

Investigation of Distributed and Parallel Performance of a Genetic Algorithm

by
Philip James Uren, BComp

A dissertation submitted to the
School of Computing
in partial fulfilment of the requirements for the degree of

Bachelor of Computing with Honours

University of Tasmania
November 2004

DECLARATION

I, Philip James Uren, declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution. To my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

.....

ABSTRACT

Genetic algorithms, a stochastic evolutionary computing technique, have demonstrated a capacity for robust, efficient problem solving through highly parallel search space exploration. This work demonstrates how an improvement in performance and efficiency over the traditional serial approach can be achieved by exploiting this highly parallel nature to produce parallel genetic algorithms. Furthermore, it is shown that by incorporating domain specific knowledge into a genetic algorithm near optimal solutions can be located in minimal time.

ACKNOWLEDGEMENTS

As the deadline for thesis submission approaches and the year draws to an end, I find my thoughts are divided between looking forward to the future and remembering the year that has just past. Certainly many people deserve my thanks for their help during this endeavour; some for their assistance with my work, others for less direct support. All are equally appreciated and I hope this small token is sufficient to express my gratitude to them.

It is logical to begin with the inception of this work, so I shall do just that. My first thanks are to my supervisor, Professor Sale. Without his help this work would never have begun let alone be reaching completion. I wish to offer my thanks for the ideas, suggestions and constant proof reading that you have provided me with over the year.

Continuing with my gratitude to the staff here, two other people stand out in my mind. Firstly, I would like to thank Dr. Peter Vamplew for his guidance as the honours co-ordinator, proof reading, and patient handling of questions. Secondly, my thanks also go to Mr. Mark Hepburn for his thorough proof reading and for making my workload in tutoring his subject more manageable than he was reasonably obliged to do. Of course, I would like to extend my thanks to all the staff here within the school of computing for their support over the past year and indeed for my entire time here.

To my fellow honours students I thank you for a most vibrant and interesting year. It would not have been so without you all. I have come to appreciate your company and welcome distractions. It is perhaps this that I will miss most.

Finally, I turn my attention to those who are not directly connected to my work. In many ways it is to them that I owe the most. To my girlfriend, Sarah, thank you for your support and love over this past year, it has meant a great deal to me. To my friends, I have greatly appreciated the chance to unwind and relax with you all. I have left extending my gratitude to my family as the final acknowledgement because it is them that I wish to thank the most. I am truly grateful for their support and love over this year and within everything I have undertaken.

TABLE OF CONTENTS

1.	Introduction.....	1
2.	Context.....	2
3.	Literature Review.....	3
3.1.	Overview.....	3
3.2.	Structure and Scope	3
3.3.	Genetic Algorithms.....	3
3.3.1	Background and Conceptual Origins	3
3.3.2	The Basic Genetic Algorithm	4
3.3.3	Genetic Operators	5
3.3.4	Parent Selection.....	6
3.3.5	Schemata	7
3.3.6	Success of Genetic Algorithms	8
3.3.7	Convergence in Serial Genetic Algorithms	10
3.4.	Parallel Genetic Algorithms.....	11
3.4.1	Algorithm Models	11
3.4.2	Design Goals	12
3.4.3	Population Granularity	13
3.4.3.1.	Global Parallelisation	13
3.4.3.2.	Coarse-Grained	13
3.4.3.3.	Fine-Grained	14
3.4.3.4.	Micro-Grained.....	16
3.4.4	Hybrid, Hierarchical and Disjoint Algorithms.....	16
3.4.5	Migration Scheme	17
3.4.6	Domain Knowledge	17
3.4.7	Convergence in Parallel Genetic Algorithms.....	18
3.5.	Capacited Vehicle Routing Problem.....	19
3.5.1	Context	19
3.5.2	Problem Definition.....	19
3.5.3	Genetic Algorithms Applied to CVRP.....	20
3.5.4	Heuristics and Domain Knowledge	21
3.6.	Grid Engine Architecture	23
3.7.	Summary	24
4.	Methodology and General Experiment Design.....	25
4.1.	Introduction.....	25
4.2.	Metric Selection	25
4.3.	Test Problem and Algorithm Mechanics	26
4.3.1	Problem Refinement	26
4.3.2	Encoding Scheme.....	27
4.3.3	Genetic Operators	28
4.3.4	Selection Scheme	28
4.3.5	Migration Scheme	29
4.4.	General Experiment Design	29
4.4.1	Requirements Separation	29

4.4.2	Serial and Parallel comparison.....	29
4.4.3	Distributed and Non-Distributed Comparison	30
4.4.4	Heuristic Comparison	30
4.5.	General Expectations	30
5.	Architecture, Topology and Configuration	32
5.1.	Environment.....	32
5.1.1	Serial and Parallel Execution Environments.....	32
5.1.2	Distributed Execution Environment.....	32
5.2.	Algorithm Testing	34
6.	Detailed Experiment Design and Expectations.....	40
6.1.	Serial – Scaling Computational Power	40
6.2.	Naïve Parallel and Serial Comparison	40
6.3.	Non-Distributed Parallel and Serial Comparison.....	41
6.4.	Distributed Parallel and Serial Comparison.....	41
6.5.	Increasing Cluster Size.....	42
6.6.	Effect of Naïve Heuristic	42
6.7.	Problem Decomposition Parallelisation.....	43
7.	Results and Observations	45
7.1.	Serial – Scaling Computational Power	45
7.2.	Naïve Parallel and Serial Comparison	46
7.3.	Non-Distributed Parallel and Serial Comparison.....	47
7.4.	Distributed Parallel and Serial Comparison.....	51
7.5.	Increasing Cluster Size.....	53
7.5.1	Results and Observed Trends.....	53
7.5.2	Four Host Cluster Variance.....	57
7.5.3	Summary	59
7.6.	Effect of Naïve Heuristic	59
7.7.	Problem Decomposition Parallelisation.....	62
7.7.1	Naïve Four-Way Decomposition	62
7.7.2	Conceptual Decomposition	64
7.7.3	Summary	65
8.	Further Observations and Experimentation	67
8.1.	Convergence.....	67
8.2.	Parallel Work Duplication	69
8.3.	Objective Convergence Measurement	71
8.3.1	Specification.....	71
8.3.2	Results.....	72
8.3.3	Discussion	73
8.4.	Expanded Cluster Size	74
8.4.1	Experiment	74
8.4.2	Results.....	74
9.	Discussion	77
10.	Conclusions.....	82
11.	Further Work.....	84
12.	References	86
13.	Appendicis	91

1. INTRODUCTION

Inspired by observations of adaptation in natural systems, genetic algorithms attempt to capture the essence of evolutions robust, clean, efficient approach to problem solving. A unique take on the classic artificial intelligence concept of problem solving by search, these techniques and approaches have proven themselves to be quite efficient at handling otherwise intractable problems.

Possessive of a highly parallel implicit structure, numerous approaches have been proposed for improving the efficiency of genetic algorithms by employing concurrent programming techniques. This work has as its aim the exploration of such efforts. The goal being to establish whether parallel genetic algorithms produce better results than serial approaches. It is expected that owing to a more efficient search of the problem space, parallel algorithms will possess superior performance. However the possibility that the extra computation power and organisational structure provided by a parallel approach leads to little or no improvement in performance is also explored. Throughout the remainder of this work, these two disjunct points are referred to simply as *the hypothesis*.

To this end, a review of the current literature is presented first, enabling a more in depth understanding of the characteristics of genetic algorithms. This is also the basis for classifying the large number of disparate parallel versions of the technique into categories. An exploration of the performance, benefits, and drawbacks offered by approaches which exemplify their class of genetic algorithm is presented. From this information a basis for designing and deploying several parallel genetic algorithms is arrived at. The design of these algorithms is elaborated and test results are presented to validate their performance. Following this, an argument for the general structure of an experimental approach aimed at establishing the proof or otherwise of the above hypothesis is presented. This approach is further refined and a final detailed description is presented.

Having deployed and evaluated the performance of the developed algorithms using several metrics the results are presented and within local scope are analysed and discussed for relevance. Drawing all of this empirical evidence together, a final discussion of the arguments for and against the proposed hypothesis is made and conclusions are expounded as to its validity. Finally, a discussion of items of interest and suggested approaches to further work on this topic is presented with several directions of intriguing research identified.

2. CONTEXT

The context and environment within which these experiments and observations are made is given thorough treatment in chapter 5, but a brief overview is given here to allow the reader a better perspective from which to view the focused exploration presented within the literature review.

As an aside to the main goal of this work, the application of grid computing as the mechanism for deploying parallel genetic algorithms upon an extensible dynamic platform is explored. As a result, the use of Sun Grid Engine¹ is not presented as a reasoned decision, but rather a fixed assumption upon which an evaluation of its performance and suitability to such tasks can be made.

The execution environment will consist of two platforms. The first of these is an 800MHz PC running windows XP upon which evaluation of most non-distributed algorithms will be possible. The second of these platforms is the grid cluster organised by way of the grid engine software, as described above. It contains four machines with processor speeds of 233MHz, each running Linux Fedora.

A problem of interest has been selected upon which to test the performance of the genetic algorithms. This problem, the *capacited vehicle routing problem* or *CVRP*, is the logical intersection of the well known *travelling salesman problem* (*TSP*) and the *bin-packing* or *knapsack problem* (often abbreviated as *BPP*). The details of the CVRP are given later, however its selection as the desired application was made due to its NP-complete nature and its conceptual composition of two sub-problems (the TSP and the BPP).

¹ Sun Grid Engine (SGE) is an open source grid computing software package, the details of which are elaborated later.

3. LITERATURE REVIEW

3.1. OVERVIEW

The purpose of this review is to present an exploration of techniques for implementing a parallel genetic algorithm on a small cluster of machines running the automated distributed computing engine, *grid engine*. The goal of the algorithm will be to solve a standardised capacited vehicle routing problem. Various approaches and techniques will be explored within this review to determine which are suitable to the architecture and application. In addition, the factors which influence, favourably or otherwise, the performance of the algorithms are also investigated. The major goal of this literature review is to provide an understanding from which to select appropriate algorithms for the investigation of the issues raised in the hypothesis.

3.2. STRUCTURE AND SCOPE

Section 3.3 will outline genetic algorithms in their serial implementation, giving a brief background and defining several important concepts, such as genetic operators, schemata, and convergence. Section 3.4 will extrapolate these concepts to parallel implementations of genetic algorithms, exploring styles of parallelism and their effect on performance and efficiency. Section 3.5 will define the capacited vehicle routing problem, upon which the performance of the genetic algorithms will be benchmarked. Finally, section 3.6 will introduce the *grid engine* software and briefly describe its function.

A complete description of the field of genetic algorithms, both parallel and serial, is beyond the scope of this review. Rather, the purpose is to provide a level of understanding and justification from which to select and implement appropriate algorithms to test the hypothesis presented above.

3.3. GENETIC ALGORITHMS

3.3.1 BACKGROUND AND CONCEPTUAL ORIGINS

The field of Genetic Algorithms (GAs) is commonly attributed to John H. Holland who devised the concept of “Genetic Plans”, later to become known as “Genetic Algorithms”. Holland believed that by applying the concept of evolution as it is found in nature to computational problems, solutions could be discovered in a

more robust fashion than other techniques for searching large problem spaces could provide (Holland, 1975).

Genetic algorithms are *search and optimisation* methods (Goldberg, 1989a). The fundamental goal of all genetic algorithms is to find some solution that satisfies certain criteria amongst many other less efficient solutions. Most often, they are applied to problems where it is difficult to see how one solution may be modified to produce a better solution.

For the purpose of this section, an overview of genetic algorithms will be presented mainly through what might be considered a *traditional genetic algorithm* (Davis, 1991). That is, one which closely resembles Holland's work. Extensions and modifications to this original approach are numerous and designed to meet varying goals. Some such variants will be discussed where appropriate.

3.3.2 THE BASIC GENETIC ALGORITHM

The general concept of a genetic algorithm approach is to represent solutions to the problem as chromosomes, made up of individual genes which in turn represent traits of the solution. Within the traditional algorithm, as presented here, chromosomes are binary strings. A function is then defined, often referred to as the *fitness function*, to evaluate the effectiveness of a given solution. Further generations are developed based on certain criteria and operations of the algorithm, with the intention of improving the value of solutions. Finally, some termination criteria will be met and the current best solution will be presented as the output of the genetic algorithm. Davis provides a top level description of genetic algorithms as the series of steps in figure 3.3.2.1

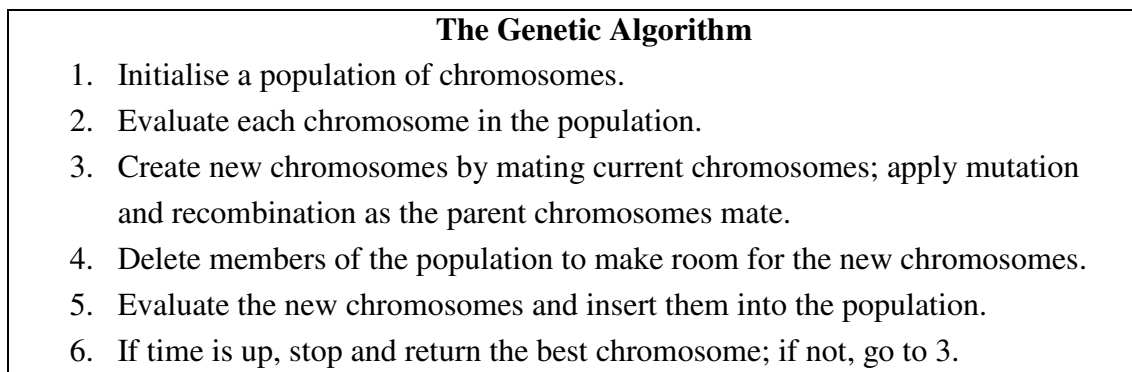


Figure 3.3.2.1: The Traditional Genetic Algorithm. Adapted from Davis (1991)

There are several input parameters that need to be specified when running a genetic algorithm. For the traditional algorithm described above, these are *population size*, *crossover rate*, *mutation rate* and *number of generations*. Insight into the problem domain is useful in selecting values for these parameters (Oszczka, 2002). This choice will influence the performance of the genetic algorithm in finding fit solutions as well as the speed at which it may discover such individuals (Prasanna Jog, 1989).

3.3.3 GENETIC OPERATORS

Holland proposed the fundamental operators of *crossover*, *mutation*, *inversion* and implicitly *replication*. These names are drawn from biological processes at the cellular level, but are not used in exactly the same senses. Various other operators have been proposed, some of which extend these basic forms or are domain specific. The replication and crossover operations are sometimes referred to as *primary operators* with others being *secondary operators* (Koza, 1992). The characteristics and function of each operation will now be explored in more detail.

Often omitted from the definition of a genetic algorithm, replication is the simplest operator. Reproduction takes a member from generation G_x and reproduces it in generation G_{x+1} . This is a unary operation, requiring the input of only one individual. Due to its simplistic and fundamental nature, reproduction is often not explicitly defined, even when it has been implemented.

Crossover is the process of deriving a new individual from two existing individuals by combining their genetic structure. Most often, this is implemented by selecting a random point within the two parent members, breaking their chromosome at this point and recombining the two halves with their corresponding complements from the other parent.

To ensure that the genetic algorithm does not converge to a local maximum, the mutation operation (as well as variants of it and other techniques) can be applied to individuals as a way of re-introducing genetic diversity (Ronald, 1995). In its simplest implementation, the mutation operation will randomly select a gene in the target chromosome and XOR it with one. This has the effect of flipping the bit value.

The inversion operator reverses the ordering of a subsection of the chromosome in an effort to bring genes with characteristics which are beneficial to the fitness of the chromosome into close proximity. This reduces the probability of these genes being split by the crossover operation and hence lowering the fitness of future offspring. The usefulness of the inversion operator has been questioned and never conclusively

proven (Davis, 1991; Goldberg, 1989a). It is often omitted to simplify implementation of the genetic algorithm, as its inclusion requires a locus independent representation scheme.² The inversion operation is expected to be of more use for representation schemes several orders of magnitude greater than those currently used (Davis, 1991).

Several more advanced operators exist that are not used within the traditional genetic algorithm, but rather are devised to cope with more elaborate representation schemes. Such a scheme may be required in a complex problem, or to more naturally represent a solution.

One such operator is that of the Partially Matched Crossover, proposed by Goldberg and Lingle (1985). This operator is designed to deal with representation schemes where information is encoded via ordering and each chromosome must have a full gene compliment. In this scheme, a crossover section is defined and swapped between two parent chromosomes; the duplicated genes in the children are exchanged to define two children with complete gene compliments. Such an operator might be useful in a travelling salesman problem, where all cities must be visited (all chromosomes must have a full gene compliment). Within such a problem the information of the solution is encoded in the ordering of cities rather than their presence or otherwise.

Other advanced operators are often modified version of the basic ones described here. For example, Goldberg (1989a) explains a modified version of the crossover operation to deal with a representation scheme that allows for chromosomes of varying length. This modified operator simply adds the step of alignment to allow for the selection of crossover point within the two variable length structures.

3.3.4 PARENT SELECTION

The operation of determining parents from a population for reproduction is crucial for emulating the natural selection process. That is, the purpose of this operation is to identify parents with a bias towards those with a higher fitness (Davis, 1991). Without this bias the algorithm would perform no better than a random search. This selection focuses the search on areas of the problem space that appear to contain promising solutions (Vamplew, 2004).

² That is, a scheme where the meaning of the gene is not determined from its location in the chromosome

A commonly applied selection scheme is that of Roulette Wheel selection. As described by Goldberg (1989a) and Davis (1991), this selection scheme is analogous to assigning a portion of a roulette wheel to each individual based on their fitness (fitter individuals receiving greater slices). The selection is then achieved by ‘spinning’ this hypothetical wheel and picking an individual based on where the wheel stops. More mathematically, each individual is assigned a segment of the number line $\{0..1\}$ proportional to their fitness; a random number in $\{0..1\}$ is then chosen which selects one individual.

Whitley (1989) explains how selection so tightly tied to fitness can cause the algorithm to stagnate or prematurely converge due to either a lack or oversupply of selective pressure respectively. He instead advocates the use of rank based selection, where each individual is ranked based on their fitness and each rank is assigned a certain number of reproductive trials.

For a more thorough exploration of selection and its effects on efficiency and effectiveness of genetic algorithms, the reader is referred to (Davis, 1991; Goldberg, 1989a; Koza, 1992). For further explanation of another common selection scheme, namely Tournament Selection, the reader is referred to (Goldberg & Deb, 1991).

3.3.5 SCHEMATA

Holland proposed the concept of schemata as a way of grouping chromosomes with similarities (Holland, 1975). The following representation of the schema theorem is derived from Tomassini’s notation (Tomassini, 1995). Holland’s *Schema Theorem* states that if the chance of reproduction is proportional to chromosome fitness, then the number of individuals m representative of a schema \mathcal{H} at time t is related to that at time $t+1$ as follows:

$$m(\mathcal{H}, t+1) = m(\mathcal{H}, t)(f_{\mathcal{H}}(t)/(f_{\mathcal{A}}(t)))$$

where $f_{\mathcal{A}}(t)$ is the average fitness of chromosomes in the population and $f_{\mathcal{H}}(t)$ is the average fitness of those chromosomes in the population which contain the schema in question (Davis, 1991; Goldberg, 1989a; Holland, 1975).

The importance of the result is that Holland’s Schema Theorem proves that a genetic algorithm allocates exponentially more trials to fitter schemata (Tomassini, 1995). This concept underpins the rationale for the effectiveness of genetic algorithms as well as the issue of convergence, both to be explored in more detail in sections 3.3.6 and 3.3.7 respectively.

The full equation, which also includes the effects of the mutation and crossover operators, is included for reference. A complete exploration of the schema theorem is beyond the scope of this review, but refer to (Holland, 1975) and (Goldberg, 1989a) for a more complete treatment of this fundamental concept. The complete schema theorem is as follows:

$$m(\mathcal{H}, t+1) \geq m(\mathcal{H}, t) (f_{\mathcal{H}}(t) / (f_{\mathcal{A}}(t))) [1 - (\mathcal{P}_c(\bullet(\mathcal{H}) / (l-1))) - o(\mathcal{H})p_m]$$

Where $1 - o(\mathcal{H})p_m$ represents the probability of a schema \mathcal{H} surviving mutation and $-(\mathcal{P}_c(\bullet(\mathcal{H}) / (l-1)))$ represents the disruption of schema by crossover, proportional to schema length.

There is, however, a major caveat: the schema theorem relies upon the genetic algorithm holding to the fundamental design, involving just the limited set of operators. Should the algorithm not conform to these specifications, the theorem is less applicable (Altenberg, 1995; Tomassini, 1995). Goldberg (1989a) provides further exploration of the schema theorem and elaborates upon the building block hypothesis which underpins it.

3.3.6 SUCCESS OF GENETIC ALGORITHMS

Various arguments have been presented as to why this non-deterministic approach can produce effective results in highly complex, non-linear problem spaces. Koza (1992) states that Genetic Algorithms violate many of the fundamental principles established in solving problems using Artificial Intelligence as well as other disciplines of not only computing but various fields of science.

Holland's justification for the success of genetic algorithms revolves around his schema theorem. He states that the algorithms work by creating combinations of genes in the chromosomes which contribute favourably to its fitness. Hence these combinations will likely be reproduced in the next generation. This is the basis for his inversion operation, which is intended to be a way of bringing certain genes within close proximity to each other. This increases the probability of the combination surviving to the next generation (Holland, 1975).

There has been significant debate into the validity of the Schema Theorem however and some believe that it provides no validation for the success of genetic algorithms. Nevertheless, the empirical data which exists demonstrates that Genetic Algorithms

do work. While many argue as to the success of the schema theorem in capturing the fundamental reasons, it is essentially accepted by all that the approach is successful (Koza, 1992). Discussion on formally proving the effectiveness of genetic algorithms is ongoing and many varying perspectives have been offered. Altenberg (1995) examines some of these in his paper and presents an alternative theorem for explaining the success of genetic algorithms.

Goldberg categorises possible search algorithms as belonging to roughly three groups: calculus-based, enumerative and “robust” search techniques. Calculus-based search techniques³ are highly effective within a limited domain, but perform very poorly in general. In addition, they are not readily applicable to combinatorial problems⁴ (Vamplew, 2004). Enumerative approaches perform equally well in all domains, but are of limited success on generic problems. Genetic algorithms then are an attempt to attain the performance characterised by the “robust” search techniques. That is, somewhat sacrificed performance in a limited domain, for overall good performance (Goldberg, 1989a). Figure 3.3.6.1 shows this hypothetical efficiency versus problem type relationship. De Jong (1975) demonstrates how genetic algorithms out-perform random search by making use of previously-learned information and avoid the problem of local maxima by searching the problem space from a number of different points.

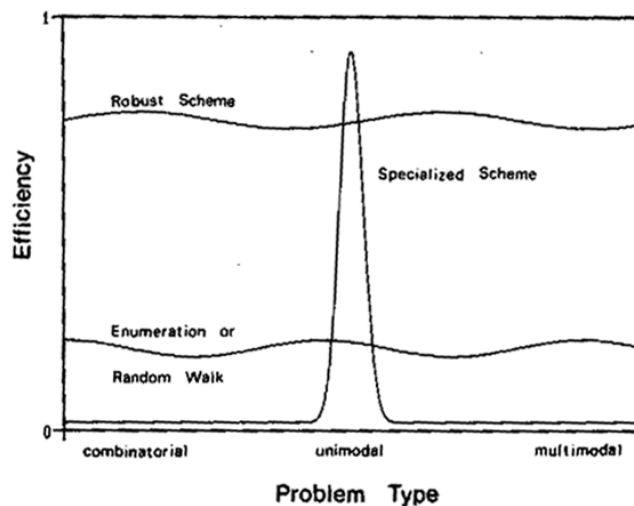


Figure 3.3.6.1: A hypothetical representation of algorithm efficiency plotted against problem type (Goldberg, 1989a).

³ Such as Hill Climbing or Gradient Descent

⁴ Combinatorial Problems are those within which solutions are found by making a number of interdependent choices.

3.3.7 CONVERGENCE IN SERIAL GENETIC ALGORITHMS

Convergence refers to the situation where genetic diversity is lost in a population and many of the individuals are of similar genetic structure. In the context of the search space, this means that the genetic algorithm has produced many solutions that are almost exactly the same. Visually represented, the population is clustered around a single or small number of points, often local maxima. Generally this is an undesirable condition, as it means the algorithm has been restricted to a very limited section of the problem space (Holland, 1975). It has been shown that elitist selection strategies, where the best individual (or individuals) of generation \mathcal{G} are automatically included into generation $\mathcal{G}+1$, can cause premature convergence (Davis, 1991). Goldberg (1989a) notes that elitism improves local search but degrades the global search capability of the algorithm. As stated above, much of the robustness characterised by genetic algorithms is derived from their ability to spread search points broadly throughout the problem space. A narrowing of this broad approach is detrimental to the capability of the genetic algorithm to effectively search the problem space.

The reason for such convergence is the allocation of trials to members of the population that perform well based on the fitness evaluation. Davis (1991) and Goldberg (1989a) discuss the two-arm and k-arm bandit problems⁵ which can be used to explain convergence. Briefly, the problem states that there is a slot machine, with two 'arms'. The user does not know which arm will produce the greatest winnings, but one will yield the optimal reward (call it the 'jackpot'). The problem then is how to allocate trials in such a way that the arm which appears to be best is used most, but the other arm is not completely neglected, in case it does eventually prove to hold the 'jackpot'. This represents a fundamental issue in adaptive systems, namely balancing exploration against exploitation of already known information (Goldberg, 1989a).

