillment of the requirements for the degree
Magister Scientia
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

2016

Publication data:

Ronald Klazar. Ant-Inspired Strategies for Opportunistic Load Balancing in the Distributed Computation of Solutions to Embarrassingly Parallel Problems. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, April 2016.

Electronic, hyperlinked versions of this dissertation are available online, as Adobe PDF files, at:

http://cirg.cs.up.ac.za/

http://upetd.up.ac.za/UPeTD.htm

Ant-Inspired Strategies for Opportunistic Load Balancing in the Distributed Computation of Solutions to Embarrassingly Parallel Problems

by

Ronald Klazar
E-mail: rklazar@cs.up.ac.za

Abstract

Computational science is a practice that requires a large amount of computing time. One means of providing the required computing time is to construct a distributed computing system that utilises the ordinary desktop computers found within an organisation. However, when the constituent computers do not all perform computations at the same speed, the overall completion time of a project involving the execution of tasks by all of the computers in the system becomes dependent on the performance of the slowest computer in the network. This study proposes two ant-inspired algorithms for dynamic task allocation that aim to overcome the aforementioned dependency. A procedure for tuning the free parameters of the algorithms is specified and the algorithms are evaluated for their viability in terms of their effect on the overall completion time of tasks as well as their usage of bandwidth in the network.

Keywords: ant algorithms, cemetary formation, division of labour, distributed systems, distributed computing, opportunistic load balancing

Supervisor : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

"The future has arrived – it's just not evenly distributed yet."

William Gibson

Acknowledgements

I would like to express my gratitude to the following people without whom I would not have completed this thesis:

- Professor Andries Engelbrecht, for taking me on as a student, reviewing all of my work tirelessly, and teaching me more about science than I could ever have hoped to learn.
- My mother, for all her support and cooking, without which I would have written much on an empty stomach.
- My colleagues and friends at the University of Pretoria and especially in the Computational Intelligence Research Group, who sat through countless presentations on hot afternoons and provided critical advice and words of encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In their paper, entitled, 'The "worm" programs - early experience with a distributed computation', Shoch and Hupp [51] briefly describe a program inspired by the science-fiction film entitled, "The Blob" [60]. This program starts out on a single computer and expands to other, connected computers that are not presently utilised by their owners, making use of the added resources to speed up its computations. During the night, presumably when office workers leave for home, the program expands to fill most, if not all, of the computers in the organisation. In the morning, the office workers return to their computers and begin using them once again. When the program detects an increase in computing load, due to the actual owners of the computers using their workstations, it pauses its computations, collects its partial results and retreats to the unused computers. This behaviour is repeated until the computations are completed.

A distributed computing system that behaves according to the "Blob" protocol is loosely coupled because the nodes that constitute the system are neither owned nor controlled by a single entity. As such, the nodes are subject to change with respect to individual performance as well as their actual connectedness to the system.

1.1 Motivation

Consider a scenario where a non-trivial amount of computing time is desired. Individuals or groups who would put large amounts of computing time to task in solving

computational problems have multiple options from which to choose their hardware infrastructure. Before looking to purchase additional infrastructure, one might consider the existing computing resources available within the organisation. If employees of the organisation have been assigned computer hardware for their work, then some or all of these computers may not be engaged all of the time and often not at all after office hours. That idle computing time could be utilised by a distributed computing system.

The concept of utilising the idle time of arbitrary computers has been explored somewhat famously under the term of *public resource computing* in the form of the SETI@home project [5]. The software underlying SETI@home was manifested in the Berkeley Open Infrastructure for Network Computing (BOINC) framework [4] and similar principles were implemented by *Condor* [40]. However, while BOINC and Condor were intended for the construction of geographically large, distributed computing systems, this study considers a much smaller scale, limited by the confines of a single administrative authority such as a single organisation or a department within an organisation. Furthermore, the study imposes a quality of service criterion in the form of the distributed computing system's performance.

A distributed computing system that is composed of disparate nodes will not necessarily be uniform with respect to the performance of the nodes. In other words, some of the nodes in the system might be slower than their peers. It is apparent that if all of the work tasks are similar in complexity, then the faster nodes in the distributed computing system will complete their tasks before their slower peers do. Consequently, the time required to complete all of the tasks will depend on the time taken by the slowest node to complete its task.

Since it is not easy to determine the time it will take an arbitrary computer to complete an arbitrary task, a task allocation algorithm that precomputes the allocation of tasks before the tasks are executed is not feasible. Furthermore, the envisioned distributed computing system presents a dynamic, stochastic environment because the owners and users of the individual nodes in the system are able to pre-empt tasks in order to utilise the nodes for their own needs at any time. Therefore, a task allocation strategy that operates during the execution of the work tasks and does not require a global view of the distributed system is desirable.

An example of decentralised control in a dynamic and stochastic environment has been observed in nature, where ant colonies exhibit seemingly intelligent behaviours that are the products of the many independent individuals that make up those colonies. This observation inspired the task allocation strategies proposed in this study.

1.2 Related Work

Montresor *et al.* [48] describes a dynamic load balancing algorithm that is based on a variation of cemetery formation. The algorithm attempts to distribute independent tasks within a computational grid in order to maximize the grid's utilization. This is achieved by altering the normal behaviour of a cemetery formation algorithm such that an ant drops a task only once the ant has observed a low frequency of similar tasks over a period of time.

Cao [19] describes a task allocation algorithm that is based on the random walk performed by ants that form cemeteries. In the case of Cao's algorithm, an ant wanders from node to node at random and, after a predetermined number of nodes, identifies the node with the highest workload. The ant then executes a second random walk of equal length to that of the first walk and identifies the node with the lowest workload amongst the visited nodes. At this point, the ant brokers a task transfer between the two recorded nodes thus redistributing the workload.

Bertelle et al. [10] present a dynamic load balancing algorithm for distributed computing systems that is based on the same principles as Ant Colony Optimization. The targeted application takes into account tasks that exhibit dependencies amongst each other and must communicate during their execution.

The task allocation algorithms proposed herein adobt a novel approach in that the new algorithms do not employ an object manifestation of an ant that must wander the computer network. Instead, the nodes themselves poll each other directly thus reducing the bandwidth required to determine a node's workload and reducing the latency between finding an imbalance in workload between two nodes.

1.3 Objectives and Contributions

The aim of this study is to facilitate the construction of a loosely coupled distributed computing system by providing a decentralised and dynamic task allocation strategy that will mitigate the impact on performance by non-uniform network configurations. The aim of the task allocation strategy will be to allow the system to consume as much idle processing time as possible while ameliorating the effects of slower resources on the overall completion time of the work tasks.

The specific contributions of this study are the following:

- The proposal of two novel approaches for dynamic task allocation, based on ant algorithms.
- The proposal of a procedure for tuning the parameters of the proposed task allocation strategies.
- An analysis of the viability of the proposed task allocation strategies.

1.4 Dissertation Outline

The study is presented in the following chapters:

- **Chapter 2**, which provides an overview of task allocation in distributed systems and defines the context within which this study is set.
- **Chapter 3**, which traces the origin and evolution of the ant algorithms that inspire the task allocation strategies proposed by this study.
- **Chapter 4**, which describes the proposed task allocation strategies in detail and illustrates the simplified model of a distributed computing system used to evaluate the proposed strategies.
- **Chapter 5**, which explains the sensitivity analysis of the free parameters, followed by the parameter optimization procedure employed to find suitable values for the free parameters of the proposed task allocation algorithms.

- **Chapter 6**, which describes the empirical analysis of the proposed task allocation strategies, including the optimization of said strategies' parameter values, the comparison of the strategies, a scalability analysis of the strategies, the presentation of all results, and the discussion of the results.
- **Chapter 7**, which summarises the findings of the empirical analysis, presents the conclusions of this study, and outlines potential avenues of future work.

Additional information about this study is provided in the following appendices:

- **Appendix A**, which lists and defines the acronyms used.
- **Appendix B**, which lists the symbols used.
- **Appendix C**, which lists the publications that were derived from this work.

Chapter 2

Overview of Task Allocation in Distributed Computing Systems

This study will seek to provide a novel solution to load balancing in a distributed computing system. This chapter defines the application area of the study by introducing the relevant aspects of distributed computing in a top-down approach. Each increasingly specific concept is defined and positioned with respect to the target application. A broad overview of the salient concepts is presented and then each concept is examined in greater detail to elucidate those characteristics of distributed computing that determine the scope of the study. Section 2.1 explains what is meant by *distributed computing* and Section 2.2 describes the scheduling of tasks in a distributed computing system. Section 2.3 summarises the chapter.

2.1 Distributed Computing Systems

A *distributed system* is a collection of hardware components that are physically separate and of software components that can be logically disjunct [52]. A salient reason for creating a distributed system is to share resources such as data, software services or hardware facilities, which include storage and processing time. An important goal, embodied by the above definition, of any distributed system is to provide users with a coherent view of a single system by hiding the fact that the system is actually made up

of many parts that may not even be located in the same vicinity as the users.

A *distributed computing system* is a specific class of distributed system that provides for the aggregation of processing resources and the sharing of processing time [52]. By combining multiple computing nodes and, by implication, processors, a distributed computing system aggregates the individually weak nodes into a single and more powerful processing resource. Such a resource is typically used in the field of scientific computing, which employs computerised simulations to analyse mathematical models of problems of interest.

Behind the façade of a unified system, the computing tasks issued by a user must be mapped to the physical and logical parts of the distributed computer. How computing tasks are mapped to the various nodes depends on the nature of the computing tasks, the nature of the computing nodes, and the quality of service required by the user.

2.1.1 Variations in Task Types

Computing tasks can exploit the resources of a distributed computing system by invoking functions provided by individual nodes or by executing directly atop one or more nodes.

In the former case, a task resides on a single node, from where it requests services from other nodes in the system. Those requests are parameterized and the computations to perform those requests are carried out by the nodes that host the requested functions, which then return the results of those computations to the requesting task.

In the latter case, a task embodies all of the programming instructions required to complete the task's work and the task is executed directly by a node. The node essentially becomes the hardware host for the task.

A task operates on data in order to produce a result for the user. When the data can be divided such that each subgroup of data can be operated upon by a different instance of the task, the task can exploit multiple processing nodes in a technique known as Single Program, Multiple Data (SPMD) (also known as Single Process, Multiple Data), which was first described by Auguin and Larbey in [7].

SPMD is a means of executing a task in parallel in order to lower the time needed to compute its result. This is achieved by placing a copy of the program used to execute the task and a different subgroup of the input data on each available processing node.

The individual results of all of the computations are combined to produce the complete result. This approach ostensibly leads to a linear reduction in computation time as the number of nodes increases.

If there exist dependencies between the individual computations where, for instance, the partial or complete results of one computation are required by another, then the time required to compute the final result depends on the order in which computations are performed and on the time that individual computations remain idle while waiting for dependent results.

Problems that can be divided into individual computations that can be executed independently of one another are sometimes referred to as *embarrassingly parallel* problems [30]. The computations of embarrassingly parallel problems do not require an ordering in execution with respect to one another. However, the time required to compute the final result depends on the characteristics of the hardware nodes that constitute the distributed computer and upon which the individual computations are disseminated.

2.1.2 Variations in Node Types

The nodes that constitute a distributed computing system vary along two axes, namely *architecture* and *performance*. A system comprised of nodes that exhibit the same architecture is said to be *homogeneous* while a *heterogeneous* system includes nodes of two or more different architectures [31].

General purpose distributed computing systems typically are homogeneous and no special care with respect to architecture need be taken when mapping tasks to nodes. However, a specialized task can benefit from execution by a computer architecture that is suited to the particular computation to be performed by that task. A heterogeneous system provides some of the flexibility of a general purpose system, in that it can be used to perform a number of task types, but additionally allows certain tasks to take advantage of specialized architectures. A heterogeneous system must be cognisant of the types of tasks that it processes as well as the types of nodes that are available for processing in order to perform an effective mapping of tasks to nodes. Braun *et. al* describe various mapping techniques in [17].

The variation in processing performance of nodes within a certain architecture deter-

mines whether a group of nodes is *uniform*, where all nodes in the group exhibit the same processing performance, or *non-uniform*, where nodes differ by their processing speeds. A typical example of a uniform system is a cluster [50], formed of multiple, identical processing nodes that are often co-located and tightly coupled by a high-speed network and control software. A computational grid [9] is an exemplar non-uniform system, being made up of geographically distributed nodes that, being subject to individual ownership and control, are more likely to differ in processing performance.

2.1.3 Quality of Service

A system is built to perform one or more functions at the behest of a user. The system is considered to be correct if it performs its functions correctly, i.e., a given input results in an expected output [52]. However, correctness does not prescribe how well a function is executed by the system with respect to predetermined criteria. Such criteria include those characteristics of a function that describe the performance of the function irrespective of its correctness.

Quality of Service (QoS) refers to the measure, according to some scale, of the performance of a system's functions that constitute that system's service to a user [2]. In the context of a distributed computing system, QoS can refer to the time required to deploy a computing task to the system, the time taken to complete a task, and the time required to retrieve the results of a computation from the system. The aforementioned criteria are herein generalised as a single criterion and labelled the *turnaround time*.

2.2 Scheduling

Given a correctly implemented and functioning distributed computing system, QoS becomes the determining factor of the system's usefulness. For instance, a lower turnaround time not only allows individual users to obtain their results in a shorter time span but also enables the system to accomplish more work for a user or to service more users. The approach taken to lower turnaround time depends on the nature of the computing nodes and the nature of the tasks that are to be executed by the system.

The process of optimising the placement and execution of tasks is referred to as

scheduling or, synonymously, *task allocation*. A distinction between the two terms can be drawn by referring to scheduling from a user's perspective and to task allocation from a computing resource perspective. Task allocation algorithms are broadly categorised by whether they are *static* or *dynamic* [20].

2.2.1 Static Task Allocation

When information about tasks and computing nodes is available before computation begins, a distributed computing system can employ static task allocation to determine the mapping of work tasks to computing nodes prior to executing the work tasks. Once a mapping is determined, the tasks are deployed to the chosen nodes and remain allocated to those nodes for the duration of the computation.

A priori determination of task allocation is indicated when tasks exhibit dependencies amongst each other. A scheduler determines the order of execution of the tasks to minimize overall execution time or to minimize communication costs between nodes that host dependent tasks.

A heterogeneous computing system will employ static task allocation to determine which nodes are best suited to executing a user's tasks, based on the fit of a task to a particular hardware architecture.

A homogeneous but non-uniform computing system will take into account the fact that some processing nodes are faster than others and will favour the allocation of tasks to those faster nodes.

However, if the time required to complete a task by a node cannot be determined, then a mapping of tasks to nodes cannot be reliably established. Furthermore, if the performance of computing nodes is subject to change during runtime, then a static mapping of tasks to nodes will become suboptimal when the performance characteristics of the distributed computer change.

2.2.2 Dynamic Task Allocation

Dynamic task allocation algorithms aim to perform the same functions as their static counterparts but do so during the runtime of a computing task. By being responsive to

environmental changes, dynamic task allocation algorithms address problems that static algorithms do not. Such problems include fault tolerance, whereby a task is reallocated when its host fails, and load balancing, which seeks to ensure that computing nodes are evenly loaded with tasks in cases where computing loads change during runtime.

In its general form, "load balancing" refers to the scheduling of differing quantities and types of work on specific resources such that a particular quality of service criterion is met. For instance, service requests are spread over multiple machines such that one machine does not become overloaded and incapable of servicing any request in some expected, minimum period of time.

A dynamic task allocation algorithm does not need to calculate the time required to complete a task in order to find a suitable hardware host for it. Unlike a static algorithm, a dynamic algorithm uses information available at runtime to determine allocation strategies. This characteristic is useful in non-uniform systems where, for instance, the very fact that a fast node becomes idle while a slow node continues to execute a task is a reason to reallocate the task in question [21, 54].

This adaptive behaviour is also suitable in cases where the composition of a distributed computing system changes during runtime. Certain computing systems are designed to allow for nodes to be added and removed over time and without disrupting the service offered by the system. Even when the nodes themselves are not physically removed, they can be repossessed temporarily by their owners who only lend idle processing time to the computing system [61]. In both cases, a dynamic task allocation algorithm will reallocate tasks, depending on the changing performance of existing nodes, the performance of newly added nodes, or the departure of nodes previously occupied by tasks.

Centralisation versus Decentralisation

A dynamic task allocation algorithm can be implemented in terms of a centralised or a decentralised architecture [61]. A centralised architecture locates the scheduling algorithm on a single, physical node or logical (software) module within the distributed computing system. Alternatively, a decentralised architecture nominates multiple (or all) nodes to perform scheduling functions.

Centralisation implies that information about the state of the system will have to be communicated to a single, predetermined location. Likewise, instructions to effect reallocation of tasks will have to be issued from a central point. This central point becomes a dependency in the system. If the point fails, then the whole system becomes inoperable. In cases where the point is not replicated, especially when it is located on a single hardware node, the scalability of the system is limited by the amount of bandwidth available to communicate with the task allocation algorithm.

Decentralisation of task allocation distributes related communication over a greater part of the system's network, thus increasing the upper bound of the system's scalability. At the same time, because there is no single point of failure for the task allocation algorithm, cognisance of its physical location in the system's topology becomes irrelevant. However, decentralisation presents its own challenge in the question of whether or not nodes should cooperate in the scheduling process.

Cooperative versus Non-cooperative Algorithms

When scheduling is distributed amongst multiple nodes, those nodes can cooperate to inform a logically composite task allocation process. The decisions made by the composite process address the global scheduling needs of the distributed system. A cooperative scheduler will incur costs in networking bandwidth and the time required for the participating nodes to communicate information about the state of the system amongst each other.

Non-cooperative algorithms are implemented as independent nodes that make scheduling decisions which do not take into account the scheduling needs of the global system. Instead, each node bases its scheduling decisions on the benefit of those decisions to the node alone. While this approach does eliminate the need for nodes to communicate scheduling information amongst each other, it either increases the difficulty of finding optimal task allocations or prohibits the possibility entirely [20].

2.2.3 Task Allocation Quality

Both static and dynamic scheduling algorithms contribute to the overall throughput of the system, not just due to the task allocation solutions they generate but due also to the

time they require to compute those solutions. A dynamic scheduling algorithm must be able to make changes to the allocation of tasks more than once, thus needing to compute multiple task allocation solutions during runtime. A static scheduling algorithm runs only once, before computing begins, and, conceivably, has more time in which to find a solution than a dynamic algorithm. However, both static and dynamic schedulers are constrained by the QoS policy in place for the distributed computer.

Static and dynamic scheduling algorithms can be designed to compute the best possible task allocation. If such a computation will be too difficult to compute in a reasonable amount of time, a suboptimal solution that is faster to compute, can be used. In the latter case, the methods by which a suboptimal solution is computed is categorised broadly as *approximate* and *heuristic*.

Approximate solutions are found by approximation algorithms, that is, scheduling algorithms that narrow the search space for a solution and pick a solution that is good enough, albeit not optimal. By excluding a portion of the search space, the scheduling algorithm reduces the amount of time taken to find a solution [1].

Heuristic scheduling algorithms employ simplified models of the distributed system that allow the algorithms to exclude details of the system deemed less important by the implementer. Heuristic scheduling algorithms make use of input parameters that indirectly affect the quality of solutions produced. While the time required to produce a sufficiently good solution can be lower than that required by approximation methods, the required input parameters may have to be tuned manually for each system topology [39].

Scheduling algorithms that operate on a fixed set of input parameters for the duration of a computation are classified as *non-adaptive* algorithms [20]. Such algorithms are sufficient when changes in one or more environmental variable do not significantly influence the quality of the solutions produced.

Adaptive algorithms modify either their scheduling policies or the parameters that control their heuristic decision methods in response to changes in the environment. This ability allows the algorithms to maintain consistency in the kinds of solutions they produce even when the distributed system's topology or performance characteristics vary widely. The configuration of adaptive algorithms can be more complex than that of

non-adaptive algorithms because the mechanisms that control how certain parameters are weighted can themselves be subject to configuration by additional parameters.

2.2.4 Remarks on Scheduling

Dynamic task allocation is not a panacea to solve all scheduling problems. Where statically allocated tasks begin execution in an optimal assignment to nodes, a dynamic task allocation algorithm requires lead time to assess the state of the system and effect necessary changes. Moving tasks amongst nodes incurs costs in terms of time spent during transit and bandwidth required to communicate a task's data and, possibly, also program code.

It is therefore necessary to employ a task allocation strategy that is suited to the structure and variability of the distributed computing system as well as the QoS required by the system's users.

2.3 Summary

The construction and operation of a distributed computing system poses, amongst others, a challenge in the allocation of tasks to computing nodes in order to meet the desired QoS criteria. This chapter presented the characteristics of systems that give rise to the problems that task allocation algorithms must solve as well as the general categories of algorithms that allocate tasks to nodes.

The scenario that this research addresses was presented and described. The specific characteristics of the problem were stated and the scope of the research was defined.

Chapter 3

Overview of Ant Algorithms

The task allocation algorithms that are proposed and studied in this work are inspired by models of ant behaviour, namely that of cemetery formation and of division of labour. The fundamental characteristics of both models include an absence of central control and the independent operation of the processes that constitute the conceptual colony. These characteristics are desirable in distributed systems where central control is associated with performance bottlenecks and critical points of failure, while dependencies are associated with a lack of robustness in unpredictable operating environments. This chapter details the aforementioned models and examines the algorithms that have been based on them. Section 3.1 describes the cemetery formation model and Section 3.2 describes the division of labour model. Section 3.3 summarises the chapter.

3.1 Cemetery Formation

Some species of ants, including *Lasius niger* and *Pheidole pallidula*, as reported in [22] and [25], respectively, organise their eggs, larvae, and dead into clusters. Each cluster contains items of the same type while items of differing types would originally have been scattered throughout the nest. Individual ants co-locate items of matching types, thereby sorting the items. The ants do not communicate directly with each other in order to reach a consensus about how different items will be sorted. Likewise, there is no central manager with a view of the whole nest that coordinates the individual workers. Instead,

each worker follows a simple set of rules and actions that effect a sorting of randomly located items. One worker alone can sort all of the items, while multiple workers, all working according to the same behaviourial pattern, complete the sorting process at a higher rate. The sorting process is henceforth referred to as *cemetery formation*.

This section describes a model of cemetery formation in ant colonies and the clustering algorithms that have been derived from it. Subsection 3.1.1 describes the simple model of cemetery formation, Subsection 3.1.2 describes the Lumer-Faeita algorithm for data clustering, Subsection 3.1.3 describes algorithms that modify or extend the Lumer-Faieta algorithm, Subsection 3.1.4 presents an alternative, minimalist model of cemetery formation, and Subsection 3.1.5 concludes with a summary of the problems to which cemetery formation algorithms have been applied.

3.1.1 Simple Model

Deneubourg *et al.* [25] have demonstrated a simple model of cemetery formation that mimicks said behaviour in ants even if it does not necessarily describe the exact, biological mechanisms involved. The model assumes that a nest is divided into a grid, that items of two different types are scattered randomly throughout the grid, that each cell contains either zero or one items, that ants do not communicate with each other or a central authority, that multiple ants cannot occupy the same cell at the same time, and that each ant has a short-term memory that recalls the last m items it has encountered. At each time step, each ant, chosen in a random order, that is currently on the grid will perceive its current cell. If the cell is empty, the ant will move to another, randomly chosen cell. If there exists an item in the cell, the ant will pick up the item with the probability P_p before moving to another cell. The pickup probability is computed as

$$P_p = \left(\frac{\theta_1}{\theta_1 + f}\right)^2 \tag{3.1}$$

where f is the estimated fraction of nearby cells occupied by items of the same type and θ_1 is a constant. Accordingly, when f is small, the probability that an ant will pick up the item in its cell is high. Conversely, a large value of f indicates that the perceived

items are already part of a cluster and therefore the probability to pick an item up is low. An ant makes use of its memory to calculate the fraction of cells containing an item of type A that it has encountered in the last m steps, as follows:

$$f_A = \frac{n_A}{m} \tag{3.2}$$

where n_A is the number of cells containing an item of type A and $n_A < m$.

If an ant is carrying an item, it will drop the item in a new cell with probablity P_d, which is calculated as

$$P_d = \left(\frac{f}{\theta_2 + f}\right)^2 \tag{3.3}$$

where f is as in Equation (3.1) and θ_2 is a constant. In this case, when f is large, the probability that an ant will drop the item it is carrying is high. A small value of f indicates that items of the same type as that carried by the ant are scarce within the ant's vicinity and therefore do not constitute an existing cluster.

Deneubourg *et al.* [25] also showed that by introducing an error into an ant's perception of item types, the number of clusters can be reduced and sorting efficiency increased at the cost of a small overlap of clusters. The error is determined by deliberately misreading a fraction of one type of item as another with the fraction of type A items calculated as

$$f_A = \frac{n_A + en_B}{m} \tag{3.4}$$

where n_A and n_B are the numbers of items of type A and type B, respectively, and e is the error rate.

