UNIVERSITY OF
NEWCASTLE

MULTIPLE CRITERIA GENETIC ALGORITHMS IN ENGINEERING DESIGN AND OPERATION

A Thesis Submitted for the Degree of

Doctor of Philosophy

October 1997

David Todd

Engineering Design Centre

Supervisor : Prof. Pratyush Sen

Department of Marine Technology

Abstract

This thesis investigates the application of Genetic Algorithms (GAs) to multiple criteria problems in engineering design and operation. The GA is an evolutionary computing technique which applies Darwinian principles such as survival of the fittest, mating and mutation to a population of individuals to evolve good solutions to a broad range of problems. GAs are normally used as single criterion optimisers. However, a Multiple Criteria Genetic Algorithm (MCGA) has been developed in this thesis which allows simultaneous maximisation and minimisation across several criteria.

The MCGA is explained, enhanced and modified throughout the thesis as new requirements are introduced. At each stage the enhancements are tested on basic test functions to assess their performance. New concepts such as a Pareto Population, Adaptive Niche Sizing and Neural Network Preferencing are introduced.

A broad range of applications have been tackled using the MCGA, all of which are combinatorial in nature. Combinatorial problems are primarily concerned with ordering or positioning of elements, the order determining the properties and attributes of the solution. As such, standard GA operators cannot be used and novel methods have to be employed.

The applications are drawn from various engineering fields. An electronic component placement problem is tackled initially with the goal of decreasing processing time as a single criterion combinatorial example. A general method of rearranging design activities so as to maximise parallelisation and reduce lead times is then performed using the MCGA. A complex scheduling model is devised and the MCGA is used to optimise various operating scenarios. Finally, the arrangement of containers in a containership is tackled.

Acknowledgements

I would like to thank the following people for their help over the three years of this work:

Prof. P.Sen of the Department of Marine Technology for his invaluable help and guidance over the last three years. I'd also like to thank him for working overtime in the weeks before submission so I could submit everything on time.

My Father and Mother for all the support, financial and otherwise which has allowed me to enjoy student life for so long and also for carrying out the enormously boring task of proof reading my thesis. My sister Sue and Phil, "The Reading Connection".

Nichola Bishop for being with me the last two and a half years and helping me enjoy myself and take my mind off things when not working.

Keith Hutchinson for supplying coffee and musical entertainment at work. Thanks also to the rest of my friends (too many to name) in the Engineering Design Centre and the Department of Marine Technology for creating a good working environment.

Finally I would like to thank the EPSRC for funding this research and also Prof. W.Hills of the EPSRC Engineering Design Centre at Newcastle.

Contents

Abstract **i**

Acknowledgements **iii**

Chapter 1 - Introduction **1**

1.1 Computing in Engineering 1

1.2 Genetic Algorithms 2

1.2.1 History 4

1.2.2 Parametric Problems 5

1.2.3 Combinatorial Problems 6

1.3 Multiple Criteria Decision Making in Engineering 6

1.4 Thesis Outline 7

Chapter 2 - Genetic Algorithm: Mechanics and Theory **10**

2.1 Introduction 10

2.2 The Mechanics of The Simple Genetic Algorithm 10

2.3 A GA Used to Solve a Mathematical Problem 14

2.4 Genetic Algorithm Theory 16

2.4.1 The Schema Theorem 17

2.4.2 The Building Block Hypothesis 19

2.5 Binary Versus Real Encoding 20

2.5.1 Generation Average 21

2.5.2 Best So Far 24

2.5.3 Obtaining Binary GA Performance 28

2.5.4 Discussion 30

2.6 Chapter Summary 31

Chapter 3 - Genetic Algorithms and the Travelling Salesman Problem **32**

3.1 The Travelling Salesman Problem 32

3.1.1 O-Schema 32

3.1.2 GAs and the TSP 33

3.2 Crossover Operators 33
3.2.1 One Point 33
3.2.2 Two Point End 34
3.2.3 Two Point Centre 34
3.2.4 Two Point/End Centre 34
3.2.5 Position 35
3.2.6 Independent Position 35
3.2.7 Partially Mapped 35
3.2.8 Ordered 36
3.2.9 Cycle 36
3.2.10 Edge Recombination 37
3.2.11 Enhanced Edge Recombination 38
3.2.12 Subtour Chunks 38
3.2.13 Alternating Edges 39
3.2.14 Inversion 39
3.3 Mutation Operators 39
3.3.1 Two-City Adjacent Swap 39
3.3.2 Two-City Random Swap 40
3.3.3 Three-City Adjacent Swap 40
3.3.4 Three-City Random Swap 40
3.3.5 Shift 40
3.4 Performance of Operators 41
3.4.1 Test Problems 41
3.4.2 Results 41
3.5 GA Local Search and Heuristics for the TSP 46
3.6 Electronic Component Placement 48
3.7 Chapter Summary 53

Chapter 4 - The Multiple Criteria Genetic Algorithm (MCGA) 54

4.1 Introduction 54
4.2 Multiple Criteria Concepts 54
4.3 Genetic Algorithms for Multiple Criteria Problems 59
4.3.1 Estimating Niche Sizes 63
4.3.2 A New Approach To Niche Sizing 65
4.4 Initial Implementation 67
4.5 Test Functions 68

4.6 The Rate of Generation of Rank 1 Solutions 71
4.7 The Pareto Population and Adaptive Niche Sizing 72
4.8 The Multiple Criteria Genetic Algorithm (MCGA) 73
4.9 Test Function Performance 77
4.10 Population Sizing and Niches 87
4.11 Run Time and Convergence Rate Measurements 89
4.12 Chapter Summary 92

Chapter 5 - Multiple Criteria Genetic Algorithm for Process Analysis 93

5.1 Concurrent Engineering 93
5.2 System Representation 93
5.3 The Design Structure Matrix 95
5.4 GA For Partitioning 97
5.5 Test Problems, Validation and Results 99
5.5.1 Problem 1 - 12 Activity (KUSIAK 91) 99
5.5.2 Problem 2 - 20 Activity (STEWARD 81) 101
5.5.3 Problem 3 - 51 Activity (AUSTIN 96) 104
5.6 MCGA For Partitioning 106
5.7 Chapter Summary 111

Chapter 6 - MCGA for Scheduling in a Manufacturing Environment 112

6.1 Scheduling Using Genetic Algorithms 112
6.1.1 Job-shop Scheduling Problem Definition 113
6.1.2 Representation / Encoding 114
6.2 Complex Scheduling 116
6.2.1 Scheduling for Made To Order Products 116
6.2.2 Operation Based Scheduling 118
6.2.3 Multiple Criteria Scheduling 118
6.3 Construction of the MCGA Scheduler 119
6.3.1 Data Representation 119
6.3.2 String Configuration 123
6.3.3 Structured GAs 123

6.3.4 Selective Gene Activation 124
6.3.5 String Construction using Selective Gene Activation 125
6.3.6 Crossover and Mutation Operators 128
6.3.7 Schedule Builder 130
6.4 Scheduler Validation and Testing 131
6.4.1 A Complex Scheduler Example 132
6.4.2 Scheduling for Robustness 134
6.4.3 Shipyard Scheduling With Shared Resources 136
6.4.4 Process Analysis of the Shipyard Problem 138
6.5 Chapter Summary 141

Chapter 7 - Diversity and Convergence 142

7.1 Introduction 142
7.2 Utilising the Pareto Population 142
7.3 Diversification 142
7.3.1 Pareto Injection 143
7.3.2 Niche Counts 143
7.3.3 Adjacent Distance 144
7.3.4 Testing of Divergence Methods 146
7.4 Convergence 150
7.4.1 Target Values 151
7.4.2 Target Ratio 152
7.4.3 Testing of Convergence Methods 152
7.5 Interactive : Neural Network Preference 156
7.5.1 Neural Networks 156
7.5.2 Back Propagation Neural Networks 158
7.5.3 Integration of MCGA and Neural Network 160
7.5.4 Neural Network Testing 162
7.6 Chapter Summary 169

Chapter 8 - MCGA for Containership Loading 170

8.1 Introduction 170
8.2 Containership Loading 170
8.3 Containership Terminology 171
8.4 The Containership Loading Problem 173
8.5 Exploratory Test Problem 175

8.6 Full Size Containership with Validation 178

8.7 Heuristic Operators 180

8.7.1 Crossover 180

8.7.2 Mutation 180

8.8 Results 181

8.9 Combined Elitist and Random Injection 186

8.10 GA, Heuristic and GA+Heuristic 192

8.11 Container 'Clean Up' Routine 195

8.12 Chapter Summary 196

Chapter 9 - Multi-Attribute Decision Making for Solution Selection 197

9.1 Introduction 197

9.2 Reducing The Decision Set 197

9.3 MADM Methods 199

9.3.1 UTA Method 200

9.3.2 TOPSIS 202

9.3.3 CODASID 205

9.4 Discussion 209

9.5 Chapter Summary 210

Chapter 10 - Conclusions and Further Work 211

10.1 Conclusions 211

10.2 Further Work 216

Appendices 219

Appendix A - GA Terminology 219

Appendix B - TSP Problem Graphs 225

Appendix C - Code Overview / User Guide 235

Appendix D - Scheduler Information 240

Appendix E - Containership Information 256

References 266

1 Introduction

1.1 Computing in Engineering

During the past decade the dramatic increase in computing power and available software tools, coupled with a correspondingly dramatic decrease in the cost of computer hardware, has seen the use of computers grow enormously. Engineering is one area of computer application that has benefited profoundly from this technology in recent years.

Computer tools can aid the engineering process in several, and often interacting, ways:

i) Design - Computers can help in the design decision making process by using captured knowledge to provide the engineer with guidance.

ii) Visualisation - Use of CAD systems and more recently Virtual Reality allows systems to be visualised and even walk-throughs to be conducted on large structures.

iii) Testing - The use of mathematical models can assist in testing designs. Instead of having to build a product initial tests can be carried out on a computer. For example component stresses can be examined via Finite Element Analysis (FEA) tools or ship hull forms can be analysed using Computational Fluid Dynamics (CFD).

iv) Operation - Through the use of simulation it is possible to model the behaviour of interacting elements of a process. Manufacturing processes can be simulated to identify bottlenecks or the control parameters in a chemical plant model can be adjusted to examine performance.

v) Optimization - It is important to get the best performance out of any engineering system. Hence computer based optimization can be used to improve many aspects of a design or performance of a system.

The last of these, optimization, is of prime concern as it is applicable across the whole engineering spectrum. The aim of optimization is to attain the best solution possible within a problems' specified limits and constraints. For example in design this may be to minimise weight, stresses and drag in the case of an aeroplane wing. In a factory the goal may be to maximise efficiency and minimise waste. This process, when performed by a designer, will involve the application of a great deal of experience, knowledge and an ability to weigh up potentially large numbers of possibilities. This process becomes harder and often intractable as the number of decisions required and the system or product complexity increases.

Conventional mathematical methods for optimization, such as differentiation, hill climbing, or algorithmic methods such as linear programming, have long been established for solving mathematically based problems. However many of these approaches lack robustness and cannot easily cope with multimodal, discrete, noisy and multiple criteria problems which are common in engineering systems. Since the late sixties a new problem solving technique has gradually emerged. This technique is called the Genetic Algorithm (GA), and the underlying principles of GAs can cope with many of the unhelpful properties outlined above.

1.2 Genetic Algorithms

GAs are one of a group of techniques known as Evolutionary Algorithms which also incorporates Genetic Programming, Evolution Strategies, Evolutionary Programming and Classifier Systems. Such techniques differ from conventional techniques in several ways (NISSEN 95):

i) They use a string or vector to represent the decision variables.

ii) They operate on a set of solutions, thus exploring several points simultaneously.

iii) They use operators that mimic some of the processes of natural evolution.

iv) The only information required for search is a quality or fitness assessment.

v) Stochastic elements are employed.

GAs were first developed in the sixties and seventies by Holland (HOLLAND 62, HOLLAND 75). They use a population of individuals called strings, each of which is a possible solution to the problem. Each string is in turn made up of a number of genes which define its properties. The analogy between natural genetics and GA terms, often used interchangeably, is shown in Table 1.1 (SOUTH 93).

Table 1.1 - GA Terminology

Natural Genetics	Genetic Algorithms
Chromosome	String
Gene	Feature or bit
Allele	Feature or bit value
Locus	String position
Genotype	Structure of an individual in the GA population
Phenotype	Decoded individual giving a possible solution to the problem

The Genetic Algorithm is best illustrated by the simple flow diagram shown in Figure 1.2. The population of strings is randomly initialised giving a diverse range of possible solutions. Each of these solutions is evaluated and given a fitness score. At this point the population is examined to see if a suitable solution has been found or improvement has slowed to such an extent that is not worth searching further. This could be when a given goal has been reached or a certain level of improvement has not been achieved over a fixed number of generations. If the stopping criteria has not been reached the GA enters a loop involving three stages. The first stage is to select a new population based on fitness. This is, in Darwinian terms, performing a 'survival of the fittest' operation on the population. The selected population, which is usually the same size as the initial population, then forms the basis of a mating pool and enters the second stage of the loop. In this stage, two genetic operators are applied to the mating pool - crossover and mutation. Crossover randomly selects pairs of strings called parents and combines their properties into two offspring strings. This is done by choosing crossing points and swapping the genetic material between these points. The principle behind this is that good genetic material from each parent will thus be combined in a favourable manner in the offspring. Mutation makes random changes to individual genes in order that genetic material is not completely lost and to introduce new material to encourage diversity. This operator aids in preventing the population converging into local maxima or minima. It should be noted that both crossover and mutation are applied with associated probabilities Pc and Pm respectively, so that not all pairs will be crossed and not all strings will be mutated. The final stage of the loop re-evaluates the evolved population and then the loop returns to the start where the stopping criteria are examined. If they are not met the population re-enters the loop, otherwise, the GA exits and the best solution found from the search is chosen.

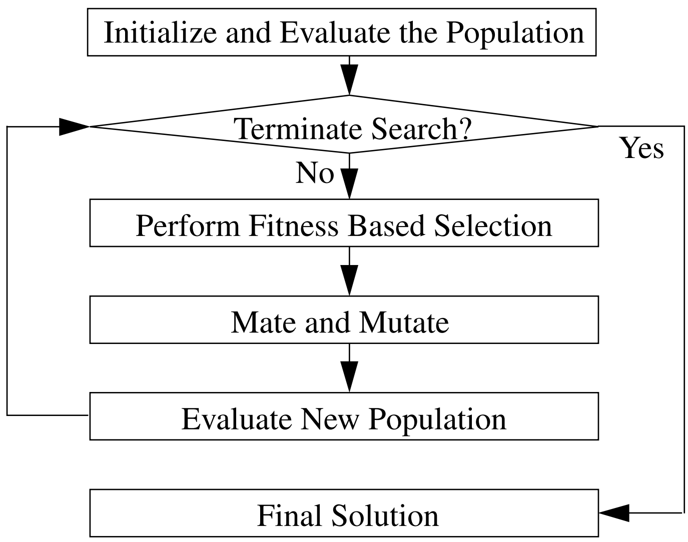


Figure 1.2 - The Simple Genetic Algorithm

1.2.1 History

Holland's book (HOLLAND 75) and De Jong's doctoral dissertation (DE JONG 75) provided the basis for much of the research in the area of GAs. Holland laid down the groundwork for his 'reproductive plans' and De Jong performed an investigation of the performance of these plans over a set of five test problems with good results. However, it was not until the early eighties that interest in GAs began to increase in earnest. In 1985 the first International Conference in Genetic Algorithms and their Applications was held in Carnegie-Mellon University in Pittsburg (GREFFENSTETTE 85) and has been convened every two years since then. Further interest was also generated by Golberg's book (GOLDBERG 89) which has become a standard text for GAs.

GAs have been applied in many areas of engineering, with applications in automotive (FISHER 95), mechanical (ROY 96), materials (GOODMAN 96), electronics (CHU 89), chemical (UNGER 93), structures (RICHE 95), aeronautical (POLONI 96), marine (SUBRAMANI 96), production (DAVIS 85), control (HUSBANDS 96) and computing (CRAMER 85) being just a few examples.

The applications of GAs can be broadly split into two categories: parametric and combinatorial. It is also possible for a given problem to involve both types in a mixed form. This thesis mainly addresses the combinatorial type problems.

1.2.2 Parametric Problems:

These problems involve the optimisation of variables or parameters. Each of these parameters are represented as a sub set of genes in a string. In their simplest form these include optimisation of mathematical functions. e.g

Minimise:

$$f(x) = x_1^2 + x_2^2 - 6x_1 - 8x_2 + 10$$

Subject to:

$$4x_1^2 + x_2^2 \leq 16$$
$$3x_1 + 5x_2 \leq 15$$

In this case the variables are x_1 and x_2, and the fitness function is $f(x)$. However, by defining a suitable representation more complex problems can be tackled. For example Keane (KEANE 96) designs a reduced vibration satellite boom in the form of a jointed frame structure by defining datums for the initial position of the joints and then uses a GA to vary the position of all the joints from this datum.

1.2.3 Combinatorial Problems:

Combinatorial problems depend on the ordering of genes in a string. Most commonly, one gene corresponds to one item that needs to be ordered. This can either be the relative ordering, i.e. addressing what comes next to what, or absolute ordering, i.e. addressing the actual position in the string that the gene occupies. A common example is the Travelling Salesman Problem (TSP). This involves a salesman travelling between a fixed set of cities and returning to the start city covering the least amount of distance. Another practical example of this would be the ordering of jobs in a manufacturing environment. Again vastly more complex problems can be tackled with a suitable representation.

1.3 Multiple Criteria Decision Making in Engineering

Many engineering problems, parametric or combinatorial, can be assessed in terms of a single measure or criterion. These are ideally suited to the use of GAs which rely on a single fitness measure. Unfortunately there are many other cases where the suitability of designs cannot be expressed in terms of a simple single value. In such cases the problems are defined as multiple criteria. Each criterion may have a different measure and may also have complex interactions with the other criteria. There is no longer a single fitness measure indicating the ‘goodness’ of a solution. One way of dealing with this class of problems is to use a weighted sum of the normalised criteria. However, this approach is not always satisfactory. The weights reflect a set of fixed priorities, and may be generated by an individual or derived from the preference structures of several people (HUANG 81) . Performing optimisation on the basis of such weights will give a single solution or solutions in compliance with these strict weights, but it will give no indications of potential local trade-offs or indications of the form of the global landscape. Ideally we would like to obtain a set of good solutions each of which has its own blend of criteria. This would give an opportunity to discuss the possible trade-offs which exist and decide upon a proper compromise solution. For example, in attempting to minimise weight and minimise cost in an aircraft design a trade-off set of possible designs needs to be generated. It can be seen from Figure 1.3 that the objective space is split into feasible and infeasible regions. In general, however, this split will not be as simple and clearly defined as the one shown here. These regions are defined by the physical limits and constraints of the problem being tackled. The trade-off set is formed from the solutions on the limits of the feasible region. This set of solutions is known as the efficient or Pareto set (STEWART 92). Pareto Optimality is named after Vilfredo Pareto (PARETO 1896) who recornised that multiple criteria solutions could be partially ordered without making any preference choices. Pareto Optimality is defined simply as:

A solution is termed Pareto Optimal if it is dominated by no other solution.

This means that if no other solution has all criteria superior or equal to the given solution then it belongs to the Pareto set. Hence, we can define good solutions in multiple criteria terms as ones which are non-dominated.

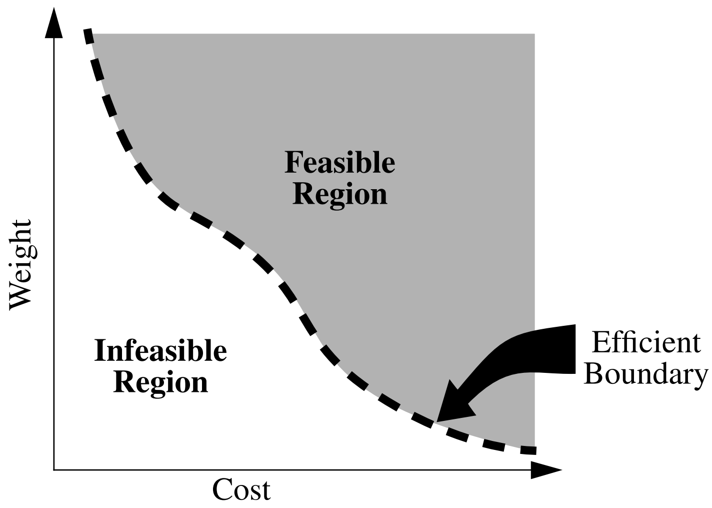


Figure 1.3 - Weight versus Cost

It is the investigation of this principle in the development of a general purpose Multiple Criteria Genetic Algorithm and the investigation of its use on complex engineering applications which is the aim of the work described in this thesis.

1.4 Thesis Outline

This thesis is arranged into 10 chapters. The organisation of the work differs from the norm in that instead of tackling a single large problem it addresses several problems from a broad range of application areas. These areas are electronics, concurrent engineering, scheduling and planning and cargo arrangement/packing. However there is a common theme throughout. Firstly all of the applications tackled are combinatorial in nature and secondly they are all solved by a specially developed Multiple Criteria Genetic Algorithm (MCGA). The MCGA is developed, enhanced and modified throughout the thesis as new requirements are introduced. At each stage the enhancements are tested on basic test functions to assess their performance and then set to work on a real application. All code used within this thesis was written specifically for the work by the author. This aided understanding and allowed greater experimentation with the GA and models.

Chapter 2 provides introduction to Genetic Algorithms, outlining the basic mechanics of the simple GA and important concepts such as Schemata Theory and the Building Block Hypothesis. It also includes a brief investigation of two encodings used within GAs.

Chapter 3 looks at how a simple genetic algorithm can be used to tackle perhaps the most well known combinatorial problem - the Travelling Salesman Problem (TSP). A comparison of the genetic operators available for this problem is carried out on a five function test bed. The use of local search and heuristics is also examined. This chapter presents an electronic component placement problem analogous to the TSP.

Following this the construction of the Multiple Criteria Genetic Algorithm (MCGA) is explained. Chapter 4 outlines the underlying principles of multiple criteria problems and summarises previous work carried out involving GAs and multiple criteria techniques. A MCGA model is defined and a second population called the Pareto Population is introduced. Some important aspects concerned with niche sizing in a multi-criteria context are also discussed using a hyper-cuboid approach. Finally, the MCGA is then demonstrated on 2,3,4 and 5 criteria problems and the results examined.