Generalising the problem to k possible 'arms', one can see how this applies to selection in genetic algorithms. Convergence occurs when the solution that appears to be the best is constantly being allocated trials, while other solutions are neglected even though they may ultimately lead to the optimal solution. Holland (1975) and De-Jong (1975) present this as allocation of trials to schema within the genetic algorithm.

⁵ A common metaphor for selection criteria of Genetic Algorithms

One can also draw comparison to the concept of greedy algorithms, which follow the ‘best next step’ approach and can miss global optima if they are surrounded by minima in the search space. Although not afflicted by this particular issue, premature convergence can result in genetic algorithms exhibiting similar characteristics. They are unable to reach the optima because it is not associated with the currently found best solution, which is dominating the search.

Many approaches have been introduced to the genetic algorithm design to control convergence, mostly through the use of meta-information gathered as the algorithm is running (Davidor, 1991; De-Jong, 1975; Goldberg & Deb, 1989). Such approaches most often focus on lowering the speed of convergence to allow a more thorough search or modifying the selection scheme to relax selective pressure (Lin, Goodman, & Punch-III, 1994). Avoiding convergence by maintaining genetic diversity allows the algorithm to distribute the search widely through the problem space. This improves the robustness of the algorithm and increases the probability and speed by which an optimal solution is found.

3.4. PARALLEL GENETIC ALGORITHMS

It is commonly accepted that genetic algorithms represent a highly parallel search mechanism. As such, the implementation of these algorithms upon parallel or distributed technology would allow for a more natural and complete exploitation of their power (Holland, 1975; Spiessens & Manderick, 1991; Whitley, 1993). Parallel implementations also allow for the application of GAs to more complex problems than would be possible with a serial approach, due to improvements in computational efficiency and speed.

3.4.1 ALGORITHM MODELS

Various models have been proposed to adapt the basic Genetic Algorithm to run on parallel hardware. In general, it is not possible to distinctly categorise these models, as many implement techniques from various styles of genetic and parallel genetic algorithms. For the purposes of this review then, different techniques and approaches will be outlined with reference to their effects on accuracy and efficiency of the algorithm. The comparison of separate techniques in parallel genetic algorithms is complicated by the changes to the fundamental operation which are introduced by various modifications. To this end, it is important to explicitly state what characteristics of the algorithm are being compared, for example quality of solution (Cant'u-Paz, 1997).

Each approach has characteristics which may or may not be favourable to certain applications. As Goldberg states, a general purpose algorithm that can be applied across a large number of domains with good performance is desirable. However, there always exists the ability to tailor the algorithm to a specific application by the use of heuristics and similar domain specific knowledge to improve performance. The key here is to be aware that such a narrowing of the algorithm's focus means that the robustness inherent in the concept of genetic algorithms is lost. Such search algorithms are redirected back towards calculus-based and similar approaches (Goldberg, 1989c).

3.4.2 DESIGN GOALS

To design a *Parallel Genetic Algorithm* (PGA) that can take advantage of the distributed nature of GAs it is necessary to distribute the workload across multiple processors. Based on the granularity, this may constitute large almost independent populations or smaller more interacting populations. The method of implementation and control of the genetic algorithm will influence how much data needs to be exchanged between the processors and whether a master processor need be designated.

Gorges-Schleuter (1989) states that to achieve an efficient parallel implementation, the following need to be satisfied: no central control, local interaction and fault tolerance. Spiessens and Manderick (1989; 1991) also argue for the global control structures of a GA to be eliminated. They cite the dependence of reproductive mechanisms on knowing the average fitness of the population as a drawback to developing parallel Gas. This is because it requires constant information exchange between processors/populations. Collins and Jefferson (1991) agree, stating that natural systems have no such global control. They continue by showing that applying appropriate population dynamics⁶, the effects afforded by such controls are shown to be emergent behaviour. It has also been Goldberg's (1989c) finding that attempts to over-control or over-specialise the algorithms lead to a reduction in robustness. Such arguments also implicitly promote the use of steady-state algorithms, as the strictly enforced generations of a traditional approach are also artificial.

⁶ A local mating scheme and conflicts, amongst other additions

3.4.3 POPULATION GRANULARITY

3.4.3.1. GLOBAL PARALLELISATION

The simplest approach to implementing a parallel genetic algorithm is to maintain the single large population of a serial algorithm but distribute the computation of fitness and application of the genetic operators across multiple processors. This approach is easy to implement, but as it leaves the behaviour of the algorithm unchanged, does not afford any improvement in premature convergence. A potential drawback to this approach is the excessive communication needed between processors. Evaluation of fitness and production of the next generation requires information about all individuals within the population, requiring all processors to exchange large amounts of data for every generation. This can cause a bottleneck in the system, limiting improvements in computational speed (Cant'u-Paz, 1997).

3.4.3.2. COARSE-GRAINED

Consider a starting point for parallelising a genetic algorithm as taking a standard GA as described in section 3.3.2 and running this algorithm in parallel on multiple processors. From time to time, the parallel processors would exchange information, and/or individuals. Such an approach is often called a distributed or coarse-grained genetic algorithm (Tanese, 1989). The sub-populations are often referred to as *islands* or *demes* (biological analogues) and the period where a subpopulation evolves in isolation is often referred to as an *epoch*. Such an approach has several advantages, not least of which is the ease of conceptualisation and implementation. Put simply, one may implement a standard serial genetic algorithm, with some minor additions for communication between the processors, and run instances on as many processors as desired.

Tanese (1989) concluded that such an algorithm with no migration of individuals between processors could produce a near linear speed-up over a standard serial genetic algorithm. It consistently out-performed the serial implementation by discovering fitter individuals. However, the algorithm was unable to maintain average population fitness comparable to the serial implementation. With the introduction of migration, Tanese demonstrated that this drawback could be eliminated. More recent work has confirmed these findings, showing that global optima are found more often using a coarse-grained algorithm with migration than a serial algorithm (Belding, 1995). Furthermore that migration of individuals between the populations in a coarse-grained algorithm can improve execution time (Wang, Maciejewski, Siegel, & Roychowdhury, 1998).

Such models promote genetic diversity due to the isolation of the populations. As a result, premature convergence is less of an issue in a coarse-grained parallel genetic algorithm (Baluja, 1993b; Li & Kirley, 2002). In addition Coohon et al. (1991) cite work in the genetics field, most notably by Wright (1932; 1964; 1982), into rapid evolution as supporting the sub-population model with occasional inter-population communication as a method of “extensive search of the adaptive landscape”. In addition, exchange of individuals should occur when subpopulations reach equilibrium – a state at which evolution has halted because phenotypic improvement is balanced with phenotypic degradation (J.P. Cohoon, Hegde, Martin, & Richards, 1988).

Baluja (1993a) notes the drawbacks to this approach however, in that once a subpopulation reaches equilibrium, introduction of new genetic material may not be sufficient to perturb this condition. Selection of migration rate and the number of individuals to transfer determines the effectiveness of the algorithm (Cant'u-Paz, 1997).

Investigation into coarse-grained parallel genetic algorithms where serial algorithms are being executed in parallel with occasional transfer of individuals can be shown to conform to De-Jong's proof of efficient allocation of trials to schema (De-Jong, 1975; Petty & Leuze, 1989). This is an important consideration, as it means theory developed in regard to serial implementations is equally applicable in these parallel algorithms.

Communication between subpopulations is generally limited in a coarse-grained genetic algorithm, allowing for more flexible deployment over parallel architectures. That is, low bandwidth, intermittent or unreliable network connections will restrict implementation of such an algorithm less than a fine-grained approach. Application on relatively cheap hardware is also possible, as no high-end parallel architecture is required (Cant'u-Paz, 1997). Finally, processing of the subpopulations may be more efficient due to the lack of any need to maintain global control structures and conduct extensive communications to maintain synchrony with other processors.

3.4.3.3. FINE-GRAINED

A fine-grained approach to parallel genetic algorithms involves splitting the population into many smaller populations (or demes). The term *massive parallelism* is commonly used to describe this style of parallel approach, as many small, interacting populations are evolved in parallel. Many fine-grained parallel genetic

algorithms define a geographical construct upon which individuals or populations are placed, often a two dimensional Cartesian plane. This allows the definition of the *distance* between individuals or populations (commonly via Cartesian geometry).

It is common for subpopulations in a fine-grained algorithm to overlap. That is, a member may belong to multiple populations. This addresses the issue of equilibrium of subpopulations described in section 3.4.3.2, as the probability of a subpopulation reaching equilibrium while its neighbour populations have not is unlikely (Baluja, 1993b).

Fine-grained algorithms suffer from the loss of genetic diversity, due to the small population size. That is, a high performance schema can quickly dominate a small population. Because of constant population interaction, this schema can be spread quickly amongst the other populations, leading to premature convergence (Baluja, 1993b). This can be overcome by having a large number of populations. Separating populations by a large distance (that is, number of population ‘hops’) allows for the uncontrolled spread of dominant schema to be retarded (Collins & Jefferson, 1991).

ASPARGOS, a fine-grained algorithm proposed by Gorges-Schleuter and Muhlenbein exhibits a self organising nature, such that global control is not required, with interacting demes leading to speciation. It requires relatively little communications and exploits natural parallelism (Gorges-Schleuter, 1989). The *neighbourhood model*, as used in ASPARGOS, is highly efficient on parallel computers (Muhlenbein, 1989) and does not require a *sharing function*⁷ or any other external control parameters. However, the algorithm is dependent on an expensive dedicated parallel architecture, which severely limits its general application (Davidor, 1991). It also contains a local hill-climbing optimisation, making it difficult to determine the effectiveness of the genetic algorithm component (Cant'u-Paz, 1997).

Spiessens and Manderick (1989) provide two motivations for eliminating global control and promoting massively parallel computation of a genetic algorithm. The first of these is that, as Goldberg states, the optimum size of a population is proportionate to the chromosome length used for the problem representation (Goldberg, 1989b; Goldberg & Richardson, 1987). Thus, as genetic algorithms are applied to more complex problems, larger populations will be required, complicating global control and requiring more overhead. Their second argument relates to

⁷ Goldberg (1989a) suggested a ‘sharing function’ as a method of curtailing the loss of genetic diversity, however it requires global information to function

approximating natural selection more accurately, where there of course exists no global control.

There exist numerous other hardware-dependent fine-grained genetic algorithms, which have outperformed their serial and coarse-grained counterparts. The reader is referred to Baluja (1993a) for an explanation of several implementations of the fine-grained parallel genetic algorithm. There is, however, a caveat: since these algorithms are often only executable on large-scale, highly-parallel architectures, an accurate empirical comparison is difficult to produce.

3.4.3.4. MICRO-GRAINED

Micro-grained parallel genetic algorithms spread the evaluation of the fitness function across multiple processors. They do not split the population, maintaining a single large group. This approach does not address the issue of convergence, but simply allows for speed improvements. Punch et al. (1993) demonstrated that a micro-grained algorithm will allow for a halving of computation time for every doubling in processors up to the population size. That is, a linear speedup over a serial algorithm is achieved. Micro-grained algorithms are employed where evaluation of the fitness function is computationally intense in comparison to other operations of the genetic algorithm (Lin et al., 1994).

3.4.4 HYBRID, HIERARCHICAL AND DISJOINT ALGORITHMS

One final observation on the topology of parallel genetic algorithms can be made in regard to the ability to combine certain techniques to create hybrid algorithms; the intention of which being to capture the positive qualities of various approaches while balancing or eliminating the negative ones. The reader is referred to Cant'u-Paz (1997) for an exploration of this concept, which is at best on the periphery of this review.

Hierarchical models, as described by (Noda, Coelho, Ricarte, Yamakami, & Freitas, 2002), allow for levels or tiers of algorithms. In such an approach, one tier may compute solutions which are then combined and processed further at a higher tier.

Certain parallel genetic algorithms have been developed that employ more than one serial algorithm specification. This is most applicable to coarse-grained genetic algorithms where each subpopulation may be evolved by a different serial algorithm. Each such algorithm can possess different rates of mutation and crossover plus a

unique population size. Such algorithms are sometimes referred to as heterogeneous (Noda et al., 2002). Running separate subpopulations with different control rates can allow selective pressure to be tailored, balancing exploration and exploitation (Tanese, 1989).

3.4.5 MIGRATION SCHEME

Migration of individuals between various subpopulations of the algorithm is crucial to its effectiveness. Too low a migration rate and the subpopulations can reach permanent equilibrium. Too high and the algorithm performs much as a serial implementation, losing the benefits of parallelisation and potentially even exhibiting slower performance due to communication overhead (Cant'u-Paz, 1997).

Generally, migration can be classified as either the Island Scheme or the Stepping Stone Scheme. The island model allows for the migration of members from any population to any other population. That is, consider the processors/populations to be vertices in a complete mesh. The stepping stone model restricts communications between subpopulations based on an imposed logical or physical limitation (Noda et al., 2002).

The choice of scheme reflects the balance desired between exploration and exploitation in concert with the balance between computation and communication (Noda et al., 2002).

3.4.6 DOMAIN KNOWLEDGE

Several approaches to implementing problem specific information into the execution of a genetic algorithm exist. Generally, domain knowledge is incorporated in the genetic operators and the fitness function. Individuals that breach certain constraints or rules of the problem domain may be penalised when assigned their fitness.

Using a travelling salesman problem, (Wang et al., 1998) implemented an approach whereby solutions are evolved to sub-tours. These sub-solutions are then recombined at various instances in the execution of the algorithm based on selecting appropriate re-combination points. Separate populations are used to derive solutions to each sub-tour and operate essentially independent from other sub-populations. Improved performance was gained through the implementation of this heuristic.

Cooperating genetic algorithms are also possible, where each algorithm computes solutions to sub-problems. Combining this with the hierarchical model, Noda et al (2002) discusses the prospect of having one tier of algorithms generate solutions to sub-problems of a higher controlling algorithm. That is, specialised sub populations for solving sub problems generate their results, which then represent a combinatorial problem for the higher tier. Such an approach can be generalised to an n-tier system, but requires domain knowledge to select sub-problems.

Goldberg cautions that by using problem-specific heuristics and complicating the genetic algorithm design, the algorithm performance is shifted away from the natural origins of the technique. The robust search space is then restricted back to the spike of calculus based and similar approaches (Goldberg, 1989c).

On a related point, Davidor (1991) supports the argument that PGAs can be designed that have naturally occurring speciation such as in the ASPARAGOS model. He uses a geographic construct and models his approach on natural forces seen to influence genetics. In this way, Davidor demonstrates how organisation is emergent from the genetic algorithm when appropriate pressures are applied.

3.4.7 CONVERGENCE IN PARALLEL GENETIC ALGORITHMS

Characteristics of certain parallel genetic algorithm approaches minimise the probability of premature convergence. As it is possible to run a parallel genetic algorithm on a single processor, it is not uncommon to find such a technique being employed simply to control convergence. This is in contrast to the intuitive speed and computational advantages that can be gained from a parallel implementation (Gondra & Samadzadeh, 2003; Lin et al., 1994; Noda et al., 2002).

Selection pressure is what leads to convergence of the population. That is, the weak members are eliminated from the population by reproductive processes that are biased towards preserving individuals with high fitness ratings. Thus, diversity in the population is lost, due to the fittest schemata becoming dominant (See section 3.3.7). Spiessens and Manderick (1989) discuss how the reduction of selection pressure in a fine-grained algorithm helps alleviate this. That is, they allow for schemata that do not perform as well to remain in the population and hence maintain diversity.

As discussed, the use of global information gathered during the running of the genetic algorithm is of detriment to the efficiency of the parallel algorithm. Instead, several approaches have been formulated that eliminate the need for such global

information, allowing the control of convergence through the creation of implicit species.

Collins and Jefferson (1991) also identify the problem of convergence, where diversity is lost and the population is localised to a single peak in the search space. They acknowledge attempts to eliminate this problem via crowding, sharing and restrictive mating, but dismiss these as requiring global knowledge. They conclude that convergence can be avoided and more peaks within the search space can be explored by introducing a local mating scheme.

Davidor (1991) developed a model to allow rapid local convergence by imposing population interactions, which leads to niche and species like behaviour. Through this controlled convergence, the algorithm can concentrate on promising areas of the search space. Importantly however, it does not lose the generality which allows genetic algorithms to be applied effectively to complex and deceptive problems. In addition, the model does not require global information to be collected, which facilitates application on parallel hardware.

3.5. CAPACITED VEHICLE ROUTING PROBLEM

3.5.1 CONTEXT

While exploring the proposed test problem, the *Capacited Vehicle Routing Problem* or CVRP, the main objective of the project should not be overshadowed. There exist several methods of solving CVRP instances that may or may not outperform genetic algorithm solutions. This fact is beyond the scope of this projects objective, which is to evaluate speed improvements gained through a parallel implementation of genetic algorithms and integration of heuristics. To this end, the CVRP has been selected as a means, rather than an end, because it represents a problem within which heuristic approaches are intrinsic. By no means does this project set out to provide an efficient, effective genetic algorithm based solution to the CVRP. The purpose of this section then is to provide a brief introduction and exploration of previous work on applying genetic algorithms to the CVRP. Heuristic techniques will also be explored.

3.5.2 PROBLEM DEFINITION

The Capacited Vehicle Routing Problem is essentially a graph optimisation problem. There exist a number of vertices in the graph, each one representing a *customer* with a specified, finite demand for a single *commodity* (there is only one

commodity in the problem). One of the vertices will be marked the *depot*, which is where the commodity is served from. The commodity is delivered by a number of vehicles, with a fixed carrying capacity. Distance between vertices is commonly determined using Cartesian geometry.

The goal of the problem is to find an optimal graph which satisfies all the demands but minimises the travel time and number of vehicles required. A *feasible solution* is defined as being a set of routes, where for each route the sum of demands for each customer on that route does not exceed the capacity of the vehicle servicing the route and each customer is present in one and only one route. This is graphically represented in figure 3.5.2.1:

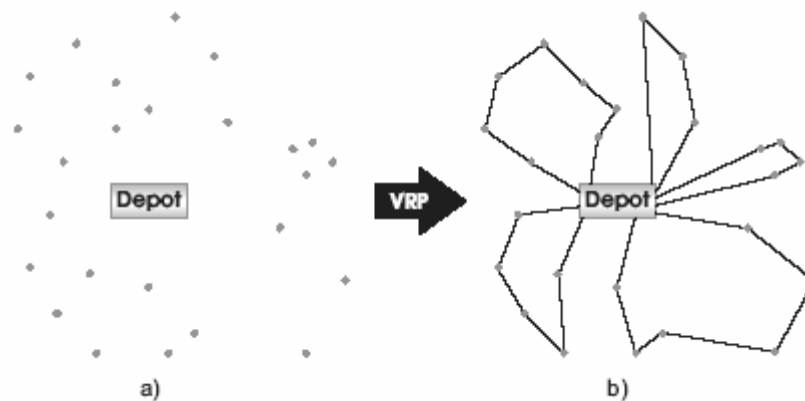


Figure 3.5.2.1: a) A distribution of clients and a depot.
b) A graph defined to supply clients
(Alba & Dorronsoro, 2004)

3.5.3 GENETIC ALGORITHMS APPLIED TO CVRP

For the purpose of investigating the application of genetic algorithms to solving CVRP instances, this review will consider the more general *Vehicle Routing Problem* (VRP) and variants of it. Approaches to solving these similar set of problems can often be directly applied or grant insight into solutions to the CVRP.

Research into providing fast, accurate solutions to the capacited vehicle routing problem have revolved mostly around applying advanced heuristics and augmented hill-climbing techniques. Application of genetic algorithms techniques, however, have been limited and in many cases shown to be less efficient than current best known approaches (Machado, Tavares, Pereira, & Costa, 2002). Berger and Barkaoui (Berger & Barkaoui, 2003) argue that this is simply because genetic algorithm

approaches have not been explored to their fullest extent and, with a more thorough investigation, can provide improved performance.

A standard approach to solving the CVRP using genetic algorithms is described in (Machado et al., 2002). The technique uses a representation scheme within which each gene represents a destination customer and the order which they are in defines the order of delivery. A special gene separating each route is introduced. The algorithm uses a fixed chromosome size, which is desirable as it simplifies implementation; however it also constrains the search. Combine this with the fact that candidate solutions are discarded if they violate the capacity constraint and the reproductive scheme is specific to the problem space and it becomes apparent that the algorithm is very tightly controlled.

Other approaches have focused on applying simple, general purpose genetic algorithms. One such approach is that outlined in Alba & Dorronsoro (2004), where a cellular genetic algorithm⁸ is used with certain heuristics to improve performance. Rather than rejecting solutions that violated constraints within the problem, these solutions simply had their fitness values penalised. Alba and Dorronsoro demonstrate in their paper that a genetic algorithm with multiple populations outperformed those with one single large population. Furthermore, with the inclusion of local search approaches and heuristics it was demonstrated that the Genetic Algorithm approach exhibited comparable performance to other well known techniques.

3.5.4 HEURISTICS AND DOMAIN KNOWLEDGE

The incorporation of domain knowledge and heuristics into the GA allows for an improvement in performance, but with the loss of generality in the algorithm.⁹ What follows is a brief explanation of several insights into the domain of the CVRP, which can be used to improve a solution algorithm.

Notice that if the edge values in the graph (that is, distance from the depot to each customer) are zero, the problem simplifies to a bin-packing problem (often called the knapsack problem). The CVRP is sometimes considered a union of the bin-packing problem and the travelling salesman problem (Machado et al., 2002; T. K. Ralphs, Kopman, Pulleyblank, & Trotter, 2001). As such, the possibility of splitting the CVRP into its constituent BPP and TSP halves represents an intriguing method of

⁸ A cellular genetic algorithm is one where the population is mapped to a landscape and individuals may only interact with their neighbours.

⁹ Meaning that the algorithm can now only be applied to a single set of problems, rather than being applicable to any problem for which a representation could be found.

applying problem specific knowledge to its solution. Ralphs et al. (2001) further describe the division of the graph, to form small, distinctly separate travelling salesman problems. These are then optimised to provide the smallest travelling distance for each sub-problem, and finally checked to ensure they meet the capacity constraint.

Another common heuristic optimisation that is often incorporated into the solution of such problems is that of the nearest neighbour operation. Put simply, this is an attempt to optimise the destinations in a route such that for each destination the next destination is the one closest to it (its nearest neighbour). Intuitively, this should reduce the travel distance, but it is possible to produce problem instances where this greedy approach is not successful. Similar approaches are used in travelling salesman problems, which can be considered a sub-part of the CVRP. An extension of this idea is that sub-graphs can be formed by applying a k nearest neighbour algorithm (KNN).¹⁰ Yang used this technique to show that population size can be reduced if representative chromosomes can be created for the initial population (Yang, 1997). However, another intriguing application is in splitting the problem into constituent parts. Wang, Maciejewski et al. explored this approach to solving instances of the travelling salesman problem and concluded that by dividing the problem into sub-problems, better solutions can be found faster (Wang et al., 1998). A cautionary note is necessary however; such results are clearly reliant on providing a suitable break-up of the problem. Within problems of a similar geometric based distribution as the TSP, of which the CVRP is one such instance, a clustering algorithm can be an effective method of dividing the problem into sub-problems. The simple KMeans algorithm (MacQueen, 1967) is one such clustering approach.

Yang (1997) investigated applying heuristic data to a genetic algorithm based solution of the travelling salesman problem. It was demonstrated that the combination of the heuristics to guide the GA in an appropriate direction and the GAs ability to avoid local optima could be balanced to provide efficient solutions. The approach incorporated domain knowledge in the genetic operators, such that reproduction was artificially biased towards producing offspring that satisfied certain heuristics.

Other heuristic approaches to solving the TSP and related problems (such as the CVRP) function by exchanging cities within a tour to decrease the length. For more

¹⁰ This works by picking a random city and selecting its K nearest cities to form a sub-graph. If one of the k nearest cities is already present in another sub-graph, a random city is selected from those yet to be allocated to a sub-graph.

details on these and other approaches to solving the travelling salesman and related set of problems, the reader is referred to (Nilsson, 2003).

Finally, Goldberg (1989c) offers the following cautionary note on the specialisation of GAs by incorporation of heuristics; such approaches should be recognised for their effects on the methodology and balanced by considering what is being achieved and at what cost. By augmenting a genetic algorithm to better handle a single set of problems, all (or many) other problems are likely to be excluded from its potential applications. As with all things, balance is important.

3.6. GRID ENGINE ARCHITECTURE

Grid engine is an open source distributed computing framework within which parallel and distributed applications can be developed. The project is sponsored by Sun Microsystems and provides “distributed resource management software for wide ranging requirements from compute farms to grid computing.” (Grid-Engine Homepage, 2001)

The grid engine architecture provides for resource sharing, allowing tasks be submitted to the software and distributed to machines based on load and performance requirements. The general purpose of the software is to allow for more efficient use of networked computational power (Lee, 2002).

Automated decision making is beyond the current capabilities of the system (Andrzejak, Graupner, Kotov, & Trinks, 2002). The architecture allows for remote execution of tasks at a level of granularity roughly equivalent to that of processes. Thus, it is the responsibility of the developer to ensure that the application is separated into sufficiently many processes to achieve the level of parallel execution desired. In addition, grid engine does not provide automated communication between the processes distributed to separate processors. If inter-process communication is required (as it often is in parallel executions, if only to return results to a “master” process) the developer is required to implement a mechanism to provide this.