Oprisan *et al.* [49] modified the simple model such that each ant maintains a memory that encompasses the complete history of object types encountered during the ant's walk. However, in this modification, an ant does not treat each observed instance of an item type in memory as having an equivalent bearing on the determination of the type's

local density. Instead, an instance of an item type carries a weight that is inversely proportional to the number of simulation steps that have elapsed since the instance was observed. The designation of weights is adjusted by a parameter, referred to as the *memory radius*, that determines how far into the past items are assigned a significant weight and beyond which weights become very small. Oprisan *et al.* [49] showed that only certain values for the memory radius allowed for the formation of clusters, and that the memory radius affected the time required to complete the sorting procedure.

3.1.2 Algorithm for Data Classification

Lumer and Faieta [41] extended the simple model introduced by Deneubourg *et al.* [25] to facilitate clustering of multidimensional data that differ along a continuous similarity measure. The Lumer-Faieta algorithm changes an ant's perception from a limited track record of recent steps to an $s \times s$ view of its surrounding, square area, where s is the number of cells that constitute the length of a side of the square. While the algorithm retains Equation (3.1), it uses the following to determine the probability $P_d(i)$ of dropping an item, i:

$$P_d(i) = \begin{cases} 2f(i) & \text{if } f(i) < \theta_2 \\ 1 & \text{otherwise} \end{cases} \tag{3.5}$$

where $f(i)$ is an estimation of the density of elements in the ant's vicinity and their similarity to item i, computed as

$$f(i) = \begin{cases} \frac{1}{s^2} \sum_j (1 - d(i,j)/\alpha) & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

where $d(i,j)$ is a dissimilarity measure used to evaluate every item j in the ant's vicinity that surrounds i and α is used to scale the dissimilarities.

The effectiveness of the algorithm was limited by its tendency to create superfluous clusters. To address this issue, Lumer and Faieta [41] studied three modifications to their proposed method. The first modification diversifies the population of ants with respect to their movement speed and the accuracy of their dissimilarity measures. A fast-moving

ant is not discriminating in its comparison of items, while a slow-moving ant compares items more accurately. Equation (3.6) is rewritten as follows to accommodate ants that differ by movement speed:

$$f(i) = \begin{cases} \frac{1}{s^2} \sum_j (1 - \frac{d(i,j)}{\alpha + \alpha(v-1)/V_{max}}) & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

where V_{max} is the number of cells that an ant travels in one time step and $v \in [1, V_{max}]$. The faster and less accurate ants essentially prime the sorting process by creating the initial clusters, while the slower and more accurate ants refine the new clusters by sorting the items that the faster ants tend to misplace. The result of this modification was that the number of superfluous clusters was reduced.

The second modification includes a memory for each ant. However, instead of using the memory to recall what types of items were encountered in the last m steps, an ant records the locations of the last m items that it dropped. The ant then compares each item that it picks up with those in its memory and transports the new item to the location of the most similar, remembered item. The result of this modification was the creation of fewer clusters with the same, statistical distribution.

The third modification addresses the premature stabilisation of the clustering process, whereby superfluous clusters are formed. Ants are unlikely to remove an item from an established cluster due to the high density of surrounding items. Therefore, a behavioural switch was introduced to cause an ant to begin breaking down a cluster if the ant has not picked up an item after a preset number of steps. The effect of the behavioural switch was to reduce the number of smaller clusters.

3.1.3 Variations of The Lumer-Faieta Algorithm

The Lumer-Faieta algorithm was combined with the k-means clustering algorithm [42] by Monmarché *et al.* [47] to produce a hybrid clustering method called AntClass. AntClass is based on a modification to the assumptions about the placement of data items in the 2-dimensional grid in which the ants operate. Specifically, multiple data items can occupy the same cell and form a *heap*. Clusters are defined by heaps instead of by spatial

arrangements of items.

As with the Lumer-Faieta algorithm, the ant population in AntClass is heterogeneous with respect to the speed of the ants. The proposed advantage of using a heterogeneous population is that the speed parameter does not need to be determined and set by a user [47]. Furthermore, each ant in AntClass maintains a memory that records the location of every encountered heap as well as the heap's centroid in order to guide ants carrying items to the heap with the least dissimilar centroid. Finally, while ants in AntClass do not implement a behavioural switch that causes them to break existing clusters after a period during which the ant manipulates no items, the ants are able to pick up one of a heap of items - that being the item that is most dissimilar with respect to the centroid of the mass to which the heap belongs.

The AntClass method proceeds in four steps:

1. apply cemetery formation to the data items that have not yet been assigned to a cluster in order to obtain an initial partition,

2. apply k-means to the initial partition in order to reduce the classification error,

3. apply cemetery formation to the heaps,

4. apply k-means to the partitioned heaps once again in order to reduce the classification error.

Monmarché [46] showed that the quality of clustering, with respect to the accuracy of assigning items to clusters and the number of resulting clusters, improved with each of the aforementioned steps in the AntClass method. The same publication also showed that AntClass produced superior results to those obtained by using k-means alone.

Wu and Shi [58] proposed a variant clustering algorithm that combines the essential concept of *pick up* and *drop* probabilities, as in the simple model, and a measure of similarity, like that of Lumer and Faieta's dissimilarity measure [41]. The variant algorithm defines the probability that an ant will pick up and drop an item as

$$P_p = \begin{cases} 1 & \text{if } f(o_i) \leq 0 \\ 1 - \beta f(o_i) & \text{if } 0 < f(o_i) \leq 1/\beta \\ 0 & \text{if } f(o_i) > 1/\beta \end{cases} \tag{3.8}$$

and

$$P_d = \begin{cases} 1 & \text{if } f(o_i) \geq 1/\beta \\ \beta f(o_i) & \text{if } 0 < f(o_i) < 1/\beta \\ 0 & \text{if } f(o_i) \leq 0 \end{cases} \tag{3.9}$$

respectively, where β is a constant and $f(o_i)$ is defined as

$$f(o_i) = \sum_{j \in N(r)} \left[1 - \frac{d(i,j)}{\alpha} \right] \tag{3.10}$$

where $N(r)$ defines a circular local area in terms of the radius r, $d(i,j)$ is the Euclidean distance between items i and j, and α is a coefficient.

Wu and Shi's algorithm executes over a range of values for α. The parameter, α, decreases as the number of time steps increases [58]. A large value of α results in a short completion time of the clustering procedure because the contribution of the dissimilarity measure to the calculation of $f(o_i)$ is reduced. Consequently, items are more likely to be placed in any cluster that is encountered, regardless of whether the item is similar to the items within the cluster. A small value of α results in a longer time to completion of the clustering procedure because the contribution of the dissimilarity measure to the calculation of $f(o_i)$ is increased and ants spend more time looking for clusters that contain items that are similar to those that they are carrying. Therefore, the algorithm starts with a large value for α and gradually decreases this value in order to produce a similar effect to that obtained by Lumer and Faieta's use of fast and slow ants. Wu and Shi's algorithm requires suitable values for its free parameters (β and r), as well as an initial value for α and, as pointed out by Monmarché [47], these values can be difficult to determine for a previously unseen problem.

In a similar vein to that of Wu and Shi's contribution, Handl *et al.* [34] proposed several modifications to both Deneubourg's simple model [25] and to the Lumer-Faieta algorithm [41] to synthesize an alternative method. The algorithm of Handl *et al.* employs the following probability definitions for the pick up, P_p, and drop, P_d, actions:

$$P_p = \begin{cases} 1 & \text{if } f(i) \leq 1 \\ \frac{1}{f(i)^2} & \text{otherwise} \end{cases} \tag{3.11}$$

and

$$P_{drop} = \begin{cases} 1 & \text{if } f(i) \geq 1 \\ f(i)^4 & \text{otherwise} \end{cases} \tag{3.12}$$

where $f(i)$ is a modified version of Equation (3.6) and is the local density of item i, defined as

$$f(i) = \begin{cases} \frac{1}{\sigma^2} \sum_j (1 - \frac{d(i,j)}{\alpha}) & \text{if } f(i) > 0 \text{ and } \forall j : (1 - \frac{d(i,j)}{\alpha}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

where σ^2 is the size of the ant's local neighbourhood and α is as for Equation (3.6). The modified local density function increases, firstly, the penalty for empty cells, thus increasing the density of clusters and, secondly, increases sensitivity towards high dissimilarities, thus effecting a greater separation of clusters. The variant probability functions were empirically derived on the basis of the variant local density function and were tailored to increase the speed of the classification algorithm.

Handl *et al.* observed that the Lumer-Faieta algorithm's use of an ant's memory to remember items that were previously dropped by the ant falls short in one respect. Specifically, a remembered item may be moved by another ant between the time that it was recorded and the time that another item is picked up. As such, the comparison of the picked up item with the remembered item will not be valid. In order to preserve the use of memory as a directional bias, the variant algorithm modifies Lumer and Faieta's use of an ant's short-term memory. Instead of comparing a newly picked up item with a remembered item, the variant visits the location of each remembered item and calculates a value for $f(i)$, where i is the carried item. Upon having visited all remembered locations, the ant will execute a single-step jump to the location with the highest value for $f(i)$, with probability P_d. If the jump is not executed, then the ant's

memory is disabled and the ant continues by attempting to drop its item at random locations.

The size of the local area that an ant perceives affects the quality and speed of clustering. A large view increases sorting quality but inhibits the early formation of clusters. A small view limits information about the dispersion of items, thus decreasing sorting quality but increasing the speed at which clusters are formed. Handl *et al.* [34] exploited the aforementioned relation by increasing the size of an ant's view of its vicinity over time. Consequently, the variant algorithm conserved computation time during the early stages of the classification process, while ensuring that initial clusters were easily formed. As time progressed, smaller clusters were destroyed by ants with an increasingly larger view of the grid.

Due to the reliance on item density by the general cemetery formation model, a classification algorithm based on said model will form initial clusters in areas where the density of items of the same type happens to be relatively high. Consequently, clusters can form very near to one another and then remain so for the remainder of the classification process. In order to increase the spatial separation of clusters, Handl *et al.* [34] introduced a temporary period of time during which the scaling parameter of Equation (3.13), σ^2, is replaced with N_{occ}, where N_{occ} is the actual observed number of occupied grid cells within an ant's vicinity. Consequently, density was no longer taken into account and only the similarity of items was considered. The effect of this was that ants gradually moved clusters apart from one another and, after the aforementioned temporary period elapsed, the ants resumed cluster formation but around the new focal points defined by the displaced clusters.

The Lumer-Faieta algorithm employs a parameter, α, to scale the dissimilarities in Equation (3.6) and Equation (3.7). An inappropriate value for α causes ants either to create clusters of dissimilar items or to create many, smaller and superfluous clusters. The nature of the data to be clustered determines what a suitable number of clusters is and therefore the value for α depends on the nature of the data. To remove the need for α to be set before commencement of the classification procedure, Handl *et al.* [34], like Monmarché *et al.* [47], employed a heterogeneous population of ants. Each ant, k, maintains its own value for α_k and, furthermore, updates the value based on its rate of

failure to pick up and drop items. Initially, $\alpha_k \sim U(0,1)$. Subsequently, α is updated according to:

$$\alpha_k \leftarrow \begin{cases} \alpha_k + 0.01 & \text{if } \frac{n_f^k(T)}{n^k(T)} > 0.99 \\ \alpha_k - 0.01 & \text{if } \frac{n_f^k(T)}{n^k(T)} \leq 0.99 \end{cases} \tag{3.14}$$

where $n_f^k(T)$ is the total number of times that an item drop was considered by the ant but did not occur because the probability of dropping the item was too low and $n^k(T)$ is the total number of moves made by ant k at time step T.

Unlike the approaches previously described in this subsection, Yang and Kamel [59] implemented the concept of heterogeneity at the colony level. The population of each colony is homogeneous with respect to how the movement of ants is determined but the colonies differ by movement paradigms. The algorithm employed by a colony is based on Deneubourg's simple model [25] and Wu and Shi's algorithm [58]. Similarity, distance, and speed are related by the local density function,

$$f(i) = max\left\{0, \frac{1}{s^2}\sum_{j \in N_{s\times s}(r)}\left[1 - \frac{d(i,j)}{\alpha(1 + ((v-1)/V_{max}))}\right]\right\} \tag{3.15}$$

where v, V_{max}, and α are as in Equation (3.7), s is the length of a side of the square area of the ant's neighbourhood, and $N_{s\times s}(r)$ denotes the ant's neighbourhood with r as the centre. The dissimilarity measure, $d(i,j)$, can be the Euclidean distance,

$$d(i,j) = \sqrt{\sum_{k=1}^{l}(i_k - j_k)^2} \tag{3.16}$$

where $i = (i_1, i_2, \cdots, i_l)$ and $j = (j_1, j_2, \cdots, j_l)$, or the cosine distance:

$$d(i,j) = 1 - sim(i,j) \tag{3.17}$$

where

$$sim(i,j) = \frac{\sum_{k=1}^{m}(i_k, j_k)}{\sqrt{\sum_{k=1}^{m}(i_k)^2 \cdot \sum_{k=1}^{m}(j_k)^2}} \tag{3.18}$$

which computes the angle between the vectors i and j. Yang and Kamel's algorithm employs both the Euclidean distance and the cosine distance so that each measure compensates for the other's limitation. For instance, two vectors may be present on the same line, thus being separated by an angle of $0°$, but may still lie on different points of that line. The pick up and drop actions are governed by the functions P_p and P_d, respectively, as

$$P_p = 1 - S(f(i)) \tag{3.19}$$

and

$$P_d = S(f(i)) \tag{3.20}$$

where S is the sigmoid function

$$S(x) = \frac{1}{1 + e^{-ax}} \tag{3.21}$$

and a affects the slope of the sigmoid function. The value of a also affects the speed of the clustering procedure. Stable clusters, which are no longer broken down or merged with other clusters, can be formed sooner by increasing the value of a. Yang and Kamel's algorithm employs three colonies of ants. The first colony's ants move at a constant speed. The second colony's ants move at speeds that are randomly chosen from $[1, V_{max}]$ where V_{max} is the maximum speed. The third colony's ants begin by moving at a specified speed and, over time, that speed is reduced by a randomly chosen fraction of the current speed. The algorithm proceeds in two phases. In the first phase, each of the three colonies independently produces a clustering of the data. In the second phase, the three classifications are combined according to a hypergraph model [59].

3.1.4 A Minimal Model of Cemetery Formation

Contrary to the predisposition of researchers to alter and extend the simple cemetery formation model, Martin *et el.* [44] proposed a minimal model of cemetery formation that reduces the simple model. The minimal model eschews the use of a memory as well as probabilities in the decision-making process for picking up and dropping items. Instead, the minimal model consists of a rule to govern the picking up of an item, a rule to govern the dropping of an item, and a rule to govern the movement of an ant.

An ant functioning according to the minimal model will pick up an item in a neighbouring cell with a probability of one. If multiple, neighbouring cells are each occupied by an item, then an item is chosen at random. An item is dropped if the ant has carried the item to at least one new cell and the current cell adjoins at least one cell occupied by an item. An ant moves by selecting a direction at random, after which it selects the distance to travel, also at random. Once an ant reaches the end of its chosen path, it picks a new path, as previously described. The maximum distance that an ant randomly chooses to travel is set by a parameter and this is the only free parameter that is required by an implementation of the minimal model.

Comparisons of the minimal model with Deneoubourg's simple model [25] showed that the latter produced an implementation that was 10 times faster than the former. Martin *et al.* [44] suggested that the addition of a memory could be responsible for the improvement in the speed of the process. However, the minimal model produced clusters of a lower density than those produced by the simple model. Most significantly, the minimal model demonstrated that as the number of ants decreased, down to one, the construction of clusters did not suffer in any respect but completion time. This last observation implies that there is no presence of an emergent behaviour in a colony of ants operating according to the minimal model.

3.1.5 Applications of Cemetery Formation Algorithms

The cemetery formation model, as proposed by Deneubourg [25], has been shown to be an effective basis for novel classification algorithms. Firstly, no global view or prior partitioning of the data is required, making the model suitable in knowledge discovery

applications and in those cases where a physical view of the environment and objects of interest is not available. Secondly, since no central authority is required to control the individual processes (ants), each process functions and fails independently of its peers, thus raising the possibility of implementing an algorithm based on cemetery formation as a concurrent program.

Furthermore, while extensions to the model pose relatively small costs in computing resources where the algorithm implementation is based entirely in software, the simplicity and limited number of behaviours ascribed to each process are advantageous in robotics applications. As each robot requires little in the way of processing ability and memory, the cost of producing each robot remains low, enabling the deployment of many, potentially disposable robots.

Cemetery formation has been used to sort structured data [41] and the Lumer-Faieta algorithm was adapted and applied for use in graph colouring and graph partitioning [37, 38]. Applications in robotics include using a swarm of robots to create annular structures of objects [57] that mimic brood care behaviour, and employing a swarm of robots to partition randomly distributed items [45].

3.2 Division of Labour

In his book, *Die Siel van die Mier* (The Soul of the White Ant), Eugene Marais [43] discusses his studies of termite colonies in South Africa. Marais' studies included an observation that identified division of labour amongst the termites. In one example, termites are divided into physiological castes, which included soldiers and workers. The workers performed different tasks such as construction and maintenance of the colony mound, tending of the larvae, and foraging for food and water. When the colony mound was damaged, soldiers gathered at the area of the damage and, after ascertaining that the perceived threat had passed, signaled an alarm by making a clicking sound. The alarm compelled workers, who were performing other tasks, to gather at the site of the damage and to begin repairing the mound.

This section describes division of labour in ant colonies, a mathematical model of the mechanism of division of labour, and the algorithms that have been derived from that

model. Subsection 3.2.1 describes the relationship between caste ratios and the division of labour in ant colonies, Subsection 3.2.2 describes a fixed threshold model, which seeks to reproduce the mechanism underlying division of labour, Subsection 3.2.3 expands the fixed threshold model with a variable threshold model that takes temporal polyethism and task specialisation into account, Subsection 3.2.4 introduces a modification to the variable threshold model in order to cause ants that perform certain tasks for long periods of time to become specialists, and Subsection 3.2.5 concludes with a summary of the problems to which division of labour algorithms have been applied.

3.2.1 Caste Ratios and Division of Labour

Edward Wilson [55] posits a correlation between caste ratios and the division of labour in colonies of the genus *Pheidole*. The study observed different species of *Pheidole*, each of which consists of minor and major workers. When the number of minor workers was reduced to less than 50% of the workforce, the major workers increased their repertoire of behaviours to assume most of the tasks of the minor workers and increased their rate of activity in performing those tasks. When the ratio of minor to major workers was returned to original levels, the major workers reduced their repertoire of behaviours and decreased their rate of activity. Conversely, when the number of minor workers was increased, no effect on either caste was observed. Wilson termed the behavioural response by major workers to changes in the colony as the *elasticity* of an individual or caste. The colony's *resilience* to environmental changes is therefore determined by elasticity.

The *raison d'être* for the existence of both minor and major workers instead of, for instance, only major workers who perform all actions at all times, is speculated by Wilson to be that a trade-off exists between specialization and energy usage. Specifically, major workers are more efficient at performing certain tasks than their minor counterparts but are also anatomically larger, requiring more energy to function. Minor workers are therefore more economical, at the colony level, at performing menial tasks, while a smaller number of major workers perform primarily specialised tasks. When the number of minor workers falls to levels where the functioning of the colony is impaired, major workers are temporarily able to assume enough of the roles of the minor workers to return

the colony to its prior state of fitness.

Wilson assumed that members of the major caste employed some means by which to detect a decline in the number of minor workers and, in [56], proposed a caste-aversion mechanism that could explain the division of labour. According to [56], major workers tended to perform brood care tasks unless they encountered minor workers near the brood. Major workers that encountered minor workers near the brood were likely to abandon the brood chamber, while minor workers showed little or no reaction. Consequently, a diminishing minor workforce would result in less avoidance of the brood by major workers, which would eventually fill the roles vacated by the previous minor workforce.

3.2.2 The Fixed Threshold Model

Bonabeau *et al.* [15] introduced a fixed threshold model (FTM) to explain some of the observations in [55]. The FTM assumes that a task to be performed emits a stimulus, γ, which represents the demand associated with said task. This demand increases over time, T, and is scaled by the size of the colony, C, according to

$$\gamma(T+1) = \gamma(T) + \delta - \frac{\tau}{C}\sum_{l=1}^{C} C_l \tag{3.22}$$

where C_l is the number of individuals of caste l performing the task, δ is the increase in stimulus intensity, and τ representes the efficiency of task performance, under the assumption that all individuals are equally efficient over time. By Equation (3.22), the demand for a task increases over time until a sufficient number of workers perform the task, at which point the stimulus either remains constant or decreases.

A threshold for performing a task is associated with each individual member of the colony. The probability P_l that an individual belonging to caste l transitions from an idle state ($X = 0$) to an active state ($X = t$) of performing task t is defined by

$$P_l(X = 0 \rightarrow X = t) = \frac{\gamma_t^2}{\gamma_t^2 + \theta_{lt}^2} \tag{3.23}$$

where γ_t is the stimulus emitted by task t and θ_{lt} is the caste's threshold associated with performing task t.

At each time step an active individual becomes inactive with probability p, given as

$$P_l(X = t \rightarrow X = 0) = p \tag{3.24}$$

Bonabeau *et al.* [15] simulated a colony consisting of two castes, namely major workers and minor workers, and considered the cases where: (i) one task was performed and only minor workers were specialised (i.e., had a lower threshold) in performing the task, (ii) two tasks were performed and minor workers were specialised in both tasks, and (iii) two tasks were performed and each caste was specialised in only one of the tasks. In the first and second cases, the FTM yielded similar results to those obtained by Wilson in [55]. The third case yielded the observation that both castes were elastic in their ability to assume the behavioural role of the diminishing caste. The FTM can be extended to any number of tasks by associating a unique threshold with each task for each worker, as described by Equation (3.23). The FTM was also shown to account for a division of labour without the need to model Wilson's proposed caste-aversion mechanism [56].

The fixed threshold model assumes that tasks are preallocated to individuals and that individual thresholds remain constant over time. As such, the FTM does not explain how division of labour occurs - only that it exists. Furthermore, the FTM does not account for task specialization within castes or for temporal polyethism, whereby individuals perform different tasks at correspondingly different stages of their lives.

3.2.3 Variable Threshold Model

Theraulaz *et al.* [53] introduced a variable threshold model whereby a threshold is decreased over time when an individual performs the related task and is increased over time when the individual does not perform the task. The variable threshold model employs the same equations as the FTM, namely Equations (3.22), (3.23), and (3.24) but introduces the following formula to update individual i's threshold θ associated with task t:

$$\theta_{it} \rightarrow \theta_{it} - x_{it}\xi\Delta T + (1 - x_{it})\rho\Delta T \tag{3.25}$$

where x_{it} is the fraction of time T that individual i spends performing task t, and ξ and ρ are coefficients that describe learning and forgetting, respectively.

Theraulaz *et al.* showed that when thresholds are determined by genes, that is, the thresholds vary amongst individuals, those individuals with low thresholds for certain tasks will perform those tasks more readily from an early age, thus accounting for temporal polyethism. When all thresholds are initialised to be equal, stochastic perturbations lead some individuals to specialise in a task, while other individuals specialise in a different task. Finally, when a group of specialists declines in number, non-specialists will eventually take over the roles of the absentees. This is because a task associated stimulus will continue to increase while an insufficient number of workers is performing said task, and the stimulus will eventually exceed the thresholds of the non-specialists.

3.2.4 Worker Specialization

Gautrais *et al.* [33] studied the effects of colony size and demand on worker polyethism under Theraulaz's variable threshold model. In order to be able to compare different colony sizes, the study incorporated a measure of demand, which specifies the proportion of the total *potential* work of the colony that is required to perform a task. The demand influences the rate of increase of a task's stimulus. Thus a small colony can be compared with a large colony when both colonies exhibit the same demand. The results of the study showed that when demand was low in a small colony, most, if not all, individuals remained at rest. In a large colony, a small proportion of workers (the specialists) exhibited heightened activity while a larger proportion (the generalists) worked sporadically or remained at rest. Larger demands increased the number of active workers in small colonies and increased the effect of specialization in larger colonies. Specialization took place when a proportion of workers were able to begin working on a task during the early stages of a simulation by having respectively low thresholds. The active workers therefore had time to "learn" the task, thereby decreasing their thresholds towards that task. Once a sufficient number of workers were performing a task, the stimulus produced

by the task was reduced, preventing workers with respectively higher thresholds from beginning to perform the task in the first place.

3.2.5 Applications of Division of Labour Algorithms

Campos *et al.* [18] compared the variable threshold model of division of labour [53] with a market-based algorithm in an application to a dynamic flow shop scheduling problem. The performance of the two algorithms, with respect to makespan, was found to be similar. However, the ant-based algorithm performed significantly better at cost optimization due to progressive specialization exhibited by the ant-based agents. Furthermore, the work demonstrated that task allocation can be viewed as a scheduling problem that is solved by the variable threshold division of labour model in a dynamic environment.