Concurrent Engineering principles are increasingly being applied to decrease lead times and increase efficiency in many industries. Chapter 5 looks at the principles involved and methods used to optimise a process model specified in terms of a matrix. These methods are compared with a GA optimisation. The MCGA is then used to solve a multiple-criteria version of the problem.

In Chapter 6 the simple Job Shop Problem (JSP) is extended to increase applicability. Using this new model the MCGA is used to tackle several complex scheduling problems examining the concept of multiple criteria scheduling. A general purpose scheduling system is designed and implemented giving a flexible method of solving a broad range of scheduling problems. The system can handle several criteria simultaneously and problem complexity to a higher level than many conventional schedule solvers. The problems tackled are a complex JSP and a Shipyard Steel Shop with a shared resource. It is further demonstrated that this tool can be used to analyse the viability of system upgrades and design robust schedules.

Diversity and convergence, in terms of multiple criteria, are two conflicting and interacting aspects of using an MCGA for search. Either can be utilised depending on the knowledge and aims of the decision maker. This provides the motivation for the design of several techniques which modify search through selection and an interactive tool incorporating Neural Networks which approximates a model of the decision maker's preference. Chapter 7 illustrates how the Pareto Population can be used to encourage diversity or convergence on a set of test problems using these techniques.

In Chapter 8 the diversity and convergence tools are demonstrated on a large scale cargo layout problem. The MCGA is used to tackle the loading of containers onto a 2000 capacity containership. A containership modeller is created which calculates various criteria involved, performs unloading of the vessel and stability analysis. It also shows how the MCGA can provide information on the correlations between criteria.

Having used the MCGA to find sets of solutions for several problems in earlier chapters, Chapter 9 looks at methods of extracting the best solution from this trade-off set. Three methods are explained and discussed and demonstrated using the MCGA containership solutions.

Chapter 10 gathers the conclusions of this study and looks ahead to further work arising from this thesis.

2 Genetic Algorithm: Mechanics and Theory

2.1 Introduction

Having described the basic algorithm in Chapter 1 this chapter discusses how the Simple Genetic Algorithm operates in terms of the action of the selection, mutation and mating procedures at a string level. Following this, two theoretical explanations of the GA are presented, the Schema Theorem and the Building Block Hypothesis. Finally a comparison of a binary and a real coded GA is carried out to examine differences between their operation.

2.2 The Mechanics of The Simple Genetic Algorithm

A small problem, one of maximising the binary value of an 8 bit string is used to demonstrate the action of the GA. In the Simple GA, roulette wheel selection, single point crossover and standard mutation are used (GOLDBERG 89). Other common methods of representation, selection, crossover and mutation are summarised in Appendix A for general reference.

(i) INITIAL POPULATION (Generation 0)

This is the starting point for the GA. In this case a population size of 6 is chosen for simplicity. The population size is very much dependent on the problem being tackled and the type of GA being utilised (GOLDBERG 89b). Usually good measures of the progress of a search are the values of Average Fitness and Best So Far Fitness. When the average moves close to the best so far fitness the search is almost converged. Both fitness curves, when plotted on a graph, usually take the form of an exponential. This is demonstrated in Section 2.3.

Initial Population

0 1 0 0 1 0 0 0	72	STRING 1
1 0 1 1 1 0 1 0	186	STRING 2
0 0 0 0 0 1 0 1	5	STRING 3
0 1 0 0 0 0 0 1	65	STRING 4
1 1 0 1 0 1 1 0	214	STRING 5
0 0 0 1 1 0 0 0	24	STRING 6

Average Fitness = 94.3

Best Design Fitness = 214

(ii) SELECTION PROCEDURE

In this case Roulette Wheel Selection is used (see Figure 2.1). Each string is given a segment of the roulette wheel proportional to it's fitness. The wheel is then spun and the segment in which the pointer lands indicates the chosen string. This is repeated six times in order to generate a mating pool.

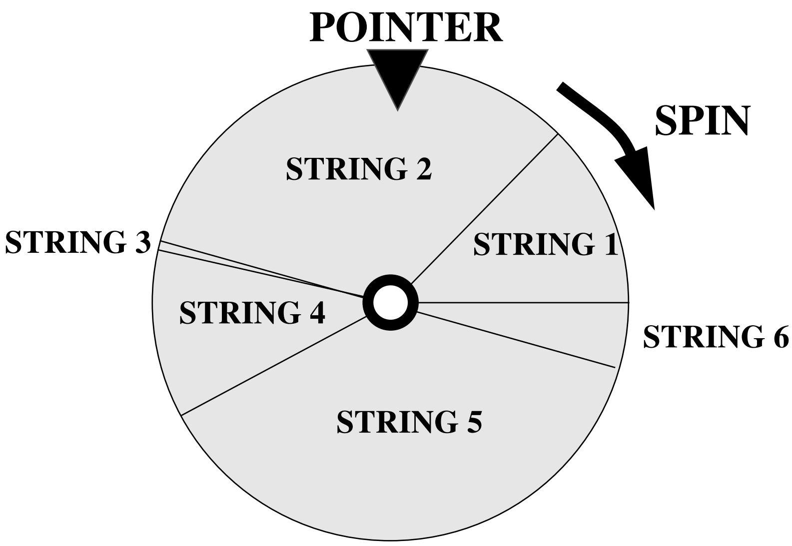


Figure 2.1 - Roulette Wheel Selection

(iii) SELECTED POPULATION

The selected population or mating pool are acted upon by the crossover and mutation operators. Those strings not chosen die out. If we repeated the selection procedure again on the same population we would end up with a different mixture of strings. However, it would still be more likely to include copies of string 5 rather than string 3.

Selected Population

1 1 0 1 0 1 1 0	214	STRING 1 - was STRING 5
1 0 1 1 1 0 1 0	186	STRING 2 - was STRING 2
0 0 0 1 1 0 0 0	24	STRING 3 - was STRING 6
1 1 0 1 0 1 1 0	214	STRING 4 - was STRING 5
0 1 0 0 0 0 0 1	65	STRING 5 - was STRING 4
1 1 0 1 0 1 1 0	214	STRING 6 - was STRING 5

(iv) CROSSOVER

The single point crossover operator randomly selects pairs of strings from the mating pool. It then decides whether to perform crossover on this pair with a likelihood of Pc, the probability of crossover (typically 0.6-0.9). If the pair is chosen for mating a random crossing point is selected and the tails of the two strings are swapped to generate two new permutations. Each string in the mating pool will only be paired up once.

Cross Strings 1 And 5 At Position 4

1 1 0 1 | 0 1 1 0	→	1 1 0 1 | 0 0 0 1	STRING 1
0 1 0 0 | 0 0 0 1		0 1 0 0 | 0 1 1 0	STRING 5

Cross Strings 3 And 4 At Position 6

0 0 0 1 1 0 | 0 0	→	0 0 0 1 1 0 | 1 0	STRING 3
1 1 0 1 0 1 | 1 0		1 1 0 1 0 1 | 0 0	STRING 4

Strings 2 And 6 Left Untouched

(v) MUTATION

Mutation is used to introduce small random changes into the population. Mutation acts on single bits with Probability Pm, typically 0.001-0.01, and will switch the current state of these bits.

Mutate String 2 Position 2

1 0 1 1 1 0 1 0 ⟶ 1 1 1 1 1 0 1 0 STRING 2

Mutate String 5 Position 7

0 1 0 0 0 1 1 0 ⟶ 0 1 0 0 0 1 0 0 STRING 5

Strings 1,3,4 And 6 Left Untouched

(vi) NEW POPULATION (Generation 1)

After the crossover and mutation stages we have generated a new set of strings. These strings can now be evaluated as before. If we look at the differences between Generation 0 and Generation 1 several things are clear. Firstly the average fitness of the population has risen from 94.3 to 159.3. Secondly the fitness of the best design has increased from 214 to 250. The GA has thus made large improvements in the quality of the population. However the rate of improvement usually follows an exponentially decreasing path. This is shown more clearly in Section 2.3.

New Population

1 1 0 1 0 0 0 1	186	STRING 1
1 1 1 1 1 0 1 0	250	STRING 2
0 0 0 1 1 0 1 0	26	STRING 3
1 1 0 1 0 1 0 0	212	STRING 4
0 1 0 0 0 1 0 0	68	STRING 5
1 1 0 1 0 1 1 0	214	STRING 6

Average Fitness = 159.3

Best Design Fitness = 250

(vii) RETURN TO (ii) OR EXIT.

At this point the terminating criteria for search are examined. If these criteria are not met we return to selection and repeat the process. However, if they are met the process is halted.

2.3 A GA Used to Solve a Mathematical Problem

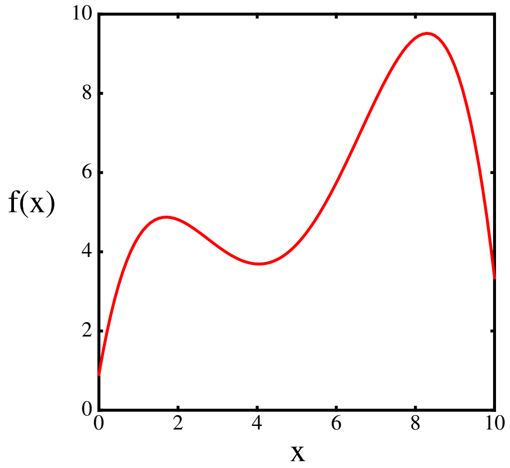


Figure 2.2 - Mathematical Test Function

This is a example of the GA used to evaluate a mathematical function. The function has one variable x and is defined as:

$$f(x) = 4.0 - ((x/1.6-0.5)*(x/1.6-3.0)*(x/1.6-6.2)*(x/1.6-2)/6.0)$$

The aim is to find a value of x over the range 0<X<10 such that f(x) is a maximum. From the plot of the function it can be seen that there are two maxima. If a hill climbing method is used, the search must be restricted to x>4 and x<10 in order to consistently find the true maxima. The GA starts from random starting points, and in this case the population size is chosen to be 20. This is a simple example to demonstrate the action of the GA. It is clear that if the hill climber was also started from 20 random points it would find the maxima in approximately 60% of the cases. However, on many problems the fitness landscape would be much more complex than in this case. This increases the difficulty of finding a starting point from which a global optima can be found using a hill climber.

The behaviour of the Population is shown in Figure 2.3. The three diagrams show the population migrating towards the maxima as the search progresses. In the initial population there is a random sprinkling of the population throughout the search space.

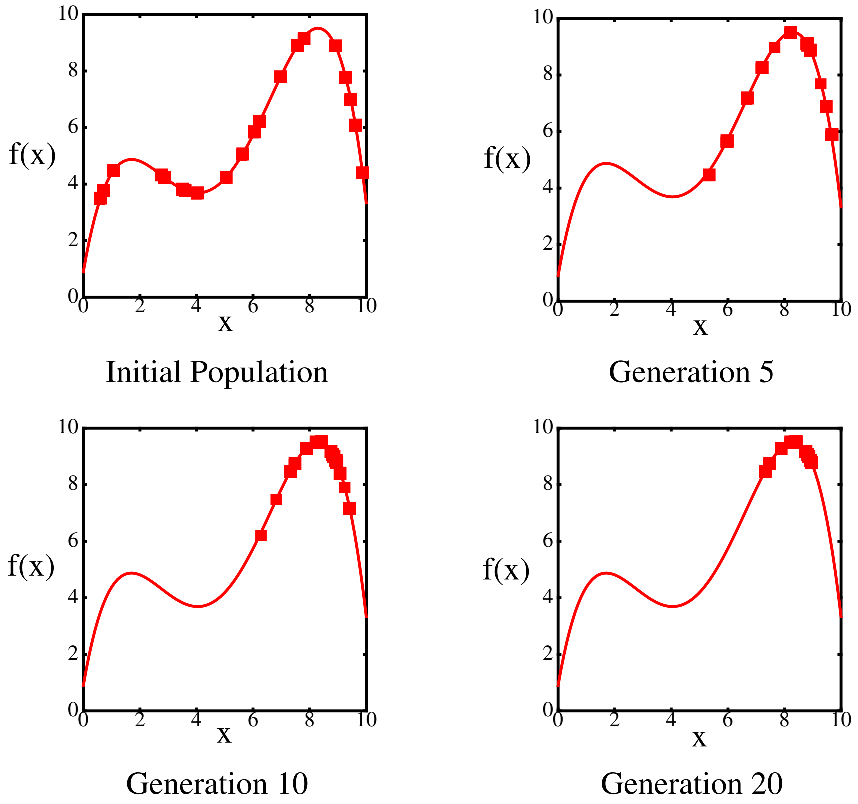


Figure 2.3 - Movement of the Population

By generation 5 all of the population has migrated to the higher hill through the selection, mating and mutation process and by generation 10 all members are clustered around the peak of the hill. By generation 20 most members were clustered on or very near the peak.

This experiment was repeated fifty times, in 36 (72%) of the fifty trials the GA was able to find the maximum value of f(x)=9.5155 at x=8.293 (or in binary 110101000101=3397 for a resolution of 4096 points from 0 to 10). For the other cases the results were within a couple of percent of the maximum. This demonstrates several important points. Firstly, a GA search is not guaranteed to give the optimum every time, and often the GA will only provide good solutions but, because of their nature, GAs can tackle problems that other solvers cannot. Secondly, the GA only needs to evaluate a small proportion of the population. In this case 20x20=400 evaluations were performed, 10% of the possible 4096 solutions. The fact that only a fraction of the search space needs to be evaluated means that problems with very large search spaces can be tackled without an exhaustive search. In many cases this percentage will be much lower than 10%, due to limits on length of run times and the sheer size of search spaces. This reduction in the total space evaluated can clearly also lead to small sacrifices in the quality of the final solution.

Over the fifty trials the average fitness of the population for each generation was also recorded. The results plotted in Figure 2.4 show the exponential improvement of the population over the 20 generations towards the maximum value of f(x). There are obviously members of the population scattered above and below the mean line but as the fitness approaches the maximum, the spread of individuals will be reduced as the population migrates towards the similar fitness values. GA search generally demonstrates these exponential properties and the rate of change of average fitness can be used to terminate search.

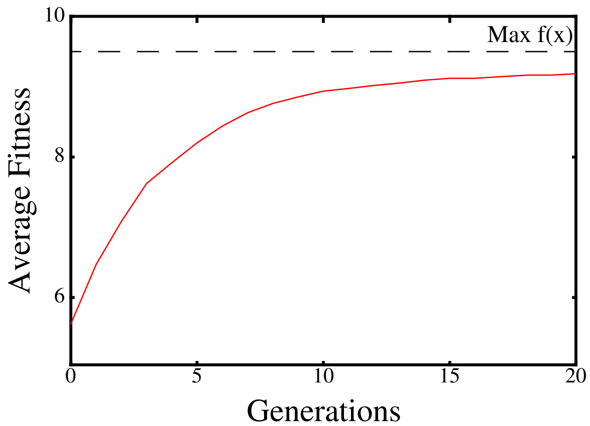


Figure 2.4 - Average Fitness Against Number of Generations

2.4 Genetic Algorithm Theory

As shown in Sections 2.2 and 2.3 Genetic Algorithms are a very straightforward means of optimization. It is simply necessary to generate a random initial population, perform selection of candidates for the next generation by biasing towards members with higher fitness, then swap substrings to form new strings and occasionally mutate. However the action of the GA and why it works is still an area of much research. Several theories have been described in the literature. The Schema Theorem and the closely linked Building Block Hypothesis are the most well known and are explained below.

2.4.1 The Schema Theorem

The Schema Theorem is used to explain the processing which takes place during each generation and to analyse the growth and decay of bit patterns known as schema (GOLDBERG 89, p28-33).

If we take the standard GA alphabet V = {0,1} and augment it with a don't care character we obtain V+ = {0,1,*}. This alphabet can be used to express any schema H.

e.g. Schema H = *11*0**. Examples of strings containing H: 1111010, 0110001, 1110001,...

A schema has two properties, order and defining length. The order is the number of fixed positions in a schema. For H = *11*0**, the order o(H) = 3. The defining length of a schema is the first and last specified bits. For H = *11*0**, the defining length d(H) = 5 - 2 = 3.

Schema Processing:

Schema provide a useful way of looking at how a GA works. By looking at what happens to a schema during selection, crossover and mutation it is possible to analyse the growth of fit schema and the decay of weak schema in a population of strings over several generations.

Selection:

At a given time t there are m examples of a given schema H within a population A(t). m is a function of H and t so we can write m = m(H,t). A string in the population A_i will get selected with probability $p_i = f_i / \sum f_j$. After selection of an entirely new population of size n it can be expected that m(H, t+1) members with schema H will be present. If the average fitness of all strings containing H is f(H), it is possible to write,

$$m(H, t+1) = m(H, t)\frac{nf(H)}{\sum f_j}$$

As the average fitness of the entire population is $\bar{f} = \sum f_j/n$, the schema growth equation becomes:

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}}$$

This equation essentially says that a particular schema grows at the ratio of the average fitness of strings containing that schema to the average fitness of the whole population. This behaviour is carried out with every schema H in the population A. The effect of this is that the number of schema of above average fitness grows and the number of schema of below average fitness decays.

Crossover:

To see how simple crossover can effect schema it is necessary to look at what kind of schema are effected by crossover. If we take a string and two of it's schema and a randomly selected crossing point 3 denoted by ^:

String A =	1 1 0 ^ 1 0 0 1
Schema H_1 =	* 1 * ^ * * * 1
Schema H_2 =	* * * ^ 1 0 * *

Unless string A's mate has an identical schema pattern, H_1 will be destroyed. It can also be seen that H_2 remains intact. We can see that there are 6 possible cut sites, each of which is equally likely to be picked. So H_1 will be destroyed with probability $p_d = d(H_1)/(l - 1) = 5/6$. Similarly H_2 will be destroyed with probability $p_d = 1/6$. It is generally true that schema with shorter defining length are less likely to be destroyed in crossover.

A lower bound on simple crossover survival probability p_s can be calculated for any schema. This is $p_s = 1 - p_d$, which is $p_s = 1 - d(H_1)/(l - 1)$. If crossover occurs with probability p_c then the survival probability is given by this expression:

$$p_s \geq 1 - p_c \cdot \frac{d(H)}{l-1}$$

Assuming the independence of the selection and crossover operators we obtain the following equation for the expected number of schema H in the next generation.

$$\mathrm{m}(H, t+1) \geq \mathrm{m}(H, t)\frac{f(H)}{\bar{f}}\left[1 - p_{\mathrm{c}} \cdot \frac{\mathrm{d}(H)}{1-1}\right]$$

The occurrences of a schema H grow in relation to the average fitness of all strings containing H and the defining length d(H) of H. The higher the average fitness and shorter the defining length, the higher the chance of survival.

Mutation :
In order for a schema H to survive mutation all of the specified positions in H must remain intact. Since a single gene survives with probability $(1 - p_m)$ and each mutation is independent, the probability that a schema survives is the same as multiplying the probability that one gene survives by itself o(H) times i.e. $(1 - p_m)^{o(H)}$. For $p_m << 1$ this can be approximated by the expression $1 - o(H).p_m$.

The Schema Theorem:
Combining all three processes of GAs, selection, crossover and mutation and ignoring cross-product terms we get :

$$m(H, t+1) \geq m(H, t)\frac{f(H)}{\bar{f}}\left[1 - p_c \cdot \frac{\mathrm{d}(H)}{l-1} - o(H) \cdot p_m\right]$$

This is known as The Schema Theorem or The Fundamental Theorem of Genetic Algorithms. Basically it says that short, low order, above average schema receive exponentially increasing trials in subsequent generations.

2.4.2 The Building Block Hypothesis

Building blocks are schema with a short defining length and high fitness. These schema are sampled, recombined and resampled to form strings of a potentially higher fitness (GOLDBERG 89). By using building blocks the complexity of the problem is reduced by creating potentially better strings using good partial solutions instead of building good strings by exhaustive search of all possibilities. Goldberg uses the analogy of a child being able to create magnificent fortresses with simple building blocks.

Much work has been done on both the Schema Theory and the Building Block Hypothesis to see if these theories are provable. Various papers in the Proceedings of the International Conference of Genetic Algorithms address this subject (BRIDGES 87, GREFFENSTETTE 89, ANTONISSE 89, HULIN 91). Investigation of this area of GA theory is not the aim of this thesis. However the next section outlines some initial investigations which look at the effect of using alternative encodings since most combinatorial problems use an increased alphabet set and not a binary one.

2.5 Binary Versus Real Encoding

Initial work with GAs commonly used a simple binary notation which has been used so far in this chapters' examples. However GAs with integer and real encodings have also been utilised (LIN 92, WRIGHT 92, THOMAS 95). The investigation detailed here was primarily carried out as a GA familiarisation exercise so is not entirely relevant to later work in this thesis. The aim was to assess the relative performance of both real and binary encodings. A real encoding is one which is not simply specified by a base 2 value, but as a full range number. A real coded GA and a binary GA were tested on De Jong's (DE JONG 75) five test functions. These functions were designed to include instances of continuous, discontinuous, convex, non-convex, unimodal, multimodal, quadratic, non-quadratic, low-dimensional and high-dimensional functions and also a function with gaussian noise.

Both algorithms were tested on the range of Pc and Pm values shown below and were evaluated on generation average and best string so far.

Pc = 0.60, 0.65, 0.70, 0.75, 0.8, 0.85, 0.9, 0.95, 1.00.
Pm = 0.0, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2.

To give a graphical representation of relative performance of binary and real GAs, each run was repeated a thousand times and averaged to give the final curve. The generation average plots give a measure of the rate of convergence. This was done by calculating the area under each average plot such as the one shown in Figure 2.10. This gives a single measure which combines rate of convergence and final value. The best so far plots give a measure of the ability of the search to improve the current best design. These are drawn by calculating the level of improvement from the first generation to the last. The ten plots, two for each of the five test functions are shown on the following pages with the binary GA coloured in grey and real GA in black.

2.5.1 Generation Average

It can be seen from Figures 2.5-2.9 that for most of the test functions the rate of convergence for the real GA is better than that of its binary equivalent. Only on DeJong 5 (Figure 2.9) has binary GA overtaken the performance of the real GA for small values of mutation.