Grid engine does not allow for dynamic allocation and removal of resources to the grid, requiring all machines to be pre-configured and listed as members of the grid. In addition, all grids require that one host be designated the master, requiring at least some hierarchy for the development (Balachandran, 2003). The software does however provide automated auditing facilities, allowing for the collection of statistics regarding resource usage and execution times.

3.7. SUMMARY

This review has provided a high-level overview of the issues inherent in the use of genetic algorithms to solve complex problems. A description of the concepts that underpin the operation of genetic algorithms was presented, including the schema theorem, convergence and the genetic operators. This was followed by an exploration of past research into exploiting the parallel nature of genetic algorithms and in particular classified approaches based on criteria such as granularity. Advantages and drawbacks of techniques were highlighted and several algorithms and heuristics designed specifically for capacited vehicle routing and associated problems were presented.

Throughout the review, the context of the project was emphasised in an attempt to prevent the focus from digressing. From the perspective provided by this review, it is now possible to select and deploy genetic algorithms upon the grid engine topology that meet appropriate criteria to test the hypothesis. The design and specification of these algorithms is explored within the subsequent chapters.

4. METHODOLOGY AND GENERAL EXPERIMENT DESIGN

4.1. INTRODUCTION

In formulating an approach with which to either prove or disprove the hypothesis presented above, the requirements must be more formally specified. This chapter will begin by selecting a series of metrics upon which to base comparative assessment of several approaches to parallel, distributed and augmented (i.e. heuristic based) genetic algorithms. Following this, an acceptable representation scheme will be formulated based on the test problem characteristics. Thereafter, genetic operators and a selection scheme will be identified that satisfy the requirements of the domain. Finally, general experiment design will be described and the expected results will be explored such that a comparison with measured empirical results can be made in a later chapter.

4.2. METRIC SELECTION

Of initial concern is a metric, or a series of metrics, upon which it will be possible to assess the efficiency and execution characteristics of any given approach. As stated by Lobo, Lima, & Martires (2004), there are three main metrics to consider in measuring efficiency in genetic algorithms: execution time, solution quality, and memory usage. Although dismissed as essentially constant in a traditional genetic algorithm by Lobo et al., memory usage within different parallel implementations may vary and could be of significance. In addition, the communication overheads of a distributed approach and CPU usage of an augmented algorithm are of concern. Therefore, the values of execution time, solution quality, memory and CPU usage, and communication requirements will be used to gauge the efficiency and effectiveness of algorithms.

As an aside, the metric of final solution quality is in fact rather coarse-grained insofar as it does not provide insight into the characteristics of the algorithm in achieving this final output. Indeed there are several concerns when considering the quality of solutions generated by the genetic algorithm. Primarily, due to the fact that the algorithm will be run over a predefined number of generations, the consideration of whether the algorithm is converging prematurely or continues to exhibit genetic diversity is of consideration. That is, having allowed the algorithm to continue running past the end of this set number of generations, would it be likely to improve the quality of its solutions or simply waste the extra computational time? At this

point a description of the techniques available for determining this are not presented. However, a more thorough motivation for their need and a complete specification of the methods used for measuring this are presented in chapters 7 and 8 respectively.

Obviously, within certain applications, some metrics may not be applicable, for example communications overhead within a single processor implementation. Furthermore, the weights attributed to each metric may vary. That is, a metric may sometimes be important and sometimes not. For example, one may desire an algorithm that produces good quality solutions no matter the execution time required. In many cases, where the measurement of a particular metric did not demonstrate any observations of interest, it is omitted from the results in an effort to avoid obfuscating important findings.

All metrics are measured over multiple executions of the algorithm in question and the observed results are averaged. Outliers are discarded where they are significantly disparate from observed trends. In comparing metrics, the stochastic nature of genetic algorithms is taken into consideration and the standard deviation of results is measured. From this, it can be determined if observed discrepancies in measurements are in fact representative of algorithmic performance differences or are simply artefacts of the implicit random nature of genetic algorithms. Throughout the remainder of this work, these metrics will be referred to by the terms *performance* metrics and *quality* metrics. The distinction between these two is made by whether the metric is measuring the quantitative performance of the algorithm, such as CPU time, run time, memory usage, etc. or the qualitative performance of the algorithm, such as quality of solutions or amount of genetic diversity.

4.3. TEST PROBLEM AND ALGORITHM MECHANICS

4.3.1 PROBLEM REFINEMENT

The test problem employed for these experiments will be that of the capacited vehicle routing problem, as described in section 3.5. A single instance of the problem, M-n101-k10, which is described in full within appendix B, will be used for every experiment to ensure there is no variance in results due to problem complexity. Note that all problem instances used within this work, excepting the test instances, are from a standard library of capacited vehicle routing problems. Their optimal solutions are also provided from this library (T. Ralphs, 2003). A variable sized vehicle fleet will be assumed with uniform, fixed vehicle capacity originating from a single depot. The test problem is a means rather than an end, and as such the effective and efficient solution of the capacited vehicle routing problem is not the

ultimate goal of these experiments; rather a uniform comparison of the algorithms employed is the desired outcome.

4.3.2 ENCODING SCHEME

The appropriate selection of encoding scheme requires consideration of the problem at hand. A complete solution to an instance of the capacited vehicle routing problem requires the selection of multiple routes containing a number of destination nodes each. All distances are symmetrical, that is $\mathcal{D}_{ij} = \mathcal{D}_{ji}$ where \mathcal{D}_{xy} represents the distance from node X to node Y , and in addition $\mathcal{D}_{ii} = 0$. Every destination node has a demand, d_i . Therefore, a solution to the problem is a permutation of the nodes into several routes where each route \mathcal{R}_q satisfies the equation:

$$\sum_{p \in \mathcal{R}_q} d_p \leq C$$

where C is the vehicle capacity which is fixed for all routes within the problem (Pereira, Tavares, Machado, & Costa, 2002).

Consider the following scheme for representing such a solution within a genetic algorithm. Each gene will represent a single destination in the problem space or a vehicle. All destinations following a vehicle designation are considered to be a route. A graphical example is given in figure 4.3.2.1, which shows two routes.

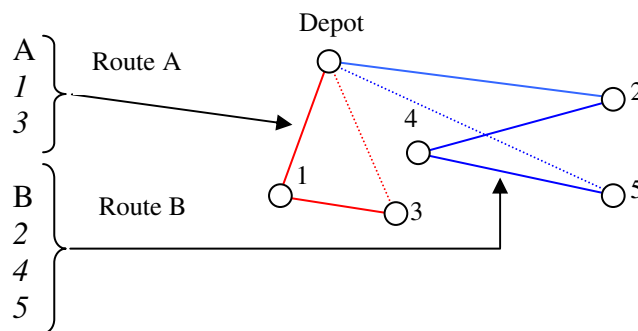


Figure 4.3.2.1: Graphical representation of a sample chromosome within the initial representation proposition

This representation does not enforce restrictions on solutions, allowing for poor solutions with good fragments to still exist. It is partially locus-independent because a destination retains its value regardless of location within the chromosome, but its

location does imply the order of a route. Undesired characteristics (for example, a solution with routes that exceed the vehicle capacity) can be penalised in the evaluation of fitness for the chromosome. There are, however several serious problems with this style of representation. Because of the order dependence and variable length of the chromosomes, the crossover operation is likely to produce a high number of invalid chromosomes; that is, chromosomes which do not represent a possible solution to the problem. Partially Matched Crossover (PMX – see section 3.3.3) can be used to deal with the order dependence of the chromosomes but its application is complicated by variable length.

As an alternative to the scheme presented above, consider instead separating the destination information from the route delineators. Each individual would now be represented by two interrelated chromosomes; one for destination order and one for route delineation. This allows for the application of the PMX operator to the now fixed length destination chromosome and the modified crossover operation (see section 3.3.3) to the variable length route delineation chromosome. This also eliminates all possibility of invalid individuals.

4.3.3 GENETIC OPERATORS

Standard genetic operators (i.e. crossover, mutation and replication) will be employed, with minor augmentations in some cases to deal with the representation scheme developed. Each will be weighted at a fixed amount for all experiments to avoid variance due to changes in operator application probability. As stated already, PMX will be used to ensure each solution contains a full gene complement. Variable length crossover will be employed to deal with the mutable size of the route delimitation chromosome. The genetic operators used within this work are described in more detail during section 5.2, where they are tested for correct performance.

4.3.4 SELECTION SCHEME

A simple rank based selection scheme where all individuals in a population are sorted based on fitness value and assigned a rank will be employed (see section 3.3.4). This will minimise the effects of super individuals on selection pressure (Goldberg, 1989a; Whitley, 1989). Probability of being selected for reproduction will be proportional to rank. That is, members with a high rank will be allocated more reproductive trials than those with lower rank.

4.3.5 MIGRATION SCHEME

The migration scheme used within the parallel variants of the algorithm will involve the migration of a random individual from each island to a random island every generation. Although the algorithm has been developed with the capacity to vary this migration scheme by selecting only high fitness individuals for migration or reducing the migration frequency, preliminary observations suggest that modification of these attributes does not result in significant improvement of performance within this domain and experimentation involving this is not explored further. Unless otherwise stated, the migration scheme in use is that which is described here.

4.4. GENERAL EXPERIMENT DESIGN

4.4.1 REQUIREMENTS SEPARATION

The stated hypothesis can be separated into several smaller intermediate goals, some of which are not directly dependent. The first of these goals is to prove or disprove the assertion that parallel genetic algorithms outperform their serial counterparts. Following this, there is the need to show that a distributed approach is either more efficient or less efficient. Finally, the effects of adding heuristics to the algorithm must be measured and determined to either improve performance or not. Measurements of performance, efficiency, and effectiveness will be based upon the metrics outlined in section 4.2. The following sections will present an experiment design aimed at addressing each of these requirements.

4.4.2 SERIAL AND PARALLEL COMPARISON

As stated in section 3.4.7, a parallel genetic algorithm need not be distributed over multiple processors. To make the comparison between the serial and non-distributed parallel variants of the genetic algorithm, the design explained in section 4.3 will be implemented and tested on the capacited vehicle routing problem as specified in section 3.5. The algorithm will then be parallelised, maintaining the operator structure and essentially all of the mechanics of the algorithm. The two algorithms will be measured based on the metrics presented in section 4.2 and comparisons drawn.

Fine-grained parallel algorithms are generally designed for specific, high end hardware. Indeed the concept of assigning a single member of the population to each processing element is not applicable to a single processor implementation. Therefore,

a coarse-grained approach will be taken within this and all further experiments herein.

The selected parallel implementations will employ a coarse-grained, multi-population approach with migration between the populations in a timely manner. In this model, each island will select another for migration to which it will send a random member of its population. There will be no restriction on the destination island for migration; that is, no topographical or geographical structure is imposed conceptually upon the island organisation.

4.4.3 DISTRIBUTED AND NON-DISTRIBUTED COMPARISON

A distributed version of the parallel algorithm introduced in section 4.3 will be tested and its results compared to that of its non-distributed and serial counterparts. The purpose during this step is to ascertain the validity of the claim that a distributed parallel algorithm will outperform the serial and non-distributed variants. Improvements from the non-distributed parallel to the distributed parallel are expected to be performance improvements only; that is, the quality of solutions found are expected to be comparable. During this stage, the number of processors available for the distributed algorithm will be incrementally increased to determine the relationship between cluster size and algorithm efficiency. All machines will have the same memory and CPU characteristics.

4.4.4 HEURISTIC COMPARISON

The final stage of experimentation will involve adding heuristics to each of the algorithms. Two heuristics based on domain knowledge of the problem will be incorporated. The first of these will allow for the re-ordering of chromosomes such that the destination selected first in any particular route is that which is closest to the depot. The second will be used within a conceptual decomposition of the problem into sub-problems for parallel evaluation.

4.5. GENERAL EXPECTATIONS

Based on the review of genetic algorithms presented in chapter 3, it is possible to make educated predictions of the results that should be expected from these experiments. This section will present the outcomes that are expected, allowing for comparison with actual observed empirical results at a later stage.

It is expected that the non-distributed parallel algorithm will outperform its serial counterpart due to the increased parallel exploration of the search space. This is due to the improved maintenance of genetic diversity and the greater search area afforded by the parallel approach, as outlined in section 3.4. Expected results are better quality solutions within the same timeframe as the serial algorithm but greater total memory and CPU usage. Communications overhead is not applicable to this experiment. In short, the parallel algorithm is expected to show improved performance at an increase in computational requirement.

The distributed parallel algorithm is expected to demonstrate an almost linear speedup over the non-distributed variant as the number of hosts increase. This is in the number of candidate solutions it can evaluate, not necessarily in the quality of solutions found. A certain amount of improvement in execution time will be lost to communications overhead. Distributed memory usage is expected to be essentially identical to the non-distributed version.

Finally, the heuristic-enhanced versions of the algorithm are expected to show improved solutions quality in a given execution time over all other variants with essentially the same memory usage and communications overhead. Average CPU usage per machine is expected to be slightly greater however.

These experiment designs coupled with the expected results provide a more detailed specification of the hypothesis. Each of these experiments will be run over a 100 generation period, unless otherwise stated.

5. ARCHITECTURE, TOPOLOGY AND CONFIGURATION

5.1. ENVIRONMENT

Within the previous chapter, the general requirements of the analysis and the overall structure of experiments designed to provide insight into the proof or otherwise of the hypothesis were presented. It is now relevant to describe the architecture upon which these experiments will be run. This, of course, is a defining factor in an appropriate low level specification of the desired experiments. Such low level specifications are explored in depth within the next chapter.

The execution platforms can be conceptually separated into two classes; the 233MHz execution class and the 800MHz execution class. These two classes of course refer to the clock speed of the relevant processors. Moreover, as was done in previous chapters, the algorithms can be separated into three classes: serial, non-distributed parallel and distributed parallel. Only the last of these three requires a cluster of machines. Descriptions of the architecture and topology used for the two former algorithms are trivial, but presented for completeness. As stated previously, use of the term parallel does not imply distributed unless this has been specifically stated.

5.1.1 SERIAL AND PARALLEL EXECUTION ENVIRONMENTS

The environments for running the serial and non-distributed parallel algorithm are the same; the only difference is in their usage. The two execution platforms of 233MHz and 800MHz are employed. In general, it will not be stated which one was used for a particular experiment unless the speed of execution or memory usage is being evaluated as part of the given experiments comparison metric. The hosts of speed 233MHz are, unless otherwise stated within the experiment, running only command-line Linux with relatively little extra load upon the system. The 800MHz machine however is running Windows XP and is often under extra load. This discrepancy in system load has been noted and is discussed within experiments where it is relevant.

5.1.2 DISTRIBUTED EXECUTION ENVIRONMENT

The execution environment for the distributed parallel algorithm is more complex and flexible than that of the serial and non-distributed algorithms. Of course, to provide an effective distributed environment, multiple hosts are required, four of

which were available for experiments in this domain. The enabling technologies to provide the cluster were *Sun Grid Engine* (SGE), *Java Remote Method Invocation* (RMI) and the standard *Network File System* (NFS). All machines are running the Linux Fedora operating system, one of which has a graphical user interface installed.

The SGE component provides the method for job scheduling and distribution. Although in general jobs can be submitted to any host and then distributed to an appropriate member of the cluster for execution, within the context of this work only submission to the master execution host was utilised. The master host monitors the load and characteristics of all hosts within the cluster and submits jobs to the execution hosts. Within the configuration used here, the master host is also an execution host. That is, jobs may be executed on the master host. Statistics of all jobs executed are stored, such as CPU and memory usage, execution time and IO. These statistics form the basis of many of the performance metrics collected for comparison purposes. Figure 5.1.2.1 shows the four hosts within the cluster. Each host has been split into three conceptual components; the grid engine component is shown at the top of each host and the flow of jobs from the master host to the execution hosts is marked by the solid line.

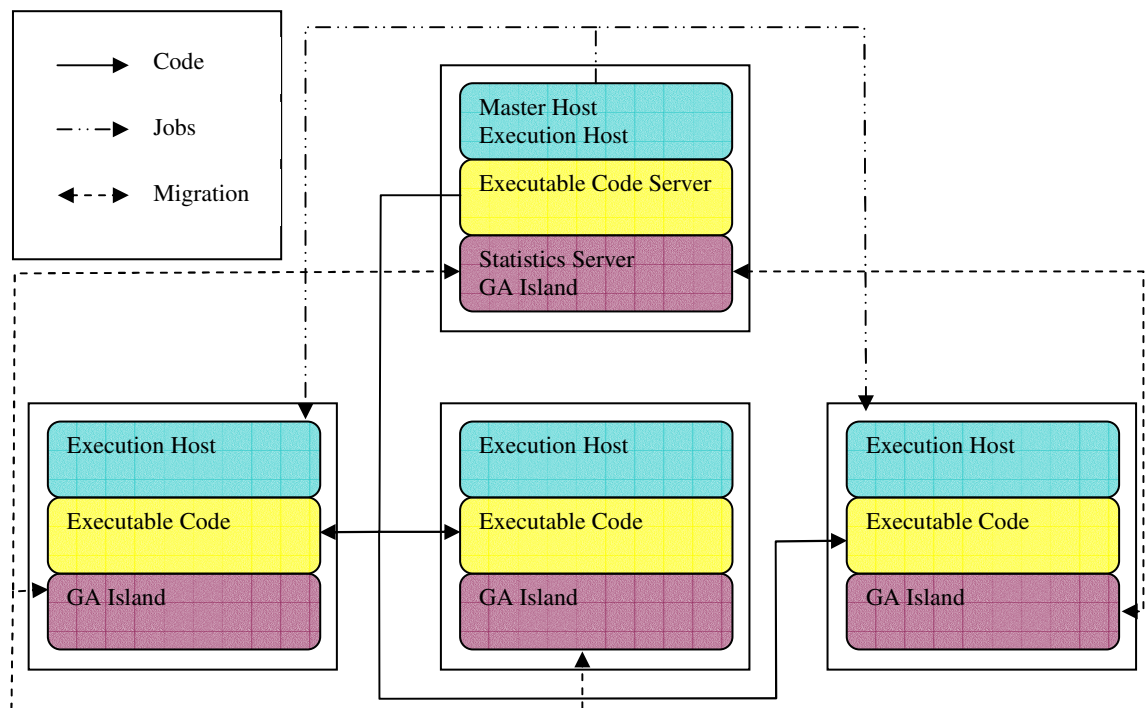


Figure 5.1.2.1: The topology of the distributed environment

The grid engine represents the method for effective job distribution, monitoring and statistical analysis of performance. However, it does not provide a mechanism for inter-host communications in a distributed parallel process. The concurrently executing java virtual machines which represent the islands within the distributed parallel genetic algorithm are such a parallel process. To facilitate their intercommunication the Java RMI technology was employed to allow indirect communication between the islands. The configuration for this is similar to that for the grid engine topology, in that one host is designated as the master and facilitates communication between all of the islands. As with the grid engine topology, the master host may also contain an island of its own. This master host also collects statistics from each of the islands about the quality of solutions found thus far and generates logs of the overall performance of the islands and the cluster in general. It is these logs from which the quality metrics used in comparison are derived. The lowest layer of the hosts shown in figure 5.1.2.1 represents the flow of migrants. Those migrants leaving the islands are sent to the master host where they will be redistributed to another island.

Finally, the NFS facility provided with the Linux distribution was employed to remove the need for duplicate code, executables and logs. That is, the code and executables were stored within a master host and this was made available to the other hosts within the cluster via an NFS share. The central segment of the hosts depicted in figure 5.1.2.1 represents the distribution of executable code from the master host.

5.2. ALGORITHM TESTING

Standard unit and system testing was carried out during development to ensure the algorithm functions correctly. To test the final algorithm, several small versions of the capacitated vehicle routing problem were devised with relatively simple solutions. The motivation for testing on small problems was to reduce the search space so as to allow a brute force enumeration of the possible solutions. It also allows visual inspection of the problem to verify solutions and manual computation of route distances to ensure correctness. One such test instance is presented here, in several modified states to demonstrate this final testing measure. All three variants are present in their entirety in appendices A0, A1 and A2.

The instance in question contains five nodes – one depot and 4 destinations. The vehicle capacity is 33 units. Due to its brevity, the full co-ordinate and demand specification is presented in table 5.2.1 and displayed graphically in figure 5.2.2.

Table 5.2.1: The test CVRP instance; node 1 is the depot. Fully presented in appendix A0

Node	X co-ordinate	Y co-ordinate	Demand
1	30	40	0
2	37	52	19
3	49	49	20
4	52	64	6
5	20	30	10

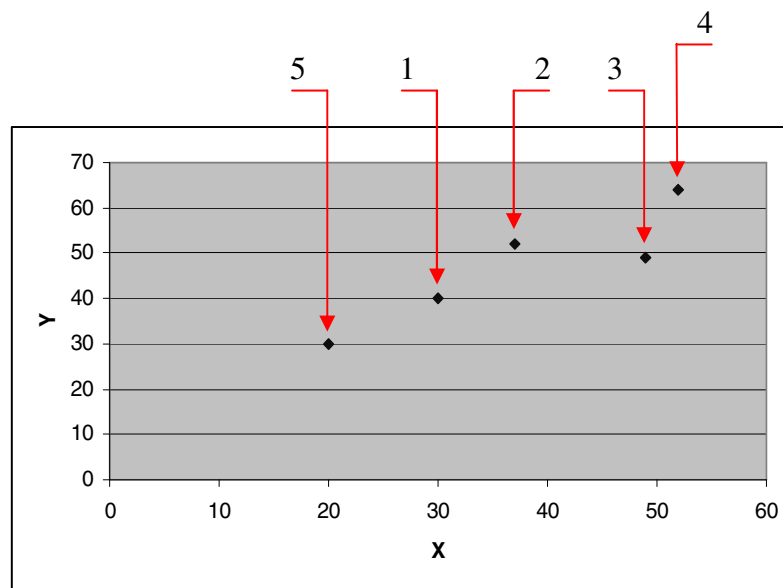


Figure 5.2.2: The test CVRP represented graphically. Node 1 is the depot.

The problem was solved first using a brute force enumeration of all the possible permutations of routes. There exist multiple solutions which minimise the total distance travelled. One such solution contains three routes, the first being to node 2, the second to node 4 and the last to nodes 3 and 4 in that order. The total distance travelled in such a solution is 124 units. Manual computation of edge weights confirms this value and, in conjunction with other similar tests¹¹, confirms the correct operation of the distance dependent computation of solution fitness. Manual inspection, although tedious, also proves this to be a minimum combination of edge weights within the graph.

Knowing the optimal value for this problem instance, the genetic algorithm was executed and found the optimal solution on average in less than 3 generations. It is promising at this stage to note the effective decrease in computational time required

¹¹ Which are omitted here as they are essentially repetition of the same approach with different instances designed to capture boundary conditions etc.

by the genetic algorithm over the brute force enumeration. However, this work is in no way aimed at comparing the performance of genetic algorithms to brute force search, or other search techniques.

A modified version of the above instance is now presented as an additional examination of correct operation. The vehicle capacity is changed from its original 33 units to a value of 45 units. The optimal solution found by the brute force enumeration contains only two routes now; one containing destinations 2, 3 and 4 and one containing only destination 5. The total distance cost is now 96. Visual inspection and manual computation once again shows this to be the correct trip cost and the smallest possible combination of edge weights within the problem constraints. The genetic algorithm finds the same solution on average within the first few generations.

One final modified version is constructed by changing the demands section of the problem, the vehicle capacity remains at its original 33 units. The new node demands are presented in table 5.2.3 and intuitively will require the dispatch of a single vehicle to each of the nodes. Indeed this is the structure of the solution presented by a brute force enumeration; one route for every node in the set of 2,3,4,5 resulting in a total distance of 163 units. Manual inspection once again verifies this trip cost and the genetic algorithm arrives rapidly at the same result.

Table 5.2.3: the final modified test CVRP instance, fully described in appendix A2

Node	X co-ordinate	Y co-ordinate	Demand
1	30	40	0
2	37	52	33
3	49	49	33
4	52	64	33
5	20	30	33

At this point within testing, the representation and problem specific operation of the algorithm has been extensively tested and shown to function correctly. However, the operation of the genetic algorithm – in particular the crossover procedures – has not been adequately reviewed and will therefore be examined in closer detail.

Within this evaluation, the correct function of the genetic operators is considered based on their specification by Goldberg (1989a) and as described in section 3.3.3. Their effectiveness in the problem domain is not considered. That is, this section will

validate the function of these operators rather than verifying that they produce useful results. Exploration of the usefulness of results obtained will be given attention during the examination of experimental performance in later chapters.

This section reviews the functionality of the partially matched crossover, the variable length crossover and the mutation operation applied to both the variable length and fixed length chromosomes. The replication operation is sufficiently trivial that its testing has been omitted here. This section also gives an opportunity for a more in-depth view of the function of the selected operators.

Standard white box testing of the partially matched crossover is presented first by demonstrating the boundary conditions and several ‘internal’ instances are handled correctly. Intuitively, there are two trivial boundary conditions; the case where the crossover section is empty or the case where it contains the full chromosome. The former is presented in table 5.2.4; however the output of both would be identical.

Table 5.2.4: Results of the PMX operator. The crossover points were both 3, forming an empty crossover section.