3.3 Summary

The models of cemetery formation and division of labour discussed in this section are examples of distributed systems in nature. Both models function on the principle that seemingly sophisticated behaviour need not be the product of a monolithic mechanism. However, as shown by Martin *et al.* [44], it is not necessarily the case that complex behaviour emerges from a collective of individually simpler behaviours. This trait in particular is potentially useful in a distributed computational system where degradation to the extreme point of only one functioning process does not compromise the system's function but merely slows it down.

Chapter 4

Dynamic Load Balancing Based on Ant Algorithms

This chapter describes the ant-inspired opportunistic load balancing strategies that were designed and implemented for this study. The strategies were studied within an experimental model that expresses a simplified distributed computing system and narrows the focus of the study to a test of each strategy's viability. The model is described here in detail and each strategy's realization within the model is articulated. The problem domain is then described in terms of the experimental model and a scope for the study is established.

Section 4.1 describes the experimental model that was devised for the purposes of this study. Section 4.2 describes the baseline task allocation strategy that was used to provide a basis for comparison of the proposed task allocation algorithms. Section 4.3 describes the cemetery formation task allocation strategy in terms of the experimental model. Section 4.4 describes the division of labour task allocation strategy in terms of the experimental model. Section 4.5 defines the problem domain, which specifies the composition of the distributed system's network within the experimental model and Section 4.6 summarises the chapter.

4.1 The Experimental Model

Each of the dynamic load balancing (DLB) strategies that was studied is defined within the context of the experimental model that was designed for the purpose of this investigation. The model includes representations of the hardware and software required to provide a general-purpose distributed computing service and is implemented as a simulator that executes distributed computations and measures the time taken to complete those computations. The goal of the experimental model is twofold: (i) to test a DLB strategy's efficacy in balancing computation loads and (ii) to do so without having to prescribe a specific system configuration or specific systems of measure by which to evaluate the results. The remainder of this section describes the experimental model in detail. The problem space is defined first, after which the architecture and component interactions of the experimental model are discussed. Thereafter, the parameters that define instances of the experimental model as well as the observable effects of those parameters are described. Finally, the assumptions made by the experimental model are listed and rationalised.

4.1.1 Elements of the Problem Space

The problem space is divided into two areas, the *service* and the *facility*, each of which is addressed by the experimental model. The service encompasses the activities for which the system is intended and includes the users of the system and the processes that they transact on the system. The facility includes the hardware and software that constitute the system and in this investigation focuses on the ownership of those assets. Each area of the problem space is further described next.

The Service

The experimental model is defined for the parallel execution of tasks that contribute to the solutions to embarrassingly parallel problems. An embarrassingly parallel problem is herein fully described as a *project*. Each project is divided into a subset of independent parts, each of which is herein referred to as a *task*. Tasks are independent of each other in that no dependencies between tasks exist, allowing tasks to be executed irrespective of

each other. A task is discrete in that it cannot be subdivided and it cannot be executed by multiple, concurrently executing processes. The execution of a task produces a *result*, which itself is a part of the solution to the project. The complete solution to the project is obtained by forming a combination of the results produced by the executions of its constituent tasks.

The Facility

While a distributed computing system harnesses multiple computers to provide a coherent service to its users, those computers are not necessarily owned and managed by a single administrative domain. One of the the aims of this study is to produce DLB strategies that support the creation of a distributed computing system that incorporates computers that are designated for a purpose other than distributed computation. Consequently, it is foreseeable that the owners of these computers may wish to utilize their hardware at various points in time and, in so doing, may preempt the execution of tasks at any time, for arbitrary durations of time. Furthermore, the aim to incorporate computers from multiple administrative domains raises the possibility that not all of the computers connected to the distributed computing system will exhibit the same performance characteristics.

In order to address both the unpredictability of computer owners and the variety of their computers' processing capacities, the experimental model makes provision for disparity in the performance of the distributed computing system's individual nodes. The experimental model defines a *non-uniform* grid, which is comprised of computers with unequal performance characteristics, and a *uniform* grid, which is comprised of computers with equal performance characteristics.

4.1.2 Component Architecture

The experimental model is designed according to a client-server architecture [23, 52] because this architecture is readily understood and uncomplicated in implementation. A single server hosts the tasks that must be processed and is herein referred to as the *project server*. Multiple processing nodes are connected to the host of the project server as well as to each other. For each physical processor there exists a client application, herein

referred to as a *resource*. Together, the resources and project server form a computing *grid*. Resources request tasks from the project server, execute those tasks and return the results of those computations to the project server. Thus the project server acts as the user entry and exit points to and from the system, respectively.

The project server effectively centralizes the source of tasks and the results repository. Even though the DLB strategies are decentralized, the system is still prone to a single point of failure. Since this study focuses on the DLB strategies themselves and does not aim to build a complete middleware layer, a distinction is drawn between *allocation* and *communication*. The experimental model facilitates the optimal allocation of tasks to the available resources and excludes means by which to correct faults in the communication of tasks and results.

Each host possesses a single network interface controller (NIC). Communication amongst resources and between the resources and the project server is connectionless and asynchronous. For the sake of simplicity, no persistent connection is established between communicating processes.

4.1.3 Component Behaviour and Collaboration

The experimental model comprises a number of components, each of which exhibits behaviour that may be translated into an executable process. These components include: the project server, the resources, and the tasks. Each of these components is described in this section.

Task

A task is a self-contained program instance in that it contains the data to be processed, the parameters for the computation, and the process logic by which to transform the data or generate a result. The single responsibility of a task is to produce a result. When a task is executed, its process logic is invoked with the associated parameters. The partial result of a task is stored with the task component until the task is completed, at which point the result may be retrieved. A task's execution may be suspended at any point in time and the current state of the task process may be serialized and transmitted to a remote host.

Project Server

As the user interface to the distributed computing system, the project server has three responsibilities: (i) enqueue projects for execution on the grid, (ii) make the constituent tasks of a project available to resources, and (iii) collect the results of completed tasks. When a user wishes to execute a project on the computational grid, he or she will enqueue the task on the project server. If no projects are pending, then the project will immediately be deployed for execution. Otherwise, the project will be placed in a queue. Once a project is deployed, its tasks are available to the resources in the grid and requests for tasks may be serviced. The project server responds to a request for a task by recording the task's unique identifier and then sending the task to the requesting process. When a result is received, the originating task's identifier is extracted from the result message, the result is stored and its associated task is marked as having been completed. When the status of a project is queried, the project server checks if any results are outstanding and, if none are, then the project server returns a positive response.

Resource

A resource has four responsibilities: (i) obtain a task, (ii) execute that task, (iii) return the results of the task to the project server, and (iv) respond to requests for the task. When a resource has no task to execute, the resource requests a task from some source, which is determined by the DLB strategy. On receiving a task, a resource begins to execute that task. Otherwise, the resource repeats its attempt to acquire a task. Once a task is complete, the resource transmits the result of the task to the project server. The time taken by a resource to complete a task is dependent upon the speed and capabilities of the resource host's processor. While in any state, a resource may receive a request for a task from another resource in the grid.

4.1.4 Parameters Defining Model Instances

An instance of the experimental model is defined by a single project server, one or more resources of one or more types, the network formed by the server and resources, and a set of tasks, collectively referred to as the project, to be executed by the resources. The

following parameters define these components.

Task Count

The task count represents the number of tasks present in a project. A corresponding result is required for each task in order for the project to be completed.

Task Duration

The duration of a task reflects the number of processing steps required to complete the task. A task is completed when the requisite steps to perform the task have been executed by one or more resources.

Resource Count

The type of a resource is determined by its processor performance. While a uniform network of resources would contain only one type of resource, a non-uniform network would contain multiple resource types. The resource count specifies the total number of resources of all types in the network.

Resource Processor Performance

Each resource is associated with a single, physical processor. A processor performs a task by performing the task's individual steps. Therefore, the performance of a resource is determined by the number of steps that its associated processor is capable of performing per unit of time.

4.1.5 Observable Effects

For the purpose of this study two measures of performance are taken into consideration and are used to determine whether or not a DLB strategy is promising. The *turnaround time* measures the effectiveness of a DLB strategy in maximizing the use of the fastest processors. The *message count* measures the cost, in terms of network communication, of achieving the measured turnaround time. These measures are the only observable effects

of a simulation run within the framework of the experimental model. Each measure is described below.

Turnaround Time

The duration of time that elapses between the user starting the project and the return, to the project server, of results from all constituent tasks is herein referred to as the *turnaround time*. The turnaround time is the number of units, in whatever measure of time is chosen, that the grid takes to complete the specified project. A DLB strategy will aim to minimize the turnaround time.

Message Count

The message count is the number of messages that are required to be carried by the grid network in order to complete the specified project. The message count is analysed in order to investigate the cost of employing the various DLB strategies. As with turnaround time, the actual unit of measure is left abstract and the message count is broken down into counts of the different types of messages transmitted during the execution of a project. Consequently, each type of message can be associated with a specific weight - for instance, the number of bits required to represent the message - in order to facilitate an analysis for a specific application. A DLB strategy will aim to minimize its message count.

4.1.6 Assumptions Made by The Model

The purpose of this study is to determine the viability of the ant-inspired DLB strategies. Instead of producing results that are specific to network topologies and resource host configurations, this study aims to present a platform upon which further research may be conducted. Subsequently, assumptions are made about the experimental model that simplify and generalize the empirical analysis. These assumptions include the following:

- Each node in the grid is connected to every other node in the grid, obviating the need to simulate routing within the network. Network topologies are determined by specific application domains. This study aims to investigate only the general

viability of the proposed DLB strategy. Therefore, no routing is simulated within the network.

- No specific software, including a stack of network protocols, is taken into consideration. This study is concerned only with the cost of communications exhibited by the DLB strategies under study. Network protocols in layers below that of the *application* layer of the Open Systems Interconnection (OSI) network model [35] are, like the network topology, determined by specific application domains.
- Nodes within the grid are free to transmit any amount of data per unit of time. This allows for the measurement of bandwidth usage within the network, providing more information than a simulation that constrains communications by limiting bandwidth and reduces results to turnaround times only.
- There exists a means by which to measure the maximum and current performance of a computer in real time such that different computers may be compared in terms of their performance values. The DLB strategies investigated by this study depend on the ability to determine the performance of resource hosts.
- Only one project is executed at a time. The experimental model does not employ any mechanisms that facilitate the use of the grid to execute multiple projects simultaneously. Such functionality is typically implemented by the distributed system middleware [52], of which the DLB strategies are only a part.
- Network communication is free of faults. In this way, the experimental model guarantees that messages sent using a connectionless protocol are always delivered.
- It is always possible to retrieve a task from a resource. The experimental model does not make provision for the case where a task and its result are not retrievable because a resource goes off-line.

4.2 The Baseline Task Allocation Strategy

In order to provide a point of reference for the evaluation of the ant-inspired DLB strategies, a performance baseline will be drawn within the framework of the experi-

mental model. The ant-inspired strategies aim to mitigate the negative effect that a non-uniform grid may have on the turnaround time of a project. Therefore, the choice of a baseline, against which the performance of the ant-inspired strategies will be compared, aims to express that negative effect.

The strategy used to determine the baseline performance data is based on the faulty premise that precipitated Amdahl's Law [3] - that the relation between the number of processors and the performance of parallel programs is linear. As Amdahl demonstrated, even small proportions of time spent executing sequential code negate the perceived benefits of high degrees of concurrency, resulting in non-linear speedup and potentially wasted investments in processor hardware. In a similar vein, a single resource host that underperforms all other resource hosts in a computational grid may become the factor that limits the minimum bound of project turnaround time. Therefore, the goals of the baseline strategy are: (i) to demonstrate that a non-uniform computational grid is subject to bottlenecks due to performance variations across resource hosts, and (ii) to establish the situations under which the bottlenecks occur.

The remainder of this section describes the method employed by the baseline strategy to allocate tasks to resources.

4.2.1 The Task Allocation Mechanism

Opportunistic Load Balancing

The essential aim of the baseline task allocation mechanism is to saturate the network with tasks indiscriminately by making use of *opportunistic load balancing* [6, 16, 31]. Opportunistic load balancing is a crude implementation of general load balancing, whereby the next task is allocated to the first available node. The desired consequence of this approach is that every resource that requests a task will obtain a task (as long as tasks are available from the project server).

Figure 4.1 depicts the states and transitions for a resource operating according to the baseline strategy. An idle resource requests a task from the project server once during each unit of time. While waiting for a response from the server, a resource remains in the *sourcing* state. If a task is available, then the project server sends the task to the resource

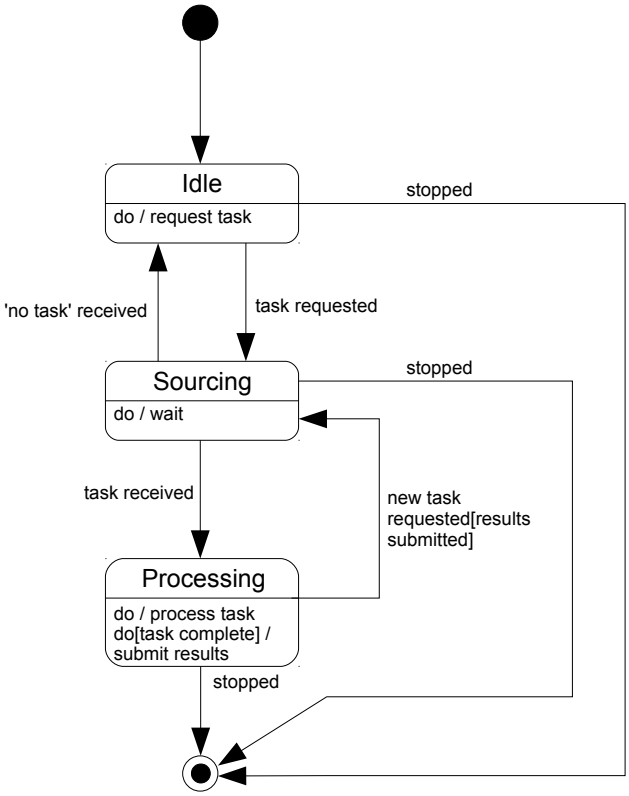

Figure 4.1: State transitions for a baseline resource.

for execution. A resource is in the *processing* state while it executes a task. When a resource completes its task, the resource returns the result of the task to the server and requests another task. The project is complete when no more tasks are available and all results have been returned.

4.3 The Cemetery Formation Task Allocation Strategy

Lumer and Faieta [41] described a model of cemetery formation (CF) by ant colonies, which, in its generalized form, describes a clustering method. However, the simpler

model described by Deneubourg *et al.* [25] and studied by Dorigo *et al.* [26] is used as the basis for the first task allocation mechanism proposed by this study. The simple model of cemetery formation is henceforth referred to as the original cemetery formation model (OCFM). In terms of the experimental model under study, tasks are viewed as ant corpses, which are moved by ants from slower resource hosts to faster ones. The mechanism by which ants in Dorigo's *et al.* model decide when to pick up a corpse and when to put it down is adapted to compare resource host performance measures. Each cell in the grid utilized by the study in [26] is represented here by a resource. As such, ants wander the network, relocating tasks to idle resources that can execute those tasks faster than those resources where the tasks are currently situated.

This section describes the method used by the CF task allocation strategy to allocate tasks to resources. The task allocation mechanism is described and the implications of the mechanism's realization within the experimental model are discussed. Thereafter, the ant actions that define the task allocation protocol are described and, finally, the parameters that control task allocation are listed and explained.

4.3.1 The Task Allocation Mechanism

Consider a grid that operates according to the baseline task allocation strategy, that is, according to opportunistic load balancing. When no more tasks are available from the project server and the only pending tasks are those being executed by resources in the grid, the turnaround time of the project will be determined by the completion time of the task being executed by the slowest resource. At the same time, a faster resource might be idling because it has completed a task and no more tasks are available for it to execute. In such a case, an incomplete task can be moved from the slow resource to the faster, idle resource in order to reduce the project's turnaround time. Therefore, the CF task allocation mechanism attempts to locate pairs of non-uniform resources, between which the reallocation of a task will reduce the task's completion time.

Task allocation may be divided into two phases: a *deployment phase* and an *optimization phase*. The deployment phase begins when a project is deployed by the project server and resources begin to request tasks from the project server. While the project server contains incomplete tasks, any resource may request a task and the next task in

the project server's queue is transmitted to the requesting resource. The project server does not order a project's tasks in any way. The aim of the deployment phase is to saturate the grid with tasks and to empty the project server's queue of tasks. The deployment phase ends when no more tasks remain in the project server's queue.

The optimization phase begins when a resource receives a response from the project server stating that no more tasks are available from the project server's queue. The implication of this response is that there is at least one resource that is idle and that there may be another resource, within the grid, that is busy executing a task and is doing so at a slower rate than what the idle resource is capable of. The optimization phase ends when no more tasks are available from amongst the resources and the project is complete.

The baseline strategy views the allocation of a task as a once-off process in that once a task is sent to a resource it remains with the resource until it is completed. Therefore, the baseline task allocation strategy employs only the deployment phase. The CF strategy complements the baseline task allocation algorithm by implementing the optimization phase, which adds a mechanism that moves tasks from slow resource hosts to faster ones when both (i) no more tasks are available from the project server and (ii) a discrepancy in performance is detected between a busy and an idle resource.

The Representation of Ants

The OCFM includes the concept of ants as individual and independent entities that wander about a defined space and move objects to form clusters. The CF task allocation mechanism discards the notion of ants as independent entities that wander from resource to resource. The first reason for this is that care is taken to avoid unnecessary expenditure of bandwidth. For instance, ants that wander the network must necessarily be transmitted from one host to another and consume bandwidth in the process. The second reason is that a population of ants as software agents would have to be managed so that ants that are lost due to a hardware failure or some similar reason could be replaced. This raises the question of how the replacement of ants is done with the consequence being that the computational grid system becomes more complex.

It is informative to note the implications of the scenario where an independent ant

colony is employed to wander the network. An ant may pick up the address of a busy resource and search for a faster, idle resource where it might drop the address, thus brokering a connection. However, while the ant searches for a better resource, the information it is carrying (the address of the busy resource) ages. The busy resource may complete its task before the ant finds a suitable resource at which it may drop the address.

Alternatively, an ant may pick up the address of an idle resource and search for a slower, busy resource where the ant might drop the address. However, as the ants do not coordinate their activities, there exists the possibility for multiple ants to search on behalf of the same idle resource. While this may decrease the time needed to locate a suitable source for a task, it also makes it possible for multiple ants to service one resource while no ants service another, potentially faster resource. To have resources act as ants, the latency inherent in waiting for optimization to take place is reduced. For the sake of simplicity, the CF task allocation mechanism avoids adding any kind of control process that coordinates the ant colony and replaces wandering ants with a more succinct method, by which the CF resource takes on the role of the OCFM ant. A resource that operates according to the CF task allocation strategy 'wanders' the network by polling its neighbouring resources, which it chooses at random.

The Orientation of The Polling Action

In the OCFM, ants perform two actions: they pick up corpses and they put them down. Likewise in the experimental model, ants record (pick up) the location of a resource and initiate a connection with another resource (put down) to facilitate the transfer of a task. In the grid, a suboptimal task allocation exists when a slow resource possesses a task while a faster resource remains idle. As described previously, a resource in the experimental model assumes the role of the ant and performs its own search by polling neighbours in the network. The polling action can be oriented in two ways.

Firstly, an idle resource polls its neighbours for a task when the project server contains no more tasks in the hope that one of the polling resource's neighbours that does contain a task is slower than the polling resource. This orientation is intuitive because polling messages are transmitted only during the optimization phase. However, in practice, when

no project is queued on the project server, all resources will be idle and all resources will attempt to locate tasks amongst themselves, thus transmitting unnecessary messages. A practical solution to this problem is found in the use of a termination detection algorithm such as the Dijkstra-Scholten algorithm [29]. A termination detection algorithm will allow resources to determine when a project is complete so that no more polling messages are transmitted.

Secondly, and alternatively, the polling action is reversed so that a busy resource attempts to locate a faster, idle resource. A resource that becomes idle will not transmit any polling messages and will wait for an offer of a task that it can complete sooner than the current host of the task. Meanwhile, only those resources that are busy processing tasks will transmit polling messages. When no tasks are available and all resources are idle, no unnecessary polling messages are transmitted. However, this alternative orientation implies that polling messages will be transmitted during both the deployment phase and the optimization phase.

Since this study considers only the optimization phase of a project's execution on a computational grid, the question of which of the two orientations is most suitable to the optimization phase is pertinent. Consider the first orientation whereby idle resources transmit polling messages. Figure 4.2 depicts two resources, one of which is faster than the other. Each resource can be in one of two states: *busy* or *idle*, denoted by B and I in the figure, respectively. In the first time step, both resources are busy processing a task so neither resource transmits a polling message. After the first time step, the fast resource completes its task and becomes idle, while the second resource continues to process its task. The idle resource transmits a polling message to its slower neighbour, which relinquishes its task to the faster resource. After the second time step, the fast resource completes the task that it acquired from its neighbour and, in the third time step, the project is complete. Only one polling message was required during the depicted execution.

Now consider the second orientation whereby busy resources transmit polling messages. Figure 4.3 depicts once again a fast and a slow resource, both of which begin the first time step processing tasks. Both resources transmit a polling message during the first time in the hopes of finding a faster, idle resource but no such resource exists and

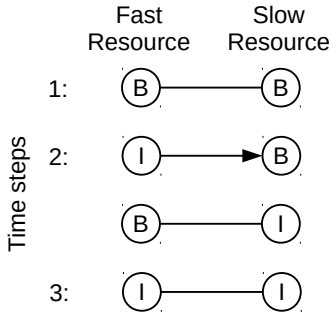

Figure 4.2: Message count for idle resources that transmit polling messages

so both resources continue to process their respective tasks. After the first time step, the fast resource completes its task while the slow resource sends out another polling message. This time around, the faster resource, being idle, accepts the slower resource's offer and acquires that resource's task. As before, the fast resource completes the second task during the second time step and the project is complete at the beginning of time step three. A total of three polling messages were required during the depicted execution.

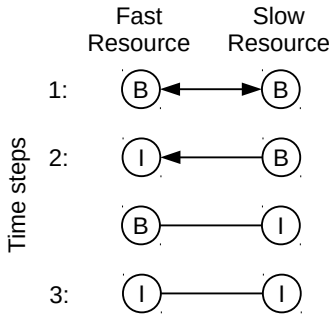

Figure 4.3: Message count for busy resources that transmit polling messages

Since the second orientation of the polling action contributes to a higher message count than the first orientation without affecting the turnaround time of the project, only the first orientation is considered by this study.

The Implementation of Actions

The OCFM defines two actions for each ant: pick up and put down. The probability that an ant will pick up a corpse that it encounters is high when the density of corpses in the ant's vicinity is low. The probability that an ant will put down a corpse that it is carrying is high when the density of corpses in the ant's vicinity is high. The CF task allocation mechanism views the address of a resource as a corpse. An ant picks up the address of a resource and puts down the address at some other resource. Thus a is connection brokered between the two resources so that a task may be transferred from the slower of the two resources to the faster one.

However, as discussed earlier, the CF task allocation strategy transfers the role of the ant to the resource and limits the lifespan of a query to one remote resource. As such it is necessary that the query records its origin. This implies that the pick up action of a resource is deterministic, with the effect that a resource that performs a search always picks up its own address. Therefore, the CF task allocation mechanism implements only the put down action probabilistically.

In order that resources can be compared, the CF task allocation mechanism compares the performance measures of each of the origin and potential destination resource hosts. A difference between the maximum performance of an idle resource and the current performance (with respect to the time at which the comparison is made) of a busy resource indicates what is herein referred to as a *shortfall* in performance. In other words, the shortfall represents a potential improvement in the performance of a task, at a specific point in time. The shortfall is based on the difference between the current performance of the slower resource and the potential maximum performance of the faster resource. The reason that the current performance of the slower resource is used is that the usage of resources by their owners, as described in Section 4.1.1, may thus be detected by the task allocation mechanism. The probability of putting down an address is dependent on the shortfall, with the probability of brokering a connection between two resources being high when the shortfall in performance between those resources is high.

Formally, if S and D are the potential source and destination of a task, respectively, and $W_{max}(R)$ is the maximum work that can be done by resource R per unit of time T,

while $W(R,T)$ is the amount of work being done by resource R at time T, the normalized shortfall s is expressed as:

$$s = \exp\left(-\alpha \frac{W(S,T)}{W_{max}(D)}\right) \tag{4.1}$$

where α determines the steepness of the shortfall. The probability P_{SD} that a connection is brokered between resources S and D is:

$$P_{SD} = \left(\frac{s}{\theta + s}\right)^n \tag{4.2}$$

where θ is a constant and n determines the steepness of the result. This function is the same as that used in the OCFM and replaces the frequency of ant corpses with performance shortfall between the two resources in question.