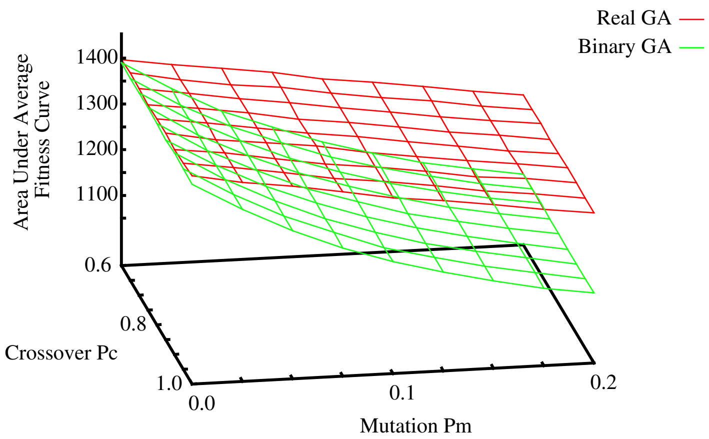


Figure 2.5 - De Jong 1 - Generation Average

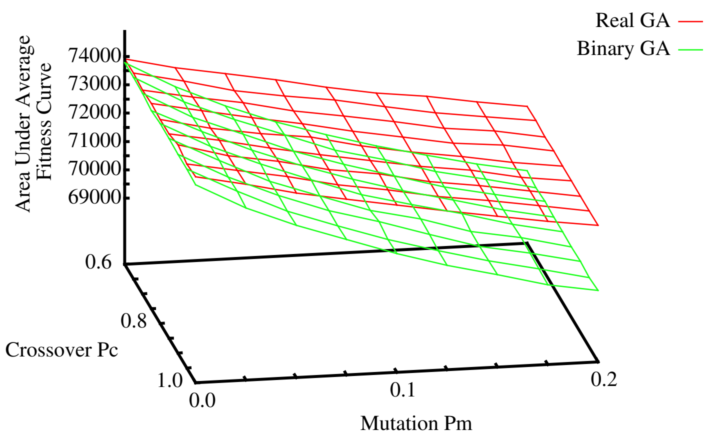


Figure 2.6 - De Jong 2 - Generation Average

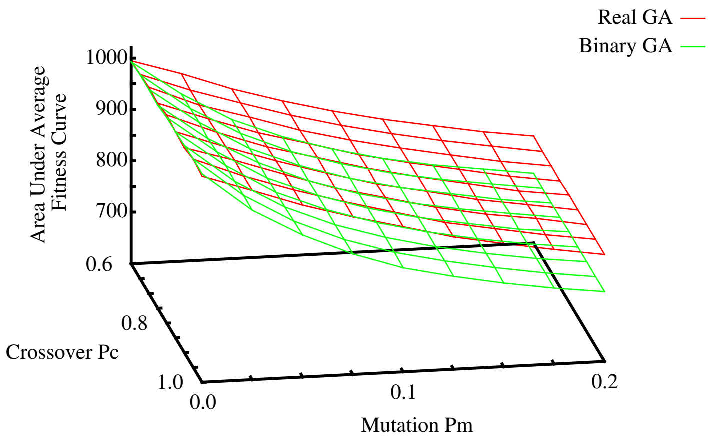


Figure 2.7 - De Jong 3 - Generation Average

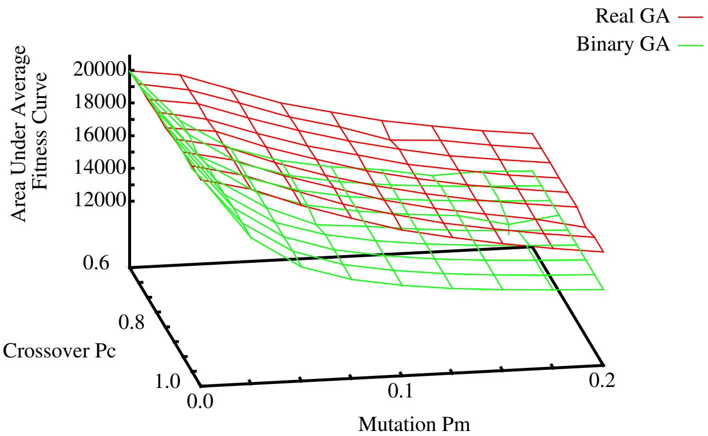


Figure 2.8 - De Jong 4 - Generation Average

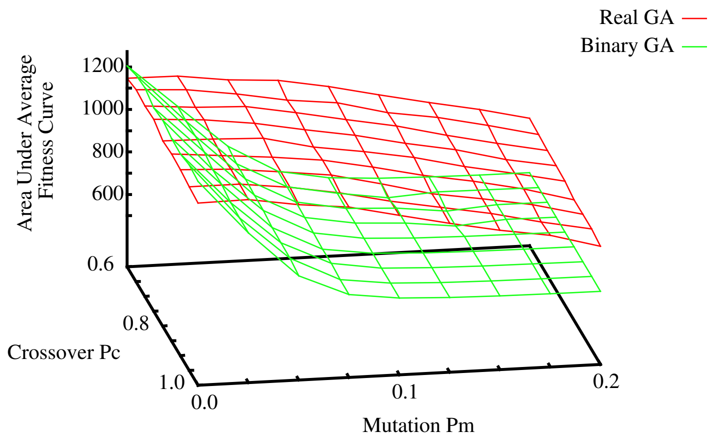


Figure 2.9 - De Jong 5 - Generation Average

In general the crossover does not have a large effect on convergence, however it can be clearly seen that increasing mutation rate decreases performance. This is expected as mutation disrupts the search by introducing random changes to the strings.

Investigations into other large alphabet search schemes have also found similar results. Goldberg (GOLDBERG 91) points out that large alphabets reduce the number of possible schema because the strings created are shorter in length than their binary equivalent. This reduction in the number of schema usually means fewer generations are required to reach population conformity.

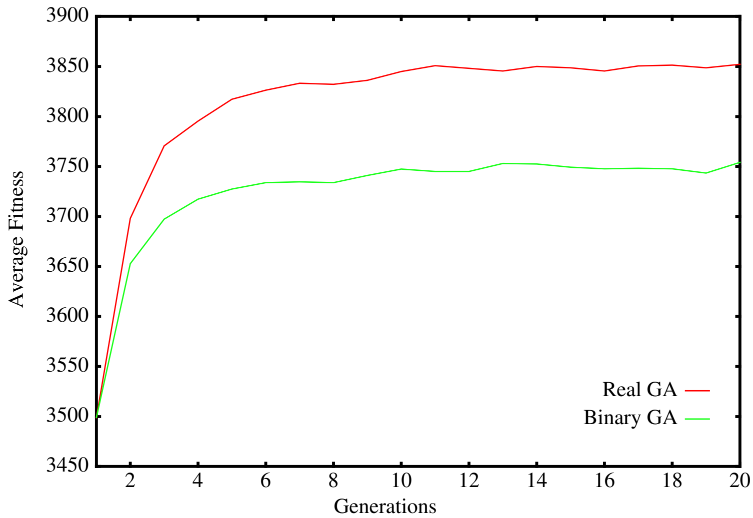


Figure 2.10 - De Jong 2 - Generation Average, Pc = 0.8, Pm = 0.1

Figure 2.10 shows a typical plot of fitness over 20 generations. The real GA clearly settles at a higher value than the binary GA, indicating that the real GA population has moved quickly towards what it thinks is the optimal point for the search.

The real GA performance is also more linear over the mutation range. This is clearly the case for test functions 4 (Figure 2.8) and 5 (Figure 2.9) where the binary GA surface drops off very steeply as mutation increases from 0. The real strings, because they are shorter, will stand less chance of individual real positions being mutated for a given value of Pm. This reduces the disruptive effect of mutation and increases the linearity over the range.

Another important subtlety is that the real GA, which is a variable-level mapping of the problem, can perform mutation over the full range of that variable i.e. if a real gene is chosen for mutation it can take on any value between the maximum and minimum of the corresponding variable. If we look at the binary case, mutation will only act upon a single bit in a string, assuming that Pm is small. This has the effect of reducing the amount of potential values reachable from the current variable state. If, for example, a variable x1 is taken with a range 0-15, then 4 bits are required to represent it, 0000 - 1111. If mutation is performed on the number 12 (1100) there are only four possible values the variable can change to, 4(0100), 8(1000), 14(1110), 13(1101).

The combined effects of the variable level mapping and full range mutation can be demonstrated by implementing the binary mutation operator as one which is analogous to that used in the real GA. Instead of mutating individual bits in a string, all of the bits representing a chosen variable will be mutated with probability Pm. This simulates the action of the real mutation operator. This change was implemented and applied to test function 5 to show the effect. Figure 2.11 below illustrates the results. As predicted it can be seen that the generation average performance of this modified binary operator is very similar to the performance of the real GA.

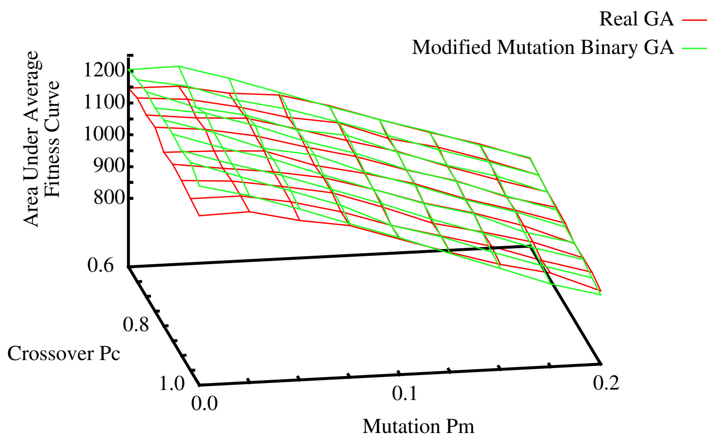


Figure 2.11 -Generation Average with Modified Mutation Binary GA

2.5.2 Best So Far

Figures 2.12-2.16 show the 'best so far' performance of the real and binary GAs on the five test functions. Compared to average performance the diversity of curves over the five functions is much greater.

The first two functions DeJong 1 and 2 are smooth functions with small numbers of variables, 3 in function 1, 2 in function 2. In both cases the binary GA shows better exploration properties than the real GA (Figures 2.12 and 2.13). Figure 2.17 shows the best so far curves for fixed values of Pm and Pc for DeJong 2. After 20 generations the binary GA reaches higher values and increases much more quickly, the real GA does move towards higher values but at a lower rate.

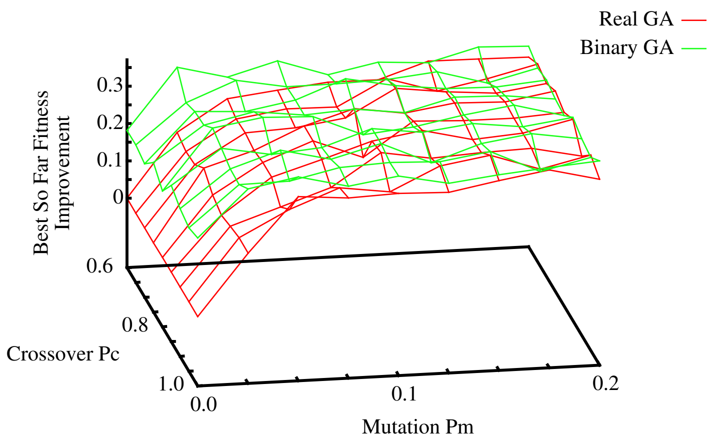


Figure 2.12 - De Jong 1 - Best So Far

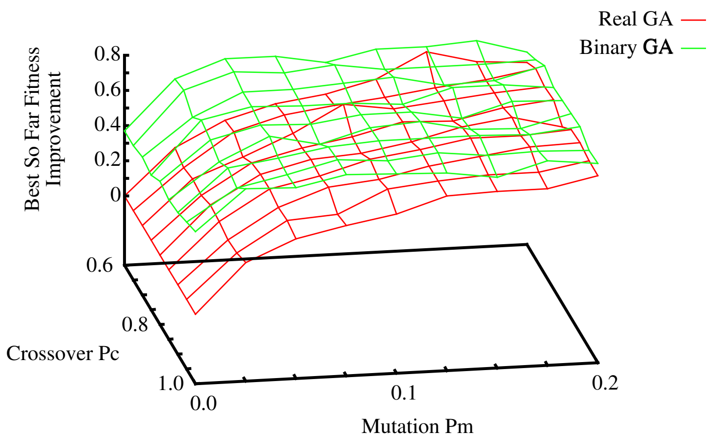


Figure 2.13 - De Jong 2 - Best So Far

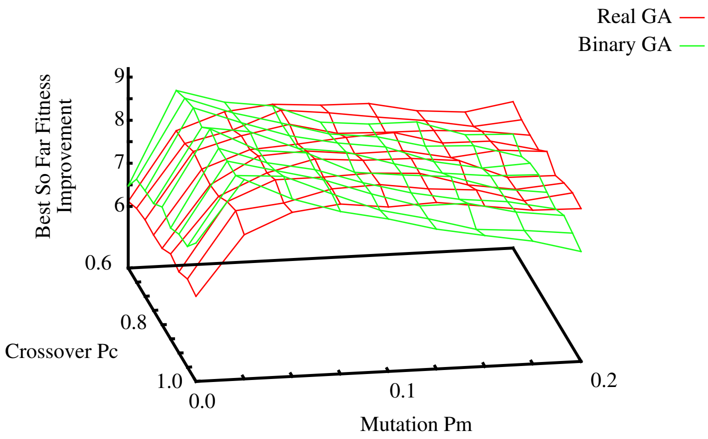


Figure 2.14 - De Jong 3 - Best So Far

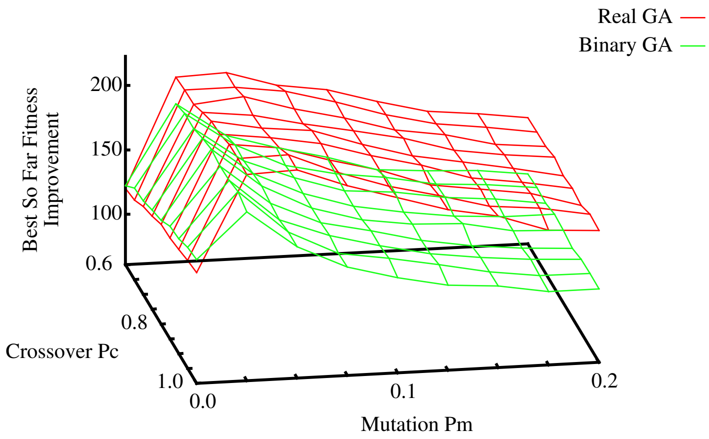


Figure 2.15 - De Jong 4 - Best So Far

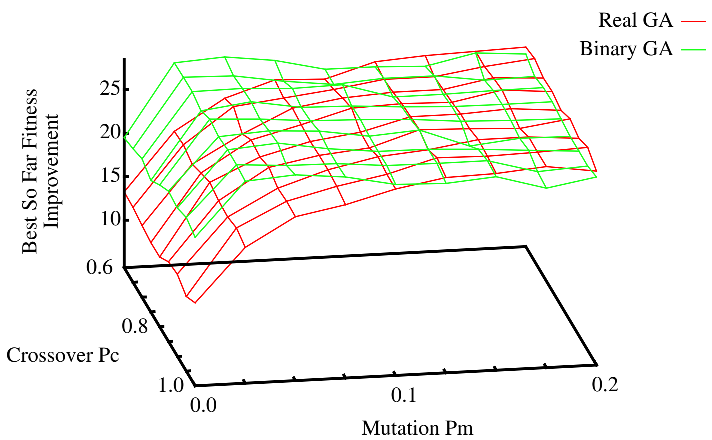


Figure 2.16 - De Jong 5 - Best So Far

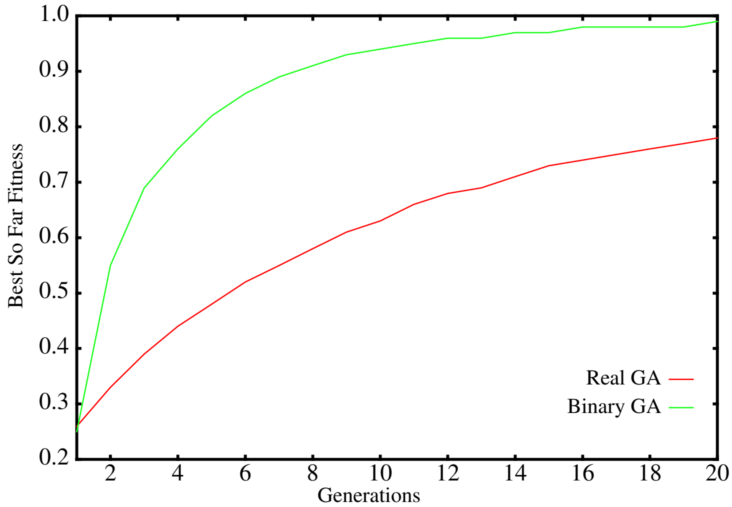


Figure 2.17 - De Jong 2 - Best So Far, Pc = 0.8, Pm = 0.1

Another phenomenon arises in the first two functions. It can be seen in Figures 2.12 and 2.13, that when Pm = 0, the real GA shows little or no improvement, yet the Binary GA is still able to synthesise new solutions. The reason for this is that the only way the real GA can create new variable values is by using mutation. Crossover in real GA is performed at a inter-variable level whereas in the Binary case it is performed at an intra-variable position. Hence the standard Binary GA will create new variable values during crossover as each variable is made up of several bits, this increases diversity and promotes wider search. This effect is shown here.

Two five bit strings :

	1 1 1 1 1	31
	0 0 0 0 0	0

Crossover Position

1	1 0 0 0 0	16
	0 1 1 1 1	15
2	1 1 0 0 0	24
	0 0 1 1 1	7

3	1 1 1 0 0	28
	0 0 0 1 1	3
4	1 1 1 1 0	30
	0 0 0 0 1	1

If these five bits represent a single variable in the problem, crossover alone can generate 8 new values for that variable. If we look at the later test functions where there are more variables, hence longer strings, the real GA is able to make improvements on its initial best so far value by creating new string combinations from currently existing strings but it will not generate new variable values. This suggests that a real GA would work better with problems with smaller variable ranges and longer strings.

Test functions 3 and 5 (Figures 2.14 and 2.16) provide very similar plots, with the real GA slowly overtaking the performance of the binary GA as mutation increases. Both functions have isolated areas of similar fitness distributed over the search space. The real GA, because it uses full range mutation of variables, gives a much broader search area for a given starting population (for larger values of Pm). This increases the chance of finding higher fitness areas, if such areas exist, hence allowing the GA to escape from local maxima/minima.

2.5.3 Obtaining Binary GA Performance

The real GA has properties and characteristics of its own. However, it can be made to mimic the binary operators by using binary-real conversion operators. This may seem a computationally expensive way of performing these operations but the number of conversions are kept to a minimum and if evaluations take a long time this conversion time is insignificant.

Initially the equivalent binary length is found for each of the variables in the string. This is not an exact mapping but an extra bit is used to give reasonable accuracy.

Crossover Operator:

When it is decided that two strings are to be crossed the variable level cross location on the real string is randomly selected. The sub-cut position is then randomly chosen within that variable based on its equivalent binary string length. If the sub-cut position is at the end of the equivalent binary string then the string is not decoded since this is the same as an inter-variable cut. Otherwise the chosen variable is converted to binary, crossed as normal and then converted back to real. The standard crossover operation then continues at a variable level swapping the rest of the strings over.

Mutation Operator:

For each string, the positions of mutations are first selected by calculating the length of the equivalent binary string and probabilistically determining where mutation occurs. If mutation does not occur within a string it is left untouched. Otherwise the variable in which mutation occurs is converted to binary, the chosen bit(s) mutated, and then re-encoded in real form.

The results of this implementation of the real GA are shown for both generation average and best so far performance on test function DeJong 3 in Figures 2.18 and 2.19. It is clear that the performance of the real GA with binary operators is comparable to the binary GA.

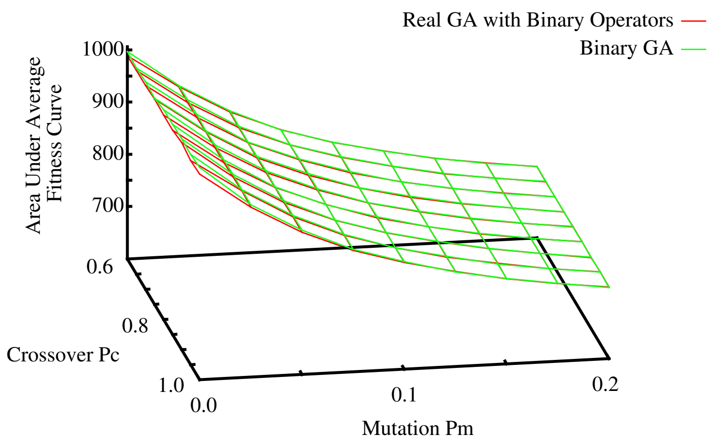


Figure 2.18 - De Jong 3 - Generation Average

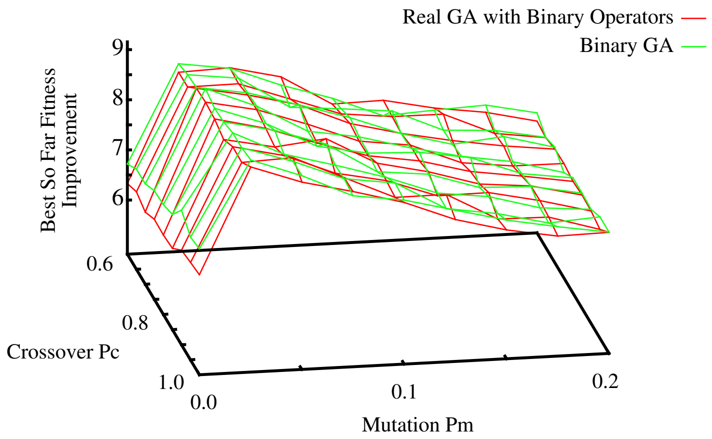


Figure 2.19 - De Jong 3 - Best So Far

2.5.4 Discussion

The real GA algorithm, although not providing significant performance advantages over the conventional binary GA, does provide several advantages in other areas. The one to one mapping between variable and gene makes the GA simple to encode and decode. There is no need to provide more complex mappings for integer, real or discrete problems that do not have exact base 2 ranges of values.