Parents	Children
1(0) 2(1) 3(2) 4(3) 5(4)	1(0) 2(1) 3(2) 4(3) 5(4)
5(0) 4(1) 3(2) 2(3) 1(4)	5(0) 4(1) 3(2) 2(3) 1(4)

Beyond this, the selection of a crossover section will contain an internal section of the chromosomes. One such example is presented in table 5.2.5. Here the crossover section is defined to be the genes in positions 0 to 3 inclusive. The ‘section crossover’ column shows the children after exchanging the two crossover sections. Notice that there are duplicate genes within the child chromosomes at this point. Following this, the duplicates are removed by selecting appropriate members within the other child chromosome to swap with. In this case, the 5 and the 1 are swapped in position 4, but the position need not be uniform.

Table 5.2.5: The PMX operator with boundaries before locus 0 and before locus 4. The crossover section is underlined.

Parents	Section Crossover	Final Children
1(0) 2(1) 3(2) <u>4(3)</u> 5(4)	<u>5(0)</u> 4(1) 3(2) <u>2(3)</u> 5(4)	1(0) 4(1) 3(2) 2(3) <u>5(4)</u>
<u>5(0)</u> 4(1) 3(2) <u>2(3)</u> 1(4)	<u>1(0)</u> 2(1) 3(2) <u>4(3)</u> 1(4)	5(0) 2(1) 3(2) 4(3) <u>1(4)</u>

Several extra tests were run of this operator and it was concluded to be functioning correctly.

Following this, the variable length crossover is tested. The trivial case of crossing two chromosomes with a length of 0 is omitted here. The first case of interest is in crossing a zero-length chromosome with a non-zero length chromosome. The possible outcomes are not changing either chromosome, which is trivial and is not shown here, or splitting the non-zero length gene into parts divided between the children. An example of the later is given in table 5.2.6 where the first three chromosomes are copied from the top parent to form the second child; the final two remain to form the first child.

Table 5.2.6: The variable length crossover operator with an empty parent.

Parents	Children
1(0) 2(1) 3(2) 4(3) 5(4)	4(0) 5(1)
<empty>	3(0) 2(1) 1(2)

The next case is where both chromosomes contain only one gene each. There is a chance of the children being identical to the parents, or one child containing both genes and the other only one. An example is not presented due to simplicity, however this functionality was verified. The majority of crossovers that occur with the variable length crossover operation however are with irregular chromosomes of length greater than one. In this case, the crossover is performed by first aligning the chromosomes and then selecting a cross point. Table 5.2.7 shows an example observed during testing this operation. The parents are aligned and the genes for crossover have been underlined. The 1 and 4 genes are exchanged between the two chromosomes and they are re-ordered to produce the children.

Table 5.2.7: the variable length crossover operator with irregular chromosomes. The crossover section is underlined

Parents	Children
<u>1</u> (0) 2(1) 3(2)	2(0) 3(1) 4(2)
5(0) <u>4</u> (1) 3(2) 2(3)	1(0) 2(1) 3(2) 5(3)

The final operators considered here are the mutation operators for the variable length chromosome and the fixed length chromosome. Both were tested for compliance with the specifications. The results of several test cases are presented here to demonstrate function and confirm the correct performance of these operations. As changing the value of a gene randomly would lead to an invalid chromosome, the fixed length mutation simply swaps two random genes within the chromosome. This is trivial hence an example test is omitted. The mutation operation for the variable

length chromosomes is slightly more complex, allowing an increase and decrease in chromosomes length in addition to swapping genes as the fixed length mutation does. In addition, because the variable length chromosome can potential be missing symbols from the full chromosome alphabet, a random mutation of a single gene into one of these missing values is also possible. Several example cases from the testing are presented in table 5.2.8.

Table 5.2.8: The variable length mutation operator. These examples demonstrate the random mutation, mutation by increasing chromosome length and mutation by decreasing chromosome length respectively.

Before Mutation	After Mutation
1(0) 2(1) 3(2) 5(3) 8(4)	1(0) 2(1) 3(2) 7(3) 8(4)
1(0) 2(1) 3(2) 5(3) 8(4)	1(0) 2(1) 3(2) 5(3)
1(0) 2(1) 3(2) 5(3) 8(4)	1(0) 2(1) 3(2) 5(3) 6(4) 8(5)

This section has presented a verification of the correct function of the algorithm developed by deploying it upon a small instance of the capacited vehicle routing problem. Brute force solutions for this small instance can be found and checked manually for consensus with the developed algorithm. Furthermore, some sample cases from the unit testing of the genetic operators deployed within this work were presented. This demonstrated their correct functionality and concurrently gave a more detailed description of their operation.

6. DETAILED EXPERIMENT DESIGN AND EXPECTATIONS

6.1. SERIAL – SCALING COMPUTATIONAL POWER

This experiment was designed to highlight the effects of scaling the computational power of the host machine that the serial genetic algorithm was being run upon. Two machine speeds are available; 800 MHz and 233MHz. The other characteristics of the machines (RAM, etc.) are essentially the same. The serial algorithm will be executed on both these machines to 100 generations and the selected metrics measured for each machine over several repeats of the experiment.

The results of this experiment are expected to demonstrate, of course, a decrease in the time required to complete 100 iterations of the genetic algorithm. Based on the difference in CPU speed, the expected increase is approximately 3–4 times. Beyond this, no improvement in quality of solution, convergence or memory usage is expected.

6.2. NAÏVE PARALLEL AND SERIAL COMPARISON

The *naïve parallel* version of the algorithm is simply several concurrently executing serial algorithms, from which the total best solution is selected upon completion of all algorithms. Both this and the serial algorithm will be executed on a single platform of the same characteristics and a comparison between the two will be made based once again upon the selected metrics. The number of concurrently executing serial algorithms within the naïve parallel algorithm will be increased and the level of improvement gained in solution quality from these increased ‘cluster’ sizes will be measured.

The results of this experiment are expected to demonstrate that the naïve parallel algorithm in general outperforms the serial algorithm based upon the measure of solution quality. Beyond this simple expectation, the constituent serial algorithms that are aggregated by the naïve parallel algorithm are expected to perform without major difference from the serial algorithm.

At this point, the previous two experiments are essentially provided to establish completeness and several desired properties of the algorithms. The expected performance to this point is indeed rather intuitive; however the results are included to confirm that these simple properties hold.

6.3. NON-DISTRIBUTED PARALLEL AND SERIAL COMPARISON

This experiment compares the performance of the non-distributed parallel and serial algorithms. As expounded earlier, the non-distributed parallel algorithm is executed entirely upon a single host, but incorporates several concurrently executing serial algorithms. The difference between this and the naïve parallel algorithm is that migration is allowed between the separate serial algorithms which are aggregated by the parallel algorithms. These constituent parts are often referred to as *islands* or *demes*; both terms were used in previous sections and will continue to be used essentially interchangeably. This experiment is also aimed at identifying the relationship between the population size within the distributed and serial algorithms. The serial algorithm will be executed with a 300 member population and compared with the results of executing the parallel algorithm with a 100 member population and 3 islands (effectively 300 members). The parallel algorithm will also be run with a single island of 300 members.

The result of this experiment is expected to demonstrate an improvement in the quality of the final solution found by the non-distributed parallel algorithm over those found by the serial algorithm. Convergence is expected to improve within the parallel algorithm as genetic diversity is favoured by the migration scheme (as was explored in section 3.4.7). It is expected that the 300 member parallel algorithm (with only a single island) will produce relatively similar results to the serial algorithm with 300 members. The parallel algorithm with three islands is expected to demonstrate improved performance due to an increase in genetic diversity as a result of the population segregation.

Trends in CPU and memory usage are expected to perform intuitively. Measurements are expected to be slightly higher for the parallel algorithms over the serial algorithm due to an increase in control structures and inter-island communications. Of course, network usage is an irrelevant metric within this experiment due to the utilization of only a single host.

6.4. DISTRIBUTED PARALLEL AND SERIAL COMPARISON

At this point within the experimentation, the distributed parallel algorithm is introduced and its performance was gauged against the serial algorithm. The parallel distributed algorithm was run with three islands of 100 members each and compared to the performance of the serial algorithm with a 100 member population. Of course

this is the first experiment where the metric of network usage is of concern. However as the serial algorithm does not generate any network traffic, there is no comparison to make. Hence it will not be explored as more than a benchmark for the network performance of the parallel distributed algorithm.

Within the comparison of the distributed algorithm with three islands and the serial algorithm, the results should demonstrate improved performance. These will be characterised by higher quality solutions after the same number of generations as the serial algorithm. Of course, there will be a corresponding increase in performance metrics for the parallel distributed algorithm due to the increased computation and the communications overhead. Convergence characteristics of the distributed parallel algorithm are also expected to offer an improvement over the serial algorithm. This is due to an increase in genetic diversity as a result of the population segregation.

6.5. INCREASING CLUSTER SIZE

The cluster size within the parallel distributed algorithm will be increased to measure the effect this has on both the quality metric and performance metrics. The variation in cluster size will be from one host to a maximum of four hosts.

The effects of the increase are expected to produce results of uniform improvement within the quality metric. Similar increases within the performance metrics are expected to reflect the increased computation and resource usage. Convergence is also expected to lessen as the cluster size increases, owing to a higher genetic diversity over the total increased population size.

6.6. EFFECT OF NAÏVE HEURISTIC

The re-ordering heuristic described in section 3.5.4 is introduced here in an attempt to improve solution quality. Briefly, this heuristic is based on the concept that optimal routes will begin with the destination closest to the depot and end with that which is farthest away. The heuristic applied here is referred to as the naïve heuristic because it is applied without any consideration of its benefit in fitness and to all solutions within each generation.

The results are expected to show that applying this heuristic increases the execution time and memory usage, but provides an improvement in the quality of the solutions found over the 100 generation trial period. Test will be conducted upon both the serial and distributed parallel algorithm but results are expected to be essentially

identical for both algorithm types. Machado et al. (2002) explored the effect of adding a K-nearest neighbour (KNN) heuristic to a genetic algorithm approach to solving the vehicle routing problem. Their results were characterised by an increase in average fitness and best individual found, as summarised in figure 6.6.1.

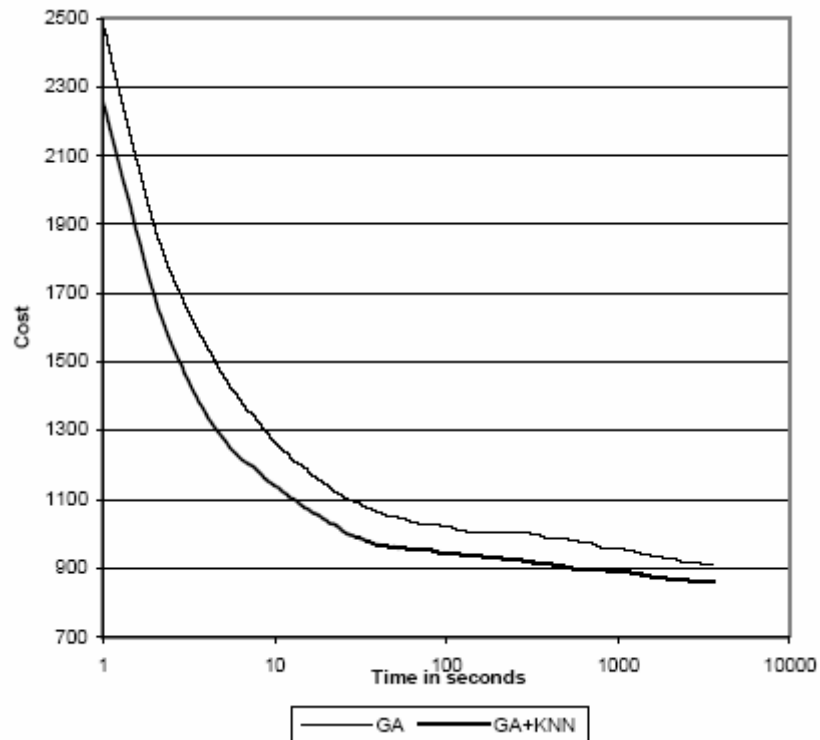


Figure 6.6.1: The results of experiments performed by Machado et al. (2002) into the addition of a KNN heuristic to their GA solution to a VRP. Cost is analogous to the measure of fitness used within this work.

6.7. PROBLEM DECOMPOSITION PARALLELISATION

The problem decomposition approach to parallelising the genetic algorithm is significantly different from the parallel algorithms used thus far. The concept of breaking the problem up into sub-problems and the constituent sub-problem parts of the capacitated vehicle routing problem are discussed in section 3.5.4. This experiment will measure the effect of two different decomposition methods. Problem decomposition is in effect a heuristic, as it requires knowledge of the problem domain to arrive at a suitable problem separation. For example, it is necessary to know that the depot node must be duplicated in every sub-problem for the decomposition to be valid.

The first problem splitting approach examined is a simple split of the problem space into four groups. This is referred to as the naïve four-way decomposition due to the

fact that it does not consider the suitability of assigning destinations to each of the sub-problems. In short, the first 25% of destinations in the problem are assigned to sub-problem one, the next 25% to sub problem two and so on. The four sub-problems are presented for reference in appendix C0, C1, C2 and C3. Each sub-problem is executed in parallel on a separate host and the results are re-unified to arrive at a final solution.

The second approach explored will be a semantic decomposition of the problem based on an understanding of its component requirements. As noted in section 3.5, the CVRP is in fact the intersection of the bin-packing problem and the travelling salesman problem. Based on this knowledge, a conceptual decomposition of the problem into these two components is possible. The bin-packing component is addressed first by finding an optimal number of vehicles and assigning destinations to vehicles such that each vehicle has as close to its maximum capacity as possible but no more. That is, minimise the wasted vehicle capacity. Following this, each vehicle route represents an instance of the travelling salesman problem and can be optimised for shortest tour length independently of other routes. The only constraint not immediately addressed here is the need to group destinations which are geographically similar into the same route. This decomposition is achieved in practise here by applying the K-means clustering algorithm to solve the bin-packing component and group close destinations into the same route. Following this, the genetic algorithm is applied to solve the constituent travelling salesman problems in parallel.

These two forms of problem decomposition also provide a significant reduction in the problem space. The naïve decomposition reduces the search space by disallowing combinations of destinations from separate sub-problems and the conceptual decomposition does the same by disallowing combinations of destinations from separate clusters. In fact, the conceptual decomposition reduces the search space even further due to the fact that it separates the original problem into more sub-problems than the naïve approach.

Assuming the validity of the heuristics used for splitting the problem, the solution quality is expected to be improved over the other variants of the algorithm due to the greatly reduced search space. Performance metrics are also expected to improve due once again to the reduced search space and hence lower computational requirements. Network bandwidth will be essentially zero, since the only stage of communication is for the initial distribution of sub-problems and the final re-unification of solutions.

7. RESULTS AND OBSERVATIONS

7.1. SERIAL – SCALING COMPUTATIONAL POWER

This experiment was aimed at measuring the performance changes in the serial algorithm by increasing the computational power available to it. The method and expected results of this experiment were presented in section 6.1.

The results demonstrated no apparent change in quality of solutions found. This can be seen from figure 7.1.1, which shows the average fitness of solutions over the sample runs for each machine speed. The average best of population and best ever results are similarly unaffected by processor speed. Execution speed however is improved, as was expected, but by a lesser amount than originally anticipated. As can be seen from figure 7.1.2, there is an effective 2.5 fold decrease in execution time, although the 800MHz processor is almost three and a half times faster. This discrepancy can be explained by the difference in operating environment. The 800MHz machine is running windows XP with extra application load, where as the 233MHz machine is running Linux in command line mode only, with relatively little extra application load. In support of this, the 800MHz machine was measured with a baseline load of approximately 15% during the execution of the experiments, where as the 233MHz machines had a baseline load of approximately 1%. Applying these offset, the effective operating speed of the 800MHz machine for the experiments is only 680MHz. This is still more than 2.5 times the effective operating speed of the 233MHz machines however. The remaining discrepancy can be put down to differences in final executable code for the different platforms and IO bottlenecks.

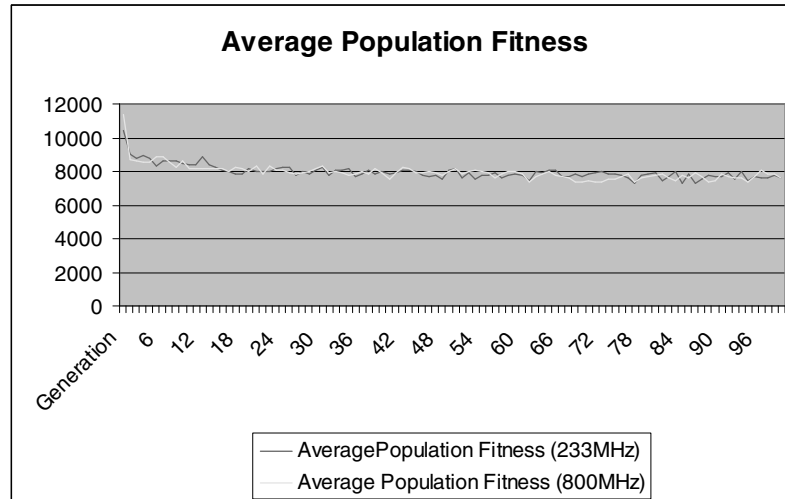


Figure 7.1.1: The average population fitness over the 100 generation test period for the serial algorithm running on both the 800MHz platform and the 200MHz platform.

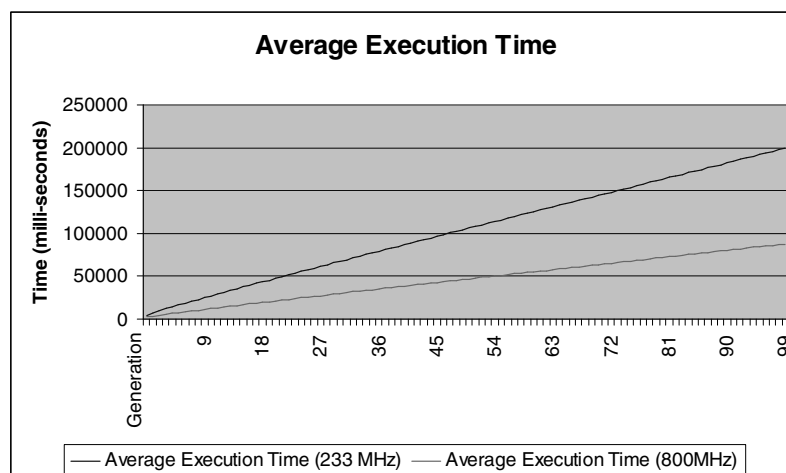


Figure 7.1.2: The Average execution time of the serial algorithm running on the 233MHz and 800MHz platforms over 100 generations.

The important result of this experiment, rudimentary though it is, was to demonstrate that external factors influence the results obtained within these experiments. Apparently small differences can lead to significant discrepancies in the results obtained. Related to this is the notion that sound theoretical principles often initially appear not to be backed up by empirical evidence due to uncontrollable variations in the environment.

7.2. NAÏVE PARALLEL AND SERIAL COMPARISON

Briefly, this experiment was intended to highlight the naïve approach to parallelising the algorithm. The method and expectations for this experiment were

described in section 6.2. The serial and naïve parallel algorithm (with an increasing number of parallel executions of the serial algorithm) were run multiple times and the average outcomes were collected. The results produced are consistent with expectations in that this naïve parallel algorithm has, on average, better performance than the serial algorithm. This can be seen from the graph in figure 7.2.1, where the average results from the experiment are presented.

Indeed, the importance of this result is that a simple approach to parallel genetic algorithms can yield benefits which may not originally have been expected. As will become evident later this simple approach does indeed produce results which are quite comparable to the more advanced algorithms.

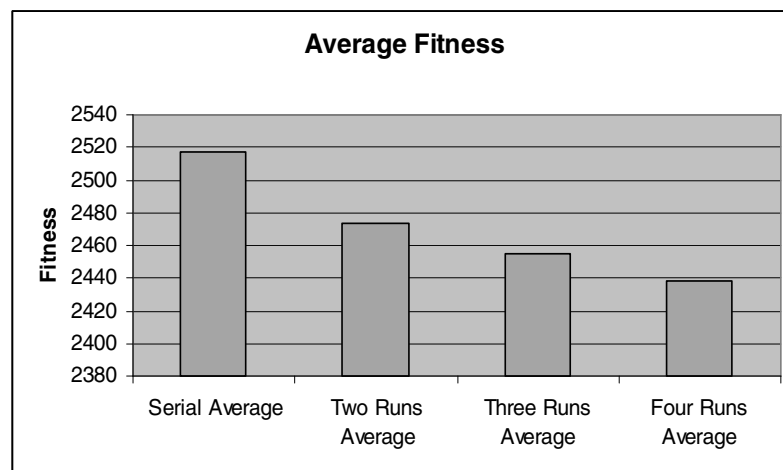


Figure 7.2.1: The average fitness of the best solutions found within the naïve parallel algorithm with increasing parallel executions.

7.3. NON-DISTRIBUTED PARALLEL AND SERIAL COMPARISON

This experiment, as described in section 6.3, was aimed at comparing the performance of the non-distributed parallel algorithm with the serial algorithm. The first set of results compares the average fitness of the best solutions found in two parallel algorithms and one serial. That is with 3 separate populations (islands) of one hundred members each¹², one single island with three hundred members¹³ and the serial algorithm with three hundred members.¹⁴ The results, as presented in figure 7.3.1, demonstrate that both of the non-distributed parallel algorithms generated similar performance, whereas the serial algorithm performs slightly worse.

¹² Marked “3x Islands” on the graphs

¹³ Marked “3x Population Parallel” on the graphs

¹⁴ Marked “3x Population Serial” on the graphs

This is somewhat surprising on one count; the parallel algorithm with a single 300 member population is conceptually identical to the serial algorithm with the same population size, however their performance is different. This will be discussed further below. The remainder of the conclusions that can be drawn from this data are in keeping with the expected results however. The two parallel algorithms produce essentially the same result. The multiple islands variant was slightly more successful. However, such a small difference could be due to the stochastic nature of the algorithm or uncontrollable changes within the operating environment.

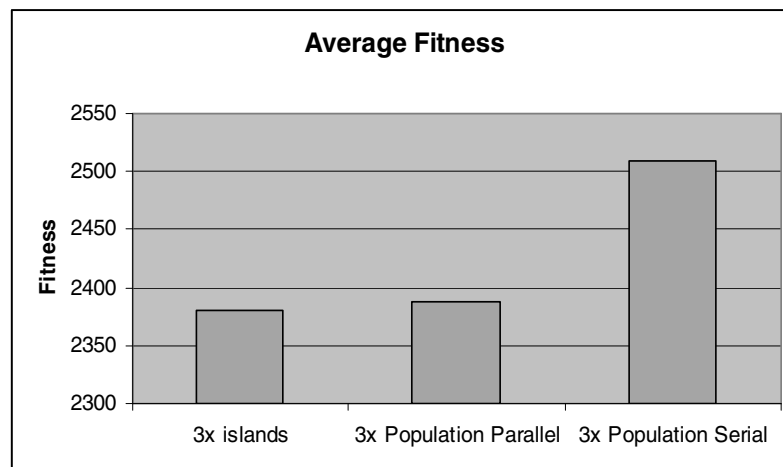


Figure 7.3.1: The average fitness of the best solutions found by the increased population serial, increased population non-distributed parallel and multiple island non-distributed parallel algorithms

As outlined in section 6.3, it was predicted that all three algorithms would produce similar results. The only discrepancy then is the difference in average result between the serial and parallel algorithms. This characteristic can likely be attributed to the migration scheme within the parallel algorithms during this experiment, which performed migration of based on fitness. Even within the parallel algorithm with only a single island, there is a migration scheme — individuals are simply migrated back to the single island. In affect, this is increasing the elitism of the algorithm, as members for migration within this algorithm are selected based on their fitness.¹⁵ There is, perhaps one counterpoint to this suggestion, and that is to note that a small increase in elitism is unlikely to produce much change in the solutions. In fact, the improvement shown in figure 7.3.1 is somewhat deceptive; there is only a one percent improvement in solution quality generated by the parallel algorithms over the serial one.

¹⁵ That is, the best individual is selected for migration during each generation which, within the single island parallel model, allows it automatic inclusion in the next generation.

As all the algorithms were executing on a single machine with the same total number of individuals, it was expected that the execution times would be similar; however this was not the case. Figure 7.3.2 presents the average execution times for each of the three types of algorithm. As can be seen, the non-distributed parallel algorithm with three islands has a significantly longer average execution time.

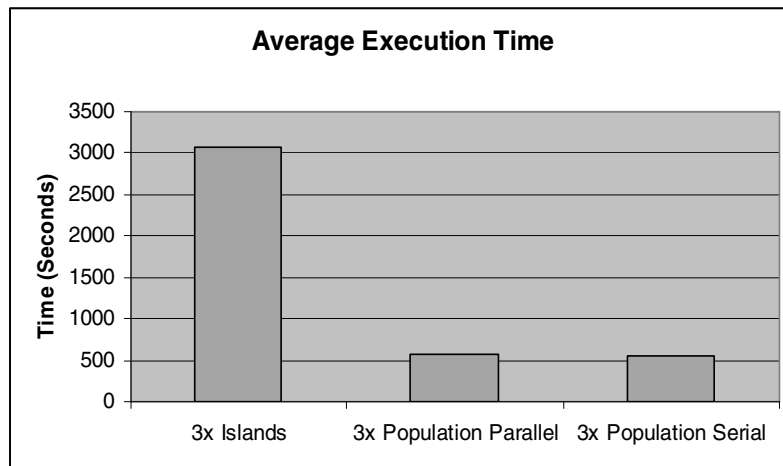


Figure 7.3.2: The average execution time of the three algorithms, demonstrating a sharp increase in the split population model (3x Islands).