Summary

The CF task allocation strategy is an adaptation of the original CF model to the experimental model under study. The nature of the computational grid expressed by the experimental model necessitates that the notion of ants and their actions, as described by the original cemetery formation model, be adapted to the experimental model. The task allocation strategy implemented for this study transfers the role of the ant to the resource, which performs the search for suboptimal task allocations that would otherwise be performed by ants wandering throughout the network. The orientation of the two ant actions, pick up and put down, may be varied depending on the behaviour of the system and is addressed in Chapter 6. The probability of brokering a connection is dependent on the performance shortfall calculated between two resources.

Figure 4.4 depicts the states and transitions for a resource operating according to the CF task allocation strategy. The baseline strategy is extended to include a *prospecting* state. When a resource requests a task from the project server but the server does not contain any more tasks, the resource will prospect for tasks amongst the other resources in the network. A resource will remain in the *prospecting* state while the resource queries other resources for available tasks.

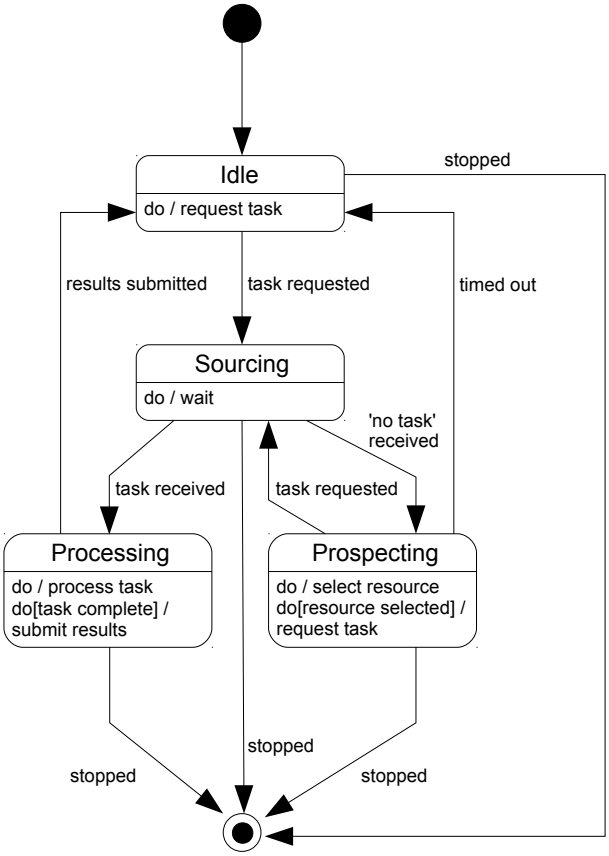

Figure 4.4: State transitions for a resource operating according to the CF task allocation strategy

4.3.2 Parameters of the Task Allocation Mechanism

In addition to the parameters controlling the shape of the probability curve generated by equation (4.2), the CF task allocation mechanism is controlled by one additional parameter. The *prospecting range* determines the number of resources that a prospecting resource will attempt to contact per unit of time. An increase in the prospecting range will increase the message count because more queries will be issued by resources during the optimization phase. Prospecting resources choose remote resources at random so a higher prospecting range, which covers more of the network, increases the chance that a suboptimal task allocation is encountered in a unit of time. However, as fewer tasks

remain toward the completion of a project, the number of remaining tasks cannot satisfy the greater number of prospecting resources and the numerous requests will increasingly contribute to wasted bandwidth. In practice, physical limitations, such as the throughput of the network interface hardware, will establish a ceiling value for this parameter.

4.4 The Division of Labour Task Allocation Strategy

As described in detail in Chapter 3, Bonabeau *et al.* [15] introduced a model of division of labour (DL) based on threshold-response functions and fixed threshold values. The fixed threshold model is used as the basis for the second task allocation mechanism proposed by this study and is henceforth referred to as the original division of labour model (ODLM). In terms of the experimental model used for this study, a resource emits a stimulus while it executes a task. This stimulus is broadcast to the whole network wherein all idle resources compare their performance measures with that reported by the stimulus. Each resource that perceives the stimulus broadcast attempts to acquire the associated task based on a probability that is dependant on the performance shortfall between itself and the resource at which the task is currently located.

The remainder of this section describes the method employed by the DL task allocation strategy to allocate tasks to resources.

4.4.1 The Task Allocation Mechanism

As with the CF task allocation strategy, the characteristics of the distributed computing system expressed by the experimental model necessitate consideration of how the ODLM is expressed in terms of the experimental model. In the case of the DL task allocation mechanism, the role of the ant in task allocation and the originator of the stimulus broadcast, i.e., busy or idle resource, are considered.

The Role of the Ant

An ant in the ODLM performs tasks and a resource in the experimental model does the same. However, while a task in the ODLM attracts multiple ants to perform it, a task in

the experimental model is executed by one resource at a time. A task attracts resources that are capable of processing it at a higher rate than its current resource can. The first idle resource to request a task that is currently being processed causes its host resource to relinquish the task to the requesting resource. Subsequent requests are answered with a response indicating that no task is available.

The Originator of the Stimulus

A stimulus message can be broadcast by either an idle resource or a busy resource. If a busy resource broadcasts a stimulus, then it does so in order to attract the attention of an idle resource that is capable of executing the busy resource's task at a higher rate. The idle resource then requests the task from the busy resource, which transfers the task to the faster host. When multiple resources in the network are idle, the stimulus message is perceived by all of them and they all send a request for the task to the busy resource. The busy resource then picks one of the requesting resources probabilistically, transfers the task to the chosen resource, and sends null messages to the other resources.

Alternatively, an idle resource broadcasts a stimulus in order to attract the attention of a slower, busy resource. A resource that possesses a task will perceive the stimulus and transmit the task to the originator of the stimulus message. When multiple, busy resources perceive a stimulus message, the potential exists for all of the busy resources to send their tasks to the originator of the stimulus at the same time. To prevent the transmission of multiple tasks, the idle resource must first choose one of the resources that expresses an interest in transmitting a task. In order for the idle resource to make a choice, each busy resource must transmit an offer to the idle resource, which will then choose from amongst the offers it receives.

The reasoning about the effect of the stimulus origin on message counts is the same as it is for the CF task allocation strategy's polling message. This implies that, in order to minimize the message count, idle resources should broadcast the stimulus. However, as described previously, due to the fact that the stimulus is transmitted to multiple resources *simultaneously*, a bidding system is required to prevent a resource from being sent multiple tasks. The bidding system incurs four exchanges of messages to complete the reallocation of a task, namely: (i) stimulus broadcast, (ii) bid, (iii) acceptance/declination

of bid, and (iv) transmission of the task. On the other hand, where busy resources broadcast the stimulus, only three exchanges of messages are required: (i) stimulus broadcast, (ii) task request, and (iii) transmission of the task/null message. Therefore, this study considers only the latter origin of the stimulus message.

Stimulus Processing in Detail

A stimulus in the ODLM increases in intensity over time while an insufficient number of ants is performing the task associated with the stimulus. Once enough ants have been diverted to a new task, its associated stimulus either remains constant or it declines. In the DL task allocation mechanism, a resource places the performance measure of its host in the stimulus message. A resource that perceives a stimulus compares the associated performance measure with a measure of its own host's performance. With S and D the potential source and destination of a task, the difference in performance is given by the normalized shortfall, s, as expressed in equation (4.1). The probability, P_{SD}, that resource D will request a task from resource S is:

$$P_{SD} = \frac{s^n}{s^n + \theta^n} \tag{4.3}$$

where θ represents the response threshold parameter and n determines the steepness of the threshold.

Since the actual stimulus is relative to the resource where it is generated, the stimulus is not increased or decreased over time. Instead, the stimulus represents the current performance of the originating resource and therefore changes as the performance of the associated resource changes. Subsequently, P_{SD} increases as the difference between $W_{max}(D)$ and $W(S,T)$ increases, thus allowing idle resources to claim tasks from busy resources that slow down or stop for any reason.

4.4.2 Summary

Within the DL task allocation mechanism, resources take on the role of the ants that perform tasks. Busy resources broadcast stimuli that idle resources may respond to. An

idle resource that reacts to a sufficiently intense stimulus will request the task currently being processed by the associated resource. A task is sent to the first resource that issues the request for the task and each task is processed by a single resource. A resource may be acquired by its owner at an arbitrary point in time, after which point the resource will not process any tasks until the owner releases it.

Figure 4.5 depicts the states and related transitions for a resource operating according to the resulting DL strategy. The states and transitions are identical to those of the CF task allocation strategy. However, while a DL resource is in the *processing* state, it also broadcasts a stimulus message to the other resources in the network. A DL resource that is in the *prospecting* state will consider stimulus messages that it receives.

4.4.3 Parameters

In addition to the parameters controlling the shape of the probability curve generated by equation (4.3), the parameter described here controls the behaviour of the resource state machine depicted in Figure 4.5.

Stimulus Period

While a resource executes a task the resource repeatedly broadcasts a stimulus message. The frequency of the repetition is controlled by the stimulus period. Higher frequencies will allow idle resources to respond sooner to suboptimal task distributions, at the cost of an increase in message count. Lower frequencies reduce the message count but increase the potential duration that task distributions remain suboptimal. The stimulus period determines the interval between stimulus broadcasts.

4.5 Problem Domain

The domain is defined by the network of resources that constitute the distributed computing system. When the resources vary in their performance capabilities, they constitute a non-uniform network. This study considered five cases of performance differences amongst the slowest and the fastest resources in the network. In each case, a range of

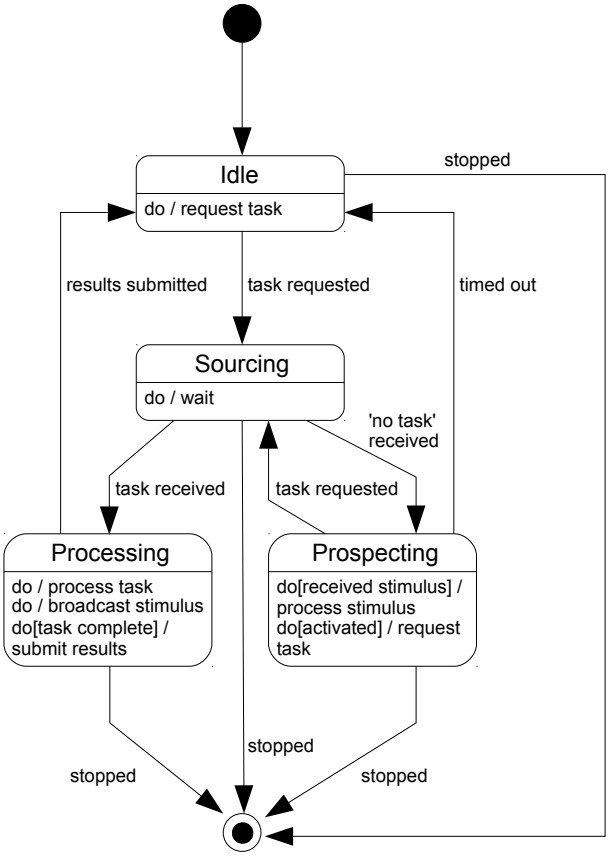

Figure 4.5: State transitions for a resource operating according to the DL task allocation strategy

resource performances was defined and each resource in the simulated network was assigned a performance capability in the specified range, such that the performances were evenly distributed within the network. Table 4.1 depicts the ranges for the five, chosen cases.

Table 4.1: Resource performance values that define the five problem cases considered.

Criterion	Case I	Case II	Case III	Case IV	Case V
Minimum Performance	1	10	100	1	1
Maximum Performance	10	100	1000	100	1000

For all cases, the network consisted of 10 resources and 10 tasks so that each evaluation could begin with the optimization phase of the algorithm. In other words, no tasks were queued on the project server. The durations of the tasks were fixed at 10000 units so that the slowest resource in *Case I* would require 10000 steps to complete a task and the fastest resource in *Case III* and *Case V* would require 10 steps to complete a task.

This study considered the five cases described above and the empirical analyses, described in the following chapters, will make references to the cases, as shown in Table 4.1.

4.6 Summary

This chapter presented the load balancing strategies that are investigated by this study and the experimental model within which the strategies were studied. A baseline strategy was established as a means by which to compare the ant-inspired strategies in terms of turnaround time and number of messages transmitted within the network. Each of the cemetery formation and division of labour strategies was adapted to the experimental model under investigation. The experimental model thus provides a basis for investigating more detailed designs and implementations of the suggested ant-inspired strategies. Finally, the problem domain was described and five cases of resource performance distribution were identified to serve as the scope of the empirical analysis.

The following chapter digresses from the topic of ant algorithms and turns to parameter optimization and the procedure by which the algorithms described in this chapter were tuned.

Chapter 5

The Parameter Optimization Procedure

The cemetery formation and division of labour task allocation strategies each employ a function that determines the probability that a task will be relinquished by one resource and transmitted to another. The curve produced by each function is shaped by a set of parameters. To determine the effect of the probability curves on turnaround time and message count, the aforementioned parameters were explored. The following sections present the technique used to test the effects of the probability curves on the performance of the cemetery formation and division of labour task allocation strategies and describe the procedure used to select suitable parameter values for the empirical study.

Section 5.1 describes the analysis conducted to determine the sensitivity of the turnaround time and message count to the individual input parameters of the two algorithms. Section 5.2 defines the general, parameter optimization problem. Section 5.3 provides an overview of the procedures that were considered to address the optimization problem. Section 5.4 describes how the chosen parameter optimization procedure was modified to suit this particular study, and Section 5.5 concludes this chapter with a summary of the key points.

5.1 Parameter Sensitivity Analysis

As described in Chapter 4, the performance of the cemetery formation and division of labour task allocation strategies is observed by measuring each strategy's turnaround time and message count. Each strategy is configured with a set of values for its respective parameters. Since both strategies were herein proposed for the first time, the effect of each parameter on a strategy's performance could, at best, only be intuited. Therefore, an empirical analysis of parameter sensitivity was carried out and is described in this section.

5.1.1 Choice of Parameter Value Ranges

The range of values for each parameter was chosen to encompass variations of the sigmoid function produced by Equation 4.2 and, where applicable and feasible, to encompass all sensible values. The individual set of values for each parameter is given in Table 5.1. The parameters α, θ, and n are those found in Equation 4.1 and Equation 4.2. The parameter λ controls the prospecting range of each resource, as described in Chapter 4.

Table 5.1: Values chosen for parameter sensitivity analysis of the cemetery formation algorithm.

parameter	**candidate values**
α	$\{\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [1, 9]\}$
θ	$\{0.1\theta \mid \theta \in \mathbb{Z} \wedge \theta \in [1, 9]\}$
λ	$\{\lambda \mid \lambda \in \mathbb{Z} \wedge \lambda \in [1, 9]\}$
n	$\{n \mid n \in \mathbb{Z} \wedge n \in [1, 9]\}$

As with the CF task allocation algorithm, the range of parameter values for the DL algorithm were chosen to encompass variations of the sigmoid function produced by Equation 4.3. The individual set of values for each parameter is given in Table 5.2. The parameters α, θ, and n are those found in Equation 4.1 and Equation 4.3. The parameter γ controls the stimulus period of each resource, as described in Chapter 4.

Table 5.2: Values chosen for parameter sensitivity analysis of the division of labour algorithm.

parameter	candidate values
α	$\{\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [1, 9]\}$
γ	$\{\gamma \mid \gamma \in \mathbb{Z} \wedge \gamma \in [1, 10, 20, ..., 90]\}$
θ	$\{0.1\theta \mid \theta \in \mathbb{Z} \wedge \theta \in [1, 9]\}$
n	$\{n \mid n \in \mathbb{Z} \wedge n \in [1, 9]\}$

5.1.2 Choice of Problem Domains

Each of the CF and DL algorithms was evaluated with the ranges of parameters described in Table 5.1 and Table 5.2, respectively, by each algorithm's application to each of the five problem cases outlined in Chapter 4.

5.1.3 Choice of Problem Instances

In the context of the simulated network, a problem represents the arbitrary initialization and subsequent order of execution of the resources within the network. Each problem was defined by a single number, which was used as the seed for a pseudo-random number generator that determined the initialization and execution orders. Problems for the purpose of optimization were generated by means of another pseudo-random number generator, which essentially produced a sequence of seeds to be used by the simulation. Therefore, the entire class of problems could be determined by a single, primary seed number. The primary seed, 15151003, was selected and kept constant for both algorithms and all cases in the sensitivity analysis. The problems that were generated during the sensitivity analysis were filtered to ensure that no problem was used more than once.

5.1.4 Analysis Procedure

Each of the CF and DL algorithms was executed for every permutation of parameter values in the previously chosen ranges. For each execution - that is, for each permutation of parameter values - the turnaround time and message count were recorded. Each case produced 6561 results.

The sensitivity of each algorithm's outputs to each respective parameter was determined by calculating the Pearson correlation coefficiant (PCC) for each set of outputs (turnaround time and message count) and each parameter.

5.1.5 Results of Analysis

The resulting PCC values for the cemetery formation algorithm are presented in Table 5.3. A visualisation of the results was produced for each parameter by selecting the median PCC value over all cases and plotting the respective performance value (turnaround time or message count) against the respective parameter value. The data for each visualisation was sorted, in ascending order, by the parameter value. Therefore, each visualisation depicts the performance value as the parameter value was increased, within the range described in Table 5.1. A trend line was computed for each visualisation to depict the degree to which a performance value is sensitive to a particular parameter. The direction of a trend line is not as important as the steepness of its gradient, where a high degree of steepness indicates high sensitivity and a low degree of steepness indicates low sensitivity.

The median value over all cases, for each parameter, as correlated with turnaround time, is depicted in Figures 5.1, 5.2, 5.3, and 5.4. Likewise, the median value over all cases, for each parameter, as correlated with message count, is depicted in Figures 5.5, 5.6, 5.7, and 5.8.

Both turnaround time and message count were correlated with each of the four parameters. That is to say that all PCC values were non-zero. However, the correlation between turnaround time and α was significantly weaker than any of the other correlations. The order of parameters, from most correlated to least correlated, with respect to turnaround time is: n, θ, λ, α. The order of the same parameters, from most correlated to least correlated, with respect to message count is: λ, n, α, θ. The aforementioned orders were determined according to the median PCC value over all cases.

Table 5.3: Pearson correlation coefficients for the cemetery formation algorithm. Turnaround time and message count are abbreviated as T and M, respectively.

	T/λ	M/λ	T/α	M/α	T/θ	M/θ	T/n	M/n
Case I	-0.163	0.320	-0.058	-0.299	0.520	-0.191	0.581	-0.305
Case II	-0.278	0.306	-0.037	-0.305	0.495	-0.222	0.560	-0.341
Case III	-0.355	0.357	-0.016	-0.242	0.513	-0.361	0.588	-0.486
Case IV	-0.262	0.347	-0.028	-0.293	0.491	-0.223	0.557	-0.339
Case V	-0.341	0.386	-0.009	-0.236	0.468	-0.355	0.536	-0.478