This direct mapping also means there are no redundant representations in that the system will only create valid string configurations through all of its operations. This can cut down on computation time and reduce the amount of problem constraints. They above all contribute to simplicity in implementation. A short string of neatly divided variable settings is more meaningful than a long binary string.

The use of real strings also avoids the occurrence of Hamming Cliffs (GOLDBERG 91). This is because each gene represents one variable so only one gene has to be changed. In the binary case there is a many to one gene-variable mapping, so, for example, in order to change from 255 to 256, 9 binary digits have to change.

The one to one mapping of the real GA means a high mutation rate is required. This problem does not arise with the binary GA does since its many to one mapping means that it is more likely to be changed for a given mutation rate. This increases the diversity of binary GA search preventing it from converging too quickly.

2.6 Chapter Summary

In this chapter the basic mechanics and theory of GAs has been outlined. This will act as a background to the work carried out later in this thesis. Also in this chapter an experiment to compare different encoding schemes was carried out as a familiarisation exercise. It was found that real coded GAs can offer some advantages over normal binary GAs. However the real GA did suffer from early convergence problems due to the compact encoding.

In the next chapter a basic combinatorial problem is tackled. The Travelling Salesman Problem (TSP) is used to test a set of general crossover and mutation operators to assess their ability in tackling combinatorial problems. The GA is also used to solve a TSP-like industrial problem dealing with component placement in electronic circuit board manufacture.

3 Genetic Algorithms and the Travelling Salesman Problem

3.1 The Travelling Salesman Problem

The Travelling Salesman Problem can be simply stated as follows:

> *Find the shortest circular route which visits N cities on a given map avoiding duplication or omission.*

Having addressed the theoretical background of GA's in the previous chapter, this chapter looks at a well known combinatorial problem, namely the Travelling Salesman Problem(TSP). The TSP is a problem which is simply stated but because of its nature is hard to solve. The problem involves a set of cities located at coordinates x,y. The aim is to order the cities within a GA string such that the sum of the adjacent distances of the cities through the string is a minimum. The difficulty lies in the number of combinations involved for a given number of cities. For n cities there are n! combinations. For example 10 cities leads to 3.6×10^6 combinations and 20 cities leads to 2.4×10^{18} combinations. The number of combinations dramatically increases for a small number of additional cities, and this explosion in complexity soon leads to the preclusion of an exhaustive search.

3.1.1 O-Schema

The TSP problem uses a slightly different definition of schema than the standard GA. O-Schema (OLIVER 87) are based on the lexical ordering of the fixed value symbols. For example the schema H=ab**c***** with order $O_o(H)=3$ is equivalent to the all of the following: ***ab**c**, *ab**c****, *****ab**c, etc. This has two advantageous effects. Firstly, there are a reduced number of schema for a given schema definition hence there are less schema to search. Secondly, many schema can be represented with lower defining lengths. For example, *b*******a is identical to a*b*******, i.e the defining length, $d_o(H)$, has been reduced from 8 to 2.

3.1.2 GAs and the TSP

Grefenstette (GREFENSTETTE 85b) carried out some initial investigations into the possibility of using the GA for the TSP using some new operators. Standard GA operators cannot be applied because duplicates and omissions would occur within the string. Since then various crossover and mutation operators have been investigated in the literature in order to improve the GA's ability to solve the problem.

Several crossover and mutation operators were obtained from the literature in order to perform comparative tests. These operators are described below with diagrams explaining their action. The reference next to each operator shows the source in the literature. The operators are all simple in that they use no heuristics or additional local search. This restriction was applied because the aim of this experiment was to find a good combinatorial operator which was not domain specific, for later use in this thesis.

3.2 Crossover Operators

3.2.1 One Point (MURATA 94)

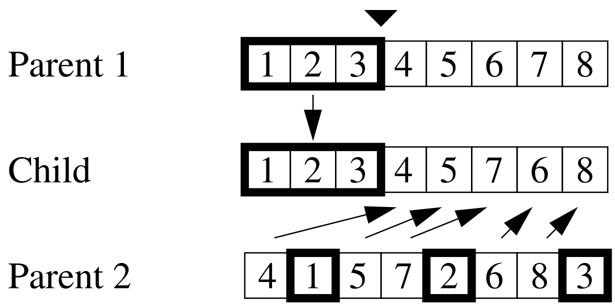


Figure 3.1 - One Point Crossover

A cross point is selected and the genes to the left of this point are copied straight across from parent 1. The rest of the child is constructed using the same order as parent 2 with the used cities removed.

3.2.2 Two Point End (MURATA 94)

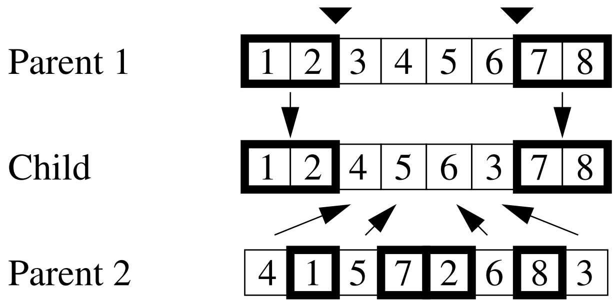


Figure 3.2 - Two Point End Crossover

Two cross points are selected and the ends of parent 1 are then copied directly into the child string. The remainder of the child is taken in the same order as parent 2 avoiding duplicate genes.

3.2.3 Two Point Centre (MURATA 94)

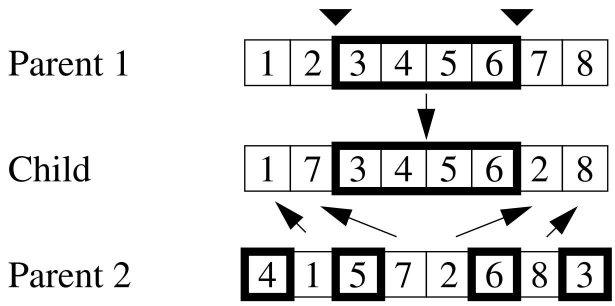


Figure 3.3 - Two Point Centre Crossover

Two cross points are selected and the centre section of parent 1 is directly copied to the child. The remainder of the child is taken in the same order as parent 2 avoiding duplicate genes.

3.2.4 Two Point End/Centre (MURATA 94)

Utilising both Two Point End and Two Point Centre this method selects, with equal probability, from one of the above two crossover operators.

3.2.5 Position (MURATA 94)

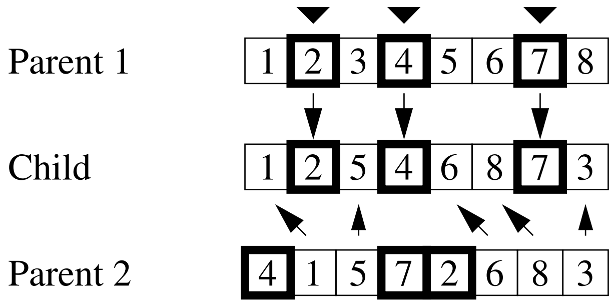


Figure 3.4 - Position Crossover

A random number of genes between 1 and the string length are selected. These points are then copied from parent 1 into the child. The remainder of the child is constructed from the order of parent 2 avoiding repeated genes.

3.2.6 Independent Position (MURATA 94)

This operator is similar to position crossover except that the crossing positions are selected with probability 0.5 for each bit in the string. Any selected gene is copied straight into the child. The remainder of the child is constructed as in Position Crossover.

3.2.7 Partially Mapped (GOLDBERG 85)

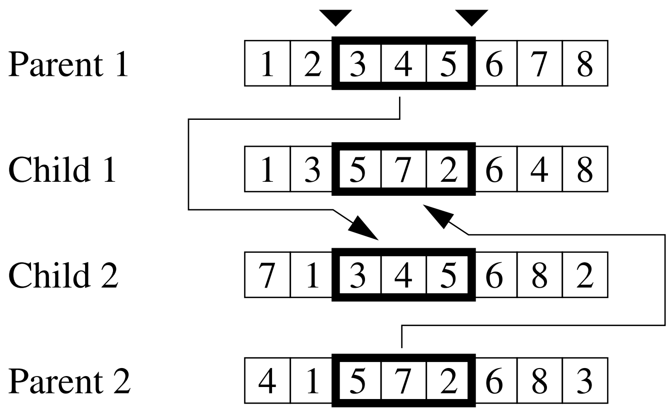


Figure 3.5 - Partially Mapped Crossover

Partially Mapped/Matched Crossover or PMX for short. Two positions are chosen on the string at random (as with Two Point Crossover). The substrings between these two points are called the Mapping Sections. These sections are then mapped across onto each other as shown. This crossover gives the mappings 3>5, 4>7 and 5>2. So in parent 1, wherever a 3 occurs it is replaced by a 5 and vice versa. The same occurs for mappings 4>7 and 5>2 and also in parent 2 where identical mappings occur. This gives two new strings with no repeated nodes.

3.2.8 Ordered (DAVIS 85)

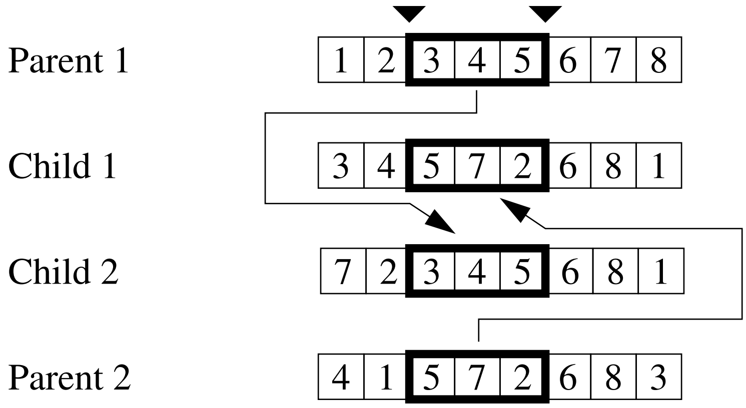


Figure 3.6 - Ordered Crossover

Two points are chosen and the genes between these points are removed from the other parent. The remaining genes are then moved round in a leftwards sliding motion leaving the mapping section empty until the first member of this parent's mapping section is at the start of the string, i.e. in the case of child 1, this is 3. This sliding motion is applied as if the string wraps-round. The empty mapping section is then filled with the mapping section of the other parent.

3.2.9 Cycle (OLIVER 87)

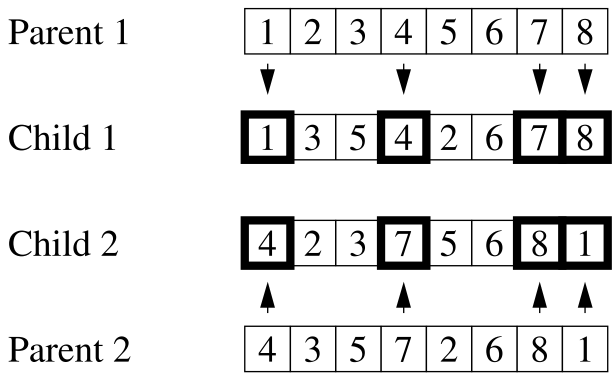


Figure 3.7 - Cycle Crossover

In parent 1 the first gene is selected and moved into child 1, the number 1 in this case. The value of the corresponding gene position in parent 2 is then found, number 4, and that gene in parent 1 is copied into child 1. Again the corresponding gene position in parent 2 is then found, number 7, and that gene in parent 1 is copied into child 1. This is repeated until a cycle is introduced, i.e a value is repeated. In this case the cycle is 1, 4, 7, 8. The remaining genes are then copied directly from parent 2 into child 1. This is repeated starting with the first gene in parent 2.

3.2.10 Edge Recombination (WHITLEY 89)

City : Edge List
1 : 2, 4, 8
2 : 1, 3, 6, 7
3 : 2, 4, 5
4 : 1, 3, 5
5 : 3, 4, 6, 7
6 : 2, 5, 7, 8
7 : 2, 5, 6, 8
8 : 1, 6, 7

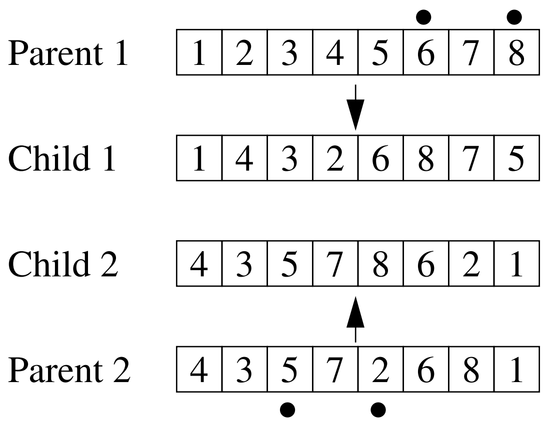


Figure 3.8 - Edge Recombination Crossover

Edge Lists are defined for all cities as the adjacent genes in a circular string in both parents with duplicates removed. Edges being the connection between cities. For example if we take city 7 it has edges 6, 8, 5 and 2 (marked with dots). To start, a city is randomly chosen from one of the two parents' starting cities, in the case of child 1, city 1 is chosen. This is then removed from all other lists. Then we need to make a choice between 2, 4 and 8. This is done first of all by selecting the city with least adjacent edges, if this is a tie, as it is here between 4 and 8, the choice is performed randomly between the two. In this example 4 has been chosen. This process is repeated until the child 1 is filled and then it is repeated for child 2.

3.2.11 Enhanced Edge Recombination (STARKWEATHER 91)

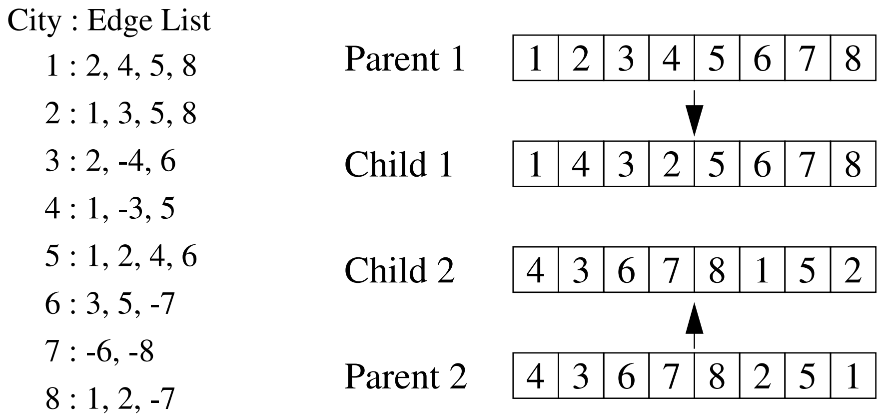


Figure 3.9 - Enhanced Edge Recombination Crossover

The edge lists are defined and built in the same way except that duplicates are now marked with a minus sign. In the case of number 7, cities 6 and 8 form the adjacent edges in both strings. The operator behaves in much the same way as normal edge recombination but priority is given to negative edge choices. It can be seen that 6, 7, 8 has been preserved in both children.

3.2.12 Subtour Chunks (GREFFENSTETTE 85b)

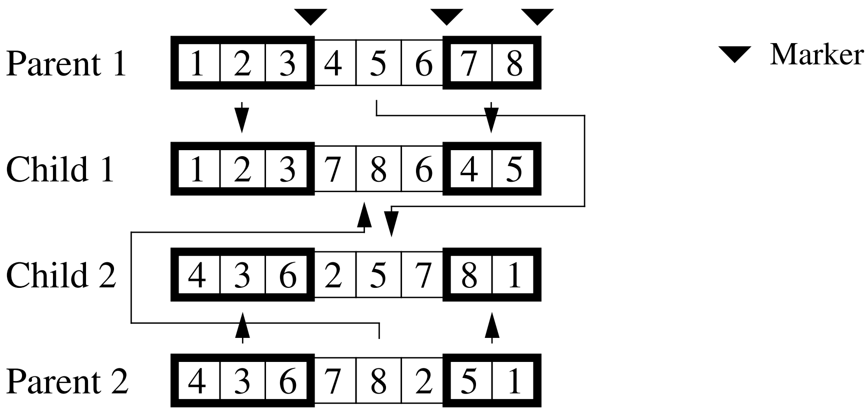


Figure 3.10 - Subtour Chunking Crossover

Random length alternate subtours are chosen from the parents, the end points of the subtours are denoted by the markers. The operator moves from left to right swapping subtours. If any city in a subtour is repeated it is replaced by a randomly selected city which has not been used yet. This usually leads to more disruption later in the string.

3.2.13 Alternating Edges (GREFFENSTETTE 85b)

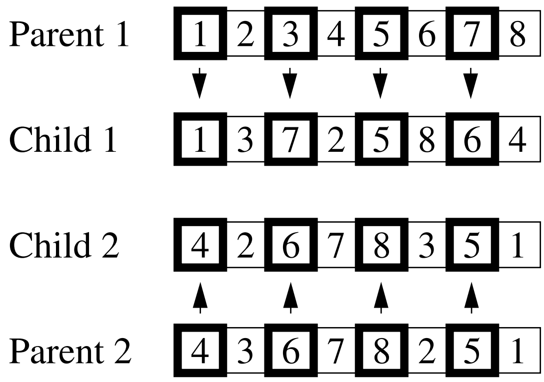


Figure 3.11 - Alternating Edges Crossover

Here alternating cities from both parents are taken and if a path is repeated it is replaced with a randomly selected path which has not yet been used.

3.2.14 Inversion (GOLDBERG 89b)

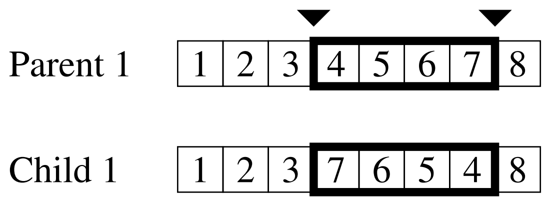


Figure 3.12 - Inversion

Not strictly a crossover operator. A section of the parent is chosen and inverted so the first city in the section becomes the last, second becomes second last and so on.

3.3 Mutation Operators

3.3.1 Two-City Adjacent Swap (MURATA 94)

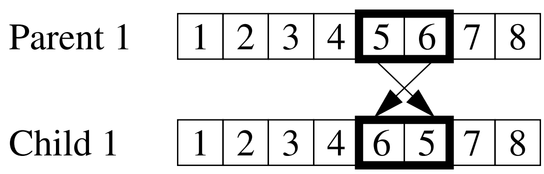


Figure 3.13 - Two City Adjacent Swap

3.3.2 Two-City Random Swap (MURATA 94)

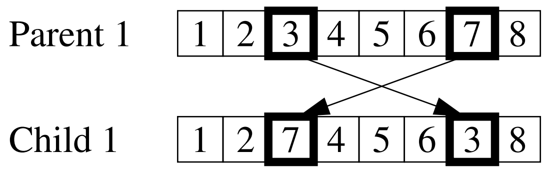


Figure 3.14 - Random Two City Swap

3.3.3 Three-City Adjacent Swap (MURATA 94)

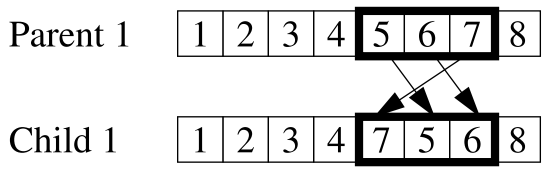


Figure 3.15 - Three City Adjacent Swap

3.3.4 Three-City Random Swap (MURATA 94)

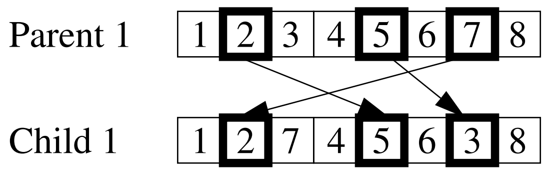


Figure 3.16 - Three City Random Swap

3.3.5 Shift (MURATA 94)

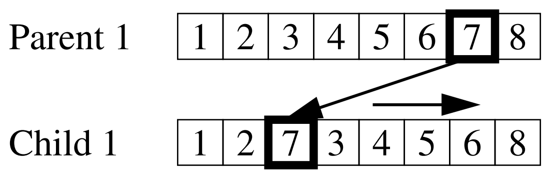


Figure 3.17 - Shift

In this operator a city is randomly selected and moved to a random location. The rest of the cities are shifted along into the gap left by the removed city.

3.4 Performance of Operators

3.4.1 Test Problems

In order to assess the quality of these various operators, with the aim of finding a good general combinatorial operator, they were applied to a set of problems of size 10, 20, 50, 100 and 200 cities. Oliver (OLIVER 87) performed a similar analysis limited to a 30 city problem with three operators. Starkweather (STARKWEATHER 91) repeated Oliver's tests with 6 operators. The problems and near optimal solutions for the tests performed here are shown in Appendix B along with a selection of graphs from the tests. In all cases the population size was fixed at 100. This was to test the search capabilities of the operators as the complexity of problem increased without the effects of changing population size. Pc=0.8 and Pm=0.01 was used in all tests.

3.4.2 Results

The results of the tests are summarised in the following two tables, one for crossover and one for mutation. The tests are based on the best so far output of the TSP search, however the generation average graphs are also shown in Appendix B. The operators are scored as follows. For Crossover the 5 best are given scores 5-1 respectively for each mutation operator. For Mutation the 3 best are given scores 3-1 for each crossover operator. These scores are summed over each of the cases and a score is given for that particular problem. These problem scores are then summed to give a total

.

Table 3.18 - Crossover Operators

	10 City	20 City	50 City	100 City	200 City	TOTAL
1.Inversion	6	0	0	5	10	**21**
2.One Point	10	3	5	4	4	**26**
3.Two Point End	4	6	10	4	7	**31**
4.Two Point Centre	0	4	3	4	3	**14**
5.Two Point End/Centre	3	5	1	3	10	**22**
6.Position	1	1	0	2	5	**9**
7.Independent Position	0	10	9	10	2	**31**
8.Patially Mapped	0	0	7	7	4	**18**
9.Ordered	3	1	0	0	0	**4**
10.Cycle	3	0	10	10	9	**32**
11.Edge Recombination	22	22	6	9	7	**66**
12.Enhanced Edge	23	23	24	17	14	**101**
13.Subtour Chunks	0	0	0	0	0	**0**
14.Alternating Edges	0	0	0	0	0	**0**

Table 3.19 - Mutation

	10 City	20 City	50 City	100 City	200 City	TOTAL
1.Adjacent 2 City	1	14	28	36	37	**116**
2.Random 2 City	37	27	9	4	4	**81**
3.Adjacent 3 City	3	6	28	32	29	**98**
4.Random 3 City	20	6	1	0	0	**27**
5.Shift	23	31	18	12	14	**98**

From Table 3.18 it can be seen that the best operators are the Enhanced Edge Recombination followed by the Edge Recombination. This is because they are the only operators to specifically attempt to maintain adjacent edges which is an important aspect in solving the TSP. By maintaining edges good subtours are built up between parents and passed on to their children. Subtour chunks and alternating edges performed the least well as they utilise a random element to repair repeated cities. This prevents the propagation of edges into children reducing the likelihood of building blocks forming. The crossover operators are the dominant operators. They have a greater effect on the convergence than mutation.