The memory usage of the three algorithms was also expected to be relatively similar, however experimental results showed quite a large variation between the algorithms with a single population and that with three separate populations. Figure 7.3.3 shows this difference.

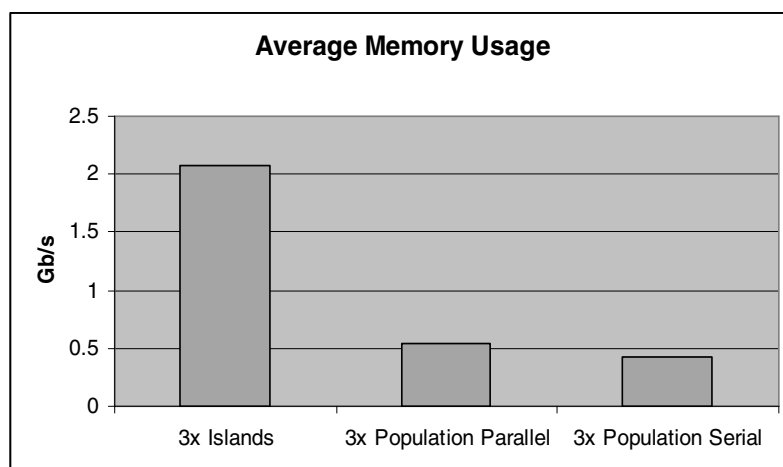


Figure 7.3.3: The average integral memory access, measured in Gb/second of the three algorithms, demonstrating a sharp increase in the split population model (3x Islands).

Comparing figure 7.3.2 with figure 7.3.3, it is apparent that the algorithm model with separate populations requires much greater memory access and longer execution times. Investigation of the structure of the code lends some insight into the memory usage trends. The parallel algorithm, even with only one island possesses several extra control structures for migration and communication over the serial algorithm, leading to the minor increase in memory usage between the 3x Population Parallel and the 3x Population Serial model. The dramatic increase in memory usage for the 3x Islands model is due to the three fold duplication of these control structures, although this only accounts for some of the increase. The remained can be explained by considering the dynamics of the memory access in question. The graph shows the amount of memory access per second; with multiple threads running concurrently – as is the case within the multiple population model (3x Island) – the memory access for context switch is increased.

A similar argument to that provided for the additional memory access is applicable for the increase in execution time required for the multi-population algorithm. However in this case the effect of increased iteration in outer loops also contributes. In general, the complexity of the algorithm is of the order $O(mn)$ where m is the number of islands within a virtual machine and n is the number of individuals.

Consider the situations presented above where the number of islands is 3, 1 and 1 respectively and the number of individuals is 300 for all experiments. This gives an expected running time for the multi-population algorithm of approximately 3 times that of the single population algorithms. The remaining discrepancies in running time between the two single population models can be explained, as was done above, by the increased control operations required in the parallel version. Similarly, the extra increase in computational time for the multi-population model is attributed to the overhead of context switch (as it is multi-threaded) and increased IO wait time.

The relevance of these results is in demonstrating the increased computational time and memory usage of the split population model over the naïve parallel and serial algorithms. In addition, the benefit of the parallel algorithms is shown in terms of solution quality. However, the results of the single island parallel algorithm suggest that the improvement is more related to an increase in elitism due to the migration scheme being used. This could be further confirmed by measuring the change in result provided by modifying the elitism within the serial algorithm.

7.4. DISTRIBUTED PARALLEL AND SERIAL COMPARISON

As described in section 6.4, it was expected that the distributed parallel algorithm would outperform the serial algorithm by improving convergence characteristics and hence allowing a more thorough exploration of the search space. Although an increase in execution time was expected as a result of the inter-island communication, an increase as profound as that which was observed was not anticipated. As can be seen from the graph in figure 7.4.1, there is an approximate two-fold increase in cumulative execution time. Note that the parallel algorithm running on each of the islands only differs from the serial one by the inclusion of a communication step. Within this communication the island updates statistics on a server and performs migration. It must be the case that this communication stage is taking approximately as long to execute as the remainder of the algorithm. This follows from the results of the previous experiment, which suggested that more than one island attempting to communicate with the server caused an increase in required execution time due to IO blocking.

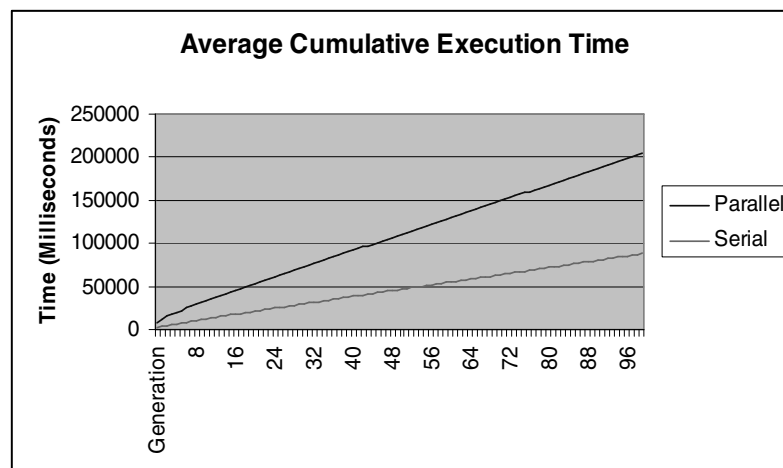


Figure 7.4.1: The average cumulative execution time for the serial and distributed parallel algorithm with three islands. The execution time for the parallel algorithm is an average of that measured for each island, rather than a total for all hosts.

The performance of the distributed algorithm was marginally better than the serial algorithm when basing the comparison on the average fitness of the best solution found after 100 generations. This can be seen in figure 7.4.2, where the two values are compared. However, the improvement is relatively minor and does not reflect the fact that the parallel algorithm has three times the computational power at its disposal and three times the population.¹⁶

¹⁶ This is true of the particular case in question, where there are three islands within the parallel algorithm with 100 members per island, rather than in general.

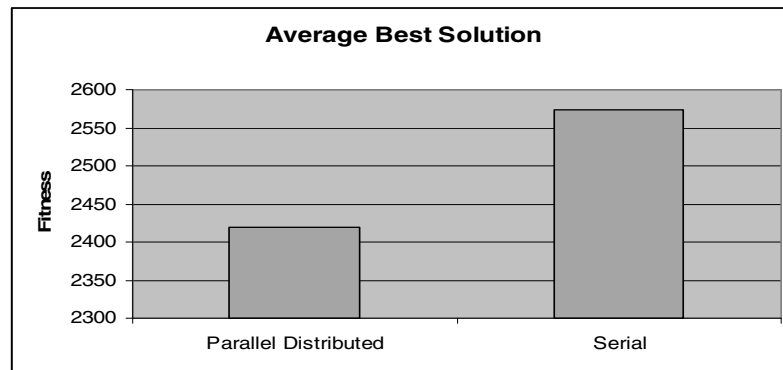


Figure 7.4.2: The average best solutions for the distributed parallel and serial algorithms with 100 member populations after 100 generations.

The parallel distributed algorithm was also expected to show improved convergence characteristics over the serial algorithm. Convergence can be estimated by comparing the fitness of the best of generation and the average fitness of all individuals in the generations. When these values are close, there are many other individuals within the generation that have a similar fitness to that of the best of generation and hence are likely to have similar genetic makeup. The results of this comparison are presented in figure 7.4.3, which seems to indicate that genetic diversity is being maintained, due to the constant separation of average and best fitness. These results are in conflict with the observed performance however, as all variants of the algorithm demonstrate an initial good rate of exploration but soon degenerate. Improvements thereafter are slow and relatively random. Further investigation of the convergence of the algorithm was clearly called for and is presented in chapter 8.

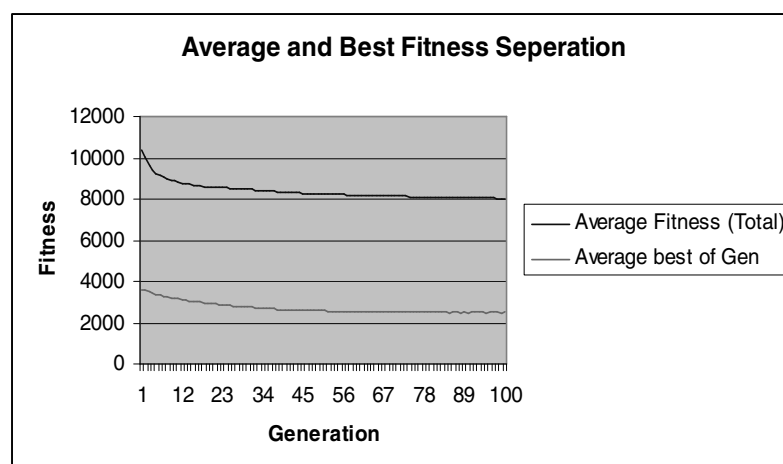


Figure 7.4.3: The separation of average fitness and the fitness of the best individual per generation within the distributed parallel algorithm with 3 islands each of 100 members, measured over 100 generations.

The results of these experiments demonstrate that the distributed parallel algorithm demonstrates improved solution quality over the serial algorithm. In fact, comparing these results with those obtained in section 7.3 for the non-distributed version of the parallel algorithm shows a slightly worse level of solution. Note however that the standard deviation within the final fitness measures of the parallel algorithms is approximately 35 units. The results presented for the distributed parallel algorithm are, in general, within one standard deviation. Hence, the discrepancy could be attributed to the stochastic nature of the algorithms. The increase in execution time is also comparable to that observed with the multi-population model of the non-distributed parallel algorithm. That is, the parallel distributed and non-distributed algorithms exhibit results which are reasonably similar. Finally, the first measure of convergence and genetic diversity made here gives the initial conclusion that diversity is being maintained within the parallel distributed algorithm. However, this result will be further explored in chapter 8.

7.5. INCREASING CLUSTER SIZE

7.5.1 RESULTS AND OBSERVED TRENDS

This experiment was aimed at measuring the effects of increasing the cluster size used to perform the parallel distributed algorithm on. The method and expected results are given in section 6.5.

The first measure made was that of required CPU time, the results of which are presented in figure 7.5.11. Generally the required CPU time per island was uniform, as expected; the times presented for the one, two and three host clusters are all within one standard deviation of each other. The only exception was that of the four host cluster. Inconsistencies were observed in the measures for all of the tests involving the four host cluster and these will be collectively explained shortly.

Wallclock time was also measured and behaves as expected. Increasing the number of hosts increases the amount of time each host spends to complete the execution of the algorithm, but does not increase the actual time spent computationally (the CPU time). Combining this with the results obtained for CPU time, it is clear that the increase in wallclock time associated with increasing the cluster size is not due to extra computation. The amount of CPU time required remains relatively constant and hence the increase is due to blocking on IO. This is either owing to an increased load on the server, the network or a combination of the two. The results obtained showing network usage, displayed in figure 7.5.1.6, corroborates this; as the cluster size increases the usage of the network also increases. Due to the fact that the

experiments were carried out on a shared, multiple access medium, the collision rate also increased. This is likely to have lead to delays in response due to packet retransmission. This explains the increase in wallclock time, but the relative stability of computational time. Tanenbaum (1996) provides an excellent description of the collision characteristics of Ethernet — the network technology that was used within these experiments.

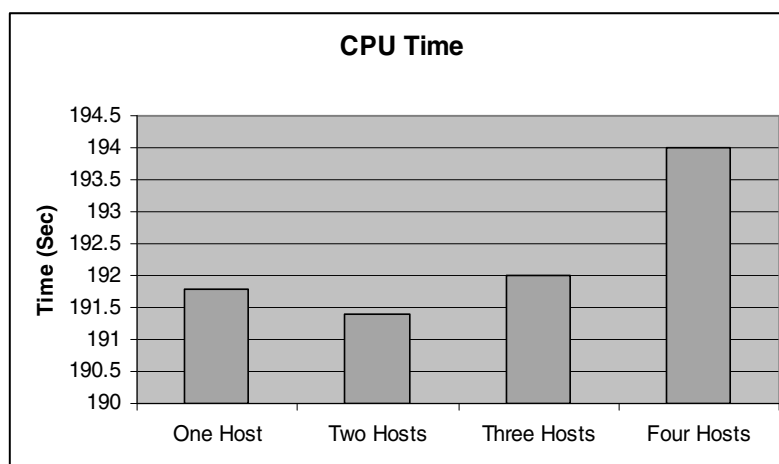


Figure 7.5.1.1: The execution time spent on the distributed parallel algorithm with varying cluster sizes for a 100 member population per host. Measured over 100 generations.

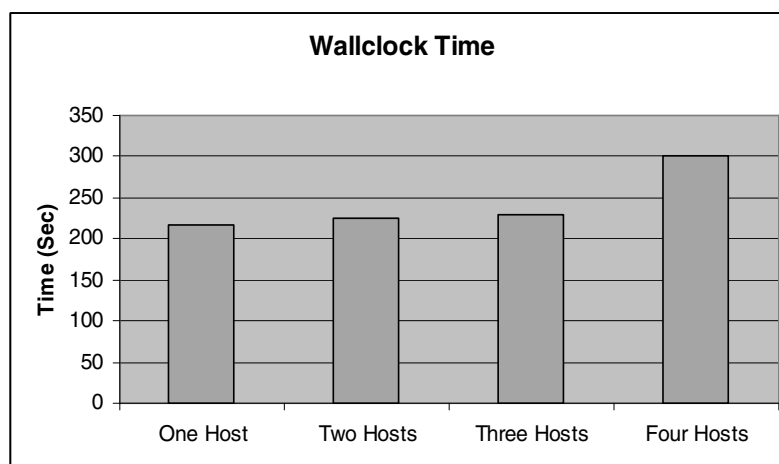


Figure 7.5.1.2: The total running time (wallclock) time for the distributed parallel algorithm with varying cluster sizes. Measured in seconds, over 100 generations with a population size of 100 members per host.

The quality of the best solutions found by each of the cluster sizes was also measured and demonstrated that the major improvement was had by increasing the cluster size from one to two hosts. Adding a third host provided only minor improvement and adding a fourth host actually resulted in slightly worse performance. However, the

outcomes of the two, three and four host clusters are quite similar, and the variation is quite possibly due to the stochastic nature of the genetic algorithm. The results are presented graphically in figure 7.5.1.3. The lack of improvement in solution quality for the larger clusters is inconsistent with the expected results and implies a lack of effective search of the problem space. The likely cause of this is premature convergence, addressed more thoroughly in chapter 8.

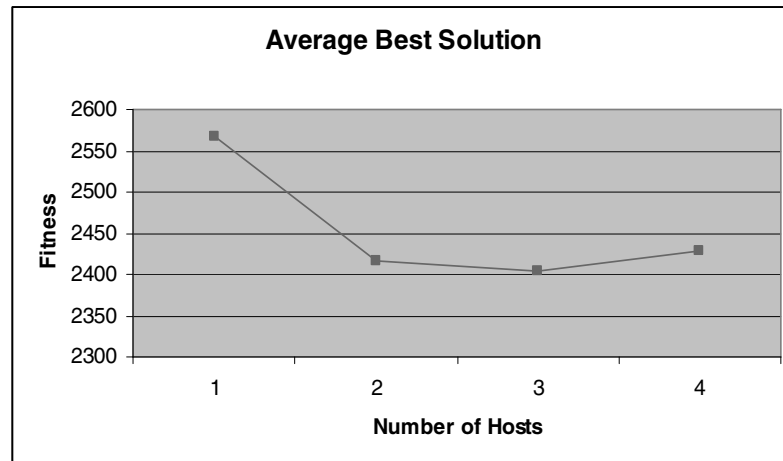


Figure 7.5.1.3: The average best solutions found by the distributed parallel algorithm with varying cluster sizes measured over 100 generations with population size of 100 members per host.

The memory access of the varying cluster sizes was also measured. Once again, the results of the four host cluster were inconsistent. This initially appears to demonstrate an improvement in memory access for the larger cluster sizes, but is actually an artefact of the increased wallclock time. The measure of memory access is the quotient of total memory accessed divided by the total execution time. Multiplying the memory access by the total time gives the total memory access, which is graphed in figure 7.5.1.5, showing that all (excepting the four host cluster, once again) cluster sizes used similar amounts of memory per host. This is in keeping with the expectations outlined in section 6.5.

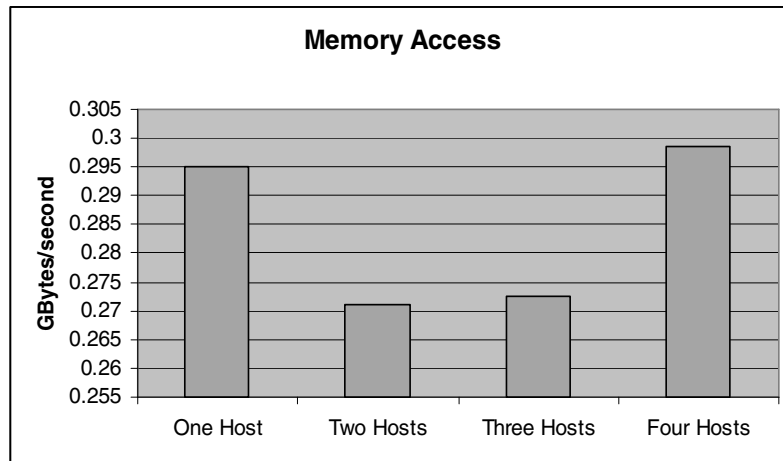


Figure 7.5.1.4: Memory access measured in gigabytes per second for the distributed parallel algorithm with varying cluster size over 100 generations with a population of 100 members per host.

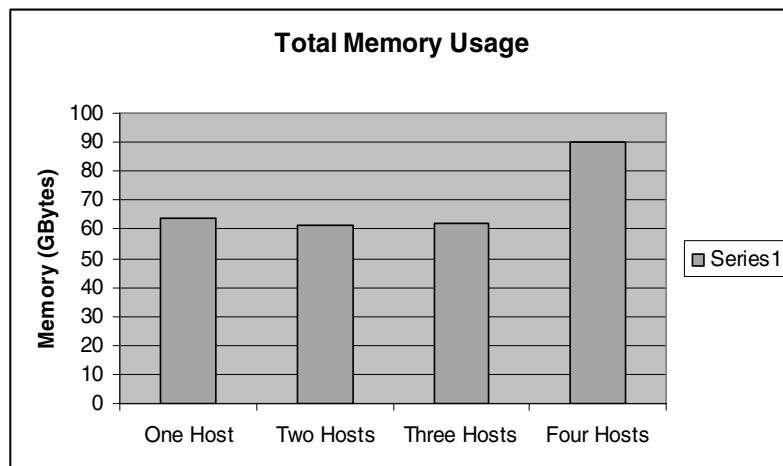


Figure 7.5.1.5: The total memory usage measured in gigabytes for the distributed parallel algorithm with varying cluster size measured over 100 generations with a population size of 100 members per host.

Network access was also measured and behaved essentially as expected, with a roughly linear increase in the amount of network traffic generated as the size of the cluster was increased. Once again, there was an exception to this trend demonstrated by the four host cluster, which is explained in the following section. The baseline network activity was measured and found to be almost negligible at approximately 1.93 kilobits per second. Recall that the network usage is not only restricted to the communication of individuals as part of the migration scheme but also includes an initial transfer of executable code for the islands as well as transmission of statistical data to the server. However, since all the cluster sizes possess these overheads, effective comparison can be made without eliminating their effects.

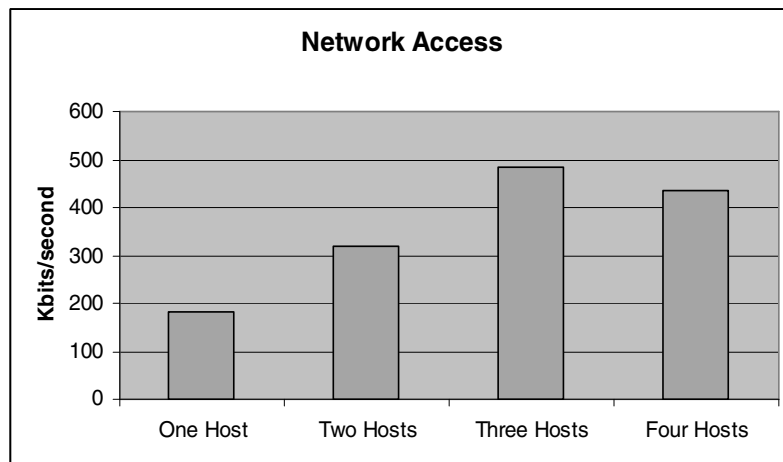


Figure 7.5.1.6: Network access measured in kilo-bits per second for the parallel distributed algorithm with increasing cluster sizes over 100 generations with a population of 100 members per host.

7.5.2 FOUR HOST CLUSTER VARIANCE

As was mentioned several times in section 7.5.1, the results obtained from the four host cluster were inconsistent with those of the smaller clusters. This is due to the topology and usage of the four machines, which was detailed in chapter 5. One of the hosts runs the KDE graphical user interface for Linux, is the master host for the grid engine, contains the migration and statistical server for the genetic algorithm and serves the executable code for the other hosts. Not surprisingly then, this machine is quite heavily loaded. For all cluster sizes of less than four machines, it had been omitted from the selection of execution hosts for the genetic algorithm by the grid engine scheduling service. However, this left only three machines suitable for hosting an island of the genetic algorithm. When the desired number of islands exceeded three (as was the case with the four host cluster), one of two possible cases occurred. The heavily loaded master host was selected to contain an island or one of the other hosts was selected to contain two islands executing concurrently on the same processor. This decision was made dynamically by the grid engine at execution time and was the cause of the unexpected results demonstrated by the cluster size of four.

In the first case, the master host was delegated the control of an island. Of course, this affected the total running (wallclock) time of the island, increasing it to between 1.5 and 2 times that of the other islands within the cluster. The amount of increase was quite variable as this was affected by the current execution environment on the server – something that was in itself quite variable. In the case where two islands were executing concurrently on a single processor, the execution time of both hosts

increased by roughly the same amount. Although the total execution (CPU) time increased in the four host cluster, it was a relatively small increase and is likely related to the overhead of context swap. Further tests would be needed to conclude this with certainty however. This then explains the results presented in figures 7.5.1.1 and 7.5.1.2, where there is a measured increase in both CPU time and wallclock time within the four host cluster.

Figure 7.5.1.5 also showed an increase in total memory access for the four host cluster. This is likely the result of the increased running time, requiring more memory swaps and manipulations due to the fact that other processes were also heavily using the memory space.

Inconsistencies were also observed in the measure of network usage; the four host cluster actually utilizing the network slightly less than the three host cluster. This is in keeping with the above explanation that an island was located on the master server or co-located with another island in every execution of the algorithm. In the former case, this reduced total network traffic by a quarter, as the island residing on the master host could access the server, executables and grid engine master without needing to access the network. This was confirmed by running a single host cluster with that host residing on the master server. The results observed showed that network access was restricted to grid engine control messages only and the observed usage of 1.54 KBits/second was significantly less than that observed for a single island residing on a separate processor. This presents another conundrum; following this logic, the network utilization for the instances where an island resided on the master host should be almost identical to that for the three host cluster. This was not the case however, with the results of such a four host cluster showing less network usage than that of the three host cluster. The explanation for this is once again due to the nature of the calculation. The measure is made by dividing the total amount of data transmitted by the total time. It has been shown that the addition of an island to the master host does not increase network usage, but it does extend the total wallclock time for the cluster, as the island executing on the master server requires a much longer time. This of course, decreases the value of network access, as the denominator is increased. Similar discrepancies appear for the case where two islands are co-located on the same host. However, further experimentation and analysis of results is required to state the cause and effects of these with any certainty.

7.5.3 SUMMARY

There are several key points to draw from these results. The first is probably the most obvious. As was demonstrated in section 7.1, theoretically sound expectations often fail to exhibit themselves in practice due to oversimplification of the operating environment when making predications about expected performance. Indeed, this is the reason for much of the data within section 7.5.1 which initially seemed to pose counterpoint to the initial observations. The next point of consideration is the effect of increasing the size of the cluster. The promising finding is that increasing the cluster size does not appear to substantially increase the execution time per host.¹⁷ Despite this, however, the results also suggest that increasing the cluster size does not significantly increase the quality of solutions found by the algorithm. Chapter 8 provides a more in depth description of why this is, but briefly there is a tendency towards each island searching the same section of the problem space as all the other islands within the algorithm. Although more wide ranging experiments would be needed for a conclusive result, these observations suggest that, with the algorithm as it currently stands, an increase in cluster size is not expected to improve performance.

7.6. EFFECT OF NAÏVE HEURISTIC

This experiment, as detailed in section 6.6, measured the effect of the naïve heuristic upon the execution of the parallel three-host cluster and the serial algorithm. The most obvious first measurement is that of quality of solution, measured over one hundred generations. This is presented in figures 7.6.1 and 7.6.2 for the serial and distributed parallel algorithms respectively by graphing the fitness of the best solution found so far at each generation. As can be seen, the simple heuristic applied here improves the quality of solutions at each stage by rearranging the gene structure of all chromosomes in the population. It does not, however, affect the rate of improvement nor would it seem to have improved convergence characteristics of the algorithm. Both the heuristic augmented version of the algorithm and the basic algorithm show little sign of continued improvement towards the end of the 100 generation trial. This separation of solution quality is similar in nature to that presented in section 6.6 as an expected result. The improvement in solution quality is different to those shown within section 6.6, but this is due to the likely premature convergence of this algorithm (which is explored more fully in chapter 8) and the fact that more trials are present in the work shown in section 6.6 than are presented here.