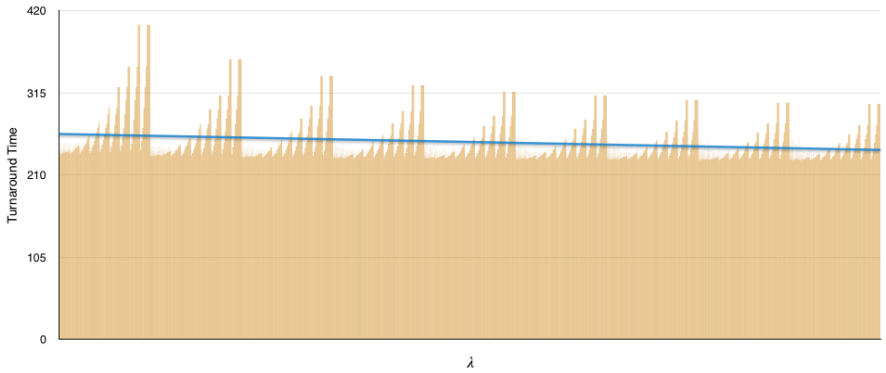

Figure 5.1: Cemetery Formation - Turnaround Time vs. λ - Mean PCC

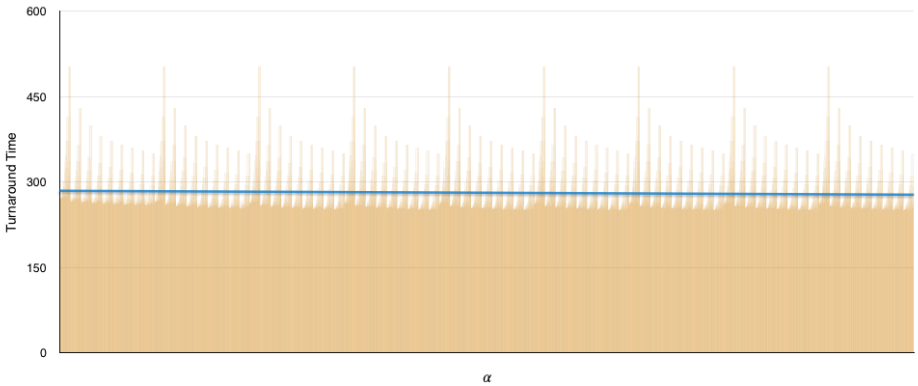

Figure 5.2: Cemetery Formation - Turnaround Time vs. α - Mean PCC

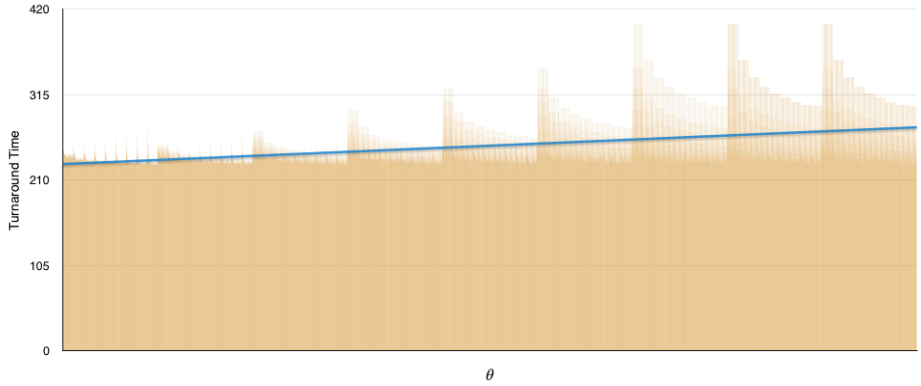

Figure 5.3: Cemetery Formation - Turnaround Time vs. θ - Mean PCC

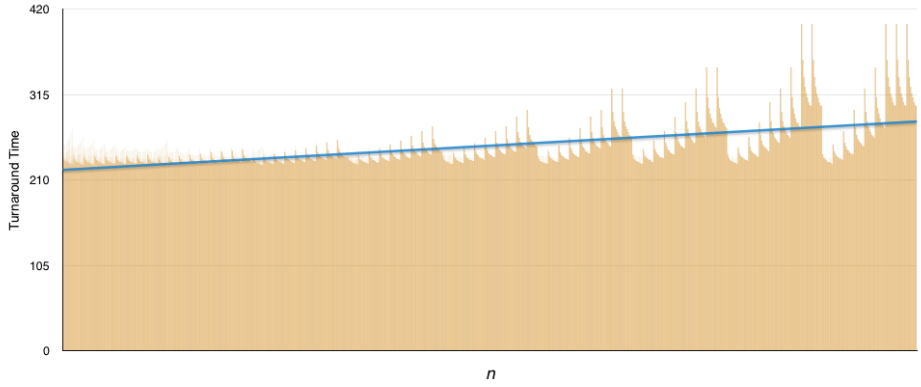

Figure 5.4: Cemetery Formation - Turnaround Time vs. n - Mean PCC

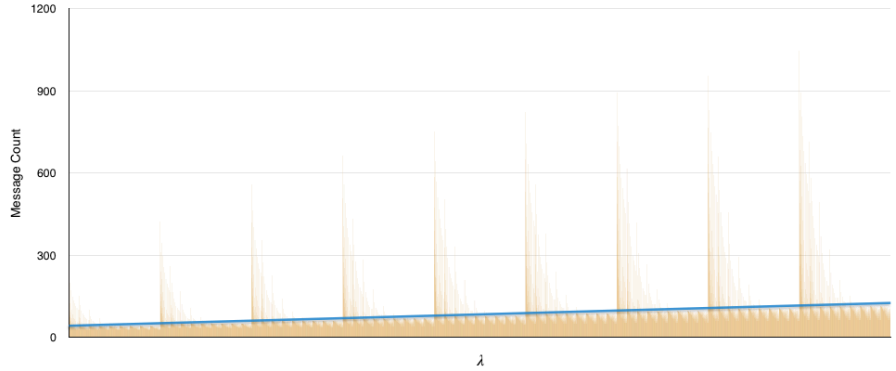

Figure 5.5: Cemetery Formation - Message Count vs. λ - Mean PCC

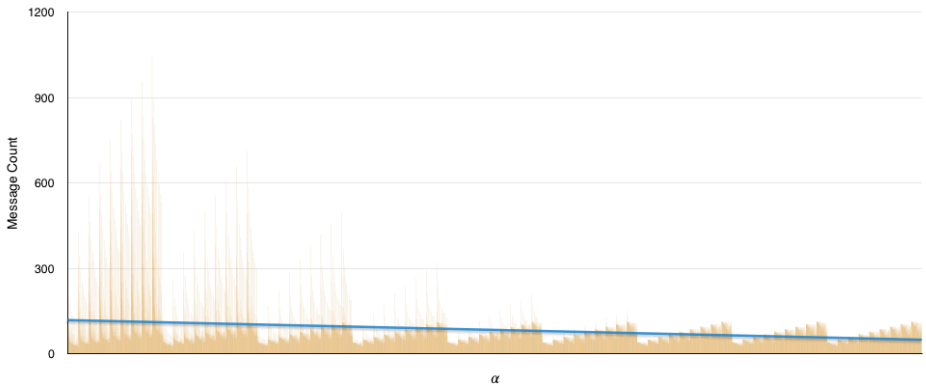

Figure 5.6: Cemetery Formation - Message Count vs. α - Mean PCC

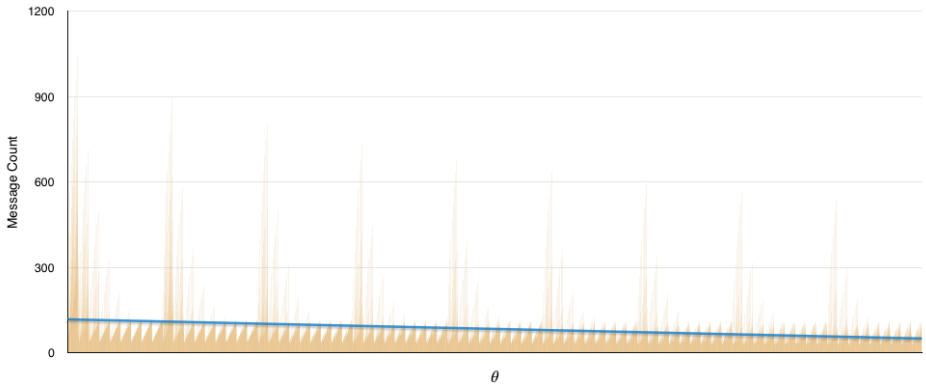

Figure 5.7: Cemetery Formation - Message Count vs. θ - Mean PCC

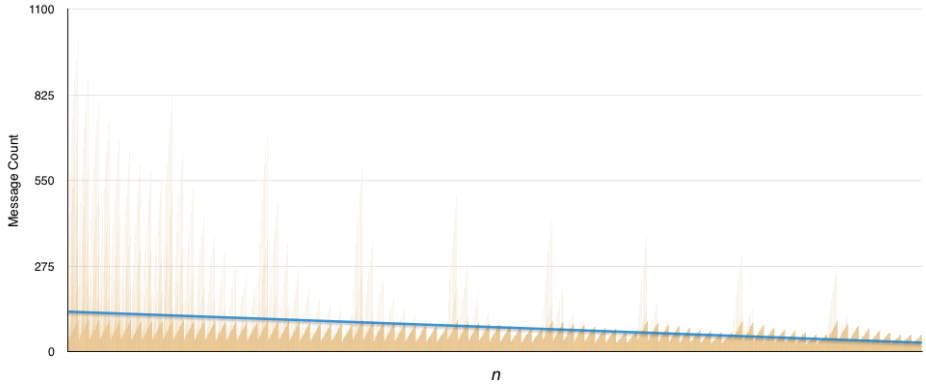

Figure 5.8: Cemetery Formation - Message Count vs. n - Mean PCC

The resulting PCC values for the division of labour algorithm are presented in Table 5.4. The visualisation of the results was produced in the same way as for the cemetery formation algorithm, as previously described. Each visualisation depicts a performance value as the respective parameter value was increased, within the range described in Table 5.2.

The median value over all cases, for each parameter, as correlated with turnaround time, is depicted in Figures 5.9, 5.10, 5.11, and 5.12. Likewise, the median value over all cases, for each parameter, as correlated with message count, is depicted in Figures 5.13, 5.14, 5.15, and 5.16.

Both turnaround time and message count were correlated with each of the four parameters. That is to say that all PCC values were non-zero. Unlike the PCC values for the CF algorithm, those obtained for the DL algorithm were more varied across cases. This suggests that the DL algorithm is more sensitive to the composition of resources in a network than the CF algorithm. The order of parameters, from most correlated to least correlated, with respect to turnaround time is: γ, n, α, θ. The order of the same parameters, from most correlated to least correlated, with respect to message count is: γ, α, θ, n. The aforementioned orders were determined according to the median PCC value over all cases.

Table 5.4: Pearson correlation coefficients for the division of labour algorithm. Turnaround time and message count are abbreviated as T and M, respectively.

	T/γ	M/γ	T/α	M/α	T/θ	M/θ	T/n	M/n
Case I	0.133	-0.395	-0.412	-0.335	-0.035	-0.179	-0.266	-0.103
Case II	0.557	-0.502	-0.120	-0.372	0.393	-0.161	-0.370	-0.138
Case III	0.984	-0.902	-0.008	-0.138	0.002	-0.089	-0.008	-0.027
Case IV	0.579	-0.534	-0.110	-0.352	0.390	-0.154	-0.360	-0.129
Case V	0.999	-0.810	-0.012	-0.197	0.006	-0.134	-0.013	-0.034

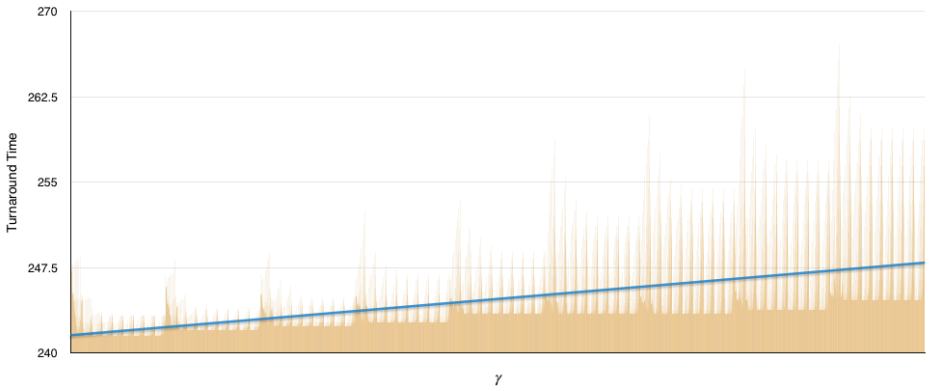

Figure 5.9: Division of Labour - Turnaround Time vs. γ - Mean PCC

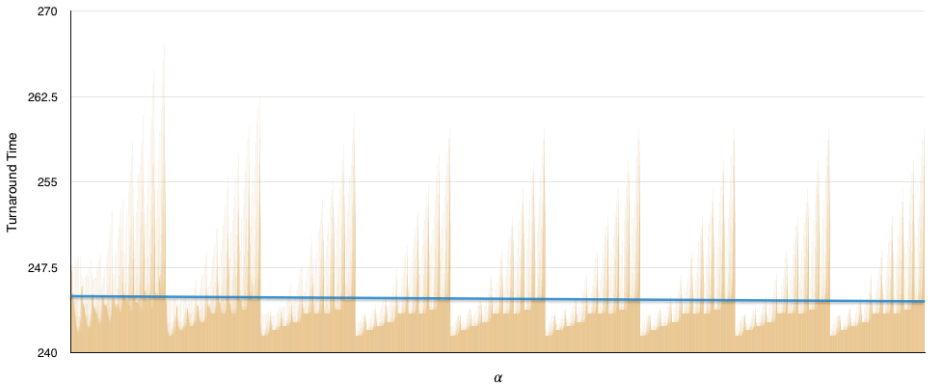

Figure 5.10: Division of Labour - Turnaround Time vs. α - Mean PCC

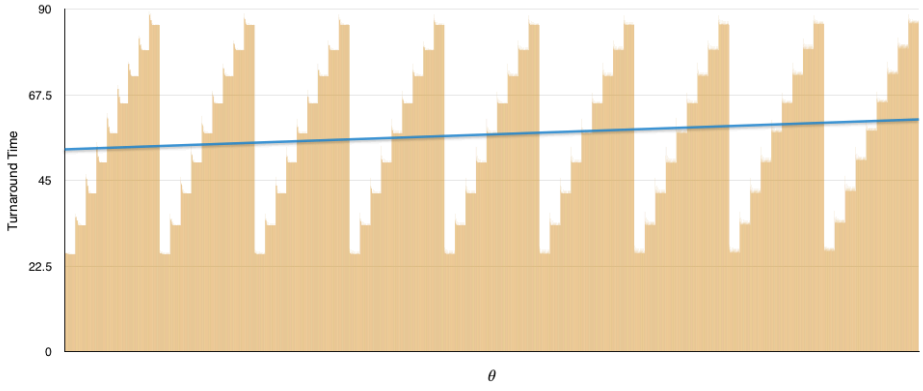

Figure 5.11: Division of Labour - Turnaround Time vs. θ - Mean PCC

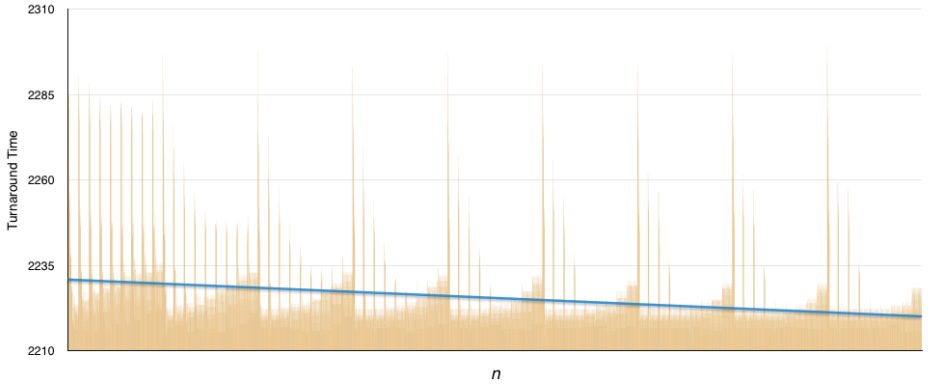

Figure 5.12: Division of Labour - Turnaround Time vs. n - Mean PCC

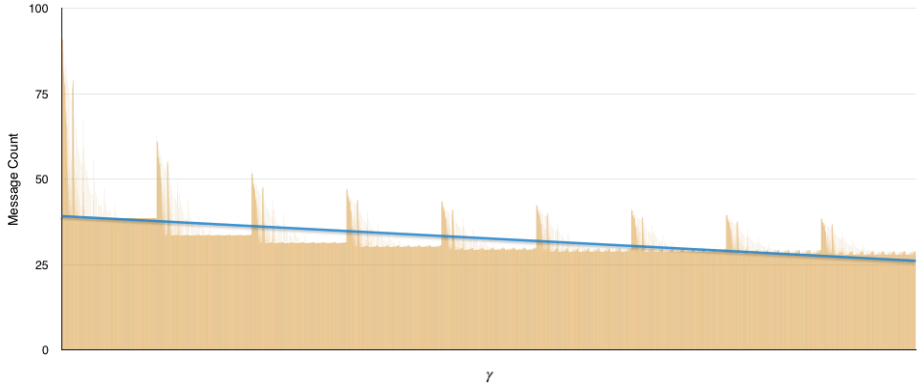

Figure 5.13: Division of Labour - Message Count vs. γ - Mean PCC

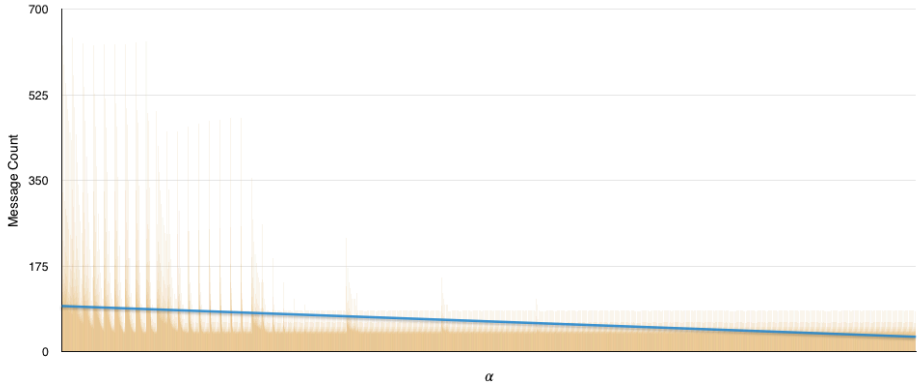

Figure 5.14: Division of Labour - Message Count vs. α - Mean PCC

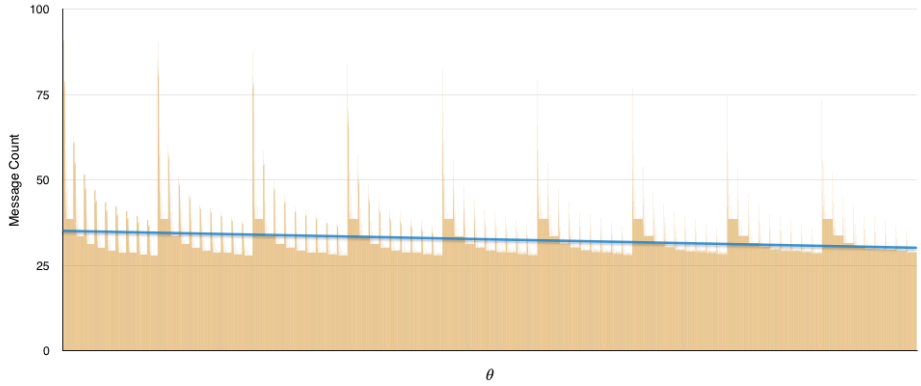

Figure 5.15: Division of Labour - Message Count vs. θ - Mean PCC

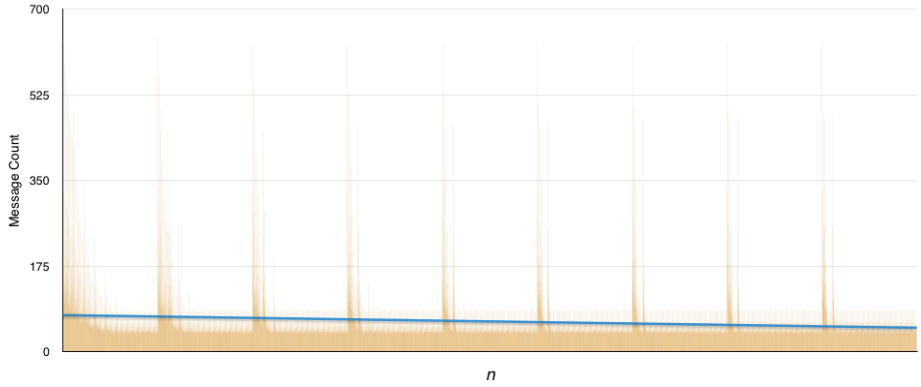

Figure 5.16: Division of Labour - Message Count vs. n - Mean glsPCC

5.1.6 Analysis Remarks

The results of the sensitivity analysis lead to two conclusions. Firstly, the fact that a correlation exists between each output and each parameter, for both algorithms, suggests that good or optimal values for the parameters will have to be determined before either algorithm is employed. However, in practice, a parameter that is weakly correlated with a performance value, as with the α parameter of the CF algorithm, need not be optimised if insufficient time exists to perform the optimisation.

Secondly, while the CF algorithm's PCC values were relatively consistent across cases, those of the DL algorithm were not. This implies that the DL algorithm's parameters will need to be optimised each time that the distribution of resources, with respect to

their performance, changes within the network. In practice, this reduces the potential to add or remove resources arbitrarily. On the other hand, the CF algorithm appears to be less sensitive to resource distribution and a set of good values for the algorithm's parameters will likely remain good as the composition of resources changes.

The study of the cemetery formation and division of labour algorithms proceeds with the optimisation of both algorithms' parameters, which is described in the next section.

5.2 The Parameter Optimization Problem

A metaheuristic algorithm that exhibits free parameters must be tuned to each class of problem to which the algorithm will be applied. The tuning of a metaheuristic requires that a set of values for the metaheuristic's free parameters be chosen and that any component algorithms or procedures employed by the heuristic are selected. The chosen set of values and component algorithms define a *configuration*.

In formal terms, the determination of a metaheuristic configuration requires the following:

1. A problem class that defines the type of problem to which the metaheuristic will be applied. The problem class defines a space, $\mathcal{I}$, from which a possibly infinite number of problem instances, $i \in \mathcal{I}$ are sampled. It is assumed that it will be possible to obtain a *training* sample from $\mathcal{I}$ such that the training sample is representative of the problem class. In other words, the configuration determined by testing the metaheuristic's application to each of the problem instances in the training sample will also be applicable to problem instances not in the training sample.

2. A space, K, from which to sample a possibly infinite number of configurations, $\kappa \in K$.

3. An objective function, F, which expresses the measure by which to determine the optimal or desired performance of the metaheuristic. Therefore, $F(\kappa, i)$ represents the performance of the metaheuristic, as configured according to κ and applied to a problem instance, i.

A suitable configuration, κ', is determined by

$$\kappa' = \arg\max_{\kappa \in, i \in \mathcal{I}} F(\kappa, i) \tag{5.1}$$

The procedure by which configurations and problem instances are sampled and by which κ' is determined depends on the type of the metaheuristic to be optimized, the type of the problem to which the metaheuristic will be applied, and the time available to complete the optimization procedure. Optimization procedures and the factors that influence their choice are discussed next.

5.3 Procedures for Parameter Optimization

The determination of a metaheuristic configuration is subject to several factors that determine how much time will be required to produce a viable, sufficient, or even optimal configuration. These factors include the following:

- The number of free parameters exhibited by the metaheuristic to be configured. The number of combinations of parameter values will increase as the number of parameters increases. While a relatively small number of parameters can be considered by a manual process, beyond some point, the employment of an automated procedure will be required.

- The sensitivity of the parameters. Parameter sensitivity contributes to how many potential values for the parameters are sampled. Highly sensitive parameters will require that many, small increments of values are sampled, while less sensitive parameters will require fewer values to be sampled.

- The type of the parameters of a metaheuristic. Parameters can be either continuous or discrete. The values of continuous-valued parameters can be subdivided *ad infinitum*. Good values may exist within those fractional ranges, making the choice of subdivisions a trade-off between optimization quality and the computational time required to evaluate the smaller increments of values. On the other hand, discrete-valued parameters readily divide a configuration search space.

- The ranges of the parameters. The range of values for a parameter delineate the configuration search space. A large configuration search space will require that a greater number of parameter values are sampled, while a smaller search space will require that a smaller number of values are sampled.

- The availability of expert knowledge of the metaheuristic and problem class. Expert knowledge can be employed to select a configuration that has been empirically derived and published in the relevant literature. If no configuration for a particular problem class is known but the effects and sensitivities of the specific metaheuristic's parameters have been studied and are known, then an appropriate space of potential configurations can be defined and searched. Either way, knowledge of the metaheuristic, the problem, or both can be utilized to constrain the configuration search space to known, feasible areas of configurations, or to obviate a search for configurations altogether.

The size of the configuration search space, along with available computing time, determines what kind of optimization procedure will be feasible to employ in order to configure the metaheuristic for a particular application. Two common procedures are discussed next and thereafter the procedure chosen for this study is introduced.

5.3.1 Brute Force Search

A brute force search is an exhaustive evaluation of every configuration that is sampled from a defined configuration search space. The advantage of a brute force search is that a definitive set of suitable configurations can be determined. However, available computing time will define an upper bound on the size of the configuration search space that is evaluated. Therefore, a brute force search is unlikely to be feasible when the search space is large or infinite.

5.3.2 Full Factorial Design

The number of potential configurations can be reduced or the search space can be expanded if configurations are sampled at some, non-zero interval. full factorial design

(FFD) [28] is a procedure for evaluating combinations of factors, f, that are sampled at different levels, b, to produce b^f evaluations. By separating tests into levels, it is possible to discern the effect of each factor on the response variable without having to test factors independently and exhaustively. Furthermore, a test that yields the sought after result might point to an area within the configuration search space where similar or better configurations are to be found. A subsequent series of tests can then be executed for configurations that are sampled from only the area near to or around the previously ascertained, promising configuration. A search for good configurations is guided by the researcher and is therefore limited in scale by a researcher's ability to consider the number of tests that will result from a large number of factors.

5.3.3 F-Race

F-Race [11] is an automated parameter optimization procedure that repeatedly tests a set of configurations and employs a statistical significance test to eliminate configurations that produce low values for F, as described in Section 5.2. The initial set of candidate configurations is chosen according to the previously described factors of parameter number, parameter type, and the availablity of expert knowledge. Each candidate configuration is then evaluated by its application to a single problem instance and the performance of each configuration is recorded. Evaluation proceeds in steps, with each step employing a new problem instance to test the set of candidate configurations. Table 5.5 depicts the stepwise evaluation of configurations by F-Race.

At the end of each step, F-Race employs the Friedman test for variance by ranks [24] in order to determine if there is a statistically significant difference between at least one pair of candidate configurations, in terms of F, in the current set of configurations. The Friedman test obviates the need to compare all possible pairs of candidates, which would be computationally time-consuming as well as wasteful in situations when no differences exist. If the Friedman test indicates that a difference exists in the set of candidates (at a confidence level of 95%), then F-Race selects the candidate with the lowest rank as the best-performing candidate, and compares all other candidates with the chosen elite candidate, in terms of F, using the Wilcoxon signed ranks test [24]. If a statistically significant difference is detected between a candidate and the elite candidate, then the

Table 5.5: Visualization of candidate configurations, problem instances, and results for m candidate configurations and k problem instances.

$\mathcal{I}/$	κ_1	κ_2	$\dots$	κ_m
i_1	$F(\kappa_1, i_1)$	$F(\kappa_2, i_1)$	$\cdots$	$F(\kappa_m, i_1)$
i_2	$F(\kappa_1, i_2)$	$F(\kappa_2, i_2)$	$\cdots$	$F(\kappa_m, i_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
i_k	$F(\kappa_1, i_k)$	$F(\kappa_2, i_k)$	$\cdots$	$F(\kappa_m, i_k)$

former is discarded. Once all comparisons are completed, the next step is executed with the set of surviving candidate configurations.

The F-Race procedure is stopped when either a pre-determined budget of evaluations of F is exhausted or when a specified, minimum number of surviving candidates remain. If multiple candidates remain when the procedure is terminated, then either the surviving candidates are indistinguishable with respect to F or additional testing will eventually discard the remaining, lesser candidates. Either way, the result of the procedure is not necessarily the determination of the optimum candidate, which may be present in the configuration search space but was not sampled for the initial candidate set.

Balakaprash *et al.* [8] proposed an iterative method, *I/F-Race*, that is intended to refine a search for good configurations by repeatedly narrowing the space of configurations. I/F-Race joins multiple iterations of the F-Race procedure such that the initial candidate set in all but the first iteration is sampled around an elite, surviving candidate from a previous iteration. Birattari *et al.* [14] suggested choosing an elite set of survivors by $N_e = min\{N_{survive}, N_{min}\}$ where $N_{survive}$ is the number of candidates remaining after the previous iteration of F-Race completes and N_{min} is a predetermined, desired number of survivors. The elite candidates are weighted by

$$w_z = \frac{N_e - r_z + 1}{N_e \cdot (N_e + 1)/2} \tag{5.2}$$

for $z = 1, \ldots, N_e$ and where r_z is the rank of an elite configuration. One of the elite survivors is then chosen with a probability that is proportional to w_z and each new candidate configuration, $\mathbf{x} = \langle x_1, x_2, \ldots, x_Q \rangle$, is sampled around the chosen elite candidate, $\mathbf{x}_z = \langle x_{z,1}, x_{z,2}, \ldots, x_{z,Q} \rangle$, where Q is the number of parameters. Each component, $x_{z,q}$, is sampled according to a normal distribution with $x_{z,q}$ used as the mean and $\sigma_{l,q}$ as the standard deviation, defined by

$$\sigma_{l+1,q} = v_q \cdot \left(\frac{1}{N_{l+1}} \right)^{\frac{1}{Q}} \tag{5.3}$$

for $l = 1, \ldots, L-1$ and where L is the number of iterations, Q is the number of components of a configuration, and v_q is the range of the component x_q. The elite candidate is included with the newly sampled candidates and all of these candidates are tested again. The implication of this design is that the bias of the sampling distribution towards the elite candidate is increased as the number of components (or parameters) is increased and as the number of candidate configurations to be sampled is increased.