It can be seen that the mutation operators compete more closely, with Adjacent Two City mutation performing the best. Shift mutation performs the most consistently across all tests, but the performance drops as the number of cities increases. Random Three city mutation performed least well as the movement of three cities will disrupt 6 edges and the likelihood of forming better edges in their new positions is very small, especially with larger problems.

In general it can be observed that as the number of cities increases the ability to evolve good solutions drops. This can be seen in Figure 3.20 which shows the best so far plots for a GA with Enhanced Edge Crossover and Adjacent 2 City Mutation. One curve uses the standard population size of 100 and the other is an additional run with a population size of 500. The line at the bottom shows the target value which is the distance covered in the near optimal solution. Additionally, route plots of near optimal and the GA Generated solutions (population size 100 and 500) are shown in Figures 3.21 to 3.23.

It is very clear that the performance of these operators is very poor. Even for the run with population size of 500 the GA is still unable to create good solutions. This poor performance is due to several effects. For a population size of 100 the diversity in the initial population is not sufficient to provide a good spread of possible edges. The Enhanced Edge Recombination operator can only utilise edges that already exist within the population and the only way to create new edges is through the mutation operator. A higher mutation rate could be used to introduce more new edges but this would slow convergence.

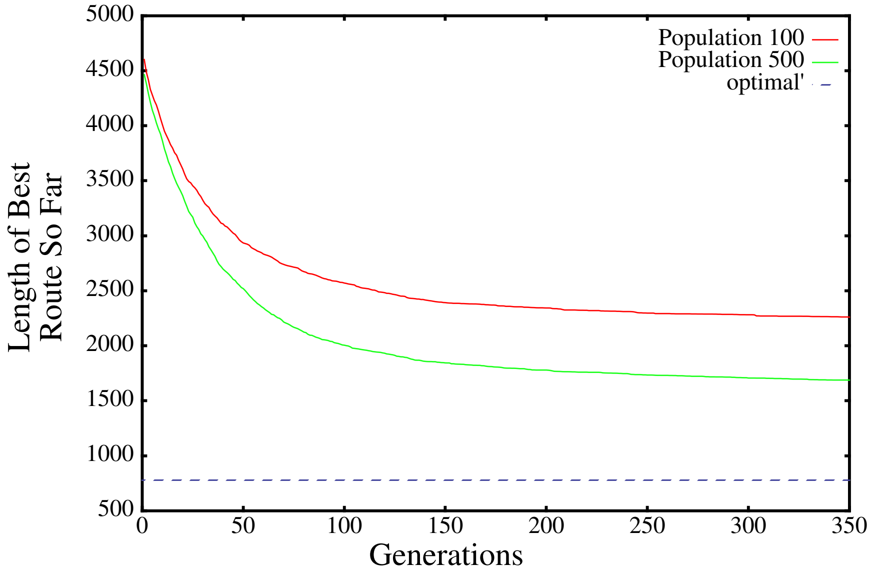


Figure 3.20 - Convergence Curves for 100 City Problem

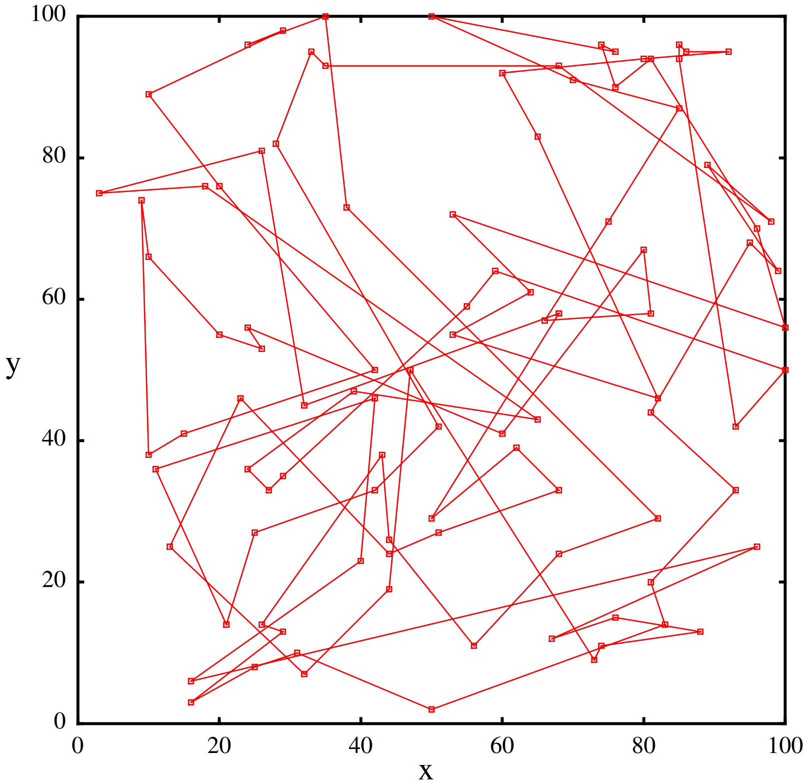


Figure 3.21 - Best Solution Generated by GA

(Population Size 100) - Length 2094.35

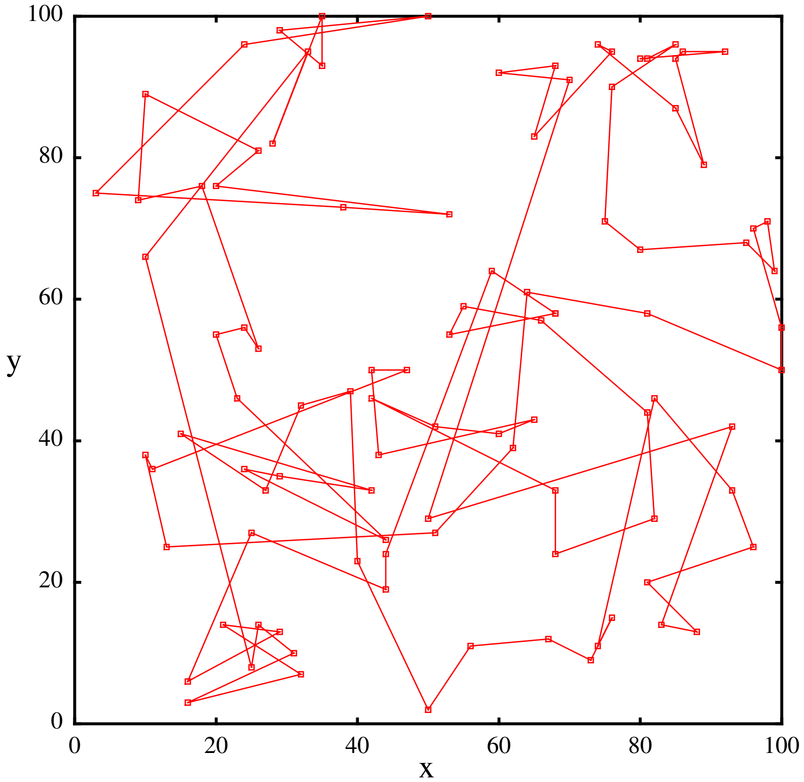


Figure 3.22 - Best Solution Generated by GA

(Population Size 500) - Length 1580.56

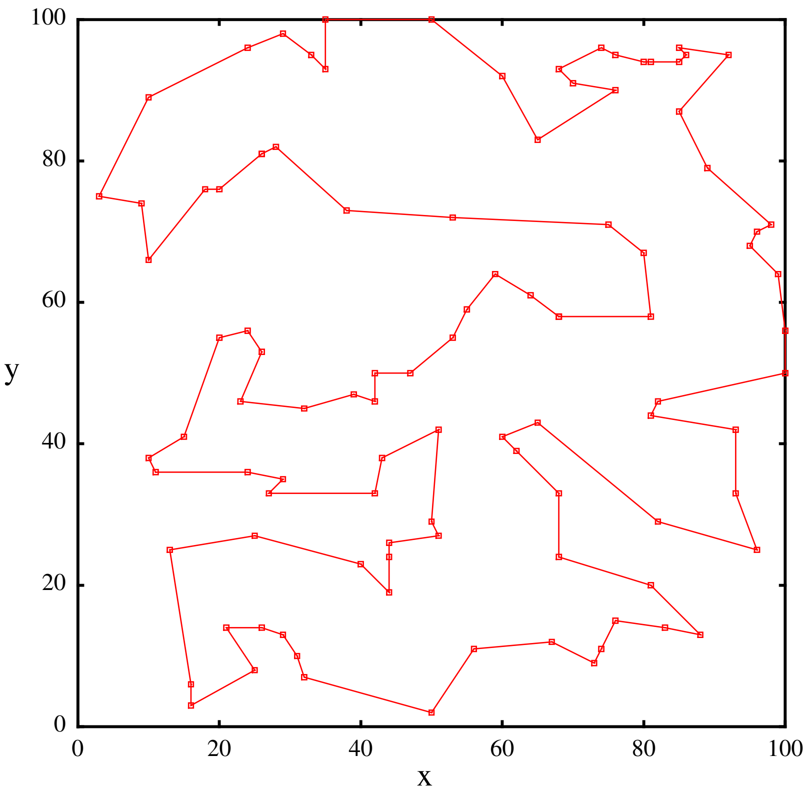


Figure 3.23 - Near Optimal Solution - Length 779.234

The reason the GA struggles is because it is fitness based. It only has information on the total length of the string. It is often the case that short fit subtours are spoilt in overall fitness terms by long linking edges. In Figure 3.21 there are good short local subtours and these are linked together by long edges. Greffenstette (GREFFENSTETTE 85b) previously indicated that selection alone may not be able to properly allocate trials to solutions with good subtours.

It is clear from these results that the GA struggles with this type of problem when scale is increased. The only way to tackle these problems using a GA is to use domain specific knowledge.These methods are described below.

3.5 Local Search and Heuristics for the TSP

Sometimes also known as greedy genetics (LIEPINS 87) these methods use domain specific rules (heuristics) or local searches to improve the speed and effectiveness of GA search for complex problems. These methods use effective but not optimal searches in order to make improvements.

In the case of the TSP an example of a heuristic could be ‘choose the next city in the string as the closest one to the current city’. A modification of this heuristic was implemented within a GA with good results (GREFFENSTETTE 85b) where the crossover operator is expressed as a 3 stage procedure:

1. For a pair of parents, start at the first position (always the same city).
2. From the current city find the next city in both parents and choose the closest of the two. If this leads to a cycle choose the other city. If this leads to a cycle choose a random city.
3. If the tour is complete then stop; otherwise go to 2.

The performance of this operator is shown in Figure 3.24 compared to the Enhanced Edge operator. The best route found is shown in Figure 3.25. There are still some residual problems with this operator. It is clear from Figure 3.25 that certain cities need to be swapped to get rid of some of the longer edges. This is because the operator attempts to ‘glue together’ good subtours (SUH 87) but occasionally has to choose a random solution which breaks the continuity of the tour.

.

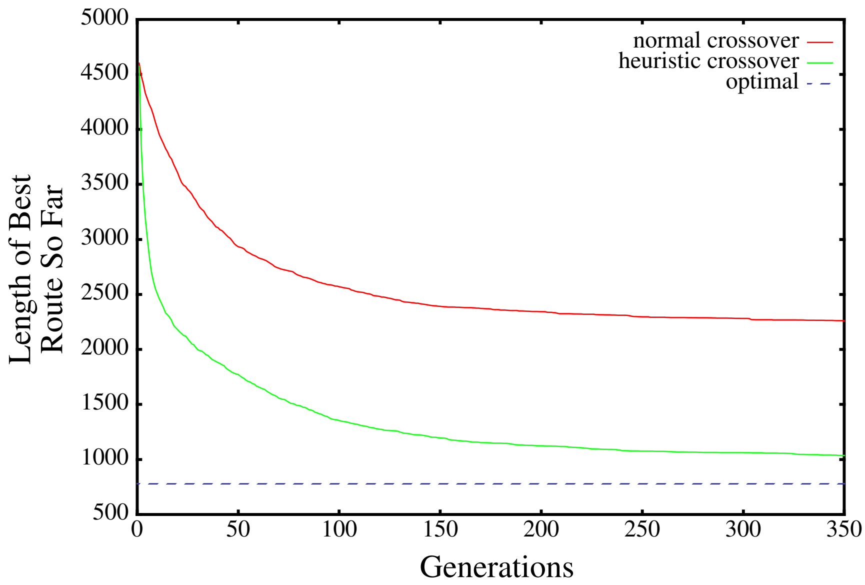


Figure 3.24 - Greffenstette's Heuristic Crossover on 100 City Problem

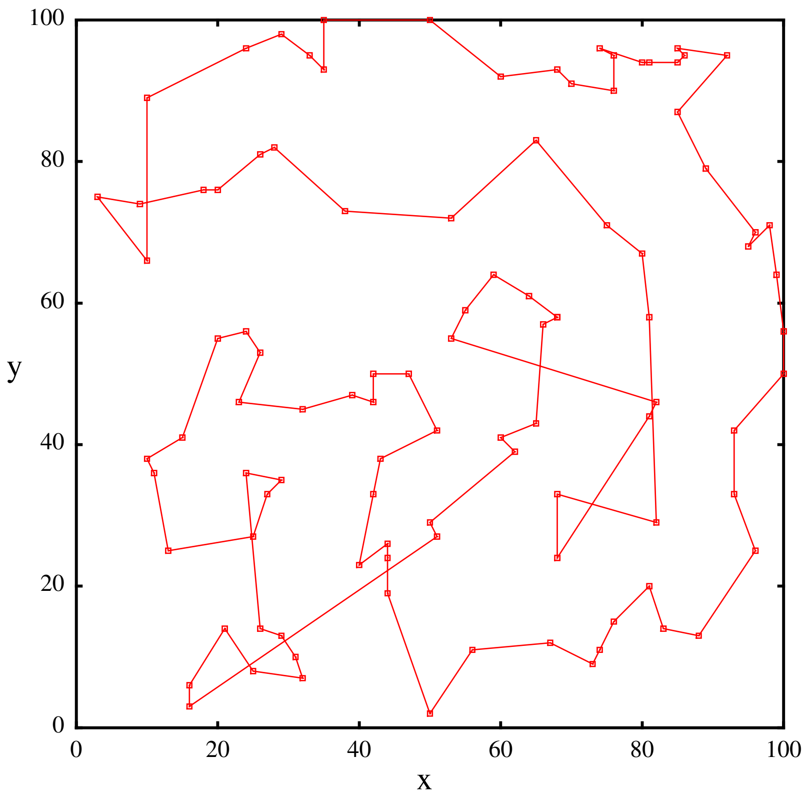


Figure 3.25 - Heuristic Crossover Solution - Length 843.5

Prompted by this early heuristic operator several other heuristics were investigated. Suh (SUH 87) investigated a local search method incorporating a variation of Lin and Kernighan's (LIN 72) n-opt operator. Their reduced 2-opt search compares the Euclidian distance of two edges and swaps if an improvement is possible. Using this operator Suh obtained optimal solutions to several TSP's. He also applied similar types of operators to the sliding block problem (SBP).

Other enhancements of the 2-opt operator were introduced by Muhlenbein (MUHLENBEIN 88) and later Jog (JOG 89). Muhlenbein applied several evolutionary methods within a parallel architecture to solve the TSP. A combination of 2-opt and Or-opt (LAWLER 85) was used to efficiently solve a 442 node problem. Jog performed similar experiments on a normal architecture using combinations of heuristic crossover and n-opt type local search. Jog also highlighted the need for crossover to move the GA out of local minima.

It is obvious that this kind of heuristic can aid in a search, however, it removes the generality of the operator. De Jong (DE JONG 89) points out that if weak methods are used the search method will have wider applicability but struggle as problem size increases. If strong methods are used, such as the ones discussed above, highly complex problems can be solved effectively but the search is not easily applied to other problems without modification. It is clear that it is important to select an effective local search method. In doing so the balance between the GA and local search must be such that there is a synergy between the two. If local search is applied too aggressively premature convergence may result.

3.6 Electronic Component Placement

This is an application for a TSP type solver. The problem involves automatic electronic component placement on a circuit board using a head feed system. A diagram of the system is shown in Figure 3.26. Initially the Printed Circuit Board (PCB) is located precisely underneath the assembler. The board may be on a conveyor system or be manipulated by a robot. Once in position the head moves around the board placing the components onto the board at the correct locations.

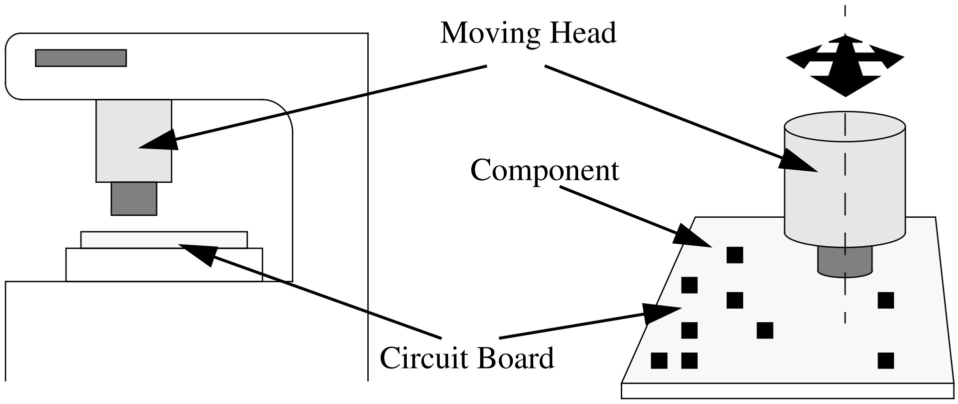


Figure 3.26 - Circuit Board Assembly

The automation of such processes can lead to significant efficiency gains in terms of reduced manufacturing time and a reduction in manpower requirements. Such systems can also improve the quality of the final product by introducing a uniformity in construction not necessarily present in manual processes. The decrease in the size of electronic devices has led to a dramatic fall in the size of circuit boards. These small circuit boards are hard to assemble by hand. This, along with the advances in surface mount technology and multi-layered printed circuit boards, has increased the need for automation.

The goal of the TSP solver is to produce the shortest placement path for the head to follow. The solver introduced here uses a slightly modified heuristic crossover and a 2-opt local search operator. The heuristic crossover discussed earlier in Section 3.5 is modified slightly in that instead of choosing a random city if a duplicate is detected the closest city which has not yet been used is inserted. To maintain diversity this is only done on half the occurrences. The solver also uses a local search method based on the 2-opt algorithm similar to Suh's TSP solver (SUH 87). However, this method is only applied in later generations (250 onwards) to try and remove edges which are crossed over and are hence sub-optimal.

The solver was tested on a UNIX circuit board shown in Figure 3.27. This 75-component board was converted to a coordinate map shown in Figure 3.28. The solver was applied 50 times and the best route chosen. This route along with the best so far convergence curves are also shown in Figures 3.29 and 3.30.