¹⁷ This is assuming that the network infrastructure and server supports the increased traffic.

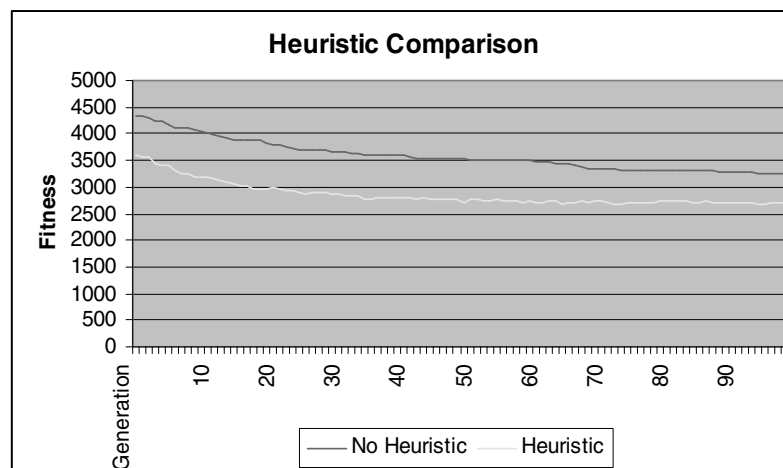


Figure 7.6.1: Comparison of the best solutions found at each generation for the serial algorithm with and without the Heuristic augmentation. The results were measured over 100 generations with a population size of 100 members.

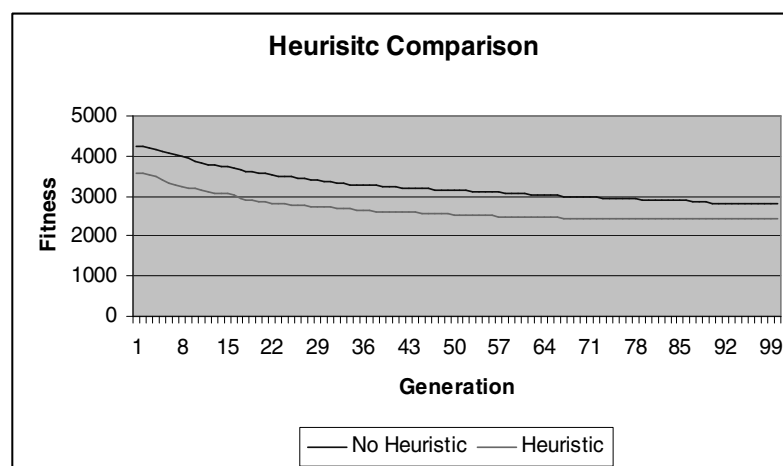


Figure 7.6.2: Comparison of the best solutions found at each generation for the distributed parallel algorithm running 3 islands with and without the Heuristic augmentation. The results were measured over 100 generations with a population size of 100 members per host.

An apparent increase in network usage can be put down to increased execution time. Following an argument identical to that of section 7.5, the network usage of both the heuristic and non-heuristic algorithms proves to be essentially identical — as was expected.

There is a sizable increase in execution time and a corresponding increase in wallclock time observed when the heuristic is applied. The particular heuristic approach used involves sorting each individual into ascending order of distance from the depot and uses the inbuilt Java `Collections.sort(List list)` method,

which makes use of a modified mergesort algorithm, with complexity of $O(n \log n)$. This must be executed for every individual within the population and must be done every generation. Therefore the complete complexity of the operation per generation is $O(mn \log n)$,¹⁸ where m is the size of the population and n is the size of the chromosome. The increase in CPU time is shown in figure 7.6.3 for the serial algorithm and demonstrates a roughly two fold increase when the heuristic is applied.

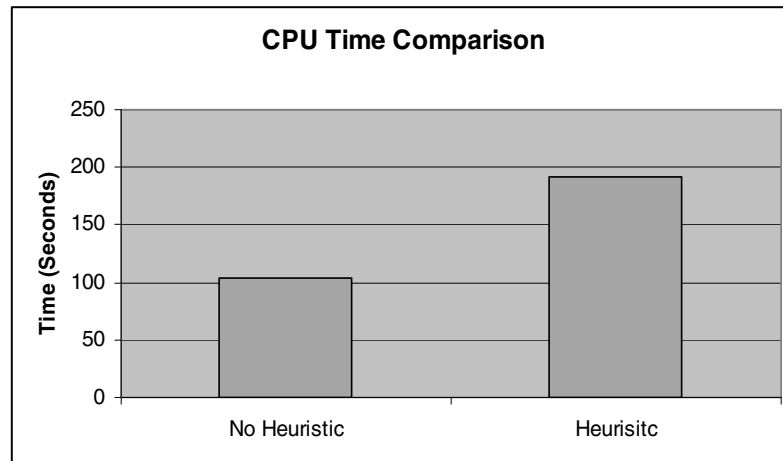


Figure 7.6.3: The observed increase in CPU time for the serial algorithm with a 100 member population when the simple heuristic was applied. Results measured over 100 generations

By way of explanation for these results, consider the following semi-formal justification. Without the heuristic, the complexity of a single generation is given by summing the complexities of the individual operations required to complete the generation. Within the serial algorithm, the operations that do not have constant complexity are generating the new population with $O(mn)$, evaluating the fitness of all new individuals, also $O(mn)$, and finding the best solution within the generation with $O(mn \log n)$. Consider then the conditions under which the results in figure 7.6.3 were obtained; population of 100 members and a chromosome size of 100. Without the heuristic, we can create a metric for the creation of a new generation by summing the complexity of the operations required:

$$(100 \times 100) + (100 \times 100) + (100 \times 100 \times \log 100) \approx 85,000$$

For the case where the heuristic is applied, the above metric plus that for the heuristic application is arrived at:

¹⁸ Throughout this explanation, the logarithmic function is assumed to be base two.

$$85,000 + 65,000 \approx 150,000$$

This is approximately a 1.75 fold increase in time complexity, which is comparable to the empirical result demonstrated by figure 7.6.3.

The simple heuristic demonstrated within this section has quite an extensive impact on execution time, but provides relatively limited improvement to the results. Within section 3.4.6 it was suggested that imposing structure upon the results (as is done by this heuristic) is redundant as the necessary structure will be emergent due to the selective pressure. Indeed, the results of this experiment suggest that the heuristic accelerated the adoption of this anticipated structure within solutions, but had little other effect. Combine this with observations within section 3.4.6 as to the emergent structure of population members. These points support the argument that the structures promoted by the re-ordering would have been emergent given sufficient trials even without the heuristic.

7.7. PROBLEM DECOMPOSITION PARALLELISATION

The problem decomposition approach to parallelising the genetic algorithm is discussed in section 3.5 and the methodology and expected results for this experiment are presented in section 6.7. Two problem decomposition approaches were made and the results compared with the sub-population approaches. The first, as described in section 6.7 is that of a simple four way split of the problem space. The second was a conceptual decomposition into the constituent components of the CVRP - the *travelling salesman* problem and the *bin packing* (or *knapsack*) problem.

7.7.1 NAÏVE FOUR-WAY DECOMPOSITION

A simplistic split of the problem space into four distinct segments is made, each representing a completely self-contained sub-problem. These segments are presented to a host within the cluster for evaluation by the genetic algorithm — one segment per host. There is no inter-host communication required; the problems are entirely disjunct. Results simply need to be combined at the completion of each section to form the full solution. Each sub-problem is executed in parallel with the others.

The major result of this experiment is the quality of solutions found after 100 generations. As presented in figure 7.7.1.1, the solutions found using this approach are a significant improvement over all the other algorithm types explored thus far.

The graph shows the average best solution found by the naïve four-way decomposition compared to the known optimal solution for the problem and the best solution found by the parallel distributed algorithm. The result of the parallel distributed algorithm required approximately 150,000 generations to locate, whereas the results presented for the naïve four-way decomposition algorithm required only 100 generations. The naïve problem decomposition algorithm was also augmented with the route-ordering heuristic discussed in section 3.5.4. With this addition, it demonstrated results from 100 generations which are comparable to those produced by the parallel distributed algorithm after approximately 150,000 generations.

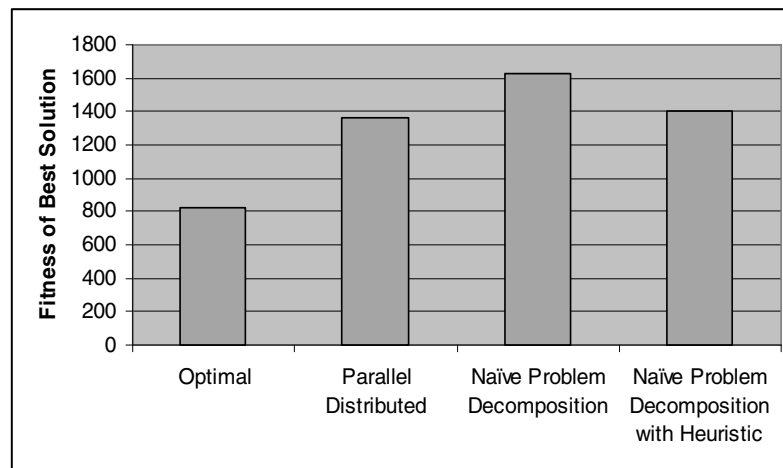


Figure 7.7.1.1: Comparison of best solutions found by the naïve problem decomposition algorithm and the distributed sub-population algorithm. The decomposition algorithm was measured over 100 generations and the sub-problem algorithm was measured over approximately 150,000 generations.

One final point on the effectiveness of this algorithm is given by the performance metrics of CPU usage and memory usage. The results given above comparing the parallel distributed and naïve problem de-composition algorithms are measured over different total generations and hence a comparison of total CPU time is meaningless. However the value of CPU time per generation is of interest. Due to the reduced problem size of the problem de-composition algorithm, the average CPU time per generation is much lower than that of the parallel distributed algorithm working on the full problem space. On average a CPU time of 1 second per generation is required for the naïve four-way decomposition algorithm. In comparison, an average of approximately 10 seconds CPU time per generation is required for the parallel distributed algorithm. Memory usage per host is also significantly less. Of course, network usage is irrelevant for the problem decomposition algorithms, as they do not perform inter-host migration or communication. The full comparison of CPU and memory usage characteristics is presented in figure 7.7.1.2 and 7.7.1.3.

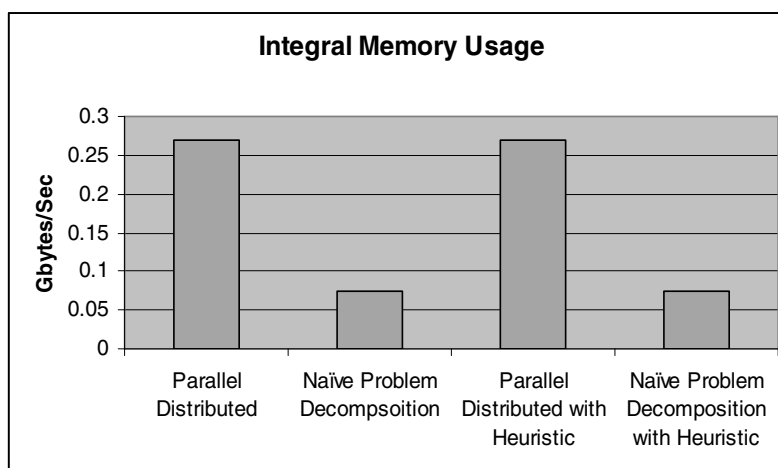


Figure 7.7.1.2: The integral memory usage of the sub-problem and sub-population algorithms measured in gigabytes per second.

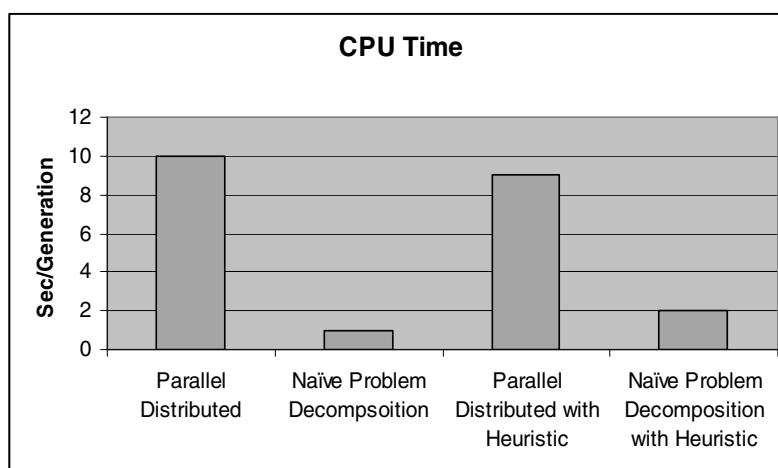


Figure 7.7.1.3: The CPU time measured in seconds per generation for the sub-problem and the sub-population algorithms.

7.7.2 CONCEPTUAL DECOMPOSITION

The parallel algorithm based on a decomposition of the problem into its constituent conceptual parts follows, by chance, essentially the same structure as the naïve decomposition. The full method is described in section 6.7 and once again the problem is broken into several sub-instances. In this case ten such instances occur rather than the four used in the naïve decomposition. The number of sub-problems is determined by the optimal number of clusters for the dataset.

Quality of solution is considered first and produces the best results of all algorithms considered thus far. Discovered solutions are on average only several units short of the optimal solution for the problem, after only 100 generations. The average quality of the best solutions found after 100 generations in the conceptual decomposition and naïve decomposition algorithm are presented in figure 7.7.2.1. Once again, the optimal solution is also presented for comparison. As can be seen, the conceptual decomposition discovers results that are, in practical terms, within such a small variation of the optimal solution as to be considered equivalent.¹⁹

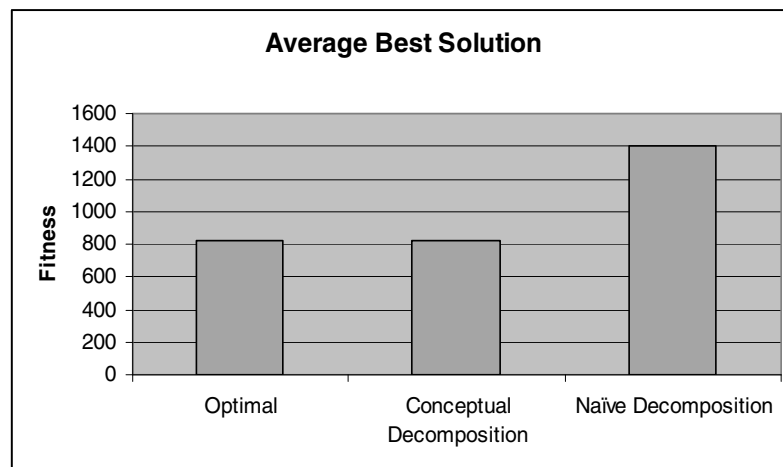


Figure 7.7.2.1: Average best solution of the conceptual sub-problem algorithm compared to the naïve sub-problem algorithm with the optimal problem solution for comparison.

Considering performance, the conceptual decomposition requires less time per generation as each sub-problem is smaller. However there are more sub-problems being executed in parallel and hence the total wallclock time required is essentially equal to the naïve decomposition. The same is true of the memory usage.

7.7.3 SUMMARY

The results observed within the problem decomposition algorithms were as anticipated; this parallel decomposition is indeed the most efficient regards all of the metrics. It has the shortest running time of all the parallel algorithms, uses less memory, requires almost no network bandwidth and produces consistent results that are close to optimal. However there is a caveat, the algorithm is far from general. Although standard machine learning approaches can be applied to finding an appropriate number of clusters (Hamerly & Elkan, 2003) the algorithm is dependent on a structural heuristic. In the case of the CVRP this is that destinations within close

¹⁹ In fact, this decomposition of the problem can never result in the optimal solution for this instance, as the optimal solution has routes with nodes which have been spread between separate clusters.

proximity to one another should be serviced in a common route. Indeed, a manual inspection of the data for the problem instance used shows a highly structured pattern to the data. In short, although this approach is likely to produce promising results on other instances of the CVRP and similar problems, the initial choice of method for decomposition is by no means trivial. It is far from obvious whether this stage can be automated or is even possible on general problems. In contrast, the sub-population approach is a general method which can be used with any problem for which a genetic algorithm is applicable.

As an aside, this decomposition provides an example where the serial algorithm is a more natural choice than a parallel variant. The sub-problems that are created are sufficiently small that the serial algorithm finds near-optimal solutions within a few generations. Although the parallel algorithm would duplicate this result, the computational complexity of the algorithm is much greater. Within the global scope of the algorithm however, the serial components are indeed executing in parallel on sub-problems. It is the application of a further layer of parallelism for the sub-problems which is unnecessary and in fact wastes resources.

8. FURTHER OBSERVATIONS AND EXPERIMENTATION

Several characteristics were observed whilst conducting experiments which were not initially anticipated, or were more pronounced than expected. This chapter details the results of extra investigations into these characteristics. In particular, considerable investigation was conducted into analysing the convergence of the algorithm in an effort to determine its cause and effect on experiment results.

8.1. CONVERGENCE

Observation of the performance of the algorithm during experimentation seems to indicate that it suffers from premature convergence. That is, many of the individuals in the population are of very similar genetic structure. The comparison of average fitness with the fitness of the best individuals found in each generation would seem to present a counter point to this. There is a significant difference, suggesting genetic diversity is still present. However, a closer examination of the population reveals this to be an artefact of the fitness scaling applied to solutions which exceed the bounds of the problem. As can be seen from the excerpt of a generation 200 population in figure 8.1.1, there is a division of the population into two classes. These two classes are made up of those solutions which are within the problem bounds and those which are not. Each class has converged. To further highlight this fact, the fitness penalty applied to the solutions exceeding the problem bound is removed. It can then be seen that the population is collapsed into a single class, where full convergence occurs. The population sample presented in figure 8.1.2 demonstrates this. Regardless, convergence can be seen in both populations and similar results were obtained from further experiments.

To ensure that this characteristic was emerging during execution rather than being present from initial population construction, the populations of earlier generations were also sampled. An excerpt from one such generation is presented in figure 8.1.3. As can be seen the level of convergence in this population is much lower, characterised by the diversity of values for each allele²⁰.

Finally, to confirm the presence of premature convergence, tests of later generations were run to confirm that the level of genetic similarity is increased. An excerpt from one such test is given in figure 8.1.4. This demonstrates that by the 400th generation,

²⁰ An allele is any of the alternative forms of a gene that may occur at a given gene locus

the level of convergence had increased over that from the 200th generation. Some alleles even contain the same gene for every chromosome in the population.

In summary, this experiment proved that the initial population of the genetic algorithm contained sufficient randomness to give rise to genetic diversity but that as the algorithm progressed, this diversity was being further and further degraded. Such a result explains the observed tendency of the algorithm for rapid early improvement in fitness but little or no improvement as the number of generations progressed. It is likely that new solutions found in later generations are predominantly the result of the mutation operation.

Chromosome	Fitness ²¹	Gene Sample - Gene (Position)
1	3761	101(0) 59(1) 65(2) 35(3) 45(4) 41(5)
2	3773	69(0) 6(1) 71(2) 75(3) 100(4) 41(5)
3	3808	69(0) 6(1) 71(2) 75(3) 100(4) 41(5)
4	3840	69(0) 6(1) 71(2) 75(3) 100(4) 41(5)
5	3859	69(0) 79(1) 71(2) 14(3) 16(4) 41(5)
6	3864	69(0) 6(1) 71(2) 10(3) 33(4) 41(5)
7	3870	59(0) 65(1) 13(2) 25(3) 71(4) 73(5)
8	3873	6(0) 69(1) 71(2) 92(3) 19(4) 41(5)
9	3889	69(0) 6(1) 19(2) 35(3) 33(4) 41(5)
10	3895	26(0) 50(1) 64(2) 99(3) 29(4) 33(5)
...
52	4472	26(0) 33(1) 13(2) 54(3) 41(4) 69(5)
53	16992	43(94) 61(95) 55(96) 65(97) 54(98) 72(99)
...
90	20346	58(94) 61(95) 55(96) 65(97) 100(98) 46(99)
91	20430	43(94) 61(95) 55(96) 65(97) 54(98) 72(99)
92	20623	43(94) 61(95) 55(96) 65(97) 54(98) 72(99)
93	20679	69(94) 5(95) 55(96) 76(97) 54(98) 43(99)
94	20682	43(94) 61(95) 2(96) 65(97) 54(98) 58(99)
95	20744	43(94) 61(95) 55(96) 65(97) 54(98) 72(99)
96	21016	69(94) 61(95) 90(96) 65(97) 54(98) 41(99)
97	21339	43(94) 61(95) 55(96) 65(97) 54(98) 72(99)
98	21418	43(94) 61(95) 55(96) 91(97) 54(98) 72(99)
99	21545	69(94) 5(95) 55(96) 76(97) 54(98) 57(99)

Figure 8.1.1: Except from the full record of a generation 200 population with fitness penalty for exceeding the problem bounds.

²¹ The difference in fitness values between the two populations (which are both from the 200th generation) is due to the removal of the scaling factor, which increases the average fitness of solutions within the problem space, essentially simplifying the problem.

Chromosome	Fitness	Gene Sample - Gene (Position)
1	1976	100(0) 10(1) 4(2) 6(3) 25(4) 48(5)
2	2031	100(0) 10(1) 22(2) 25(3) 24(4) 65(5)
3	2073	73(0) 10(1) 4(2) 6(3) 25(4) 48(5)
...
97	2558	93(94) 96(95) 99(96) 89(97) 83(98) 88(99)
98	2622	93(94) 96(95) 99(96) 100(97) 83(98) 88(99)
99	2628	90(94) 96(95) 99(96) 100(97) 83(98) 88(99)

Figure 8.1.2: Excerpt from the full record of a generation 200 population with no fitness penalty for exceeding the problem bounds.

Chromosome	Fitness	Gene Sample - Gene (Position)
1	4334	39(0) 71(1) 65(2) 40(3) 94(4) 14(5)
2	4456	89(0) 93(1) 7(2) 38(3) 67(4) 65(5)
3	4506	89(0) 50(1) 53(2) 39(3) 56(4) 61(5)
4	4556	66(0) 17(1) 19(2) 43(3) 39(4) 73(5)
5	4563	33(0) 18(1) 90(2) 4(3) 23(4) 79(5)

Figure 8.1.3: Excerpt from a generation 1 population

Chromosome	Fitness	Gene Sample - Gene (Position)
1	3341	59(94) 55(95) 71(96) 77(97) 78(98) 97(99)
2	3429	59(94) 55(95) 72(96) 77(97) 78(98) 91(99)
3	3462	59(94) 55(95) 72(96) 77(97) 78(98) 15(99)
4	3486	59(94) 55(95) 71(96) 77(97) 78(98) 97(99)
5	3488	59(94) 55(95) 72(96) 77(97) 8(98) 76(99)

Figure 8.1.4: Excerpt from a generation 400 population

8.2. PARALLEL WORK DUPLICATION

It is apparent from the results obtained during the investigation of the parallel algorithms that much of the extra computational power is not being effectively used. Although the amount of computation is significantly increased, the quality of solution and number of generations required to reach results similar to the serial algorithm is not improved. A closer exploration of the computation being undertaken during the parallel algorithm was made in an attempt to identify where this extra computational power was being lost. Several excerpts from the full set of results are presented in figure 8.2.1. It can be seen that during the given generation, the parallel

islands contain members with very similar genes in the 99th position. As with the results presented in section 8.1, this trend was apparent for all alleles within the chromosomes and increased with generation. The experiment was conducted several times and the results were similar to those presented in figure 8.2.1. From these results, it is apparent that each island is concurrently exploring the same section of the problem space and thus duplicating the work of other islands. In addition, each island exhibits the convergence characteristics observed in section 8.1 for the serial algorithm.

It is highly likely that this duplication of work within the parallel algorithm can be attributed to the convergence characteristics of the serial algorithm and the migration scheme. The scheme employed herein was fitness based migration. The islands converge to a point in the search space and the exchange of high scoring individuals between the islands directs all islands to a similar part of the problem space. Modification of the migration scheme to explore different techniques and a further investigation into eliminating premature convergence from the serial algorithm would be necessary to fully explore this hypothesis and possibly improve performance.

Chromosome	Island 1 Gene Value (Position)	Island 2 Gene Value (Position)
1	57(99)	57(99)
2	57(99)	71(99)
3	57(99)	57(99)
4	27(99)	71(99)
5	27(99)	57(99)
6	27(99)	57(99)
7	57(99)	57(99)
8	57(99)	27(99)
9	27(99)	57(99)
10	27(99)	27(99)
11	27(99)	27(99)
12	27(99)	27(99)
13	27(99)	57(99)
14	27(99)	57(99)
15	57(99)	57(99)
16	27(99)	27(99)
17	27(99)	57(99)
18	66(99)	57(99)
19	57(99)	57(99)
20	57(99)	71(99)

Figure 8.2.1: Examples of the observed values for the 99th allele in two islands of a parallel distributed genetic algorithm. Note the similar gene values not only within each island but between the islands also.