The computational budget, B, is distributed over all iterations according to

$$B_l = \frac{B - B_{used}}{L - l + 1} \tag{5.4}$$

where $l = 1, \cdots, L$ and B_{used} is the computational budget used up to and including iteration $l - 1$.

Each iteration of F-Race is stopped when at most N_{min} candidate configurations remain, where

$$N_{min} = 2 + round(log_2 Q) \tag{5.5}$$

The following are the advantages of F-Race and I/F-Race:

- a large number of candidate configurations are evaluated in a structured procedure,

- the resulting configuration is justifiable by virtue of it having been selected by the results of statistical significance tests,
- the means by which an initial candidate set is sampled can be determined by the researcher, thus accommodating expert knowledge, and
- the computational cost of the procedure can be controlled by an evaluation budget and, in the case of I/F-Race, by limiting the number of iterations.

However, when the need to find a good or optimal configuration outweighs the need to limit the amount of time to be spent on searching for a good configuration, the choice of an evaluation budget is no longer justifiable. The following section describes a novel extension to the F-Race procedure, proposed by this author, to provide a reasonable stopping condition.

5.4 A New F-Race Termination Condition

When the constraint of minimising the time required to perform the tuning procedure can be relaxed in favour of minimising the number of surviving candidates, the choice of a computational budget for F-Race becomes arbitrary.

Firstly, it stands to reason that if all of the candidate configurations selected for a tuning procedure are statistically significantly distinguishable with respect to the algorithm to be tuned and the problem instances upon which the configurations are evaluated, then the tuning procedure can be run until only one candidate configuration remains. However, if two or more candidates are not distinguishable, then an execution of the tuning procedure with an unlimited budget will run indefinitely. Since chosen candidate configurations cannot be distinguished before the tuning procedure is executed, neither an outcome of one surviving candidate nor indefinite execution can be predicted.

Secondly, even when it is possible to execute a tuning procedure until only one configuration remains, the computational time required to do so may nevertheless be infeasible. The following tuning procedure was executed in order to observe the p-values produced by the Friedman test after each step. A set of instances of the travelling salesman problem was produced such that each instance contained 100 nodes, randomly distributed

on a 1000×1000 unit grid. The ant system (AS) variant of the ant colony optimization metaheuristic was chosen for its relative simplicity as the algorithm to tune and the candidate configurations were chosen according to Table 5.6.

Table 5.6: Initial set of candidate configurations chosen for the observation of p-value generation.

parameter	**candidate values**
α	$\alpha \in \{0.1\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [1, 10]\}$
β	$\beta \in \{\beta \mid \beta \in \mathbb{Z} \wedge \beta \in [1, 10]\}$
ρ	$\rho \in \{0.1\rho \mid \rho \in \mathbb{Z} \wedge \rho \in [1, 10]\}$

As suggested by Dorigo and Stützle [27], the value for τ_0, the initial pheromone value, was determined by calculating $\bar{C}^{nn}$, the mean shortest path over all problem instances, as calculated by the nearest-neighbour heuristic for each instance, and computing $m/\bar{C}^{nn}$, where m is the number of ants. The number of ants was fixed and equal to the number of nodes. Each run of AS was limited to 10 iterations.

The tuning procedure was executed until only one candidate configuration remained. A new set of candidate configurations was chosen by reducing the ranges of possible parameter values such that each new configuration was much closer (in Euclidean space) to the reference candidate. The reduced ranges of the new candidate configuration parameter values are described in Table 5.7. The aim of choosing the new set was to test candidates that were potentially difficult to distinguish from each other by statistical significance.

Table 5.7: Subsequent set of candidate configurations chosen for the observation of p-value generation.

parameter	**candidate values**
α	$\alpha \in \{0.01\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [90, 100]\}$
β	$\beta \in \{0.1\beta \mid \beta \in \mathbb{Z} \wedge \beta \in [4, 6]\}$
ρ	$\rho \in \{0.01\rho \mid \rho \in \mathbb{Z} \wedge \rho \in [50, 70]\}$

Figure 5.17 depicts the p-values produced in the first 20 000 steps of the second

execution. Initially, before sufficient evidence to distinguish the candidates has been gathered, the p-values appear close to 1.0. As more problems are evaluated, the p-values decline and when the p-value drops below the critical value (0.05 in this case), the candidates are evaluated in pairs and those that are determined to be statistically significantly different from the chosen best candidate thus far are discarded. The step immediately following a discard once again bears little evidence to distinguish the new set of surviving candidates so the p-values increase. The crucial observation to be made is that the number of steps that elapse between discards of candidates increases after each discard. This occurs as the poor candidate configurations are discarded, leaving increasingly similar candidates to be tested. Consequently, a larger number of problems must be evaluated in order to gather sufficient evidence to distinguish the remaining candidate configurations.

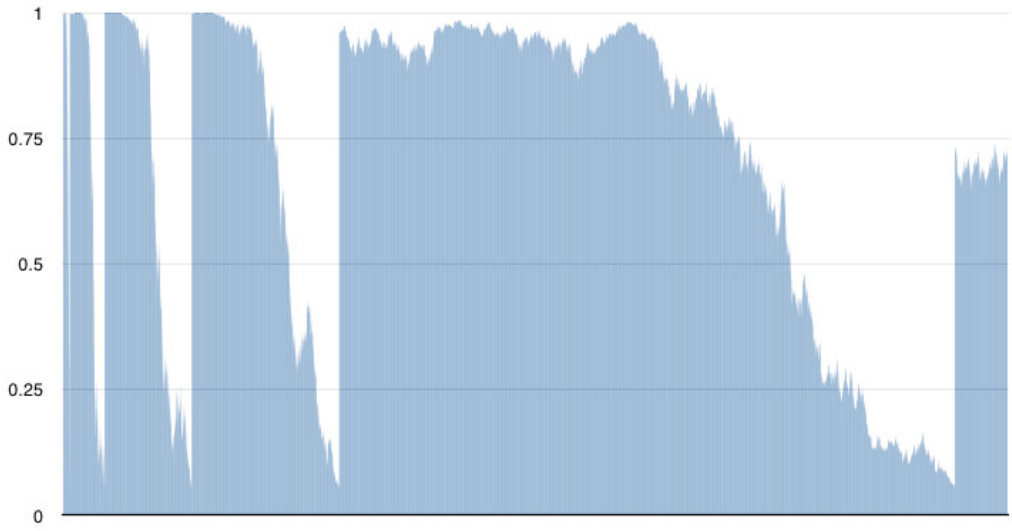

Figure 5.17: p-values for the first 20 000 steps.

Figure 5.18 depicts the p-values produced in the last 20 000 steps of the second execution. No discards take place in the depicted set of steps and although the depicted p-values are decreasing, the rate of decrease is now much lower than that of the p-values produced during the initial 20 000 steps. Naturally, it is not possible to know how many steps will be required to distinguish all of the candidates before the tuning procedure

is executed. If the time required to evalute one configuration upon one problem is relatively high, then the amount of time required to complete the entire execution can be unacceptably large even though one might initially have decided that computational time is not an important constraint. Therefore, it is not possible to choose a computational budget *a priori* such that an appropriate number of steps is completed and the tuning procedure is terminated before the number of steps required to reach the next discard potentially becomes either infeasible or infinite.

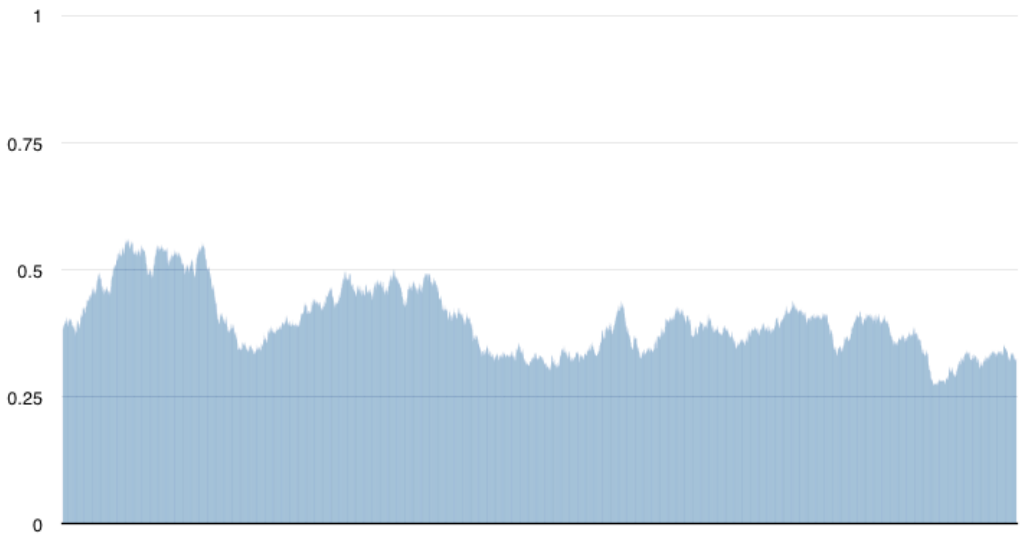

Figure 5.18: p-values for the last 20 000 steps.

In order to provide a justifiable stopping condition for the optimization of the load balancing algorithms proposed by this study, a heuristic termination mechanism was added to F-Race, as described next.

5.4.1 Overview of the Termination Heuristic

F-Race employs the Friedman test for variance by ranks to determine if a statistically significant difference with respect to f exists within a set of candidate configurations. The termination heuristic, proposed in [36], makes use of the p-values of the Friedman test computed since the last step wherein candidates are discarded as an indicator of the likelihood that statistically significant differences will be found in further testing.

The termination heuristic gathers a minimum number of p-values in order to form a sample from which to compute a trend. The trend is determined by means of least squares linear regression, whereby a line is fitted to the sample of p-values. The optimization procedure is permitted to continue as long as the slope of the trend line is decreasing. Once the slope of the line becomes absolutely constant or begins to increase, the optimization procedure is stopped. The size of the p-value sample is increased by adding to the sample the p-value of each new test as long as no candidates are discarded. The p-value sample is discarded whenever candidates are discarded and a new sample is built and evaluated in the subsequent F-Race steps.

The minimum size of the p-value sample is a constant and is not used as a parameter. The p-value sample size is related to a computational budget in that changes to the minimum sample size would be subject to the same *a priori* estimates. For this study, a minimum of ten p-values were sampled before the termination condition was evaluated.

The termination heuristic was evaluated upon a single, customized instance of the travelling salesman problem (TSP) in [36] to determine the viability of the heuristic method. Thereafter, additional instances of the TSP were evaluated in order to constitute a more substantial body of evidence to support and characterize the termination heuristic. The additional empirical analysis is described next.

5.4.2 Empirical Analysis

The fundamental behaviour of the termination heuristic that determines the heuristic's feasibility is the heuristic's use of computational time. The ideal case is one where the heuristic uses only as much computational time as is required to differentiate all but one candidate or a group of similar candidates. The empirical analysis was therefore begun by posing the following questions:

1. If a set of candidate configurations contains statistically significantly distinguishable candidates, then does the termination heuristic allow for enough computational time to elapse in order to differentiate the candidates at least once?

2. If a set of candidate configurations contains statistically significantly distinguishable candidates, then does the termination heuristic stop the optimization proce-

dure after the candidates have been reasonably distinguished, leaving either only one candidate or a group of very similar or identical candidates?

To determine the behaviour of the termination heuristic with respect to the above questions, a series of trials was run. The goal of each trial was to determine a good configuration for an application of the AS variant of the ant colony optimization (ACO) metaheuristc [27] to an instance of the TSP. A configuration's fitness was measured by the length of the shortest path reported after an execution of the AS algorithm according to the respective configuration. Therefore, in relation to Equation (5.6), a good configuration, κ', was determined by

$$\kappa' = \arg\min_{\kappa\in,i\in\mathcal{I}} F(\kappa, i) \tag{5.6}$$

where κ is a configuration, F is the AS algorithm, $\mathcal{I}$ is the TSP problem set, and i is a problem instance chosen from $\mathcal{I}$.

The implementation of F-Race used for this study evaluated ten problem instances before beginning testing using the Friedman test. The reason for this is that the Friedman test statistic is approximated by the χ^2 distribution when the number of ranks (candidate configurations) is three and the number of observations (problem instances) is greater than nine and when the number of ranks is four and the number of observations exceeds four [32]. Therefore, in order to ensure a good approximation by the Friedman statistic of χ^2, the minimum number of evaluations performed before statistical testing was begun was ten times the number of configuration candidates. The significance level used for the Friedman test was set to 95.0% and that of the Wilcoxon signed ranks test was set to 97.5%, as described by Birattari *et al.* [13] (later published in [12]).

F-Race was executed four times for each trial. The first execution employed the proposed termination heuristic and the number of evaluations made during the execution was noted and used as the reference budget for the subsequent, three executions. The remaining executions were performed without the termination heuristic and with a constant budget of evaluations. As in [36], the constant budgets were multiples of the reference budget and were determined by the multipliers, 0.0, 0.5, and 2.0. A budget multiplier of 0.0 indicates that only the minimum number of evaluations were performed,

as described in the preceeding paragraph, and that no budget was allocated beyond the minimum number of required evaluations.

No limit on the number of surviving candidates was set and each execution was terminated when the computational budget was exhausted, when the termination heuristic determined that no further testing should be conducted, or when only one configuration remained. The candidate configuration that survived after each execution was selected as the best configuration for the associated budget. If more than one candidate configuration survived, then the candidate with the lowest rank, by the Friedman test, was selected as the best configuration.

Three parameters of the AS algorithm were chosen for optimization: pheromone influence, heuristic information influence, and pheromone evaporation rate, which in [27] are α, β, and ρ, respectively. The ranges of values from which initial candidates were generated were informed by the suggested settings for ACO algorithms without local search in [27]. The initial set of candidates was sampled using FFD, as described in Subsection 5.3.2, and each parameter range was divided evenly into a number of levels. A total of 5184 candidates were sampled. Table 5.8 lists the AS parameters that were chosen for optimization and shows the range and number of levels for each parameter.

Table 5.8: Initial set of candidate configurations chosen for the comparison of budgets.

Parameter	Range	Levels
α	[0.0,1.0]	6
β	[1.0,11.0]	6
ρ	[0.0,1.0]	6
τ_0	[0.0,1.0]	6
g (number of ants)	[1,100]	4

Furthermore, each run of the AS algorithm was allowed to perform ten iterations and to execute to completion. The distance of the shortest path reported after the ten iterations were completed was taken as the fitness value by which the candidate configuration was evaluated.

The problem class used for the trials was a set of TSP instances that were generated at random. Each instance consisted of 100 nodes in a fully connected graph in a two-dimensional space of size u^2 for $0 \leq u \leq 1000$ and $u \in \mathbb{Z}$. Euclidean distances between nodes were computed by the AS implementation. Each execution in a trial started with the same problem instances and employed as many problem instances as required, up to termination. Executions within the same trial employed the same problems instances up to the point where their budgets diverged. This ensured that executions were identical if their budgets were identical and differed if and only if their budgets differed and only after the point at which their budgets differed. However, problem instances were not reused amongst trials. In total, 100 trials were completed to produce 100 configurations for each of the four budgets. For each budget, the 100 selected configurations were each evaluated with 100 problems to produce 10 000 shortest path results. Each candidate was evaluated on a different set of 100 problems. However, the same 10 000 problems were used for each budget. The budget variations were then compared in terms of their associated sets of 10 000 shortest path results using the Student's t-test.

Table 5.9 lists the median, mean, and standard deviation of the shortest paths obtained by the configurations for each budget variation.

Table 5.9: Configuration evaluation results for budget multipliers 0.0, 0.5, 1.0, and 2.0, where the multiplier 1.0 refers to the reference budget.

Budget Multiplier	Median	Mean	Standard Deviation
0.0	8656.9650	8652.2766	360.0252
0.5	8644.6200	8640.7622	362.7801
1.0	8636.4850	8629.0478	363.8898
2.0	8630.5100	8628.6663	363.1426

As was expected, the mean and the median decrease as the budget increases. The reason for this is that a larger computational budget allows F-Race to perform a larger number of evaluations, which serve to distinguish the better performing configurations from other configurations in the set of candidates. In other words, where candidates were

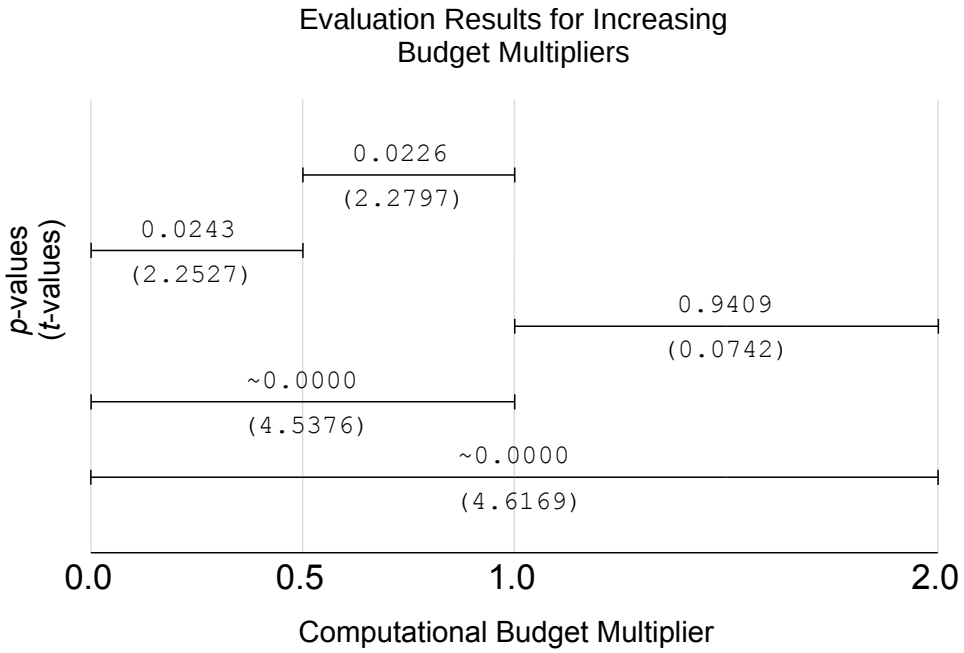

Figure 5.19: Configuration comparison results for budget multipliers 0.0, 0.5, 1.0, and 2.0, where the multiplier 1.0 refers to the reference budget.

evaluated more often, more problem instances were tested. A larger number of problem instances constitutes a more accurate sample of the problem class and therefore serves as a more accurate way to differentiate the candidate configurations.

Figure 5.19 depicts the values for the comparisons of the configuration evaluations for the four budget variations. Each horizontal bar between two budget variations indicates that those variations were compared. The label above each horizontal bar depicts the p-value for the Student's t-test and the label below the bar depicts the value of t.

The results of the statistical significance tests yielded the following observations:

1. The configurations for the budget multipliers of 0.0 and 2.0 produced statistically significantly different sets of shortest paths.

2. The configurations for the budget multipliers of 0.0 and 1.0 produced statistically significantly different sets of shortest paths.

3. The configurations for the budget multipliers of 1.0 and 2.0 were not statistically significantly differentiable.

4. The configurations for the budget multipliers of 0.0 and 0.5 produced statistically significantly different sets of shortest paths.

5. The configurations for the budget multipliers of 0.5 and 1.0 produced statistically significantly different sets of shortest paths.

The first observation shows that the candidate configurations were differentiable at least once. The second observation shows that the termination heuristic allowed F-Race to continue to evaluate candidates until the candidates were differentiated at least once, thus answering the first of the two questions posed for the empirical analysis. The third observation shows that the termination heuristic tended to stop F-Race from performing evaluations at a point when the candidate configurations had been reasonably differentiated. However, a caveat should be noted here. The p-value for the comparison of multipliers 1.0 and 2.0 suggests that, in at least some of the trials, a better configuration might have been ascertained, given enough of a computational budget. With reference to the second question posed for the empirical analysis, what is 'reasonable' here depends on how much real time a larger computational budget implies. For the purposes of this study, the fact that twice the reference computational budget did not yield a superior configuration was sufficient to accept the configuration yielded by the use of the termination heuristic.

It should be noted that even though a statistically significant difference was observed between multipliers 0.0 and 1.0 and that none was observed between multipliers 1.0 and 2.0, the number of evaluations performed under the guidance of the termination heuristic is not necessarily optimal. Therefore, a multiplier of 0.5 was used to determine if candidates might have been reasonably differentiated sooner than what the termination heuristic suggests. The fourth and the fifth observations above show that the candidate configurations were differentiable at least twice before the termination heuristic stopped F-Race. If F-Race had been stopped at some point between the multipliers of 0.5 and 1.0, the second differentiation might not have taken place. Once again, this observation shows that the termination heuristic did not use an optimal amount of computational

time, but that the computational time spent by the tuning procedure was reasonable.

5.5 Summary

The overview of optimization presented in this chapter provided an introduction to the problem of finding good parameters for a metaheuristic or any other algorithm that requires configuration prior to its application. The optimization problem was stated in formal terms, after which the factors that influence a potential solution to the problem were discussed. While the approaches to parameter optimization presented in this chapter are not comprehensive, they do provide an illustration of the challenges of sampling a parameter search space and evaluating parameters within a feasible period of time. The F-Race procedure was presented as a means by which to address the aforementioned challenges of parameter optimization. Thereafter, an extension to F-Race was introduced in order to provide a justifiable way to terminate an optimization procedure without compromising the search for a good configuration. The proposed extension was evaluated empirically and the results showed that the extension allowed for a sufficient amount of computational time to be expended in order to find a good configuration. Subsequently, the proposed extension was accepted for use in this study.

Chapter 6

Comparison of The Proposed Load Balancing Algorithms

In order to determine the viability of the proposed task allocation algorithms, each algorithm is herein compared to a baseline. Section 6.1 describes the procedure used to select the parameters for the load balancing algorithms. Section 6.2 details the design of the experiments employed to compare the algorithms. Section 6.3 presents the results of the comparisons and Section 6.4 discusses the findings. Section 6.5 describes the method that was used to investigate and compare the scalability of the cemetery formation and DL algorithms. Section 6.6 presents the results of the scalability analysis. Section 6.7 posits remarks on the findings. Section 6.8 summaries this chapter.

6.1 Parameter Optimization

For each algorithm, a set of parameter values was found in order to minimize the *turnaround time* and another set of values was found in order to minimize the number of task transmissions, henceforth referred to as the *message count*. F-Race, as described in Chapter 5, was used to perform the optimization. The following subsections describe each aspect of the optimization procedure in detail and the resulting parameter values that were obtained.

6.1.1 Problem Instances

As described in Chapter 5, a problem is defined by a pseudo-random number generator seed and seeds are produced by a primary random number generator. The primary seed, 1212141827, was selected prior to the commencement of optimization and testing and was kept constant for the remainder of the study.

The problems that were generated during the optimization procedure were filtered to ensure that no problem was used more than once. All of the problems used during optimization were recorded so that a new, unique set of problems (from the same seed) could be generated for the purpose of testing, after optimization was completed.

6.1.2 Parameter Search Space

The initial set of parameters (candidates) to evaluate was chosen by means of full factorial design, which was informed by knowledge of the proposed task allocation algorithms. The initial set of candidates for both the minimization of *turnaround time* and *message count* within the CF task allocation algorithm were chosen to encompass variations of the sigmoid function produced by Equation 4.2. The individual set of values for each parameter is given in Table 6.1. The parameters α, θ, and n are those found in Equation 4.1 and Equation 4.2. The parameter λ controls the prospecting range of each resource.

Table 6.1: Initial set of candidate configurations chosen for the CF task allocation algorithm.

parameter	candidate values
α	$\{\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [1, 10]\}$
θ	$\{0.01\theta \mid \theta \in \mathbb{Z} \wedge \theta \in [1, 100]\}$
λ	$\{\lambda \mid \lambda \in \mathbb{Z} \wedge \lambda \in [1, 9]\}$
n	$\{n \mid n \in \mathbb{Z} \wedge n \in [1, 10]\}$

As with the CF task allocation algorithm, the initial set of candidates for the DL algorithm were chosen to encompass variations of the sigmoid function produced by Equation 4.3. The individual set of values for each parameter is given in Table 6.2. The

parameters α, θ, and n are those found in Equation 4.1 and Equation 4.3. The parameter γ controls the stimulus period of each resource.

Table 6.2: Initial set of candidate configurations chosen for the DL task allocation algorithm.

parameter	candidate values
α	$\{\alpha \mid \alpha \in \mathbb{Z} \wedge \alpha \in [1, 10]\}$
γ	$\{\gamma \mid \gamma \in \mathbb{Z} \wedge \gamma \in [1, 10, 20, ..., 100]\}$
θ	$\{0.01\theta \mid \theta \in \mathbb{Z} \wedge \theta \in [1, 100]\}$
n	$\{n \mid n \in \mathbb{Z} \wedge n \in [1, 10]\}$

6.1.3 F-Race Configuration

The F-Race procedure was augmented with the termination heuristic described in Section 5.4.1 and, in each case, the procedure was run for one iteration only. The best parameter configuration was chosen from amongst the surviving candidate configurations by picking the configuration with the lowest rank sum. F-Race evaluated at least ten problems before commencing statistical significance testing. The termination heuristic's minimum sample size was also set to ten. To ensure that the load balancing algorithm, as fitness function, was evaluated accurately the algorithm was executed 100 times for each configuration, per problem, and the mean fitness value was utilized by F-Race to rank the candidates.