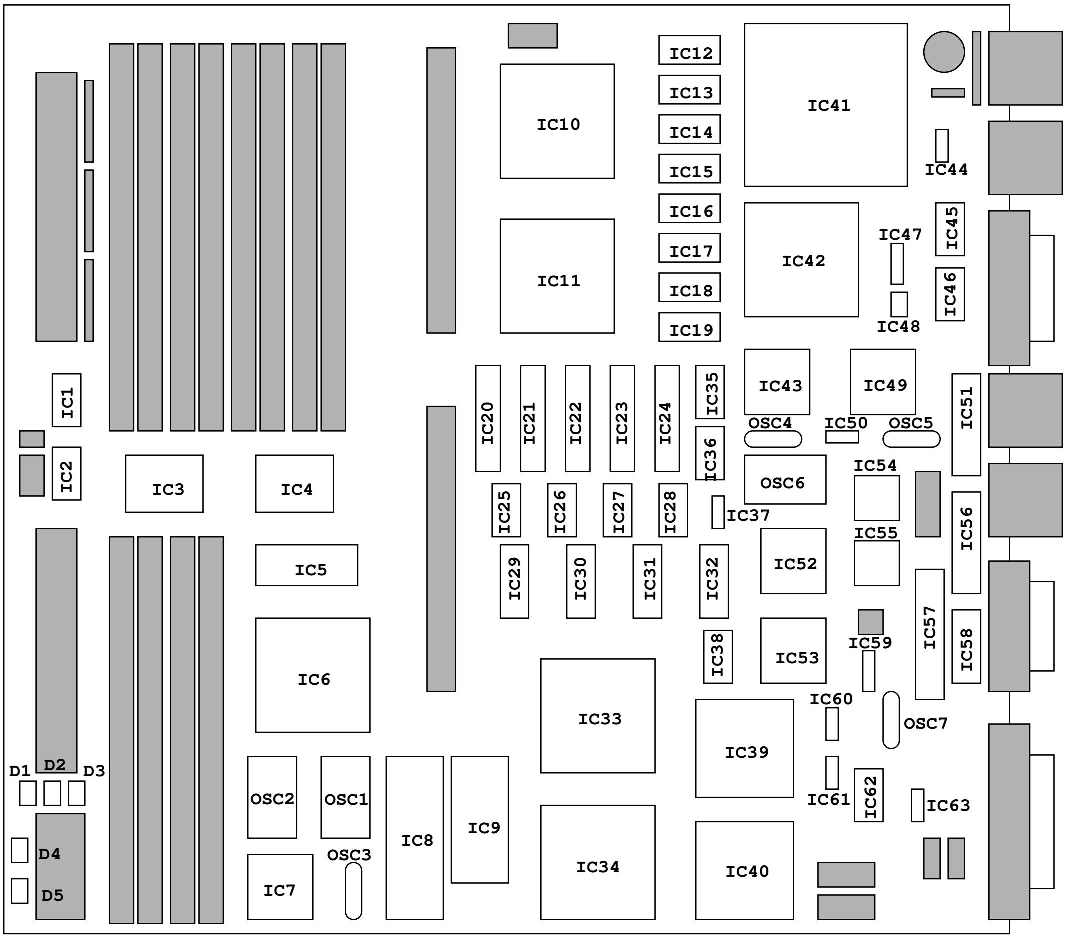


Figure 3.27 - Circuit Board Layout

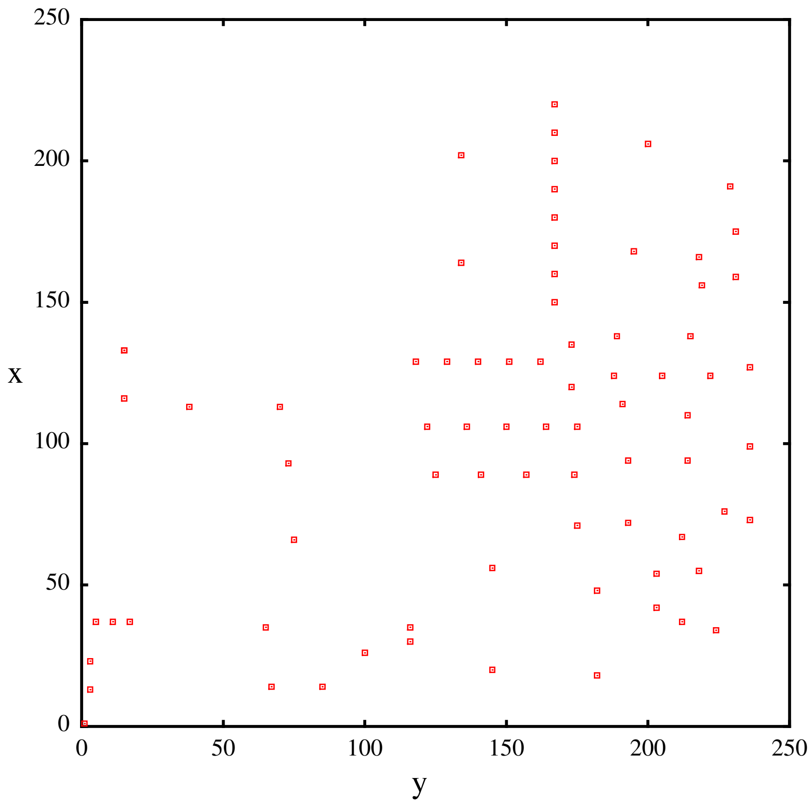


Figure 3.28 - Component Centres

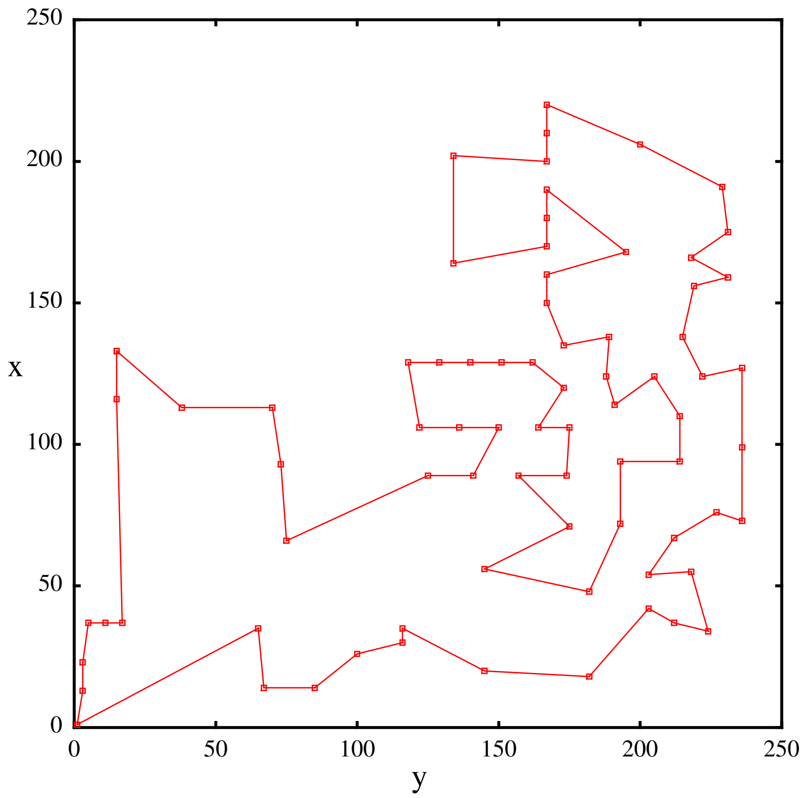


Figure 3.29 - Route Given by Solver (Length = 1594.12)

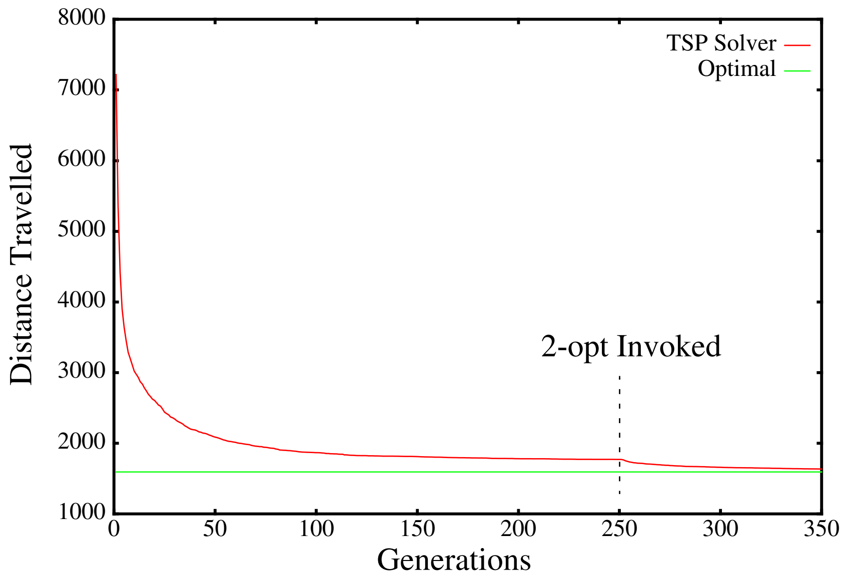


Figure 3.30 - Route Given by Solver (Length = 1594.12)

It can be seen that the solver performs very well. It finds good solutions consistently. The worst solution found over the 50 runs was 1664.2. which is a 1% error on the optimal. From Figure 3.30 it can be seen that using the improved heuristic crossover operator alone will not give such good results, because it is unlikely to achieve the optimal. The 2-opt solver invoked at 250 generations (vertical dotted line in graph) is seen to make improvements, even giving the optimal in one case.

The performance gains due to routing in terms of time for the manufacture of an individual board are often small as all manufacturers perform heuristic searches of their own, however unstructured they may be. However, for mass produced boards, the time saving accumulates giving significant production benefits. The reduced travel will also minimise the wear on the machine thereby reducing maintenance costs.

3.7 Chapter Summary

In this chapter various domain independent crossover and mutation operators were explained for reference purposes and applied to the travelling salesman problem in order to asses their effectiveness at performing combinatorial search. The aim was to find a good general operator for combinatorial type problems. Enhanced Edge Recombination Crossover and Adjacent Two City Mutation performed the best during the tests. However, these operators still struggled when problem size increased. These tests provided an essential insight into the recombinative effects which must be present to provide constructive crossover in combinatorial problems. Such effects have been used later in the thesis when defining new operators.

Using domain specific heuristic and local search did improve performance significantly. Using such techniques it is possible to gain near optimal and sometimes optimal solutions for large TSPs. However, the use of local search and heuristics does prevent the general applicability of the search. This means that for new problems new operators and local search techniques have to be implemented.

This chapter also showed the kind of efficiency gains that can be sought by using optimization methods on combinatorial engineering problems. The next chapter explains the important concepts involved in designing a Multiple Criteria Genetic Algorithm (MCGA), so that multiple objectives can be tackled simultaneously. It also describes how the basic MCGA is developed.

4 The Multiple Criteria Genetic Algorithm (MCGA)

4.1 Introduction

Having looked at using a GA to solve basic combinatorial problems in the last chapter, this chapter moves on to the development of a Multiple Criteria Genetic Algorithm (MCGA). The concepts and principles of multiple criteria problems are first outlined. This is followed by a summary of how GAs have been used to solve multiple criteria problems to date. The MCGA is then explained along with several new concepts. The MCGA is then tested on a set of 5 functions and the performance examined.

4.2 Multiple Criteria Concepts

Multiple Criteria Decision Making (MCDM) can be broken down into two distinct types, problems of selection and problems of synthesis. Selection problems involve choosing one of several possible alternatives. Synthesis involves creating solutions whose aim is to attain a set of goals. Synthesis can also be looked upon as an investigation into what is achievable within the constraints of a problem. If we have a two criteria search space, as shown in Figure 4.1, this can be split into a feasible and an infeasible region.

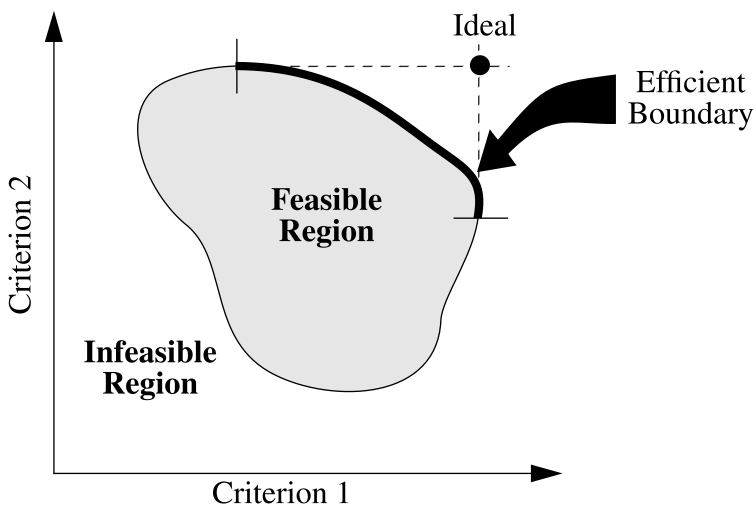


Figure 4.1 - Multiple Criteria Search

The precise form of this region will be defined by the problems, constraints and limitations. If the goal is to maximise criterion 1 and 2 we would like to generate solutions distributed across the efficient boundary. This boundary contains all points which are Pareto optimal or non-dominated in that an improved value of one criterion can only be obtained at the expense of the other criterion. This condition is only satisfied on the efficient boundary.

In order to generate this boundary a general method must be found. For this purpose the GA was investigated and the Multiple Criteria Genetic Algorithm (MCGA) developed. If it is possible to generate this set of efficient solution points the synthesis problem becomes one of selection as no solution dominates another on this boundary, as observed earlier. At this point the underlying conflict can be resolved by one of a number of methods further investigated in Chapter 9.

The efficient boundary can be defined as either concave or convex. This is dependent upon the direction of optimal search. However most problems in real life operate in a region of diminishing returns and this inevitably leads to convexity. Figure 4.2 shows this for the various cases of optimisation. There are a total of four cases for a two criteria problem, eight for three criteria, sixteen for four criteria and so on.

The surface may not necessarily be simply convex or concave but a complex combination of the two. As dimensionality increases it is also possible to have concave surfaces following convex paths and vice versa. In complex cases the shape of the surface will also define which points are Pareto optimal. This can be clearly seen in Figure 4.3 where Pareto dominance can still be used to identify dominated solutions. In other words, even for points on the edge of the feasible region some sections on this surface may be surpassed by other points on the surface. The dashed lines show this.

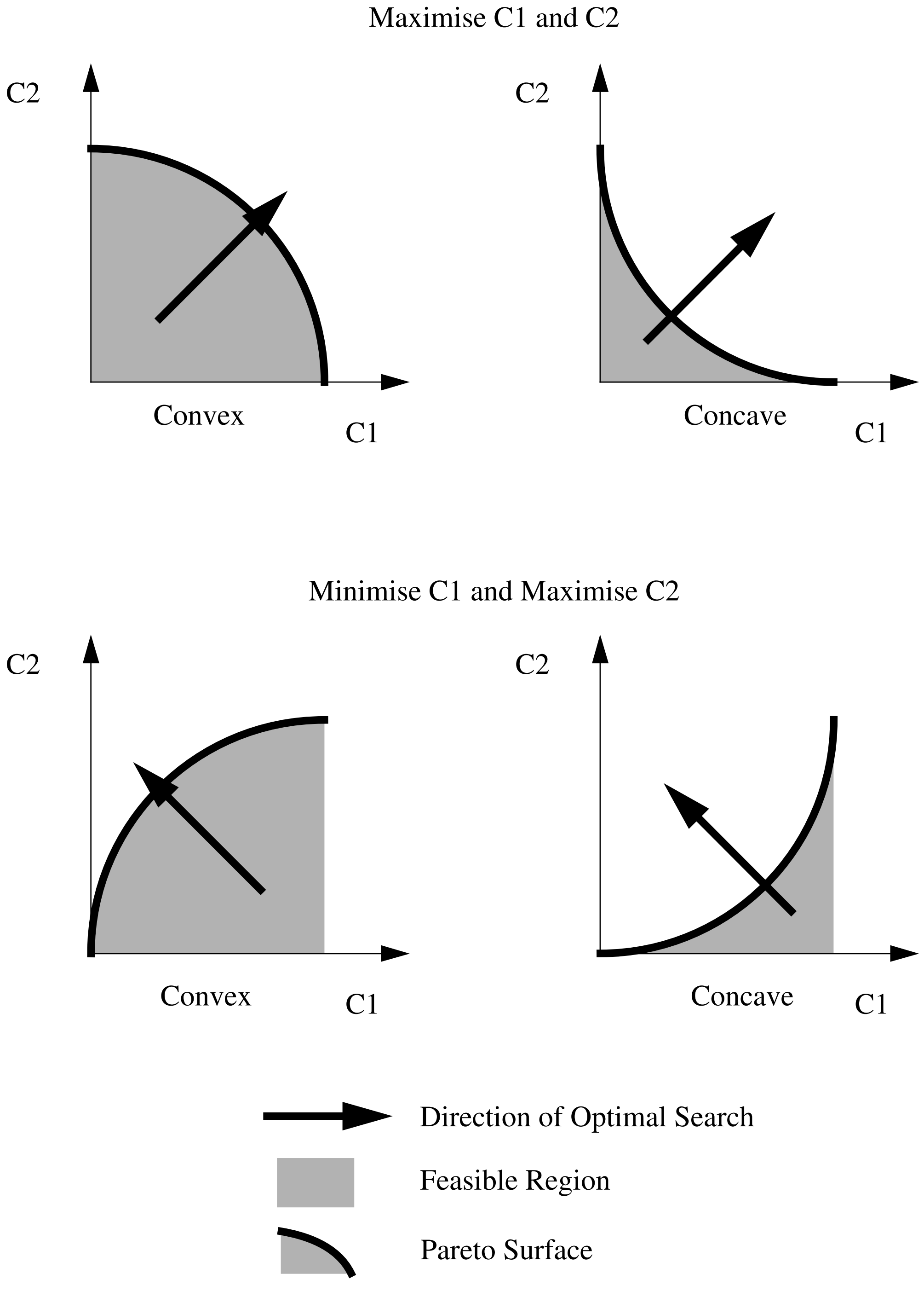


Figure 4.2 - 2D Convex and Concave Surfaces

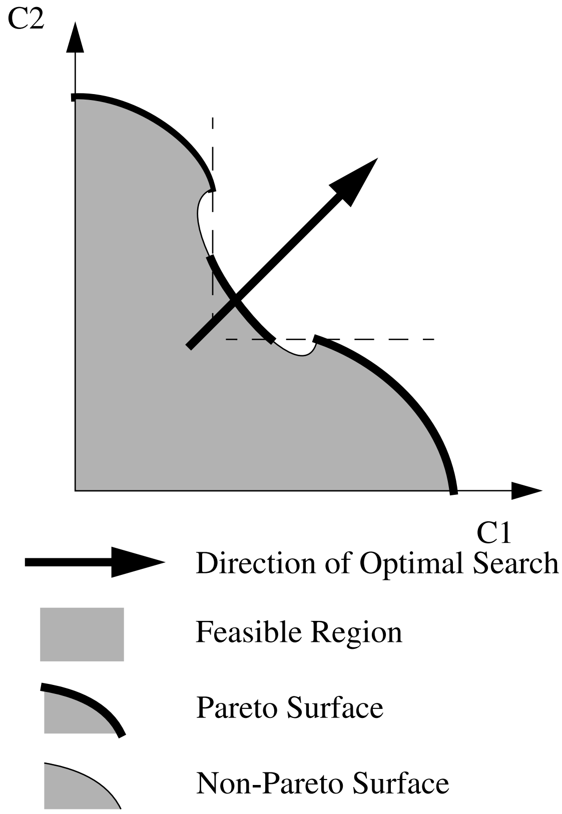


Figure 4.3 - A More Complex 2D Surface

All multiobjective problems can be reduced to single objective ones, either directly or indirectly. The cases in Figure 4.4 show some of the different types. If knowledge about the search space is available and it is of the form shown in case (i) then this optimisation is not multiobjective as there is no conflict between points and the dominance relationships are clear. This problem can therefore be solved by choosing either criteria. In cases (ii), (iii) and (iv) there are solutions which conflict in some way. Case (ii) is a combination of a single and multiple criteria problem. In case (iii) even though there is only one optimal solution, some resolution of multiple criteria might need to take place to reach this solution. In the final case (v) if no knowledge of the search space is available it must be assumed to be a multiple criteria problem. If it turns out to be similar to case (i) it will reduce to a single criterion problem.

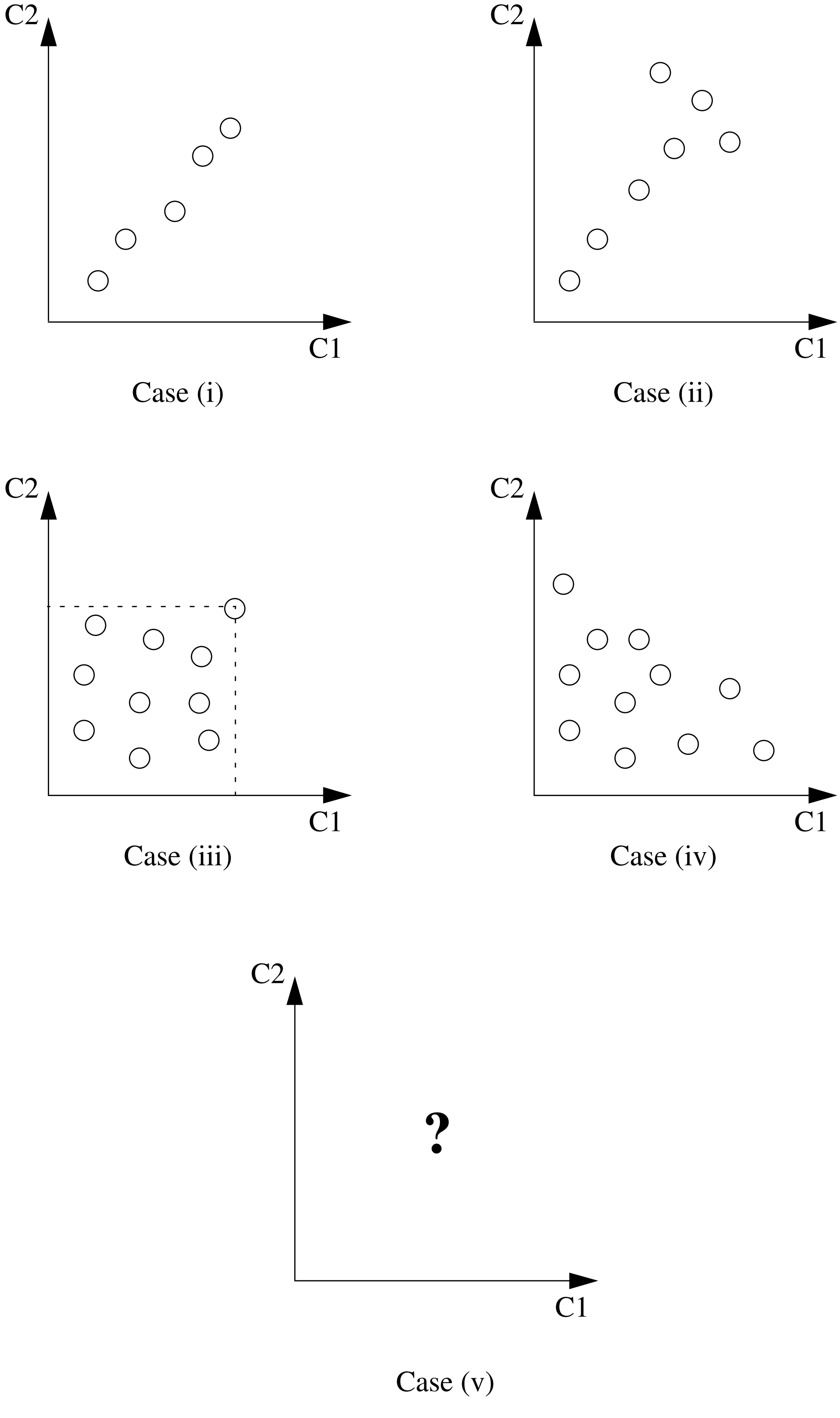


Figure 4.4 - Types of Search space.

4.3 Genetic Algorithms for Multiple Criteria Problems

The concept of using GAs for multiple objective problems is not a new one. Commonly these types of problems have often been tackled by using a weighted sum of objectives. This method gives only one solution and no indication of the trade-offs involved. However, different criteria are typically non-commensurate preventing meaningful combination within such a sum. The Vector Evaluated Genetic Algorithm (VEGA) (SCHAFFER 85) was the first attempt made at extending the GA into the multiple objective domain. Schaffer used a special selection mechanism which chose n equally sized subgroups of individuals from the population based on their performance in each of n criteria. These subgroups were then shuffled together and genetic operators applied. It was recognised that this would favour solutions with extreme performance in at least one objective. To combat this Schaffer suggested applying fitness penalties to locally dominated points and redistributing the deducted fitnesses to non-dominated ones. He found that this caused premature convergence because in populations with few non-dominated points these points were given large fitness values. It was also suggested that individuals performing well in one criterion should be mated with individuals performing well in others. Unfortunately this was found to have a detrimental effect. For this reason random mating was used throughout Schaffer's experiments.