8.3. OBJECTIVE CONVERGENCE MEASUREMENT

8.3.1 SPECIFICATION

It is clear from the above results and observations that an objective measure for convergence was needed. Within this experiment, a new metric is introduced to allow the measurement of convergence, automatically and objectively, such that comparison of genetic diversity within generations can be made.

The metric is based heavily upon a string difference approach. Each individual is represented by two chromosomes, each of which can be expressed as a string of genes with integer values. A measure of the difference between two members of a population is then simply the string difference between the chromosomes that it aggregates and those aggregated by the member to be compared against. The difference algorithm used is the Levenshtein Distance (Levenshtein, 1965), the specifics of which have been omitted here.

Now having a measure for the difference between two members of a population, the concept can be scaled to provide a measure of distance within the population as a whole. The algorithm for doing so has the following structure:

1. For every member, compute the distance to every other member within the population for which the difference between the two has not already been computed.
2. Find the average distance of all these values and take this as the average distance from this member to the rest of the population.
3. Find the average of the distances computed for all members and take this as the measure of genetic diversity within the population.
 - a. Large values represent high diversity, small values represent low diversity.

The major drawback to this approach is the computational complexity. The comparison of all individuals to all other individuals has a worst-case running time $O(n^2)$. The comparison operation has $O(nm)$, therefore the algorithm in general has $O(n^3)$, where n is the number of individuals for comparison and m is their length. If the metric is to be measured for every generation, the complexity becomes $O(pn^3)$ where p is the number of generations. It is for this reason that the application of the

algorithm to obtain experimental results was limited to generalised cases. Application to all variants of the genetic algorithms to determine their effect on genetic diversity would have been too computationally expensive. This metric was measured for both the serial and distributed parallel algorithms without any heuristic modification; measurements were made every ten generations to reduce run time.

8.3.2 RESULTS

The results of measuring the genetic diversity of the serial algorithm confirmed the observed trends in genetic diversity made in sections 8.1 and 8.2. It is clear from the graph presented in figure 8.3.2.1 that a reduction in genetic diversity occurs as the algorithm progresses. The value found for generation 0 represents an initial diversity of a randomly generated population. A reduction from this initial level to the next reading at generation 10 is expected due to the focussing of the search. However, the result of interest is the general declination of genetic diversity over the course of the 100 generations for which it was measured. In fact, the decrease is not as pronounced as expected and indeed there is no point of reference from which to base an assertion as to whether such levels of genetic diversity are good or bad. Rather, what this result provides is a point from which to judge the effect of parallelising the algorithm on genetic diversity within the population.

As an aside, it is useful at this point to consider the granularity of this measure. Despite a reasonable measure for general genetic diversity, a loss of diversity per allele can still occur without necessarily lowering the overall population diversity. This is the situation that is presented by the analysis in section 8.1.

Applying the same measure to the distributed parallel algorithm demonstrates a more considerable loss in genetic diversity over the course of 100 generations. The graph shows an initial genetic diversity similar to the serial algorithm, but a more dramatic degradation. The increased loss of diversity in the parallel algorithm can be attributed to the added selective pressure generated by the migration scheme.

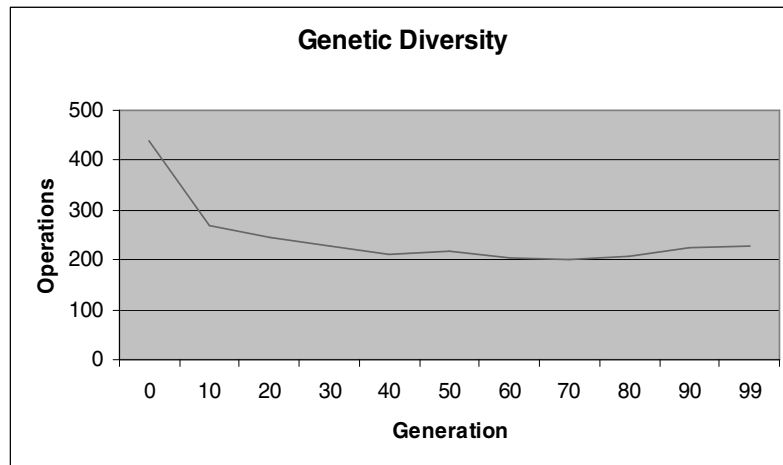


Figure 8.3.2.1: The genetic diversity measured periodically over 100 generations within the serial algorithm in terms of average number of operations required to transform a member into any other member.

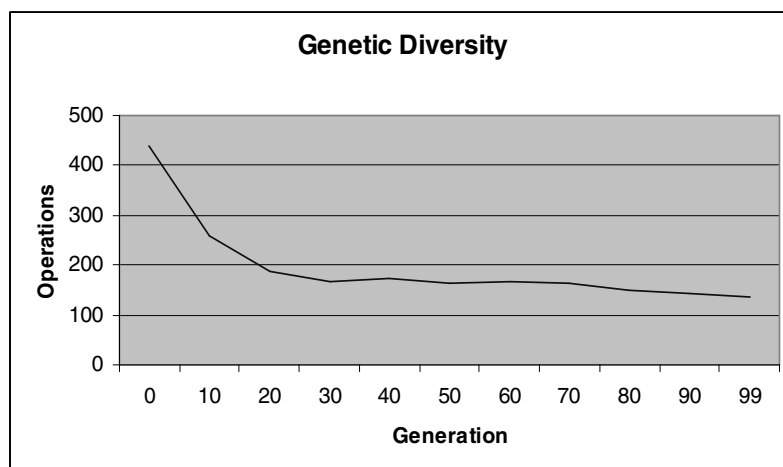


Figure 8.3.2.2: The genetic diversity measured periodically over 100 generations within the distributed parallel algorithm in terms of average number of operations required to transform a member into any other member.

8.3.3 DISCUSSION

The most relevant conclusion to draw from this experiment, in conjunction with the other analyses of convergence given throughout chapter 8, is that parallel genetic algorithms do not necessarily improve convergence. This can be seen by comparing the two graphs in figures 8.3.2.1 and 8.3.2.2. In section 3.4 parallel algorithms were discussed as a way to improve premature convergence. The claim clearly needs to be qualified by stating that improvement, if any, is dependent on the mechanism of parallelisation. Finally, the issue of whether the level of genetic diversity within these algorithms is sufficient is an open question. This is tied to the crucial issue of

exploitation versus exploration that was presented in section 3.3. Indeed, this is a fundamental issue in most all forms of artificial intelligence and life. Given a more thorough exploration of the problem instance this question could possibly be answered in a very restricted domain. This is definitely beyond the scope of this work however and would result in little useful information. Hence such an analysis has been omitted.

8.4. EXPANDED CLUSTER SIZE

8.4.1 EXPERIMENT

Within section 7.5, the performance of the parallel distributed genetic algorithm was examined within a varying cluster size. It was noted that a cluster size of four hosts resulted in performance that was not in keeping with the trends observed over smaller cluster sizes. The explanation provided was that of extra load upon the master machine due to a need for it to perform as both a server and one of the GA islands. Within this experiment the cluster size is further increased by two machines, giving the potential to run a four host cluster without such interference. In addition, experiments can be conducted with 5 hosts to further support the results obtained in section 7.5.

8.4.2 RESULTS

The first result of interest is the quality of solutions found within the various cluster sizes. Without the influence of the heavily loaded 4th host, the results for the four host cluster are more indicative of observed trends. As can be seen within figure 8.4.2.1, the fitness of solutions continues to improve as the cluster size is expanded. However, improvements are relatively small beyond the two host cluster. As has been addressed earlier, this is due to the lack of genetic diversity across the islands. This further supports the argument that little advantage can be gained from continuously increasing the cluster size if the fundamental issue of convergence and genetic diversity is not addressed.



Figure 8.4.2.1: The effect of increased cluster size on solution quality within the distributed parallel genetic algorithm.

Of interest also is the wallclock time, shown in figure 8.4.2.2, of the algorithm as the cluster size increases. Intuitively, the required total wallclock time increases with the cluster size due to amplified wait time for IO as the server becomes more heavily loaded. This is backed up by the fact that CPU time remains relatively constant, although the graph is omitted here due to simplicity.

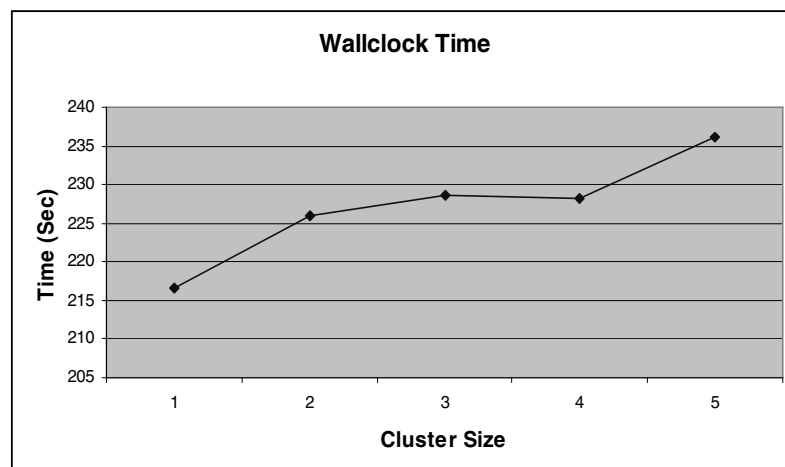


Figure 8.4.2.2: The effect of increased cluster size on wallclock time within the distributed parallel genetic algorithm.

Network usage also increases as expected, demonstrating an almost linear relationship with cluster size. There is however another attribute influencing network usage which is related to the cluster size. With a larger cluster size, there is an increased wait time for server access. Thus, the measure of network usage over a time period is less than it would otherwise have been. This can be observed in the graph by noting that the increase in network usage per second between cluster sizes

decreases as the cluster size increases. However, if the IO blocking could be eliminated, the relationship would be linear.

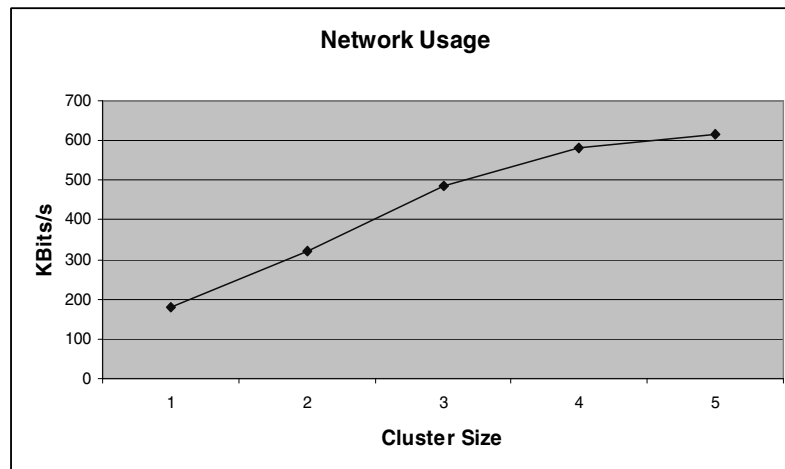


Figure 8.4.2.3: The effect of increased cluster size on network usage within the distributed parallel genetic algorithm.

These experiments have demonstrated that the observed discrepancy in the performance of the four host cluster was simply an attribute of increased host load. Furthermore, the observed trends within the one, two and three host clusters are extensible to the larger clusters. Finally, performance does not greatly improve as a result of increasing the cluster size within the distributed parallel algorithm due to poor genetic diversity between and within islands. These results suggest that arbitrarily increasing cluster size without addressing this issue will yield little or no improvement.

9. DISCUSSION

Having now presented the experiments, expectations, and results aimed at proving or disproving the hypothesis, the observations can be drawn together to form a reasoned argument aimed at addressing this issue. First, however, a restatement of the hypothesis is given. Broadly speaking, the hypothesis is that the parallel implementation of a genetic algorithm will outperform its serial counterpart and that with the addition of domain specific knowledge and heuristics, performance will be further improved. A slightly more constrained version was given in chapter 4, which defined the metrics upon which the performance and comparison of these algorithms would be based. It stated that the parallel algorithms would produce results with a higher average fitness over 100 generations. Furthermore, the inclusion of domain specific knowledge into the algorithms would additionally improve the fitness of these results over the 100 generation test period. Finally, it stated a small expected increase in computation time and memory usage for the heuristic algorithms and an approximately linear increase in network usage as the cluster size is increased.

It is of some concern that most of the experimentation was carried out on a single instance of the capacited vehicle routing problem. Of course there is always the possibility of this being a particularly unusual instance. However, if one examines the nature of the hypothesis and conclusions it is apparent that this is of limited concern. The conclusions presented here disprove the vast majority of the hypothesis due to its generality. Therefore, even a single instance is sufficient for this counter-proof. There is no reason to expect that these results will not be extensible to other instances of the capacited vehicle routing problem, however it is realised that such an extension is not possible based solely on this work. With this in mind though, further work will be presented within a subsequent chapter aimed at extending the methods and experimentation of this work to allow more general conclusions to be drawn.

In general, the assertion within the hypothesis that the parallel algorithms will produce better results is disproved due to its generality. The parallel algorithms did not always outperform the serial algorithm in terms of solution quality after 100 generations. Where they did, it was sometimes not by great enough margins to eliminate the possibility of such a result being due to the stochastic nature of genetic algorithms. That is, it is not possible to state that parallel genetic algorithms will outperform serial algorithms under all conditions. Indeed, quite the opposite has been demonstrated; it is possible to engineer problems upon which a serial algorithm is

more appropriate than a parallel one. In light of this, it is not possible to prove that parallel genetic algorithms conclusively outperform their serial counterparts.

The assertions regarding the heuristics are also overly-general and for this reason cannot be conclusively proved. This work does not demonstrate that heuristic-augmented genetic algorithms will always outperform those without heuristics. Indeed, it is quite obvious that a poor heuristic would adversely affect the performance of a genetic algorithm. However, within the context of this work alone, it was demonstrated that the simple heuristic can accelerate the incorporation of desired structure within the solutions generated by the genetic algorithm. Whether this ultimately leads to improved performance depends entirely upon the accuracy of the heuristic and characteristics of the problem in question. The problem decomposition algorithm, which in itself uses a heuristic approach for decomposition, demonstrates the balance between generality and speed. The optimal solution was not attainable after applying the heuristic, however much better solutions were found much faster than with any of the other algorithms. Whether this improved speed is worth the possibility of eliminating the optimal solution from the search space is a matter dependent on the desired application and on how much weight is given to each of the attributes of speed and eventual accuracy. Whether the use of a specialised approach grants sufficient improvement over a more general approach to warrant its selection is also not immediately answerable.

With regards to the performance expectations stated within the hypothesis, they were accurate but the generality of these statements was once again of concern. They are proved within the limited context of this work, but the results are certainly not extensible to genetic algorithms in general. As with the other results, they are entirely dependent on the choice of architecture and algorithm design. The problem decomposition algorithms present a situation where the application of a heuristic improves performance due to reducing the complexity of the problem. In general this will be true of all genetic algorithms. Where the application of a heuristic does not simplify the problem and hence reduce the computational complexity, at best comparable run time can be expected to the variant without the heuristic. The naïve heuristic observed in section 7.6 exemplifies this. In contrast, if the heuristic allows a simplification of problem complexity, as was the case with the decomposition heuristic used in section 7.7, improvements in execution time can be achieved. In the worst case however, the execution time is not improved. Memory usage follows an almost identical argument.

The results of the experiments intended to contrast the performance of the sub-population parallel algorithms with that of the serial algorithm were inconclusive.

They produced no firm evidence to suggest the sub-population algorithms handle the problem of convergence more effectively or are able to more efficiently explore the search space. Indeed, considering the computational cost, the sub-problem parallel algorithms are less efficient. They consume more resources in terms of computation time and network bandwidth to reach results which are not significantly better than those of the serial algorithm. It was shown that the extra computational time of the parallel algorithms can easily be wasted due to a repetitive search of the same area within the problem space. However, this is not to say that the sub-population algorithms are incapable of better performance. Genetic algorithms in general depend on the selection of a number of attributes, relating to such things as selection and reproduction. Parallelising these algorithms introduce even more attributes upon which performance depends. Indeed, much research has been conducted into the performance of these algorithms and yet there is no optimal set of values that will always produce good results; it is unlikely that such a set exists. Ironically, genetic algorithms themselves represent an optimisation problem — the exact problem they are intended to solve.

The Sun Grid Engine was used as a mechanism for allowing the parallel execution of the genetic algorithm islands. Its use raised several interesting considerations. Foremost, it was very useful for collecting auditing information and monitoring the execution of the various algorithm components. However, grid computing is not necessary within this limited capacity. The cluster size was only four hosts; there was and is little likelihood of it being increased far beyond this number. For this reason, the dynamic allocation of tasks was not particularly necessary and indeed the dedicated nature of the cluster negates the need for effective resource allocation. Furthermore, the granularity of the application in most cases was large and hence there was little need for constant rescheduling. This is not to say that grid computing does not have a use within such an application; this is not the case. Within a larger cluster which was not dedicated, the dynamic allocation of jobs with effective resource allocation is most definitely a desirable operation. Moreover, if the granularity of parallel break-up is small, the need for rescheduling will be greater and automation is necessary. In short, the problem is amenable to grid computing, however within this work the environment was sufficiently small and controlled that it was not necessary. Indeed, the conceptual target of grid computing is not small isolated experimental clusters but large scale, multi-purpose collections of machines — the goal being to reclaim wasted computational capacity.

The programming model used also warrants examination. Although it was not explained in detail, the design was focused on extensibility and generality. That is, the problem specifics were very carefully separated from the genetic algorithm

mechanics. Although this is good object oriented design, it leads to more verbose code and in several instances reduced efficiency of execution. The language of choice was Java, a hybrid interpreted-compiled language with a (perhaps somewhat unfounded) reputation for slow execution speed. The chosen problem instance is also quite expansive and combining all these factors lead to some of the algorithms having quite a slow execution speed. A more efficiency-based approach may have been beneficial in allowing measurements to be made over a larger number of generations. The sub-population parallel distributed algorithm was run up to approximately 150,000 generations on one occasion, which required an execution time of approximately 4 days. This was certainly a major advantage of the problem decomposition approach, which had running time in the order of minutes rather than days. The client-server architecture also presents concerns for scalability. Increasing the number of host increases the load on the server and will eventually cause a reduction in execution time per host. A decentralised approach is definitely more desirable, improving scalability and more closely mirroring the intended target of grid computing.

The problem decomposition approach to parallel genetic algorithms revisits issues that were touched upon in section 3.4.6 and are fundamental to the development and use of genetic algorithms. This is the issue of generality and applicability. The abstract concept of genetic algorithms provides a general approach to problem solving and indeed in its original form is applicable to many different problem types. However, there are problems that are not effectively representable within the constraints of the traditional genetic algorithm. Many modifications have been proposed to both the representation component of genetic algorithms and the actual functional component to cope with this. Numerous modifications are aimed at addressing specific applications. These either enable the use of genetic algorithms upon problems that would otherwise not be amenable to such an approach or to enhance the performance of genetic algorithms within existing applications. Generality of solution is, with good reason, a desired attribute within computer science; it simplifies re-application and conceptualisation amongst other benefits. The problem decomposition approach presented herein cannot be generalised to arbitrary problems. At best, it can be generalised to capacited vehicle routing and possibly related problems. In short, what has been found is an effective means for finding good solutions to the capacited vehicle routing problem in short running time, but not a general improvement to the function of genetic algorithms.

The issue of convergence was also identified as important within this work. As was apparent, the one dimensional metric for the comparison of genetic diversity within different populations was not sufficiently descriptive. It overlooks the fact that the

loss of diversity within a single allele can be as detrimental to the explorative capabilities of the algorithm as the complete loss of diversity on a global scale within the population. In addition to this, the lack of any sort of measure of acceptable genetic diversity (i.e. how diverse should the population be to allow continued exploration) leads to problems in establishing whether the algorithm has achieved premature convergence or not.

The problem decomposition algorithm presented within this work is indeed quite efficient at solving the capacited vehicle routing problem. However its characteristics are highly related to the discussion of algorithm generality which was presented in section 3.4.6. This algorithm is highly specific and as such, it loses many of the advantages of genetic algorithms and evolutionary computing techniques in general, such as an ability to be applied to a large range of problems. Although the concept of decomposing the CVRP into the BPP and the TSP is not unique, the application of genetic algorithms to this approach is relatively un-examined. Indeed, this is likely as a result of the above consideration of generality. The use of a separate approach for the decomposition does to a certain extent alleviate this as the genetic algorithm used is indeed still the traditional approach. However purists may consider it a violation of the generality of the genetic algorithm.

10. CONCLUSIONS

This work began by presenting a hypothesis regarding the expected improvement in performance that could be offered by employing a parallel genetic algorithm in place of a serial variant. Following an introduction to the concept of genetic algorithms, the various work regarding parallel and distributed versions was explored. A method of classification was arrived at and several approaches to parallelising genetic algorithms that exemplify the apparent classes were examined. The motivation was the understanding and thereby selection of appropriate algorithms to explore the hypothesis. A coarse-grained approach was selected and a test problem of sufficient complexity was identified: the capacitated vehicle routing problem. Metrics for comparison were explored and a collection of measures based upon solution fitness, memory usage, CPU utilization, and bandwidth usage were identified. Measures for convergence were also explored and a string difference approach was selected as an objective metric. A distributed architecture was presented as the environment for experimentation and several algorithm variants were constructed to explore the issues presented within the hypothesis statement. The algorithms developed were tested for consensus with known results. Experiments were proposed to explore the hypothesis. Finally, results were presented and discussed which uncovered a more complex nature to the hypothesis.

The empirical evidence shows that parallel genetic algorithms can indeed outperform their serial counterparts by making use of more computational power. However, the characteristics of the algorithm and the problem in question bear heavily on this capacity. It was demonstrated that parallel algorithms can be devised which grant little or no improvement over the serial variant from which they are derived, despite being more computational intense. Heuristic enhancement can be used to force structural requirements to be incorporated into the solutions of a genetic algorithm more rapidly, and hence lead to better solutions within smaller timeframes. However, results suggest that these structures will be emergent within the algorithm without the need for heuristics, a conclusion supported by previous work. The effectiveness of the heuristics is dependent on their validity and characteristics; it was demonstrated that heuristic measures can in the long term reduce the effectiveness of genetic algorithms.

The most promising algorithm presented in terms of the metrics used for comparison was the problem decomposition approach to parallelising genetic algorithms. This incorporated heuristic decomposition with parallel processing to produce results

comparable to optimal solutions within very minimal run time. Despite excellent performance however, this algorithm was found to be highly specific and it is not possible to generalise these results to applications other than the capacited vehicle routing and related problems.

Ultimately, the general nature of the hypothesis prevents a conclusive proof or disproof of it from the results obtained. More accurately, the hypothesis that parallel genetic algorithms outperform their serial counterparts and that heuristic augmentation can further improve performance can be proven within a restricted domain and concurrently shown to be untrue in highly general circumstances.

11. FURTHER WORK

Several intriguing directions for further exploration and research have been identified during the course of this work. This chapter presents some of these observations and a series of directions upon which further work could be based.

As noted in the discussion of results, a useful measure of algorithm convergence was lacking and caused an inability to effectively determine the effect that genetic diversity was having on the effective search of the problem space. The one-dimensional metric presented within section 8.3 allows an approximation of genetic diversity, but a determination of what level of diversity is sufficient is not available. Such a measure would no doubt be dependent on the dimensions of the problem space, and vicariously the frequency of possible gene values for each allele that is present within the population. The development of an effective metric for this presents a possible direction for future work.

The problem decomposition approach to parallelising the genetic algorithm represented an efficient technique in terms of both performance and quality metrics. In its current form the KMeans algorithm was used to decompose the problem instance into smaller TSP instances and in doing so solve the BPP component of the problem. Further work upon this approach could replace the KMeans algorithm with a hierarchical genetic algorithm. Such an algorithm would have as the goal of its upper tier the effective solution of the BPP component and at its lower tier an effective solution of the TSP sub-problem. It is likely this approach will perform less efficiently in terms of performance metrics such as CPU time and memory usage. However, it is also likely to be more generally applicable to less well behaved instances of the capacited vehicle routing problem and possibly more generally applicable to other applications. Further work upon this could compare an expected loss in speed and memory usage with observed empirical results. In addition, an increase in general applicability could be explored by comparing results of both approaches upon well behaved and less amenable instances.

Although developing genetic algorithms which handle the issues of convergence and genetic diversity effectively was beyond the scope of this work it did arise as a serious issue. Effective measures for ensuring a sufficient level of genetic diversity could be explored as further work, linking with the issue of defining an effective metric of genetic diversity discussed above. As was covered within the literature review, determining effective setting for the genetic algorithm regards such

influences as selective pressure is very much an unrefined process centralised around empirical observations. Significant research has been undertaken into this concern but as yet little in the way of concrete theory has been produced which can be effectively applied to genetic algorithms in general.

Connected with the issue of ensuring effective genetic diversity is the consideration of a method for reducing the level of duplicated work in the parallel islands. Several potential starting points include reducing the frequency of migration or selective migration based upon the genetic difference between the migrant and the destination island. This is also connected with determining a more efficient method of measuring genetic diversity, as the method present herein is too computationally complex to be used practically. Once this has been achieved an exploration of larger cluster sizes becomes a much more useful endeavour.