6.1.4 Resulting Parameter Values

The candidates chosen at the end of the optimization procedure in each case, for each algorithm, to favour either *turnaround time* or *message count* are depicted in Tables 6.3, 6.4, 6.5, and 6.6.

The chosen parameter values were used to conduct the evaluations and comparisons of the CF and DL task allocation algorithms, which are presented next.

Table 6.3: Chosen parameter values for the cemetery formation algorithm to favour *turnaround time*.

Case	α	θ	λ	n
I	9	7	0.15	1
II	9	7	0.10	1
III	9	7	0.10	1
IV	9	7	0.13	1
V	9	7	0.12	1

Table 6.4: Chosen parameter values for the cemetery formation algorithm to favour *message count*.

Case	α	θ	λ	n
I	1	1	0.59	10
II	1	1	0.59	10
III	1	1	0.59	10
IV	1	1	0.59	10
V	1	1	0.59	10

Table 6.5: Chosen parameter values for the division of labour algorithm to favour *turnaround time*.

Case	α	θ	γ	n
I	10	1	0.58	7
II	10	1	0.59	10
III	1	1	0.59	10
IV	10	1	0.59	10
V	1	1	0.57	8

6.2 Comparison Design

This study considered two criteria of algorithm performance, previously introduced as the *turnaround time* and the *message count* (number of task transmissions between

Table 6.6: Chosen parameter values for the division of labour algorithm to favour *message count*.

Case	α	θ	γ	n
I	100	1	0.88	8
II	100	1	0.59	10
III	100	1	0.10	1
IV	100	1	0.59	10
V	100	1	1.00	5

resources). For each criterion, both task allocation algorithms were compared with a baseline algorithm and with each other. The baseline algorithm is simply the simulated network of resources without any task allocation strategy. That is, each resource executes its task to completion and tasks are not moved between resources. The baseline algorithm thus serves also to demonstrate that the absence of a task allocation strategy in a non-uniform, distributed computing system, leads to an unnecessary increase in the turnaround time of a project.

As described in Section 6.1, the problems used during the optimization procedure for each task allocation strategy were recorded and a new, different set of problems was generated for the purpose of testing. This served to illustrate that the strategies were not tuned for a specific set of training problems but that the parameter values were applied to the general class of problem (the sequence of random numbers due to the chosen primary seed). The same set of testing problems was used for all of the evaluations to ensure that the strategies could be compared.

Each strategy was evaluated to produce samples of 30, 100, and 1000 turnaround time and message count values. Each sample was tested for normality using D'Agostino and Pearson's omnibus test. At sample sizes of 100 and 1000, the samples could not be considered to be normally distributed. Therefore, the strategies were compared using the Mann-Whitney U test.

Furthermore, the standard error of the mean (SEM) was calculated for each sample (for turnaround time). At a sample size of 1000, the SEM was found to be approximately 3% of the mean. At a confidence level of 95%, with a $z*$-value of 1.96, the sampling

margin of error would therefore be approximately $+5.88\%/-5.88\%$ of the mean, which was considered to be acceptable.

Consequently, the largest sample (1000 values) affordable in terms of processing time was used to ascertain the most accurate results.

The median, mean, and standard deviation were computed for each sample. The results of the evaluations and comparisons are presented, as follows.

6.3 Comparison Results

The baseline, CF, and DL algorithms were compared according to each of the two criteria for each case. The p-value for each comparison is depicted in Table 6.7 and Table 6.8.

Table 6.7: Comparison p-values of the algorithms under study for *turnaround time*.

Case	Algorithm	Cemetery Formation	Division of Labour
I	**Baseline**	0.000	0.000
I	**Cemetery Formation**	-	0.001
II	**Baseline**	0.000	0.000
II	**Cemetery Formation**	-	0.000
III	**Baseline**	0.000	0.000
III	**Cemetery Formation**	-	0.000
IV	**Baseline**	0.000	0.000
IV	**Cemetery Formation**	-	0.000
V	**Baseline**	0.000	0.000
V	**Cemetery Formation**	-	0.000

The median, mean, and standard deviation for the *turnaround time* criterion of each algorithm are depicted in Tables 6.9, 6.10, and 6.11.

The median, mean, and standard deviation for the *message count* criterion of each algorithm are depicted in Tables 6.12, 6.13, and 6.14.

The results obtained by evaluating the task allocation algorithms described in this study and depicted in this section are discussed next.

Table 6.8: Comparison p-values of the algorithms under study for *message count*.

Case	Algorithm	Cemetery Formation	Division of Labour
I	Baseline	0.000	0.000
I	Cemetery Formation	-	0.000
II	Baseline	0.000	0.000
II	Cemetery Formation	-	0.000
III	Baseline	0.000	0.000
III	Cemetery Formation	-	0.000
IV	Baseline	0.000	0.000
IV	Cemetery Formation	-	0.028
V	Baseline	0.000	0.000
V	Cemetery Formation	-	0.000

Table 6.9: Median turnaround time for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	10000.000	667.000	64.000	1429.000	152.000
Cemetery Formation	2184.500	222.000	24.000	241.000	26.000
Division of Labour	2139.000	214.000	23.000	233.000	25.000

Table 6.10: Mean turnaround time for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	8115.877	651.109	63.749	2503.910	423.923
Cemetery Formation	2253.739	228.307	24.274	250.097	26.900
Division of Labour	2220.619	221.388	23.241	242.363	25.700

6.4 Comparison Discussion

The comparisons of the three task allocation strategies yielded p-values that suggest that all of the results, for each of the two criteria of *turnaround time* and *message count*, were statistically significantly different at a confidence level of 95%. This result was expected because the algorithms are fundamentally different from one another. Therefore, the

Table 6.11: Standard deviation of the turnaround time for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	2686.369	216.050	20.427	2703.261	981.840
Cemetery Formation	347.108	33.491	3.493	44.415	4.881
Division of Labour	371.437	34.275	3.460	46.086	4.794

Table 6.12: Median message count for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	20.000	20.000	20.000	20.000	20.000
Cemetery Formation	59.000	25.000	21.000	27.000	21.000
Division of Labour	37.000	27.000	20.000	27.000	22.000

Table 6.13: Mean message count for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	20.000	20.000	20.000	20.000	20.000
Cemetery Formation	60.644	25.366	20.793	27.213	21.586
Division of Labour	37.032	27.002	20.000	27.481	21.976

Table 6.14: Standard deviation of the message count for each algorithm and each case.

Algorithm	Case I	Case II	Case III	Case IV	Case V
Baseline	0.000	0.000	0.000	0.000	0.000
Cemetery Formation	13.553	2.444	0.856	3.336	1.275
Division of Labour	4.507	1.837	0.000	1.914	1.843

mean, median, and standard deviations were taken at face value and used to categorise the algorithms that were studied. The following subsections describe the results for each task allocation strategy in detail.

6.4.1 Baseline Strategy

The turnaround times obtained for the baseline strategy are the highest amongst those for all of the algorithms in all of the cases. This result is in line with expectations for the reason that tasks were not moved from slower to faster resources during runtime. Therefore, the turnaround time was always determined by the slowest resource in the network.

The standard deviations for the turnaround times are relatively large because the turnaround times obtained were entirely dependent on the performance of the slowest resource within the network. If the performance of the slowest resource was closer to the highest possible value in the specified range of performance values, then the turnaround time would have been lower. Likewise, if the performance of the slowest resource was closer to the lowest possible value in specified range of performance values, then the turnaround time would have been higher.

The message counts obtained were exactly as expected. In each case, 20 transmissions occurred – one transmission for each of the 10 tasks from the project server to a resource and one transmission for each task back to the server on completion. There were no other transmissions because no task allocation was performed during runtime.

6.4.2 Cemetery Formation Strategy

The mean and median turnaround times obtained for the CF strategy were lower than those of the baseline in all cases, which is reasonable because the baseline strategy did not perform any task allocation. Furthermore, the mean and median turnaround times of the CF strategy are higher than those of the DL strategy in all cases.

The standard deviations for the turnaround times of the CF and DL strategies are similar, with those of the CF strategy being lower in cases I, II, and IV and higher in cases III and V. This suggests that the greater range of resource performance, as in cases III and V, contributes to an increase in the CF strategy's turnaround time deviations in a similar fashion to that of the baseline strategy. In other words, while the CF algorithm does reallocate tasks during runtime, thus mitigating its dependency on the performance of the slowest resource, it is not independent of the distribution of the resources in the

network.

The mean and median message counts obtained for the CF strategy were higher than those of the baseline in all cases because the CF stragegy performed additional task allocation during runtime, in addition to the 20 basic transmissions incurred by the baseline stragegy. The mean message counts obtained for the CF strategy were higher than those of the DL strategy in cases I and III and lower in cases II, IV, and V. The median message counts obtained for the CF strategy were higher than those of the DL strategy in cases I and III, lower in cases II and V, and equal in case IV. Taken together with the turnaround times, this suggests that the CF algorithm is generally weaker than the DL algorithm in that it produces higher turnaround times, often at greater or equivalent cost in terms of task transmissions (message count). Where the message counts are lower, the higher turnaround times are seen as a tradeoff.

The standard deviations of the message counts for the CF strategy are higher than those of the DL strategy in all cases except for case V. The reason for this is attributed to the polling strategy employed by each of the aforementioned task allocation strategies. The CF strategy requires a resource to choose another resource to poll at random, while the DL strategy has a resource broadcast its poll to the entire network. The former strategy can cause a task to be moved between two resources prematurely – that is, before a better resource is located by the random polling. The latter strategy locates the best pairing of resources between which to exchange a task after only one poll broadcast. In other words, minimal transmission of tasks in the CF strategy depends on the fortuity of the polling.

6.4.3 Division of Labour Strategy

The mean and median turnaround times obtained for the DL strategy were lower than those of the baseline in all cases, which, as with the CF algorithm, is due to the baseline strategy not having perfomed any task allocation. As described in the previous subsections on the baseline and CF strategies, the DL strategy exhibited the lowest mean and median turnaround times in all cases.

With reference to the standard deviations of the turnaround times obtained for the CF strategy, those obtained for the DL strategy suggest that the latter strategy is most

resilient to the distribution of the resources in the network. This is due to the polling message being broadcast, thus enabling the best resource to be located each time.

As described previously with regards to the CF strategy, the DL strategy tended to exhibit higher mean message counts in cases II, IV, and V and higher median message counts in cases II and V. The reason for this difference is attributed to the poll broadcast. While the lower standard deviations of message counts obtained for the DL strategy were attributed to this broadcast, the broadcast also appears to present opportunities for task reallocation more often than the CF strategy. The latter strategy must poll iteratively, hoping to find an opportunity to reallocate a task, while the latter strategy perceives opportunities for reallocation more often. This explains why the DL strategy exhibits the lowest turnaround times – tasks are reallocated more favourably, more often. The occassional increase in cost of task transmissions (message count) is seen as the tradeoff for this increased performance.

6.5 Scalability Analysis Design

In order to determine the effect of problem and network scale on the proposed task allocation algorithms, increasing numbers of tasks and resources within the network were tested. Once again, each problem case was tested separately for each algorithm and each performance measure. For each case, the number of tasks to complete was minimally 10 and maximally 100. For each set of tasks, the number of resources ranged from 10 to 100. Therefore, 100 instances were computed for each performance measure of each algorithm, for each case.

Since problem cases I, II, and III differ by an order of magnitude in *both* minimum and maximum resource performance, cases II and III were not included in the scalability analysis. Instead, the analysis focused on cases I, IV, and V, wherein the difference *between* the minimum resource performance and the maximum resource performance grows from one order of magnitude, in case I, to three orders of magnitude, in case V.

6.6 Scalability Analysis Results

The results of the scalability analysis for the CF algorithm are depicted in Figures 6.1, 6.2, and 6.3, for turnaround time. Every set of 10 values represents an incremental increase in the number of tasks and every result within the set represents an incremental increase in the number of resources.

As the number of tasks increases, so too does the turnaround time. This is expected because processing more tasks takes more time. In each set, the turnaround time decreases as more resources are added. Again, this is expected because adding more resources increases the capacity for processing more tasks in parallel.

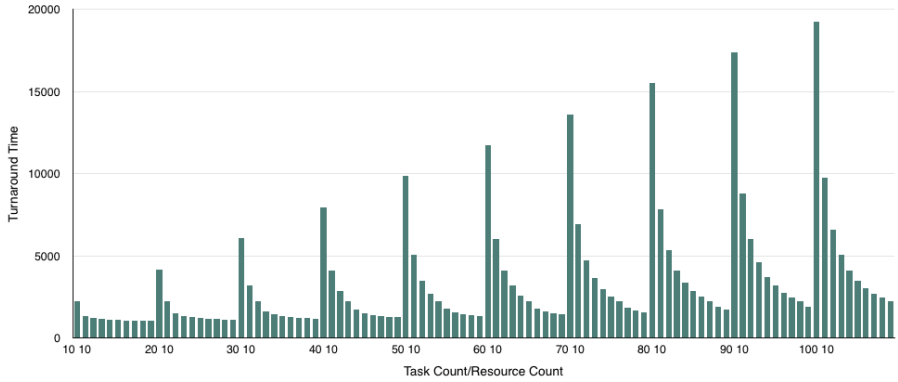

Figure 6.1: Cemetery Formation - Turnaround Time vs. [Task Resource] Count - Case I

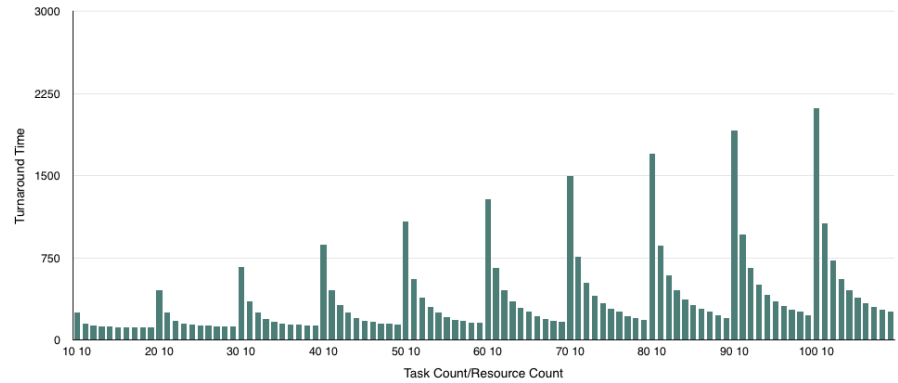

Figure 6.2: Cemetery Formation - Turnaround Time vs. [Task Resource] Count - Case IV

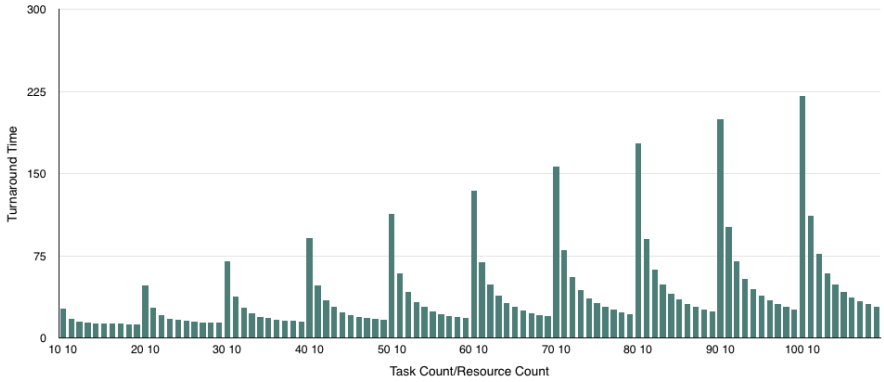

Figure 6.3: Cemetery Formation - Turnaround Time vs. [Task Resource] Count - Case V

With reference to the first set of results for case I, where the task count is 10 and the resource count ranges from 10 to 100, the turnaround time decreases as the number of resources increases from 10 to 40. The reason for this decrease is that the resource performance values, as specified for case I, were evenly distributed amongst the set of resources. Therefore, as the number of resources increased, the absolute number of fast resources also increased, thus providing the algorithm with better resources with which to complete the 10 tasks. In other words, the algorithm moved some tasks from slower to faster resources when more, faster resources were available. However, the trend does not appear to continue beyond 50 resources. The reason for this is that the tasks were already being completed in as short a time as physically possible and the addition of yet more resources afforded no further gains.

With reference to the last set of results for case I, where the task count is 100 and the resource count ranges from 10 to 100, the turnaround time continues to decrease with each increment in resource count because each additional 10 resources allows for 10 additional tasks to be processed in parallel.

The same results were observed for cases IV and V. Where the maximum resource performance increased, as specified for each case, the turnaround time decreased correspondingly.

The results for the message count of the CF algorithm appear in Figures 6.4, 6.5, and 6.6. The number of messages transmitted within the network increased as the number of tasks increased. Naturally, more messages are transmitted simply by virtue

of the greater number of tasks as well as their results being moved around the network. Furthermore, as the number of resources increased, more prospecting messages were sent out by idle resources and tasks were moved more frequently, thus adding to the total count of transmitted messages.

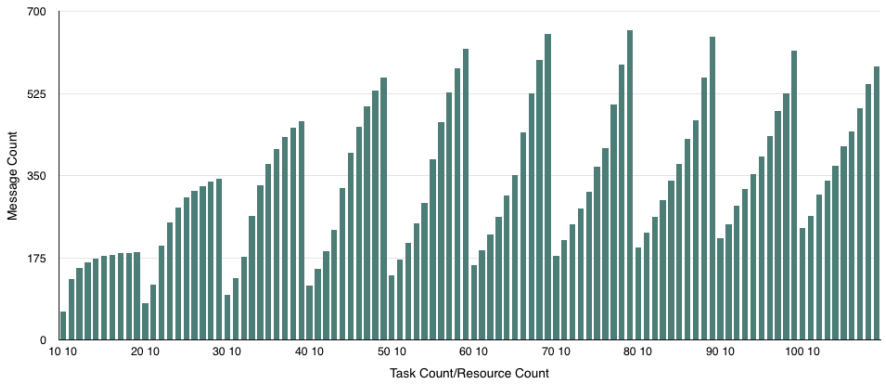

Figure 6.4: Cemetery Formation - Message Count vs. [Task Resource] Count - Case I

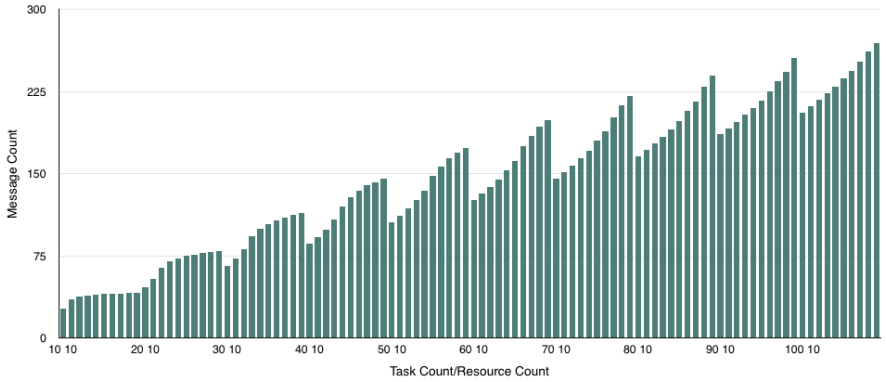

Figure 6.5: Cemetery Formation - Message Count vs. [Task Resource] Count - Case IV

With reference to the first five sets of results for case I, where the task count ranges from 10 to 50, the trend within each set appears as a concave curve. The reason for this is that an increasing number of resources produced an increasing number of prospecting messages, thereby increasing the total message count. However, the rate of increase slowed as more resources were added. This is because there were relatively few tasks

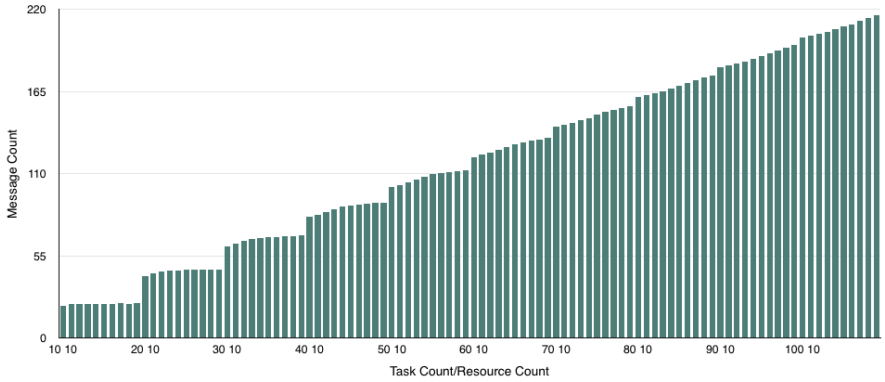

Figure 6.6: Cemetery Formation - Message Count vs. [Task Resource] Count - Case V

present in the network and these tasks were acquired by the faster resources, which completed the tasks relatively quickly, giving the slower resources less time to transmit polling messages. Essentially, the sooner a project was completed, the less time other resources had to transmit polling messages.

Now consider the second five sets of results for case I, where the task count ranges from 60 to 100. The trend within each set appears first as a convex curve and then, within the last set, as a linear increase. The reason for this change in the shape of the trend is that as more of the network was saturated with tasks, more of the slower resources acquired tasks, too. Then, when the faster resources completed their tasks, not only did those faster resources transmit polling messages but those polling messages also resulted in tasks being moved from the slower resources to the faster ones. Those task movements account for the increase in overall message count.

The trends described above are less pronounced in the results for case IV, while case V depicts mostly linear, increasing trends in the sets of results. The maximum performance of resources was higher in case IV than it was in case I and was highest in case V. The effect of this difference was that the performance of the faster resources in each test was higher in each case. Therefore, where the number of tasks was small, the faster resources completed the project quickly and little time was spent by slower resources on transmitting polling messages and moving tasks. Where the number of tasks was large, the network behaved as in case I.

The results for the DL algorithm are depicted in Figures 6.7, 6.8, and 6.9, for

turnaround time.

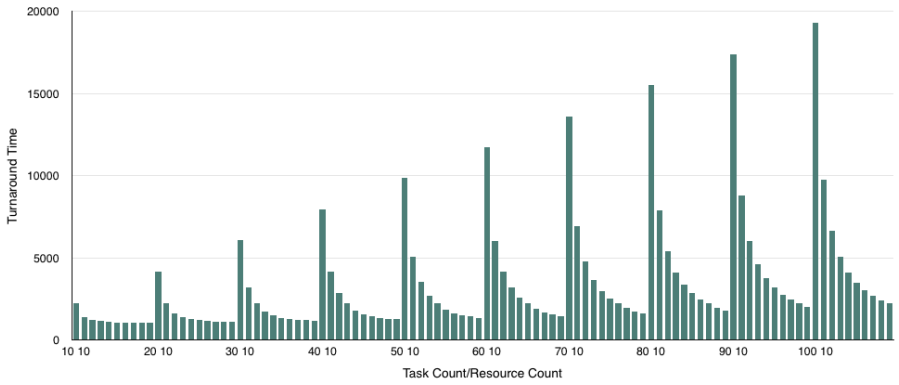

Figure 6.7: Division of Labour - Turnaround Time vs. [Task Resource] Count - Case I

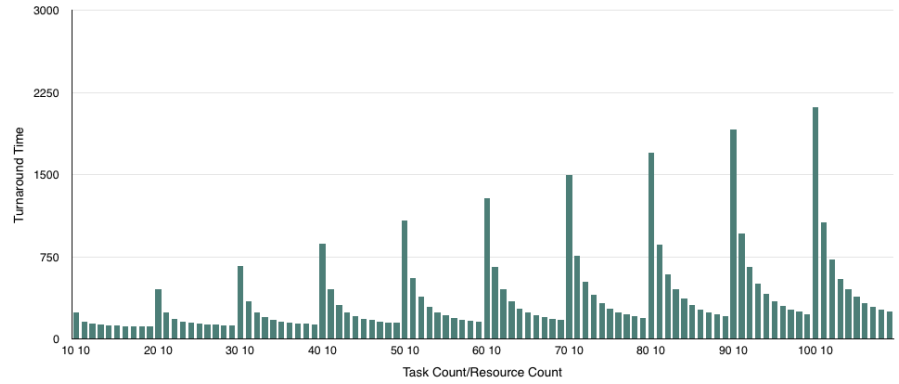

Figure 6.8: Division of Labour - Turnaround Time vs. [Task Resource] Count - Case IV

The results for the DL algorithm, with respect to turnaround time, are essentially the same as those observed for the CF algorithm. In each set, the turnaround time decreased as the number of resources increased. The turnaround time increased across sets as the number of tasks increased.