Later analysis of VEGA's performance showed that fitness was effectively linear combination of criteria where the weights in this linear combination were defined by the distribution of the population (RICHARDSON 89). Due to the nature of selection, which was biased towards strings which were strong in at least one criteria, the population contained many extreme individuals and few compromise ones. This is especially the case in concave trade off surfaces where the population moves out to the extremes of the trade-off surface.

Another approach is the use of an independent sampling method. Using this method a front is built up using a series of independent runs with the weightings between criteria being varied. Fourman (FOURMAN 85) used several formulae to search for the trade-off surface systematically. Ritzel (RITZEL 94) and Cieniawski (CIENIAWSKI 95) applied constrained and weighted factor methods respectively to water resource management problems. While Tsoi (TSOI 95) and Chang (CHANG 95) used a factor and varied it between 0 and 1 to change the relative criteria weightings. However, because these methods are linear aggregative, they are biased towards convex proportions of the trade-off surface.

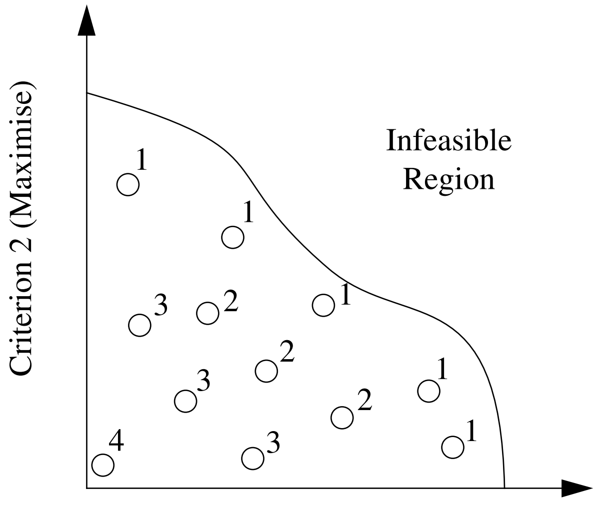
Other methods have used selection, like the approach taken by Schaffer (SCHAFFER 85), to select a range of solutions. Kursawe (KURSAWE 91) used a selection method with random criteria and a crowding mechanism (DE JONG 75) to maintain diversity, while Hajela (HAJELA 92) selected a set of sub-populations based on different weightings. All of these methods have been criticised for bias towards extremes based on previous observations (RICHARDSON 89, GOLDBERG 89). Murata (MURATA 96) proved this fact empirically and implemented a procedure which could find concave fronts using random weights to select pairs of strings.

Since the publication of Goldberg's book (GOLDBERG 89) in which he defines a method called non-dominating sorting, the direction of research has changed. Techniques have started to use the concept of Pareto optimality though selection and ranking methods.

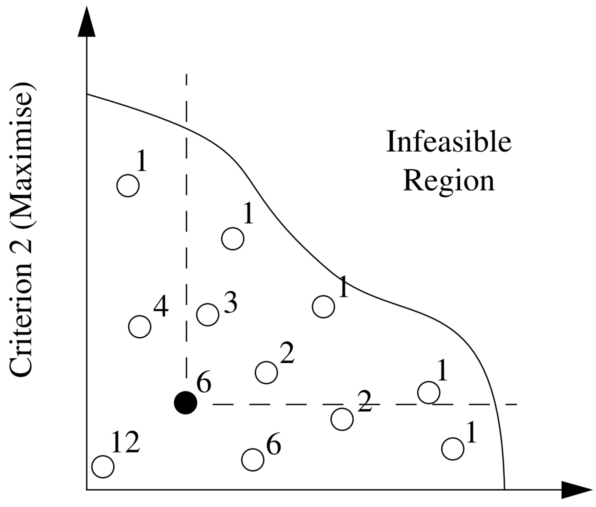
The selection methods use the concept of Pareto optimality to select individuals via an elitist or tournament selection procedure. In the elitist method Pareto individuals are passed directly into the next generation or used in crossover and mating (BELEGUNDU 94). Using tournament selection Pareto competitions are held between parents and their offspring and the two winning strings propagate to the next generation (LOUIS 93). Gero applied this type of technique to shape optimization (GERO 95).

Ranking methods are used to grade the population in terms of Pareto dominance. Goldberg suggested non-dominated sorting (GOLDBERG 89). This method involves first finding all of the Pareto optimal points within a population, giving them a rank of one and removing them. The remaining population members are again processed to find non dominated individuals and these are given rank two and removed and so on until all of the population has been ranked. Both Richardson (RICHARDSON 90) and

Ritzel (RITZEL 94) implement this type of ranking. Fonseca (FONSECA 93) outlined another ranking scheme called multiobjective ranking. In this scheme each individual is given a rank according to how many individuals dominate it. If an individual is non-dominated it is given a rank of 1, but if five individuals dominate it then a rank of 6 is assigned and so on. This method gives a greater range of ranks and will also penalise areas of high density solutions. Both of these ranking methods are shown in Figure 4.5 for a given spread of points.



Non-Dominated Sorting (GOLDBERG 89)



Multiobjective Ranking (FONSECA 93)

Figure 4.5 - Ranking Methods

One of the aims of multiobjective optimization is to generate and maintain a diverse range of Pareto optimal points. In order to achieve this diversity a technique originally used for single criteria GAs is used. Fitness Sharing or Niching (GOLDBERG 87) ensures that the population does not all migrate to localised regions of the search space. Goldberg originally demonstrated this technique in the context of finding multiple maxima/minima in a multimodal search space. This is done by trying to encourage individuals to move away from crowded areas by evenly sharing what resource is available among all individuals within that local area. In its simplest form, if an individual has a fitness value of 1 and it has n other individuals within its close proximity (i.e within its niche), it will be given a new fitness value of 1/n. The size of the niche must be specified before the search begins.

Niching has been employed in several multiple criteria approaches. The Multiple Objective Genetic Algorithm (MOGA) uses multiobjective ranking and performs fitness sharing between sets of solutions of the same rank (FONSECA 93). The Niched Pareto Genetic Algorithm (NPGA) holds a type of binary tournament selection called a Pareto domination tournament (HORN 93). In this tournament the two strings are compared to a sample of the population and the number of dominating points in the sample is counted for both individuals. The individual with the least number of dominating points survives. If both points have the same number then the one with the lower niche count survives. The size of the sample used can be varied depending on the level of selection pressure required. Obviously larger size samples will increase the bias towards stronger solutions. This method is analogous to performing a local version of Fonseca's ranking method. Both MOGA and NPGA perform sharing within criteria space. A third method, the Non-dominated Sorting Genetic Algorithm (NSGA) (SRINIVAS 95) uses Goldberg's (GOLDBERG 89) ranking method. However, sharing is performed in the phenotypic space by measuring the vector distance between the decoded design variables.

Mating restriction (DEB 89) is used in MOGA (FONSECA 93) to ensure profitable search. Mating restriction is also used to ensure that individuals that are far away from each other do not mate. This can form what are called 'lethals' which are low fitness individuals formed from two high fitness individuals which have dissimilar properties.

Like fitness sharing, a niche size is defined within which a suitable partner is allowed to mate. If a suitable partner cannot be found a random partner is chosen.

4.3.1 Estimating Niche Sizes

The size of the niche, defined by σ_{share}, used for fitness sharing and mating restriction must be estimated before search commences. Deb (DEB 89) in his sharing mechanism for single criteria problems assumes some knowledge of the search space before search starts. Deb's sharing distributes the population in each optimal region in proportion to that optima's fitness. In multiobjective space Pareto solutions are all equally fit so the aim is to generate an even spread of points over the trade off surface. Fonseca (FONSECA 93) outlines a method for estimating niche sizes for multiple criteria problems.

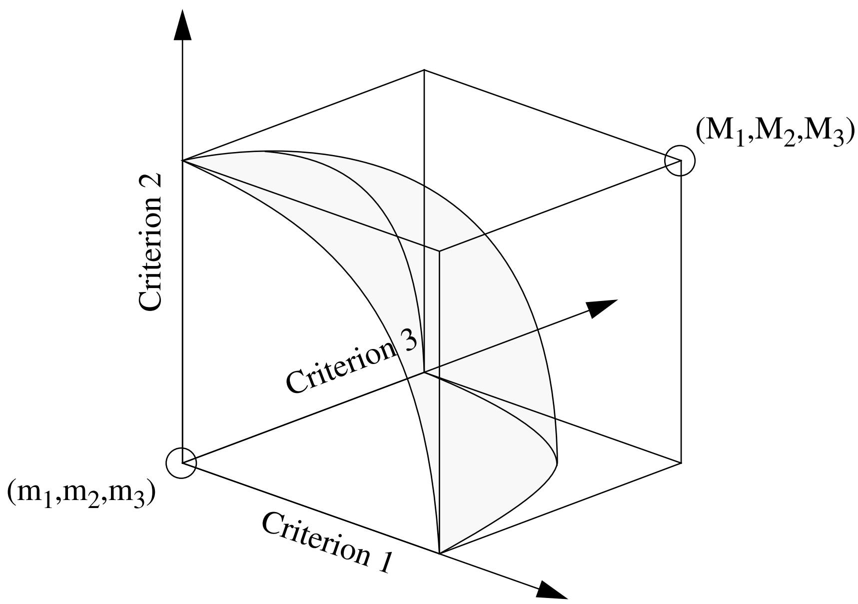


Figure 4.6 - The Trade-off Surface

σ_{share} defines how close individuals need to be to get penalised. In a multiple criteria context this is defined for each orthogonal direction. The number of individuals needed to describe the surface is dependent on the size of the surface. Figure 4.6 shows how the Pareto surface in the objective domain is bounded for all criteria by maximum and minimum values. From the diagram the ranges of the criteria are :

$$m_1 < \text{Criteria } 1 < M_1,\ m_2 < \text{Criteria } 2 < M_2,\ m_3 < \text{Criteria } 3 < M_3.$$

The hyperarea of this surface will be less than the sum of the areas of each face given by:

$$(M_1 - m_1) \times (M_2 - m_2) + (M_1 - m_1) \times (M_3 - m_3) + (M_2 - m_2) \times (M_3 - M_3)$$

This is shown in Figure 4.7. In general terms for q criteria this hyperarea will be:

$$\text{Hyperarea} = \sum_{i=1}^{q} \prod_{\substack{j=1 \\ j \neq i}}^{q} (M_j - m_j)$$

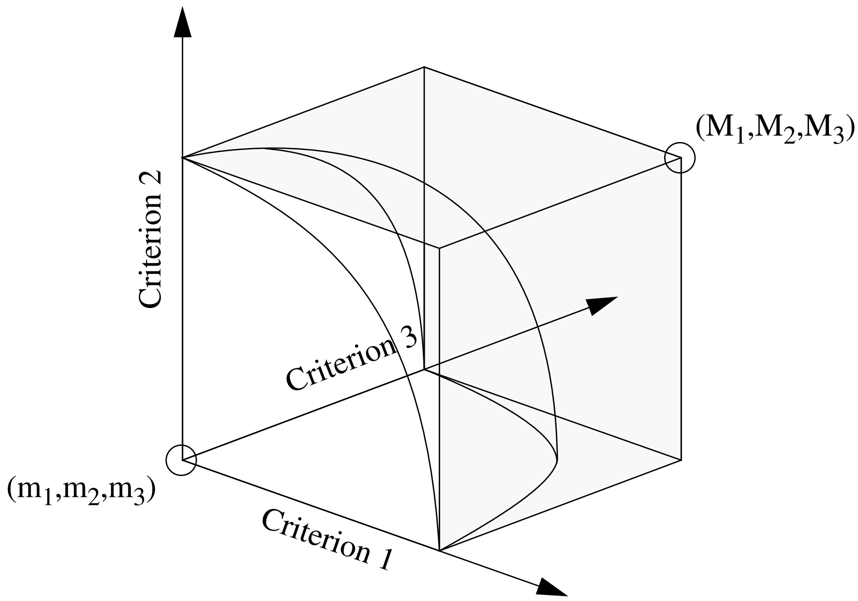


Figure 4.7 - Upper Bound Area Trade-off Surface

Fonseca then defines the number of hypercubes, N, of size σ_{share} and volume $\sigma^q{}_{share}$ to be (see also Figure 4.8):

$$N = \frac{\prod_{i=1}^{q} (M_i - m_i + \sigma_{share}) - \prod_{i=1}^{q} (M_i - m_i)}{\sigma^q{}_{share}}$$

In order to find σ_{share} the following (q-1)th order polynomial equation has to be solved:

$$N\sigma^{q-1}{}_{share} - \frac{\prod_{i=1}^{q} (M_i - m_i + \sigma_{share}) - \prod_{i=1}^{q} (M_i - m_i)}{\sigma_{share}} = 0$$

for $\sigma_{share} > 0$.

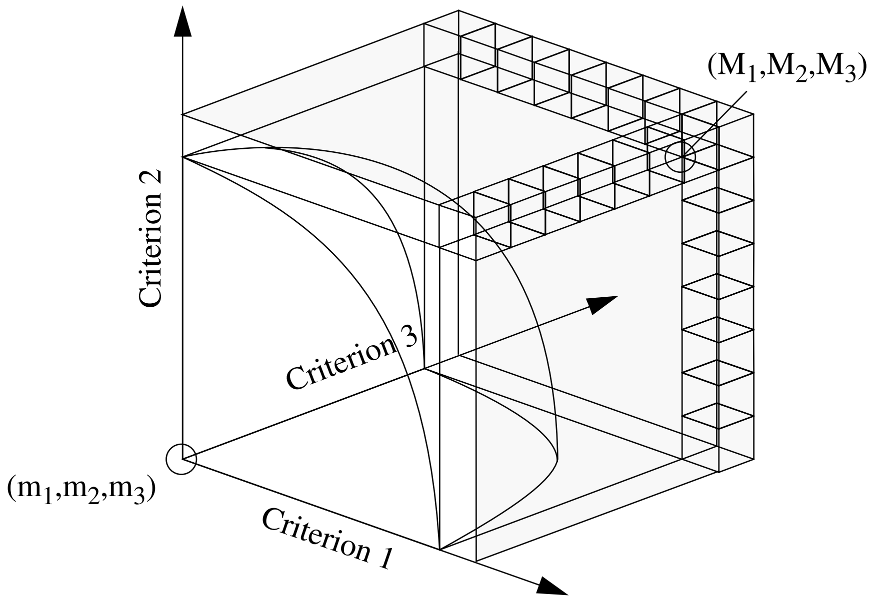


Figure 4.8 - Hypercubes Defining the Trade-off Surface

4.3.2 A New Approach To Niche Sizing

Instead of having to solve this polynomial equation a simpler method to find σ_{share} has been derived. Figure 4.9 shows the number of points needed to define a 2D Pareto surface. As we approach the limit, i.e. the Pareto surface tends towards two sides of a square, this gives the maximum number of points needed to define the surface.

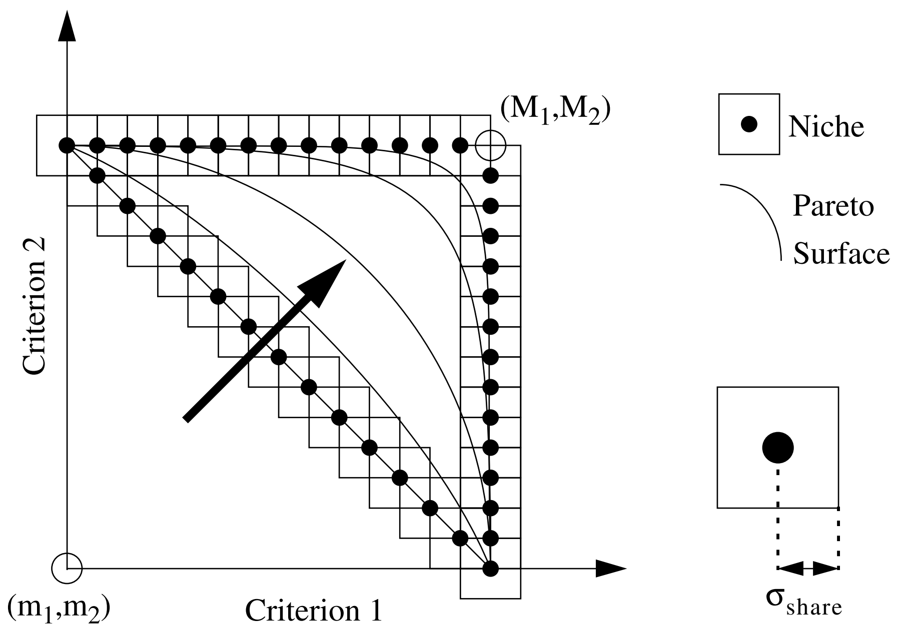


Figure 4.9 - Defining the Pareto Surface

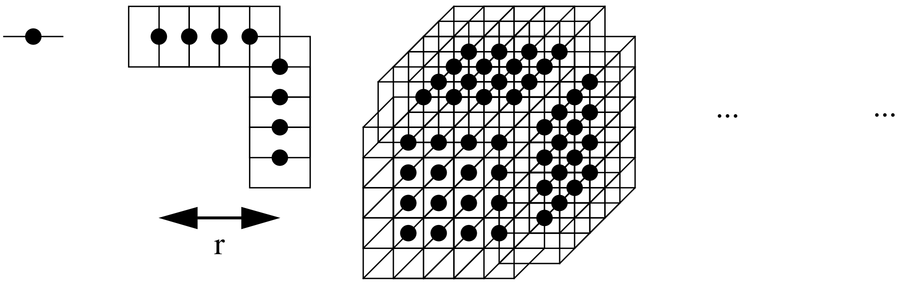
In this case every solution has no other within the boundary of its niche (defined by σ_{share}). The dimensions of the surface are still defined by (M_i-m_i) for each of the q criteria. Now instead of defining either a value for σ_{share} or N, the level of resolution, r, required along the criteria axes is defined.

Number of Criteria:

1	2	3	4	i

Number of Hypercubes:

1	2r	$3r^2$	$4r^3$	ir^{i-1}



Number of Hypercubes (Resolution, r = 4):

1	8	48	256	$i4^{i-1}$

Figure 4.10 - Defining Hypercubes

This method is shown in Figure 4.10 for r = 4. For a desired resolution of r, the size of σ_{share} for criteria i is given by:

$$\sigma_{ishare} = \frac{M_i - m_i}{r}$$

Additionally, the minimum number of points required to specify the surface is given by :

$$N = ir^{i-1}$$

This method allows the niche dimensions to be easily found for all criteria. It allows the user to specify the resolution and also provides a quick estimate for the number of points required to define the surface.

4.4 Initial Implementation

An examination of the functional requirements of a Multiple Criteria GA indicated seven fundamental elements that needed to be addressed. These elements are:

(i) A String Representation
(ii) An evaluation Mechanism for n criteria
(iii) A Ranking Scheme
(iv) A Sharing Scheme
(v) A Pareto Selection Mechanism
(vi) A Restricted Crossover Mechanism
(vii) A Mutation Mechanism

An initial version of the MCGA was implemented in C++ in order to test its behaviour on a set of multiple criteria functions. The system devised uses a binary notation and evaluates the strings by decoding them into variables. Fonseca's (FONSECA 93) ranking scheme is used and is converted into a fitness for each individual(i) through the following equation:

$$\text{fitness} = (PS+1 - R(i))^a$$

Where PS=Population Size, R(i)=Rank of individual i and a=control factor to adjust selection pressure(set to 1 for these tests).

Sharing was performed on the population by estimating the values of m_i and M_i for each criteria. This was simple as the test functions in Section 4.5 are well defined. Then a resolution r was specified and used to estimate a reasonable population size. Sharing was performed on every individual by counting the number of other points within its niche and dividing its fitness by this count. This is shown in Figure 4.11. If the point being penalised has fitness F, then if there are n individuals in its niche, it's new fitness after sharing will be $F_{new} = F/n$.

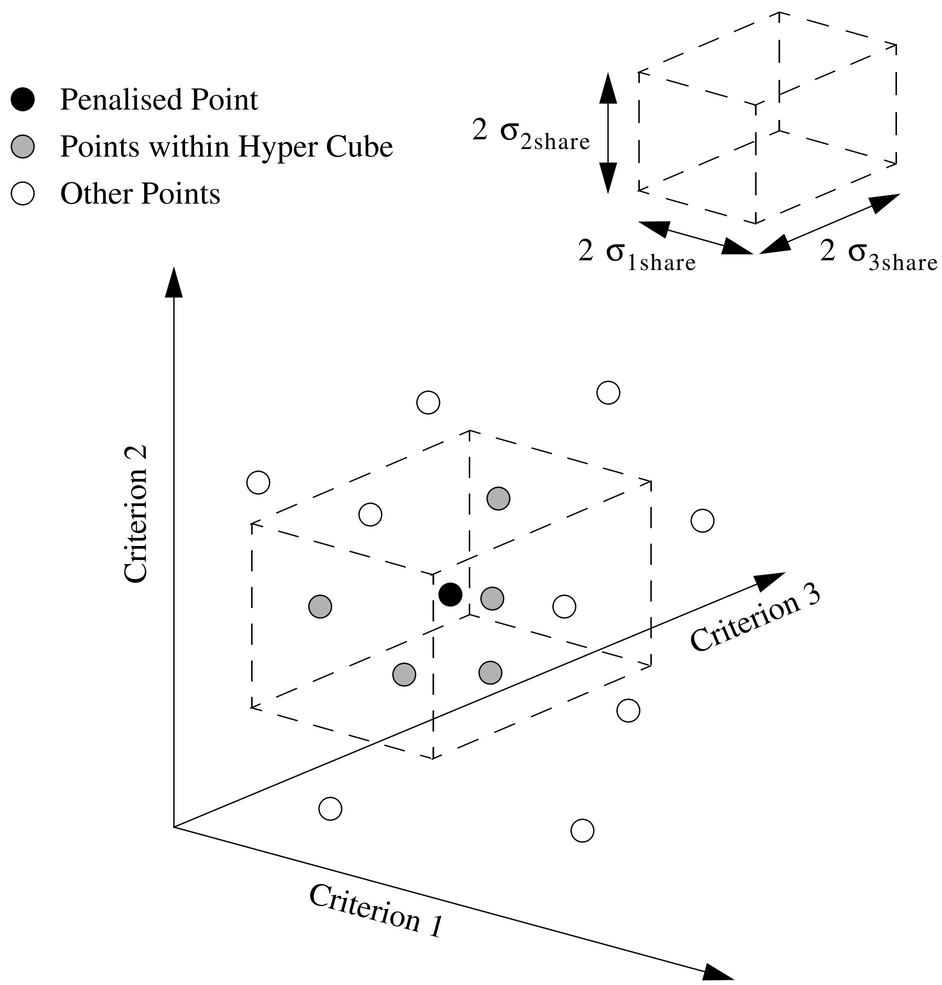


Figure 4.11 - Fitness Sharing

A Pareto elitist mechanism (BELEGUNDU 94) was used to pass individuals into the mating pool. This was followed by roulette selection based upon string fitness after sharing. Crossover was then performed using two point crossover. Restrictions were applied allowing only individuals in the same niche to be selected for crossover. However, if no mates were available a random mate was chosen. Simple binary mutation was then applied to the population.