Finally, the centralised model used within this work has several drawbacks, the most obvious of which is the server load. Within larger cluster sizes there are issues of asymptotic performance improvements due to a maximum capacity of the server. A much more efficient mechanism would involve a decentralised approach where migration was entirely a peer-to-peer operation. Of course, this involves significant modification of the underlying topology and organisation.

The algorithm performance is quite slow, as is apparent by examining the execution times presented throughout the results chapter. Developing a more efficient algorithm in terms of execution would enable a much more thorough investigation of results by eliminating the considerable time penalty. Preliminary observations suggest that the evaluation of solution fitness is a major contributor to execution time. A suggested avenue for exploring a more efficient algorithm is in applying a parallel algorithm which distributes the evaluation of fitness. Client machines could compute fitness and return results while the server continues to execute the genetic algorithm — such an approach was discussed in section 3.4.3 of the literature review.

12. REFERENCES

- Alba, E., & Dorronsoro, B. e. (2004). *Solving the Vehicle Routing Problem by Using Cellular Genetic Algorithms*. Paper presented at the Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP, Coimbra, Portugal.
- Altenberg, L. (1995). The Schema Theorem and Price's Theorem. In *Foundations of Genetic Algorithms 3* (pp. pp23- 49): Morgan Kaufmann Publishers.
- Andrzejak, A., Graupner, S., Kotov, V., & Trinks, H. (2002). *Control Architecture for Service Grids in a Federation of Utility Data Centers*. Palo Alto: Hewlett-Packard Laboratories.
- Balachandran, R. (2003). *Computational Grid Support for Cluster Schedulers*. Unpublished Masters Thesis.
- Baluja, S. (1993a). *Structure and Performance of Fine-Grain Parallelism in Genetic Search*. Paper presented at the Fifth International Conference on Genetic Algorithms, University of Illinois.
- Baluja, S. (1993b). *Structure and Performance of Fine-Grain Parallelism in Genetic Search*. Paper presented at the Proceedings of the Fifth International Conference on Genetic Algorithms, University of Illinois.
- Belding, T. C. (1995). *The Distributed Genetic Algorithm Revisited*. Paper presented at the Sixth International Conference on Genetic Algorithms, San Francisco.
- Berger, J., & Barkaoui, M. (2003). A Hybrid Genetic Algorithm for the Capacitated Vehicle Routing Problem. *Lecture Notes in Computer Science*, 2723, 646 - 656.
- Cant'u-Paz, E. (1997). *A Survey of Parallel Genetic Algorithms* (Technical Report No. 97003): University of Illinois at Urbana-Champaign.
- Cohoon, J. P., Hegde, S. U., Martin, W. N., & Richards, D. (1988). *Distributed Genetic Algorithms for the Floor Plan Design Problem* (Technical Report): School of Engineering and Applied Science, Computer Science Department, University of Virginia.
- Cohoon, J. P., Martin, W. N., & Richards, D. S. (1991). *A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes*. Paper presented at the Fourth International Conference on Genetic Algorithms, University of California, San Diego.
- Collins, R. J., & Jefferson, D. R. (1991). *Selection in Massively Parallel Genetic Algorithms*. Paper presented at the Fourth International Conference on Genetic Algorithms, University of California, San Diego.

- Davidor, Y. (1991). *A Naturally Occurring Niche & Species Phenomenon: The Model and First Results*. Paper presented at the Fourth International Conference on Genetic Algorithms, University of California, San Diego.
- Davis, L. (1991). *Handbook of Genetic Algorithms*: Van Nostrand Reinhold.
- De-Jong, K. A. (1975). *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. Unpublished Doctoral Thesis, University of Michigan, Ann Arbor.
- Goldberg, D. E. (1989a). *Genetic Algorithms in Search, Optimisation, and Machine Learning*: Addison-Wesley.
- Goldberg, D. E. (1989b). *Sizing Populations for Serial and Parallel Genetic Algorithms*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Goldberg, D. E. (1989c). *Zen and the Art of Genetic Algorithms*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Goldberg, D. E., & Deb, K. (1989). *An Investigation of Niche and Species Formation in Genetic Function Optimisation*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Goldberg, D. E., & Deb, K. (1991). *A comparative analysis of selection schemes used in genetic algorithms*. Paper presented at the Foundations of Genetic Algorithms.
- Goldberg, D. E., & Lingle, R. (1985). *Alleles, loci, and the traveling salesman problem*. Paper presented at the International Conference on Genetic Algorithms and Their Applications.
- Goldberg, D. E., & Richardson, J. (1987). Genetic Algorithms with Sharing for Multimodal Function Optimisation.
- Gondra, I., & Samadzadeh, M. H. (2003). AI and computational science: A coarse-grain parallel genetic algorithm for finding Ramsey Numbers.
- Gorges-Schleuter, M. (1989). *ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Grid-Engine Homepage. (2001). *What is the Grid Engine project?* Retrieved June, 2004, from <http://gridengine.sunsource.net/>
- Hamerly, G., & Elkan, C. (2003, December). *Learning the k in k-means*. Paper presented at the seventeenth annual conference on neural information processing systems (NIPS).
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*: MIT Press.
- Koza, J. R. (1992). *Genetic Programming*: MIT Press.

- Lee, W. (2002). *An Exploratory Study of the Economics of the Computational Grid*. London: Birkbeck College, University of London.
- Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4), 845-848.
- Li, X., & Kirley, M. (2002). *The Effects of Varying Population Density in a Fine-grained Parallel Genetic Algorithm*. Paper presented at the IEEE World Congress on Computational Intelligence (Congress on Evolutionary Computation).
- Lin, S.-C., Goodman, E. D., & Punch-III, W. F. (1994). Coarse-Grain Parallel Genetic Algorithms: Categorisation and New Approach.
- Lobo, F. G., Lima, C. F., & Martires, H. (2004, June 26-30). *An architecture for massively parallelization of the compact genetic algorithm*. Paper presented at the Genetic and Evolutionary Computation Conference (GECCO), Seattle, Washington USA.
- Machado, P., Tavares, J., Pereira, F. B., & Costa, E. (2002). *Vehicle Routing Problem: Doing it the Evolutionary Way*. Paper presented at the Genetic and Evolutionary Computation Conference (GECCO).
- MacQueen, J. B. (1967). *Some Methods for Classification and Analysis of Multivariate Observations*. Paper presented at the Fifth Berkeley Symposium on Mathematical Statistics and Probability.
- Muhlenbein, H. (1989). *Parallel genetic algorithms, population genetics and combinatorial optimisation*. Paper presented at the International Conference on Genetic Algorithms, George Mason University.
- Nilsson, C. (2003). *Heuristics for the Traveling Salesman Problem*.
- Noda, E., Coelho, A. L. V., Ricarte, I. L. M., Yamakami, A., & Freitas, A. A. (2002). *Devising Adaptive Migration Policies for Cooperative Distributed Genetic Algorithms*. Paper presented at the IEEE SMC, Hammamet (Tunisia).
- Osyczka, A. (2002). *Evolutionary Algorithms for Single and Multicriteria Design Optimisation* (Vol. 79): Springer-Verlag.
- Pereira, F. B., Tavares, J., Machado, P., & Costa, E. (2002, 12-13 September). *GVR: a New Genetic Representation for the Vehicle Routing Problem*. Paper presented at the 13th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2002), Limerick, Ireland.
- Petty, C. C., & Leuze, M. R. (1989). *A Theoretical Investigation of a Parallel Genetic Algorithm*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Prasanna Jog, J. Y. S., Dirk Van Gucht. (1989). *The Effects of Population Size, Heuristic Crossover and Local Improvements on a Genetic Algorithm for the*

Traveling Salesman Problem. Paper presented at the International Conference on Genetic Algorithms, George Mason University.

- Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., & Enbody, R. (1993). *Further Research on Feature Selection and Classification Using Genetic Algorithms*. Paper presented at the Fifth International Conference on Genetic Algorithms, University of Illinois.
- Ralphs, T. (2003, October 3). *Vehicle Routing Data Sets*. Retrieved November, 2004, from <http://branchandcut.org/VRP/data/>
- Ralphs, T. K., Kopman, L., Pulleyblank, W. R., & Trotter, L. E. (2001). On the Capacitated Vehicle Routing Problem. *Mathematical Programming*, 94(2-3), 343 - 359.
- Ronald, S. (1995). Preventing Diversity Loss in a Routing Genetic Algorithm with Hash Tagging. *Complexity International*.
- Spiessens, P., & Manderick, B. (1989). *Fine Grained Parallel Genetic Algorithms*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Spiessens, P., & Manderick, B. (1991). *A Massively Parallel Genetic Algorithm*. Paper presented at the Fourth International Conference on Genetic Algorithms, University of California, San Diego.
- Tanenbaum, A. S. (1996). *Computer Networks* (Third ed.): Prentice-Hall.
- Tanese, R. (1989). *Distributed Genetic Algorithms*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Tomassini, M. (1995). A Survey of Genetic Algorithms. *Annual Reviews Of Computational Physics*, 3.
- Vamplew, P. (2004). Personal Communication. In P. Uren (Ed.). Hobart.
- Wang, L., Maciejewski, A. A., Siegel, H. J., & Roychowdhury, V. P. (1998). *A Comparative Study of Five Parallel Genetic Algorithms Using the Travelling Salesman Problem*. Paper presented at the 12th. International Parallel Processing Symposium, Orlando, Florida.
- Whitley, D. (1989). *The GENITOR Algorithm and Selection Pressure*. Paper presented at the Third International Conference on Genetic Algorithms, George Mason University.
- Whitley, D. (1993). A genetic algorithm tutorial.
- Wright, S. (1932). *The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution*. Paper presented at the Sixth International Congress of Genetics.
- Wright, S. (1964). Stochastic Processes in Evolution. In J. Gurland (Ed.), *Models in Medicine and Biology* (pp. 199-241).

- Wright, S. (1982). Character Change, Speciation, and the Higher Taxa. *Evolution*, 36(3).
- Yang, R. (1997). *Solving Large Travelling Salesman Problems with Small Populations*. Paper presented at the Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA), Second International Conference on.

13. APPENDICIS

APPENDIX A0 – CVRP INSTANCE TEST0-N5-K3

NAME : TEST0-	NODE_COORD_SECTION	2 19
N5-K3	1 30 40	3 20
COMMENT : none	2 37 52	4 6
TYPE : CVRP	3 49 49	5 10
DIMENSION : 5	4 52 64	DEPOT_SECTION
EDGE_WEIGHT_T	5 20 30	1
YPE : EUC_2D	DEMAND_SECTION	-1
CAPACITY : 33	1 0	EOF

APPENDIX A1 – CVRP INSTANCE TEST1-N5-K2

NAME : TEST1-	NODE_COORD_SECTION	2 19
N5-K2	1 30 40	3 20
COMMENT : none	2 37 52	4 6
TYPE : CVRP	3 49 49	5 10
DIMENSION : 5	4 52 64	DEPOT_SECTION
EDGE_WEIGHT_T	5 20 30	1
YPE : EUC_2D	DEMAND_SECTION	-1
CAPACITY : 45	1 0	EOF

APPENDIX A2 – CVRP INSTANCE TEST2-N5-K4

NAME : test1_vrp	NODE_COORD_SECTION	2 33
COMMENT : none	1 30 40	3 33
TYPE : CVRP	2 37 52	4 33
DIMENSION : 5	3 49 49	5 33
EDGE_WEIGHT_T	4 52 64	DEPOT_SECTION
YPE : EUC_2D	5 20 30	1
CAPACITY : 33	DEMAND_SECTION	-1
	1 0	EOF

APPENDIX B – CVRP INSTANCE M-n101-k10

NAME : M-n101-k10	44 33 35	99 58 75	52 10
COMMENT :	45 32 30	100 55 80	53 10
(Christophides et al,	46 30 30	101 55 85	54 20
No of trucks: 10,	47 30 32	DEMAND_SECTION	55 40
Optimal value: 820)	48 30 35	1 0	56 10
TYPE : CVRP	49 28 30	2 10	57 30
DIMENSION : 101	50 28 35	3 30	58 40
EDGE_WEIGHT_TY	51 26 32	4 10	59 30
PE : EUC_2D	52 25 30	5 10	60 10
CAPACITY : 200	53 25 35	6 10	61 20
NODE_COORD_SEC	54 44 5	7 20	62 10
TION	55 42 10	8 20	63 20
1 40 50	56 42 15	9 20	64 50
2 45 68	57 40 5	10 10	65 10
3 45 70	58 40 15	11 10	66 10
4 42 66	59 38 5	12 10	67 10
5 42 68	60 38 15	13 20	68 10
6 42 65	61 35 5	14 30	69 10
7 40 69	62 50 30	15 10	70 10
8 40 66	63 50 35	16 40	71 30
9 38 68	64 50 40	17 40	72 20
10 38 70	65 48 30	18 20	73 10
11 35 66	66 48 40	19 20	74 10
12 35 69	67 47 35	20 10	75 50
13 25 85	68 47 40	21 10	76 20
14 22 75	69 45 30	22 20	77 10
15 22 85	70 45 35	23 20	78 10
16 20 80	71 95 30	24 10	79 20
17 20 85	72 95 35	25 10	80 10
18 18 75	73 53 30	26 40	81 10
19 15 75	74 92 30	27 10	82 30
20 15 80	75 53 35	28 10	83 20
21 30 50	76 45 65	29 20	84 10
22 30 52	77 90 35	30 10	85 20
23 28 52	78 88 30	31 10	86 30
24 28 55	79 88 35	32 20	87 10
25 25 50	80 87 30	33 30	88 20
26 25 52	81 85 25	34 40	89 30
27 25 55	82 85 35	35 20	90 10
28 23 52	83 75 55	36 10	91 10
29 23 55	84 72 55	37 10	92 10
30 20 50	85 70 58	38 20	93 20
31 20 55	86 68 60	39 30	94 40
32 10 35	87 66 55	40 20	95 10
33 10 40	88 65 55	41 10	96 30
34 8 40	89 65 60	42 10	97 10
35 8 45	90 63 58	43 20	98 30
36 5 35	91 60 55	44 10	99 20
37 5 45	92 60 60	45 10	100 10
38 2 40	93 67 85	46 10	101 20
39 0 40	94 65 85	47 30	DEPOT_SECTION
40 0 45	95 65 82	48 10	1
41 35 30	96 62 80	49 10	-1
42 35 32	97 60 80	50 10	EOF
43 33 32	98 60 85	51 10	

APPENDIX C0 – CVRP INSTANCE D_{N0} -M-N101-K10

NAME : DN0-M-	7 40 69	25 25 50	15 10
n101-k10	8 40 66	26 25 52	16 40
COMMENT : (naive	9 38 68	DEMAND_SEC	17 40
decomposition of M-	10 38 70	TION	18 20
n101-k10)	11 35 66	1 0	19 20
TYPE : CVRP	12 35 69	2 10	20 10
DIMENSION : 26	13 25 85	3 30	21 10
EDGE_WEIGHT_TY	14 22 75	4 10	22 20
PE : EUC_2D	15 22 85	5 10	23 20
CAPACITY : 200	16 20 80	6 10	24 10
NODE_COORD_SEC	17 20 85	7 20	25 10
TION	18 18 75	8 20	26 40
1 40 50	19 15 75	9 20	DEPOT_SECTION
2 45 68	20 15 80	10 10	1
3 45 70	21 30 50	11 10	-1
4 42 66	22 30 52	12 10	EOF
5 42 68	23 28 52	13 20	
6 42 65	24 28 55	14 30	

APPENDIX C1 – CVRP INSTANCE D_{N1} -M-N101-K10

NAME : DN1-M-	32 10 35	50 28 35	41 10
n101-k10	33 10 40	51 26 32	42 10
COMMENT : (naive	34 8 40	DEMAND_SECTION	43 20
decomposition of M-	35 8 45	1 0	44 10
n101-k10)	36 5 35	27 10	45 10
TYPE : CVRP	37 5 45	28 10	46 10
DIMENSION : 26	38 2 40	29 20	47 30
EDGE_WEIGHT_TY	39 0 40	30 10	48 10
PE : EUC_2D	40 0 45	31 10	49 10
CAPACITY : 200	41 35 30	32 20	50 10
NODE_COORD_SEC	42 35 32	33 30	51 10
TION	43 33 32	34 40	DEPOT_SECTION
1 40 50	44 33 35	35 20	1
27 25 55	45 32 30	36 10	-1
28 23 52	46 30 30	37 10	EOF
29 23 55	47 30 32	38 20	
30 20 50	48 30 35	39 30	
31 20 55	49 28 30	40 20	

APPENDIX C2 – CVRP INSTANCE D_{N2} -M-N101-K10

NAME : DN2-M-	57 40 5	75 53 35	65 10
n101-k10	58 40 15	76 45 65	66 10
COMMENT : (naive	59 38 5	DEMAND_SECTIO	67 10
decomposition of M-	60 38 15	N	68 10
n101-k10)	61 35 5	1 0	69 10
TYPE : CVRP	62 50 30	52 10	70 10
DIMENSION : 26	63 50 35	53 10	71 30
EDGE_WEIGHT_TY	64 50 40	54 20	72 20
PE : EUC_2D	65 48 30	55 40	73 10
CAPACITY : 200	66 48 40	56 10	74 10
NODE_COORD_SEC	67 47 35	57 30	75 50
TION	68 47 40	58 40	76 20
1 40 50	69 45 30	59 30	DEPOT_SECTION
52 25 30	70 45 35	60 10	1
53 25 35	71 95 30	61 20	-1
54 44 5	72 95 35	62 10	EOF
55 42 10	73 53 30	63 20	
56 42 15	74 92 30	64 50	

APPENDIX C3 – CVRP INSTANCE D_{N3} -M-N101-K10

NAME : DN3-M-	82 85 35	100 55 80	90 10
n101-k10	83 75 55	101 55 85	91 10
COMMENT : (naive	84 72 55	DEMAND_SECTIO	92 10
decomposition of M-	85 70 58	N	93 20
n101-k10)	86 68 60	1 0	94 40
TYPE : CVRP	87 66 55	77 10	95 10
DIMENSION : 26	88 65 55	78 10	96 30
EDGE_WEIGHT_TY	89 65 60	79 20	97 10
PE : EUC_2D	90 63 58	80 10	98 30
CAPACITY : 200	91 60 55	81 10	99 20
NODE_COORD_SEC	92 60 60	82 30	100 10
TION	93 67 85	83 20	101 20
1 40 50	94 65 85	84 10	DEPOT_SECTION
77 90 35	95 65 82	85 20	1
78 88 30	96 62 80	86 30	-1
79 88 35	97 60 80	87 10	EOF
80 87 30	98 60 85	88 20	
81 85 25	99 58 75	89 30	

APPENDIX D0 – CVRP INSTANCE D_{c0}-M-N101-K10

NAME : DC0-M-	63 50 35	64 50
n101-k10	64 50 40	65 10
COMMENT :	65 48 30	66 10
(Decomposition of	66 48 40	67 10
M-n101-k10)	67 47 35	68 10
TYPE : CVRP	68 47 40	69 10
DIMENSION : 12	69 45 30	70 10
EDGE_WEIGHT_T	70 45 35	73 10
YPE : EUC_2D	73 53 30	75 50
CAPACITY : 200	75 53 35	DEPOT_SECTION
NODE_COORD_SE	DEMAND_SECTION	1
CTION	1 0	-1
1 40 50	62 10	EOF
62 50 30	63 20	

APPENDIX D1 – CVRP INSTANCE D_{c1}-M-N101-K10

NAME : DC1-M-	4 42 66	5 10
n101-k10	5 42 68	6 10
COMMENT :	6 42 65	7 20
(Decomposition of	7 40 69	8 20
M-n101-k10)	8 40 66	9 20
TYPE : CVRP	9 38 68	10 10
DIMENSION : 13	10 38 70	11 10
EDGE_WEIGHT_T	11 35 66	12 10
YPE : EUC_2D	12 35 69	76 20
CAPACITY : 200	76 45 65	DEPOT_SECTION
NODE_COORD_SE	DEMAND_SECTION	1
CTION	1 0	-1
1 40 50	2 10	EOF
2 45 68	3 30	
3 45 70	4 10	

APPENDIX D2 – CVRP INSTANCE D_{c2}-M-N101-K10

NAME : DC2-M-	32 10 35	34 40
n101-k10	33 10 40	35 20
COMMENT :	34 8 40	36 10
(Decomposition of	35 8 45	37 10
M-n101-k10)	36 5 35	38 20
TYPE : CVRP	37 5 45	39 30
DIMENSION : 10	38 2 40	40 20
EDGE_WEIGHT_T	39 0 40	DEPOT_SECTION
YPE : EUC_2D	40 0 45	1
CAPACITY : 200	DEMAND_SECTION	-1
NODE_COORD_SE	1 0	EOF
CTION	32 20	
1 40 50	33 30	

APPENDIX D3 – CVRP INSTANCE D_{C3}-M-N101-K10

NAME : DC3-M-	43 33 32	43 20
n101-k10	44 33 35	44 10
COMMENT :	45 32 30	45 10
(Decomposition of	46 30 30	46 10
M-n101-k10)	47 30 32	47 30
TYPE : CVRP	48 30 35	48 10
DIMENSION : 14	49 28 30	49 10
EDGE_WEIGHT_T	50 28 35	50 10
YPE : EUC_2D	51 26 32	51 10
CAPACITY : 200	52 25 30	52 10
NODE_COORD_S	53 25 35	53 10
SECTION	DEMAND_SECTION	DEPOT_SECTION
1 40 50	1 0	1
41 35 30	41 10	-1
42 35 32	42 10	EOF

APPENDIX D4 – CVRP INSTANCE D_{C4}-M-N101-K10

NAME : DC4-M-	83 75 55	84 10
n101-k10	84 72 55	85 20
COMMENT :	85 70 58	86 30
(Decomposition of	86 68 60	87 10
M-n101-k10)	87 66 55	88 20
TYPE : CVRP	88 65 55	89 30
DIMENSION : 11	89 65 60	90 10
EDGE_WEIGHT_T	90 63 58	91 10
YPE : EUC_2D	91 60 55	92 10
CAPACITY : 200	92 60 60	DEPOT_SECTION
NODE_COORD_S	DEMAND_SECTION	1
SECTION	1 0	-1
1 40 50	83 20	EOF

APPENDIX D5 – CVRP INSTANCE D_{C5}-M-N101-K10

NAME : DC5-M-	1 40 50	55 40
n101-k10	54 44 5	56 10
COMMENT :	55 42 10	57 30
(Decomposition of	56 42 15	58 40
M-n101-k10)	57 40 5	59 30
TYPE : CVRP	58 40 15	60 10
DIMENSION : 9	59 38 5	61 20
EDGE_WEIGHT_T	60 38 15	DEPOT_SECTION
YPE : EUC_2D	61 35 5	1
CAPACITY : 200	DEMAND_SECTION	-1
NODE_COORD_S	1 0	EOF
SECTION	54 20	

APPENDIX D6 – CVRP INSTANCE D_{c6}-M-N101-K10

NAME : DC6-M-	22 30 52	23 20
n101-k10	23 28 52	24 10
COMMENT :	24 28 55	25 10
(Decomposition of	25 25 50	26 40
M-n101-k10)	26 25 52	27 10
TYPE : CVRP	27 25 55	28 10
DIMENSION : 12	28 23 52	29 20
EDGE_WEIGHT_T	29 23 55	30 10
YPE : EUC_2D	30 20 50	31 10
CAPACITY : 200	31 20 55	DEPOT_SECTION
NODE_COORD_S	DEMAND_SECTION	1
ECTION	1 0	-1
1 40 50	21 10	EOF
21 30 50	22 20	

APPENDIX D7 – CVRP INSTANCE D_{c7}-M-N101-K10

NAME : DC7-M-	71 95 30	74 10
n101-k10	72 95 35	77 10
COMMENT :	74 92 30	78 10
(Decomposition of	77 90 35	79 20
M-n101-k10)	78 88 30	80 10
TYPE : CVRP	79 88 35	81 10
DIMENSION : 10	80 87 30	82 30
EDGE_WEIGHT_T	81 85 25	DEPOT_SECTION
YPE : EUC_2D	82 85 35	1
CAPACITY : 200	DEMAND_SECTION	-1
NODE_COORD_S	1 0	EOF
ECTION	71 30	
1 40 50	72 20	

APPENDIX D8 – CVRP INSTANCE D_{c8}-M-N101-K10

NAME : DC8-M-	93 67 85	95 10
n101-k10	94 65 85	96 30
COMMENT :	95 65 82	97 10
(Decomposition of	96 62 80	98 30
M-n101-k10)	97 60 80	99 20
TYPE : CVRP	98 60 85	100 10
DIMENSION : 10	99 58 75	101 20
EDGE_WEIGHT_T	100 55 80	DEPOT_SECTION
YPE : EUC_2D	101 55 85	1
CAPACITY : 200	DEMAND_SECTION	-1
NODE_COORD_S	1 0	EOF
ECTION	93 20	
1 40 50	94 40	

APPENDIX D9 – CVRP INSTANCE D_{c9}-M-N101-K10

NAME : DC9-M-	1 40 50	14 30
n101-k10	13 25 85	15 10
COMMENT :	14 22 75	16 40
(Decomposition of	15 22 85	17 40
M-n101-k10)	16 20 80	18 20
TYPE : CVRP	17 20 85	19 20
DIMENSION : 9	18 18 75	20 10
EDGE_WEIGHT_T	19 15 75	DEPOT_SECTION
YPE : EUC_2D	20 15 80	1
CAPACITY : 200	DEMAND_SECTION	-1
NODE_COORD_S	1 0	EOF
ECTION	13 20	