Figures 6.10, 6.11, and 6.12 depict the results for the DL algorith with respect to message count.

Once again, the results followed a similar pattern to that of the results for the CF algorithm. However, with reference to the first set of results for case I, of both algorithms,

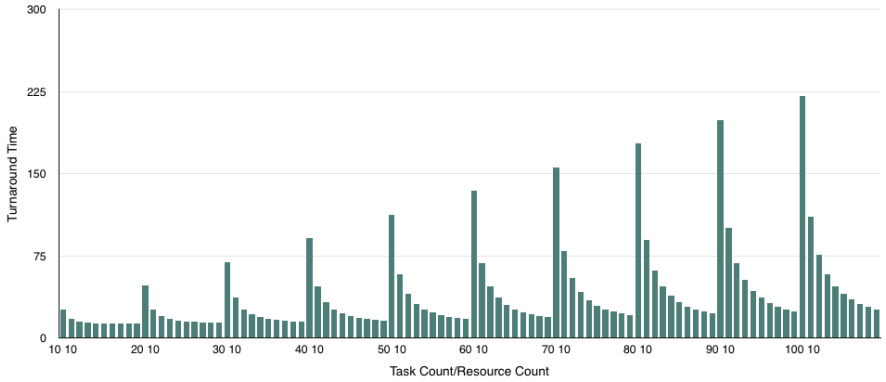

Figure 6.9: Division of Labour - Turnaround Time vs. [Task Resource] Count - Case V

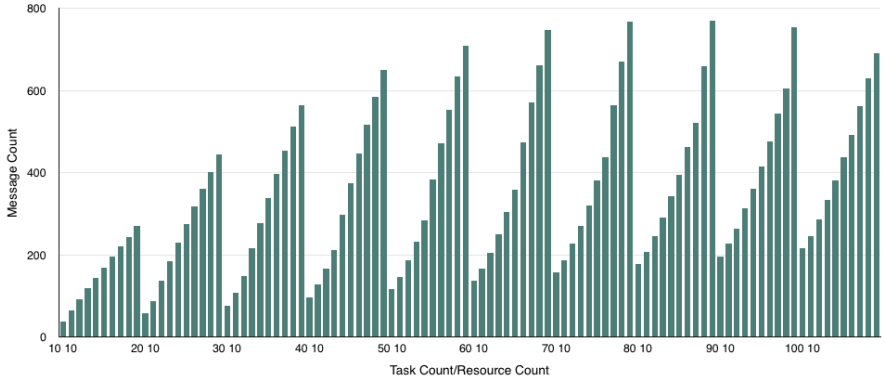

Figure 6.10: Division of Labour - Message Count vs. [Task Resource] Count - Case I

a distinct difference is apparent. As more resources were added, the CF algorithm exhibited a diminishing increase in message count, while the DL algorithm exhibited a linear increase in message count. Since a resource operating according to the DL algorithm transmits a stimulus message only while it is processing a task, the fact that the message count increased as resources were added suggests that tasks were exchanged more frequently as more resources were added. In other words, resources exhibited thrashing while moving tasks.

As the difference between minimum resource performance and maximum resource performance increased in case IV and again in case V, the increase in message count became much more gradual in each set of results. When the faster resources possessed

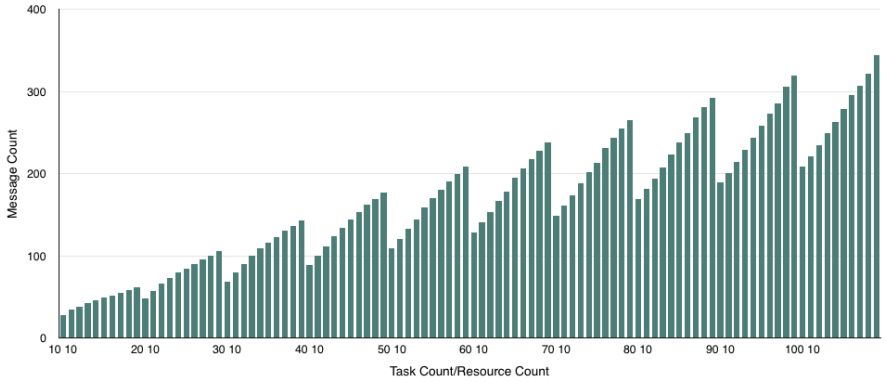

Figure 6.11: Division of Labour - Message Count vs. [Task Resource] Count - Case IV

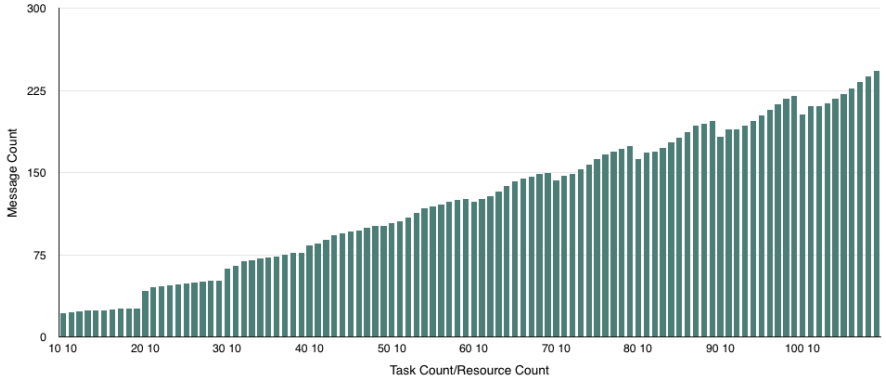

Figure 6.12: Division of Labour - Message Count vs. [Task Resource] Count - Case V

tasks, their stimulus messages advertised substantially greater performance than the slower resources were able to compete with so the probabilistic function that determines whether or not to acquire a task from another resource was not frequently activated. In other words, thrashing was reduced because the difference in performance between the slow resources and fast resources was large.

6.7 Scalability Analysis Remarks

The results of the scalability analysis are generally in line with the expectations that an increase in resources will result in a decrease in turnaround time and an increase in mes-

sage count. However, some specific observations have yielded practical considerations.

Firstly, both algorithms incur an unavoidable cost in message count while resources remain idle. Therefore, the number of tasks should ideally either meet or exceed the number of resources in a network. Secondly, the DL algorithm is more prone to thrashing than the CF algorithm when the network is composed of resources with similar performance capabilities.

6.8 Summary

This chapter described the empirical methodology followed to complete the study. The procedure used to find suitable values for the free parameters of the task allocation strategies was specified and the chosen parameter values were presented. The design of the testing procedure was detailed and the results of evaluating and comparing the task allocation strategies were depicted. The statistical significance tests used to compare the strategies under study revealed that both proposed task allocation strategies outperformed the baseline and that while both proposed strategies exhibited similar results, they did provide a tradeoff with respect to the two criteria of turnaround time and task transmission (message count) chosen for this study. Additionally, a scalability analaysis of the proposed task allocation algorithms was conducted. The results of the analysis provided evidence to show that the algorithms perform as designed and also that careful attention to the composition of the network is required when using the algorithms. The following chapter summarises the findings of this chapter and presents some final remarks to conclude the study.

Chapter 7

Conclusions

The findigs of the empirical analysis of the task allocation strategies proposed by this study were reported in Chapter 6. This chapter summarises the findings of the empirical analysis and suggests possible directions for further studies. Section 7.1 outlines the conclusions of the study and Section 7.2 describes potential future work.

7.1 Summary of Conclusions

This study isolated the turnaround time of completing an embarassingly parallel project using a distributed computing system as the key problem to solve and identified task transmission (bandwidth usage) as the cost of doing so. The resulting contributions are dynamic task allocation algorithms that facilitate decentralised control in a stochastic environment. The findings of the study are detailed in the following subsections.

7.1.1 Parameter Sensitivity Analysis

The cemetery formation and division of labour algorithms were evaluated by means of a parameter sensitivity analysis. The analysis was used to determine the relative importance of each parameter to each of the two measures used to evaluate algorithm performance, as well as the necessity of finding optimal parameter values before applying either algorithm.

The sensitivity analysis found that each parameter, of each algorithm, has a bearing on each algorithm's performance. However, the parameters are not all equally important, with some parameters having exhibited a substantially smaller influence on performance than others. Subsequently, a practical application of either algorithm may proceed by optimising the most influential subset of parameters and may exclude the least significant parameters from optimisation where the time available for optimisation is limited.

The most significant parameters of the Cemetery Formation algorithm, for minimal turnaround time, are, in order from most to least significant, n, θ, λ, with α being substantially less important. The most significant parameters of the CF algorithm, for minimal message count, are, in order from most to least significant, λ, n, α, θ, with no parameter being substantially less important than the others.

The most significant parameters of the Division of Labour algorithm, for minimal turnaround time, are, in order from most to least significant, γ, n, α, θ, with θ and n being substantially less important *in some but not all cases*. The most significant parameters of the DL algorithm, for minimal message count, are, in order from most to least significant, γ, α, θ, n, with θ and n being substantially less important again in some but not all cases.

As is inferred from the fact that the DL algorithm's parameters are not important in all of the cases that were studied, the DL algorithm was found to be more sensitive than the CF algorithm to the distribution of resource performance. The consequence of this characteristic is that the DL algorithm's parameters will need to be optimised for each application of the algorithm and that the network within which the algorithm operates cannot be modified arbitrarily without potentially requiring a new set of parameter values for the DL algorithm.

7.1.2 Optimization Procedure

The procedure to tune the parameters of the task allocation strategies proposed by this study was based on an augmented version of F-Race. The modification to F-Race added a termination heuristic that allowed for the execution of the procedure to be stopped when a reasonably good set of parameter values was found. The heuristic method was evaluated empirically and found to produce reasonable results. An advantage conveyed

by the heuristic was that it provided a concrete substantiation for the time expended to determine the parameter values that were ultimately chosen.

On its own, the termination heuristic is a small but novel contribution to the field of numerical optimisation and, specifically, to the evolution of the research embodied by the F-Race procedure.

7.1.3 Statistical Significance Tests

The two task allocation strategies that were proposed by this study were inspired by the cemetery formation and division of labour ant algorithms, respectively. The empirical analysis of the proposed strategies showed that the proposed means of task allocation were able to overcome the dependence of turnaround time on the performance of the slowest resource in the network, thus achieving lower turnaround times than that of the baseline.

While the task allocation strategies performed similarly with respect to turnaround time and the number of effected task transmissions, the division of labour strategy was found to achieve the lowest turnaround time in general. However, the cemetery formation strategy reallocated tasks less often than the division of labour strategy. The two algorithms present a trade-off of turnaround time for number of task transmissions.

This trade-off can be employed to favour either performance or bandwidth. Specifically, if the tasks to be performed by the distributed computing system would require a large amount of bandwidth to transmit from one resource to another, then the cemetery formation strategy could be employed. Alternatively, the division of labour strategy could be employed where the tasks are small and a low turnaround time is desired.

7.1.4 Scalability Analysis

A scalability analysis of the proposed task allocation strategies was conducted in order to determine how the CF and DL algorithms performed as the numbers of tasks and resources were increased. In general, increasing the number of resources reduces the turnaround time of a project for both algorithms. However, increasing the number of resources beyond approximately twice the number of tasks leads to diminishing returns.

Furthermore, the DL algorithm is more prone to thrashing – that is, moving tasks between resources unnecessarily – than the CF algorithm when most of the resources in the network exhibit similar performance. In other words, the DL algorithm is better suited to networks where the resources can be separated into two, distinct categories, with one category of resources being significantly faster than the other.

7.2 Future Work

Beyond the scope of task allocation in a dynamic, distributed computing system, the following issues were not addressed and may be considered as future work:

- Minimization of the control messages transmitted by the task allocation algorithms.
- Alternative threshold-response functions that take more information about the network into account.
- Robustness of the distributed computing system in the event of nodes failing unexpectedly.
- Separation of the task from its data and how that data might be handled within the network.

Bibliography

[1] William Aiello, Baruch Awerbuch, Bruce Maggs, and Satish Rao. Approximate Load Balancing on Dynamic and Asynchronous Networks. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 632–641, 1993.

[2] Rashid Al-Ali, Kaizar Amin, Gregor von Laszewski, Omer Rana, David Walker, Mihael Hategan, and Nestor Zaluzec. No Title. *Journal of Grid Computing*, 2(2):163–182, 2004.

[3] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485. ACM, 1967.

[4] David P Anderson. BOINC : A System for Public-Resource Computing and Storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

[5] David P Anderson, Jeff Cobb, Eric Korpela, and Matt Lebofsky. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[6] Robert Armstrong, Debra Hensgen, and Taylor Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pages 79–87, 1998.

[7] Michel Auguin and Francois Larbey. OPSILA: an advanced SIMD for numerical analysis and signal processing. In *Microcomputers: developments in industry, business, and education*, pages 311–318, 1983.

[8] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement. Technical Report May, Université Libre de Bruxelles, 2007.

[9] Fran Berman, Geoffrey Fox, and Tony Hey. The Grid: past, present, future. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 9–50. Wiley, 2003.

[10] Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, and Damien Olivier. Organization Detection for Dynamic Load Balancing in Individual-Based Simulations. *Multiagent and Grid Systems*, 1(1):141–163, 2007.

[11] Mauro Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, 2004.

[12] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stutzle. Automated Algorithm Tuning using F-races: Recent Developments. *MIC 2009: The VIII Metaheuristics International Conference*, pages 1–10, 2009.

[13] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-Race and iterated F-Race : An overview. Technical Report June, Université Libre de Bruxelles, 2009.

[14] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-Race and iterated F-Race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer Berlin, 2010.

[15] Eric Bonabeau, Guy Theraulaz, and Jean-Louis Deneubourg. Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in Insect Societies. *Proceedings of the Royal Society B: Biological Sciences*, 263(1376):1565–1569, November 1996.

[16] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, Bin Yao, D. Hensgen, and R.F. Freund. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop (HCW '99)*, pages 15–29, San Juan, 1999.

[17] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computer*, 61(6):810–837, 2001.

[18] Mike Campos, Eric Bonabeau, Guy Théraulaz, and Jean-Louis Deneubourg. Dynamic Scheduling and Division of Labor in Social Insects. *Adaptive Behavior*, 8(2):83–95, March 2000.

[19] Junwei Cao. Self-Organizing Agents for Grid Load Balancing. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, number November, pages 388–395. IEEE Computer Society, 2004.

[20] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, 1988.

[21] Timothy C.K. Chou and Jacob A. Abraham. Load Balancing in Distributed Systems. *IEEE Transactions on Software Engineering*, SE-8(4):401–412, 1982.

[22] L. Chrétien. *Organisation spatiale du materiel provenant de lexcavation du nid chez Messor Barbarus et des cadavres douvrieres chez Lasius Niger*. PhD thesis, Universite Libre de Bruxelles, 1996.

[23] Douglas E. Comer. *Computer Networks and Internets*. Prentice Hall, 2001.

[24] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, 3rd editio edition, 1999.

[25] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-hanks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: robot-like ants and ant-like robots. In J.-A. Meyer and S.W. Wilson, editors, *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 356–363, 1991.

[26] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, June 2000.

[27] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[28] R. A. Fisher. *The Design of Experiments*. Oliver & Boyd, Oxford, England, 1935.

[29] Wan Fokkink. *Distributed Algorithms: An Intuitive Approach*. MIT Press, 2013.

[30] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.

[31] Richard F. Freund and Howard Jay Siegel. Heterogeneous Processing. *Computer*, 26(June):13–17, 1993.

[32] Milton Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[33] Jacques Gautrais, Guy Theraulaz, Jean-Louis Deneubourg, and Carl Anderson. Emergent polyethism as a consequence of increased colony size in insect societies. *Journal of theoretical biology*, 215(3):363–73, April 2002.

[34] J Handl, J Knowles, and M Dorigo. Ant-Based Clustering: A Comparative Study of Its Relative Performance with Respect to k-Means, Average Link and 1D-SOM. Technical Report i, Université Libre de Bruxelles, 2003.

[35] Joint Technical Committee ISO/IEC JTC 1. *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. ISO/IEC, 2 edition, 1994.

[36] Ronald Klazar and Andries Engelbrecht. Parameter Optimization by Means of Statistical Quality Guides in F-Race. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–6, Beijing, 2014. IEEE.

[37] Pascale Kuntz and Snyers Dominique. New results on an ant-based heuristic for highlighting the organization of large graphs. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99)*, pages 1451–1458, 1999.

[38] Pascale Kuntz, Dominique Snyers, and Paul Layzell. A Stochastic Heuristic for Visualising Graph Clusters in a Bi-Dimensional Space Prior to Partitioning. *Journal of Heuristics*, 5(3):327–351, 1999.

[39] Thomas Kunz. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, 1991.

[40] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Comput. Soc. Press, 1988.

[41] Erik D. Lumer and Baldo Faieta. Diversity and Adaptation in Populations of Clustering Ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 501–508, 1994.

[42] J MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 233, pages 281–297. University of California Press, 1967.

[43] Eugene N. Marais. *The Soul of the White Ant*. Review Press, 2009.

[44] Marc Martin, Bastien Chopard, and Paul Albuquerque. Formation of an ant cemetery: swarm intelligence or statistical accident? *Future Generation Computer Systems*, 18(7):951–959, August 2002.

[45] Chris Melhuish, Owen Holland, and Steve Hoddell. Collective sorting and segregation in robots with minimal sensing. In *5th International Conference on the Simulation of Adaptive Behaviour*, pages 465–470, 1998.

[46] N. Monmarché. On Data Clustering With Artificial Ants. In *In AAAI-99 & GECCO-99 Workshop on Data Mining with Evolutionary Algorithms: Research Directions*, pages 23–26, 1999.

[47] N. Monmarche, M. Slimane, and G. Venturini. AntClass: Discovery of Clusters in Numeric Data by an Hybridization of an Ant Colony with the K-Means Algorithm. Technical report, Laboratoire d'Informatique, University of Tours, 1999.

[48] Alberto Montresor, Hein Meling, and Özalp Babaolu. Messor: Load-Balancing through a Swarm of Autonomous Agents. Technical Report September, University of Bologna, 2003.

[49] Sorinel Adrian Oprisan, Viorel Holban, and Bogdan Moldoveanu. Functional self-organization performing wide-sense stochastic processes. *Physics Letters A*, 216(6):303–306, June 1996.

[50] Gregory F. Pfister. *In Search of Clusters*. Prentice Hall PTR, 1998.

[51] John F. Shoch and Jon A. Hupp. The worm programs—early experience with a distributed computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[52] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice Hall, 2006.

[53] Guy Theraulaz, Eric Bonabeau, and Jean-Louis Deneubourg. Response threshold reinforcement and division of labour in insect societies. *Proceedings of the Royal Society Biological Sciences*, 265(1393):327–332, 1998.

[54] Marc H Willebeek-LeMair and Anthony P Reeves. Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993, 1993.

[55] Edward O. Wilson. The relation between caste ratios and division of labor in the ant genus Pheidole (Hymenoptera: Formicidae). *Behavioral Ecology and Sociobiology*, 16(1):89–98, November 1984.

[56] Edward O Wilson. Between-caste aversion as a basis for division of labor in the ant Pheidole pubiventris (Hymenoptera: Formicidae). *Behavioral Ecology and Sociobiology*, 17(1):35–37, 1985.

[57] Matt Wilson, Chris Melhuish, and Ana Sendova-franks. Creating Annular Structures Inspired by Ant Colony Behaviour Using Minimalist Robots. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 53–58, Yasmine Hammamet, 2002.

[58] Bin Wu and Zhongzhi Shi. A Clustering Algorithm Based on Swarm Intelligence. In *Proceedings of the International Conference on Info-tech and Info-net*, pages 58–66, Beijing, 2001.

[59] Yan Yang and Mohamed Kamel. Clustering Ensemble Using Swarm Intelligence. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 65–71, 2003.

[60] Irvin Yeaworth. The Blob, 1958.

[61] Mohammed Javeed Zaki, Li Wei, and Parthasarathy Srinivasan. Customized Dynamic Load Balancing for a Network of Workstations. In *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*, 1996.

Appendix A

Acronyms

The following is a list of acronyms used throughout the text and is arranged in alphabetical order. Each acronym is typeset in bold and its expanded form is displayed alongside.

ACO ant colony optimization 79, 80

AS ant system 75, 79–81

BOINC Berkeley Open Infrastructure for Network Computing 2

CF cemetery formation vii, 43–45, 48–52, 58, 59, 64, 67, 68, 86, 87, 90, 93–97, 100, 101, 103, 105–107

DL division of labour 51, 53, 54, 59, 64, 85–87, 90, 93–95, 99–101, 103, 105–107

DLB dynamic load balancing 34–40

FFD full factorial design 70, 80

FTM fixed threshold model 29, 30

NIC network interface controller 36

OCFM original cemetery formation model 43, 45, 48, 49

ODLM original division of labour model 51, 53

OSI Open Systems Interconnection 40

PCC Pearson correlation coefficiant 60–67

QoS Quality of Service 9, 13, 14

SPMD Single Program, Multiple Data 7

TSP travelling salesman problem 78, 79, 81

Appendix B

Symbols

The following is a list of all mathematical symbols used throughout this text. Each symbol is defined under the chapter in which it first appears.

B.1 Chapter 3: Overview of Ant Algorithms

α	Coefficient
α_k	Coefficient for ant k
β	Constant value
γ	Stimulus
γ_t	Stimulus emitted by task t
δ	Increase in stimulus
θ_1	Activation threshold for pick up
θ_2	Activation threshold for put down
θ_{it}	Activation threshold for individual i to perform task t
θ_{lt}	Activation threshold for caste l to perform task t
ξ	Coefficient
ρ	Coefficient
σ	Size of an ant's local neighbourhood
τ	Efficiency of task performance

a	Coefficient
C	Number of individuals in a colony
C_l	Number of individuals in caste l
$d(i,j)$	Euclidiean distance between items i and j
e	Error rate
f	Fraction of nearby cells occupied by items
$f(i)$	Fraction of nearby cells occupied by item i
i	Index of an item
j	Index of an item
l	Index of a caste
m	Number of steps
$N(r)$	Circular, local area
N_{occ}	Observed number of occupied cells
n	Number of items
P_p	Probability of an ant to pick up an item
P_d	Probability of an ant to put down an item
P_l	Probability of an ant in caste l to change state
p	Probability of an ant to become inactive
r	Radius
S	Sigmoid function
s	Length of a side of the square neighbourhood of an ant
T	Time step
t	Index of a task
v	Ant velocity
V_{max}	Maximum ant velocity
X	State of an ant
x_{it}	Fraction of time that individual i performs task t

B.2 Chapter 4: Dynamic Load Balancing Based on Ant Algorithms

θ	Activation threshold
D	Destination of a task
n	Constant value
P_{SD}	Probability of a connection to be brokered between S and D
R	Type of resource (S or D)
S	Source of a task
s	Normalised shortfall
$W_{max}(R)$	Maximum work that a resource is capable of performing
$W(R,T)$	Work done by a resource at time T

B.3 Chapter 5: The Parameter Optimization Procedure

γ	Stimulus period
	Configuration space
κ	A configuration
λ	Prospecting range
ρ	Evaporation rate
$\sigma_{l,q}$	Standard deviation of parameter q in iteration l
τ_0	Initial pheromone value
B	Computational budget
B_{used}	Computational budget used up to iteration $l-1$
b	Number of levels
$\bar{C}^{nn}$	Mean shortest path, according to the nearest-neighbour heuristic
F	Objective function
f	Number of factors

g	Number of ants
$\mathcal{I}$	Problem space
i	Index of a problem instance
k	Number of problem instances
L	Number of iterations
l	Index of an iteration
m	Number of candidate configurations
N_e	Number of elite surviving configurations
N_{min}	Minimum number of surviving configurations
$N_{survive}$	Number of surviving configurations
Q	Number of parameters in a configuration
q	Index of a parameter
r_z	Rank of configuration z
u	Length of a side of the square of a TSP problem
v_q	Range of parameter q
w_z	Weight of configuration z
$\mathbf{x}$	New candidate configuration
$\mathbf{x}_z$	Elite candidate configuration
x_q	Parameter q of candidate configuration x
z	Index of a configuration

Appendix C

Derived Publications

The following list of publications were derived from this study:

- Ronald Klazar and Andries P. Engelbrecht, Dynamic load balancing inspired by division of labour in ant colonies, *2011 IEEE Symposium on Swarm Intelligence (SIS)*, IEEE, 2011.
- Ronald Klazar and Andries P. Engelbrecht, Dynamic load balancing inspired by cemetery formation in ant colonies, *Swarm Intelligence*, pages 236-243, Springer Berlin Heidelberg, 2012.
- Ronald Klazar and Andries P. Engelbrecht, Parameter Optimization by Means of Statistical Quality Guides in F-Race, *2014 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2014.