4.5 Test Functions

Four test functions were used to experiment on the MCGA. The functions are based on a hill function which is overlaid several times onto the search space, depending on the number of criteria. Diagrams of the hill function and the hill centre plots for the four test functions are shown in Figure 4.12

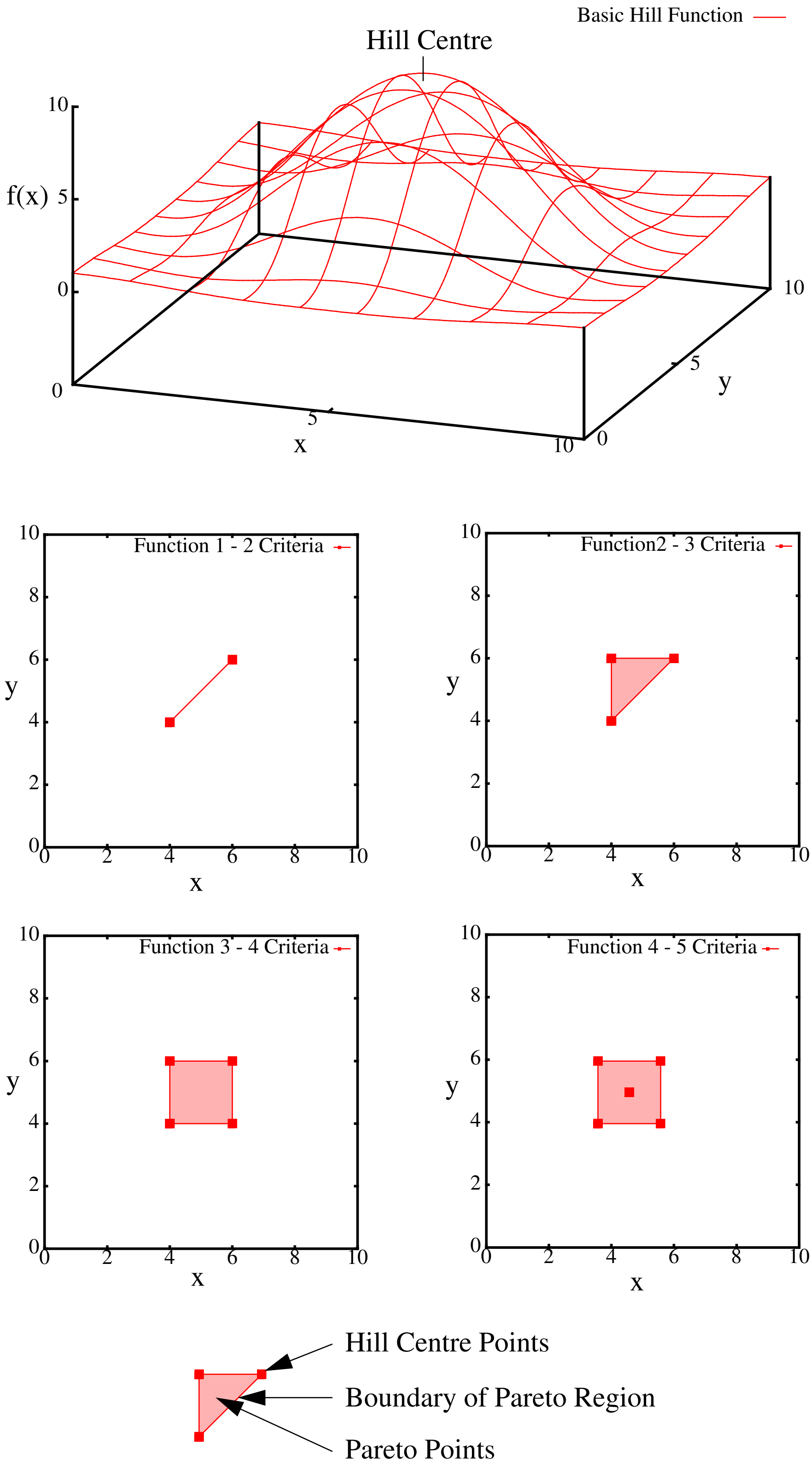


Figure 4.12 - Variable Space Plots of The Test Functions

The hill is defined by the following exponentially decaying cos function, in this case centred at 5,5:

$$F(x,y) = \cos(((x-5)^2+(y-5)^2)/10)*1/\exp(((x-5)^2+(y-5)^2)/10)*8+1$$

This hill represents one criteria in the x,y domain where $0 \le x, y \le 10$. Using several of these functions shifted around in variable space it is possible to create multiple criteria functions where each hill represents a single criteria. The four x,y plots in Figure 4.12 show the hill centres for each of the four test problems. The shaded areas show the regions of the variable space which contain the Pareto solutions. These areas are bounded by lines drawn between the hill centres. The variables x and y were each defined as 10 bit binary hence the total string length was 20.

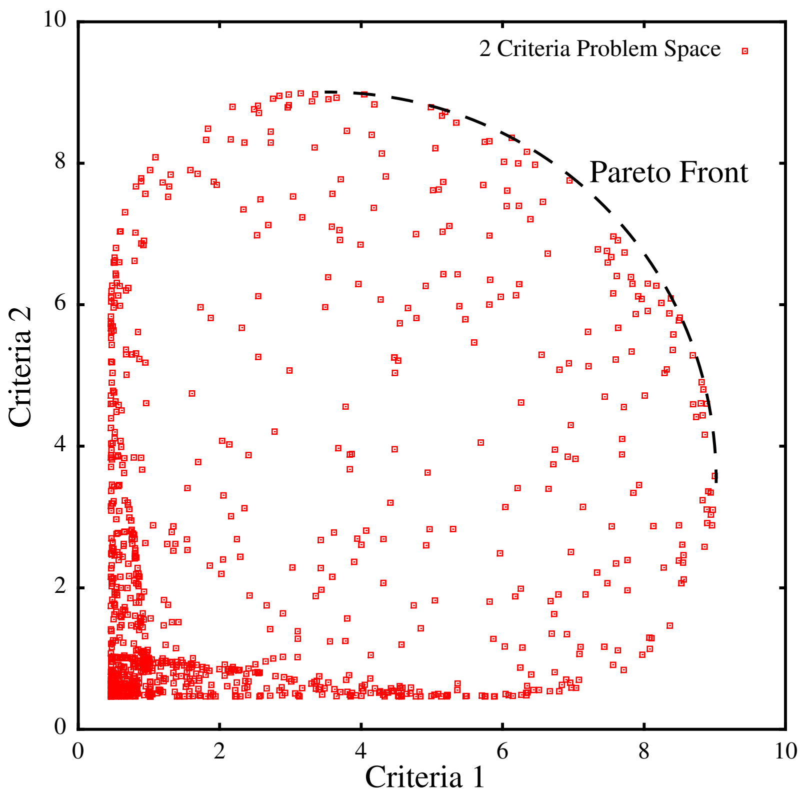


Figure 4.13 - Criteria Space Plot of Function 1

The variable space is mapped into the criteria space through the evaluation function. Figure 4.13 shows Function 1 in the test set mapped into the criteria space using 1000 randomly scattered points in the x,y domain. From this Figure it can be seen that a large proportion of the solutions are gathered near the axes. However, the aim, as with all of the test functions is to obtain the maximum of all criteria. The Pareto front is shown on the diagram by the dotted line. At the termination of the GA run the trade-off surface should be clearly defined by a distributed set of points. This mapping is easy to demonstrate in the two criteria case. However for higher dimensional spaces this becomes very difficult. For this reason the x,y variable mapping will be used to look at the distribution of solutions. As the Pareto areas are known the non-dominated solutions can be easily identified.

4.6 The Rate of Generation of Rank 1 Solutions

Having constructed a set of test functions and defined the GA initial experiments were carried out on the system. Various metrics were evaluated during the search: maximum, average and minimum for each criteria across the population, number of rank 1's in population, the variable space representation of the population and the criteria space representation of the population. Additional to this a second population, the same size as the first, was added to act as a store for rank 1 solutions. This store would be updated every generation with copies and surpassed individuals being removed. This was a suggestion originally made by Schaffer (SCHAFFER 85) but not followed through. Several authors (KURSAWE 91, CIENAWSKI 95, KRAUSE 95) did suggest a similar approaches but only used it in limited experiments.

After running the MCGA several times it was observed that this off-line population was becoming full yet the GA itself could only maintain around half the total population size as rank 1's. The number of rank 1's in the normal population could be increased by using a lower crossover rate or by passing elitist individuals through to the next generation instead of into the mating pool. However, this would reduce the diversity of search. Alternatively the population size could be increased to hold more rank 1 individuals. Figure 4.14 shows the rate of generation of rank 1's in the population itself and in the off-line population. It is clear that the number of solutions produced by the MCGA which are rank 1 is greater than the number which can be maintained within the population itself. This may not always be the case as certain problems may have only a small number or even a single solution. However, this is not known beforehand . Having generated and evaluated the solutions it is inefficient to loose good solutions through the genetic process. If the evaluation time per solution is long it becomes even more inefficient in terms of computational cost. Even if the population size is increased the number of good solutions actually generated still exceeds those that are maintained in the population.

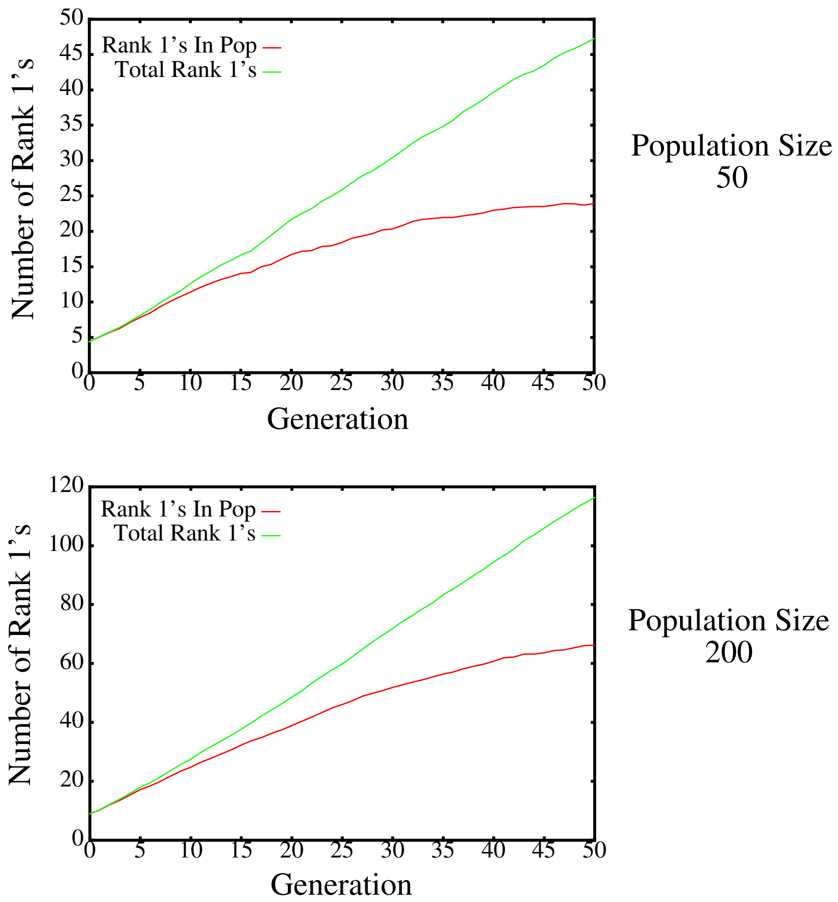


Figure 4.14 - The Rate of Generation of Rank 1 Solutions

4.7 The Pareto Population and Adaptive Niche Sizing

The loss of rank 1 solutions from the population prompted the addition of a second population called the Pareto Population. Within this population the current rank 1 individuals are stored. For every generation the rank 1 solutions from the normal population are copied and added to the Pareto Population. The Pareto Population is then genotypically checked to see if any duplicates exist. Genotypic comparison is used as more than one solution may have the same criteria value combinations. Then the Pareto Population is checked for any individuals which may be dominated and any such individuals are removed.

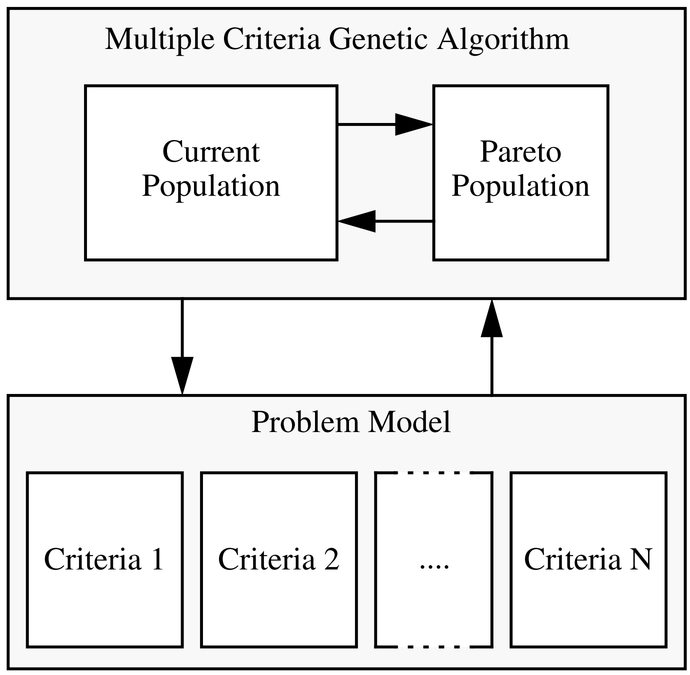
The use of a population which stores the developing front also provides some other useful information. The population as it develops can be used to analyse the rate of generation of new solutions. This could be used as a stopping criteria for search as could the average Pareto Population fitness for each criteria. However, another use for the Pareto Population has been investigated, here the population has been used to adaptively set the size of the niches used in fitness sharing and mating restriction as follows.

In order to perform sharing and mating restriction a niche size must be estimated at the start based on an estimation of the maximum and minimum of each criteria. A problem arises when no knowledge of the fitness function is available. The only way to find out more about the solution space is by performing some evaluations. However, if the search space is large or the time per evaluation is long only a restricted search can be performed and therefore only an approximate and probably unreliable estimate can be obtained.

The Pareto Population gives a snapshot of the current status of the developing front. From this front the maximum and minimum values of all criteria can be obtained and used to estimate niche size in all dimensions. Using this method the niche size is updated adaptively based on current knowledge of the Pareto front. Problems may arise if sufficient definition is not available in the Pareto Population. This is dealt with by using the normal population to control niche size until sufficient solutions exist within the Pareto Population. This method has been implemented within the MCGA. A full description of the MCGA is presented below.

4.8 The Multiple Criteria Genetic Algorithm (MCGA)

The MCGA was implemented in C++. However, full Object Orientation has not been implemented. Appendix C shows the full code structure along with the description of all the procedures. These descriptions are supplemented by structure charts illustrating the action of the system. The MCGA system is shown in diagrammatic form in Figure 4.15 together with the principal features of the algorithm (TODD 97). The MCGA System is made up of two interacting parts - the MCGA and the problem model which can be removed and replaced depending on the application area. However, due to the lack of good general operators the MCGA crossover and mutation processes may have to be changed in order to create meaningful solutions for a specific problem.



1. Create population.
2. Evaluate population on all criteria.
3. Rank population using dominance.
4. Update the Pareto Population.
5. Perform Fitness Conversion.
6. Perform Fitness Sharing.
7. Selection:
    - Step 1: Elitist strategy.
    - Step 2: Roulette Wheel Strategy.
8. Perform Restricted Crossover.
9. Perform Mutation.
10. Evaluate population on all criteria.
11. Return to 3 unless end reached.
12. Output Results

Figure 4.15 - The Multiple Criteria Genetic Algorithm

A full summary of how the MCGA operates is outlined below:

1. Create population: Randomly generate a population for the given problem.

2. Evaluate population on all criteria: Each string of the generated population is sent to the problem model then it is mapped into the form(s) used by the model. Each criteria is then evaluated and the results returned to the MCGA.

3. Rank population using dominance: Instead of dealing with a single criteria or fitness value, as is the case for standard GAs, the MCGA deals with several criteria simultaneously. The aim is not to find a single solution but to generate a set of solutions distributed along the Pareto front. The MCGA also avoids the use of combining criteria in the form of weighted sums, which leads to single preferred solutions and provides no information about the search space. Additionally, by maintaining this separation it is possible to both maximise and minimise individual criteria within a single optimisation. In order to reduce the problem into a single fitness measure the MCGA uses a concept called dominance defined in section 4.3.

4. Update the Pareto Population: The Pareto Population allows the MCGA to maintain a full set of currently non-dominated solutions, including ones which have been lost from the normal population. The Pareto Population is updated every generation with the new non-dominated solutions of the current population. The population is ranked and any duplicates or dominated solutions, which have been surpassed by newly discovered solutions, are removed. This population is also utilised later within the fitness sharing, selection and crossover procedures.

5. Perform fitness conversion: The rank will hold a value between 1 (non-dominated) and population size (totally dominated). This value can be converted directly to a fitness value as explained previously (Section 4.4.) by the equation: fitness = $(PS+1 - R(i))^a$. Where PS=Population Size, R(i)=Rank of individual i and a=control factor to adjust selection pressure (set to 1 for these tests).

6. Perform fitness sharing: This is done by forming a hyper-cuboid in criteria space around a given point and penalising that point relative to the number of points in its surrounding cuboid. This procedure is carried out on all points.

The most important factor in implementing sharing is choosing the appropriate niche size; that is, the length of each of the sides of the cuboid. Sharing once again utilises the Pareto Population. Using this population the maximum and minimum values of each criteria can be found and the niche size for each criterion calculated by subtracting the minimum from the maximum and dividing by the resolution(r) required to define the front.

7. Selection: Selection is a two step strategy.
Step 1-The pareto individuals within the population are passed directly through to the mating pool.
Step 2-The remainder of the mating pool is filled from the current population using roulette wheel selection.

8. Perform restricted crossover: This operator has to be restricted using the same technique used in fitness sharing. Any string can only mate with strings in the confines of the surrounding hyper-cuboid. This restriction is used to prevent individuals which are a large distance apart in criteria space from crossing together. Once again the sides of the cuboid are related by the maximum and minimum values of the criteria in the Pareto Population.

9. Perform mutation: Apply mutation operator to population.

10.Evaluate population on all criteria: Following these operators the new population is re-evaluated.

11. Return to 3 unless end reached: The process is repeated until a specified stopping criteria is reached.

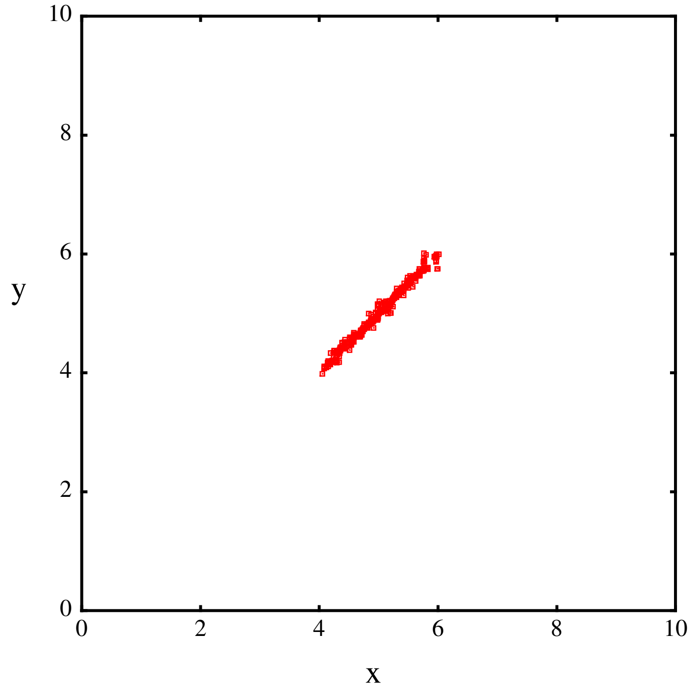
12. Output Results: A list of Pareto optimal solutions is displayed. From this list solutions which most closely meet the user's requirements can be selected. This may be done using some multi-attribute decision making tools (See Chapter 9). The selected solutions can then be output and modified if necessary.

4.9 Test Function Performance

The MCGA described in the previous section was evaluated by examining its performance and characteristics on the four test functions described earlier in Section 4.5. In all tests Pc = 0.7 and Pm = 0.01. Population Size was set to 500 and the MCGA was executed for 100 generations. This is 50,000 evaluations or 4.8% of the total search space of 1024^2 (10 bits are used to define each variable). Resolutions (r) were set at 250, 12.9, 5 and 3.17 derived from $N = ir^{i-1}$ by inserting N = 500 and finding r for each value of i. The runs for each test function were carried out 50 times giving averages, maxima and minima of the both populations. Additionally, the number of rank 1's in both populations were also tracked. Runs were also carried out to give graphs of criteria values and x-y values of each generation. and the x-y plots were then converted into distribution graphs.

Figures 4.16-4.19 show the final location of the members of the Pareto Population and their distributions over the x-y variable space for each test function. It is clear from the plots that the MCGA is able to identify and investigate the Pareto regions of the search space successfully. In all cases the Pareto Population defines the Pareto region in the x-y domain with few points outside. Only in Figure 4.19 does the MCGA have problems defining part of the x-y Pareto space. In Function 4 the fifth criteria which is centred on coordinates 5,5 appears to have deceived the GA and one of the quadrants of the square has been left under defined. This suggests that genetic drift has caused the population to under allocate resources to this region. This occurrence is investigated in greater detail later in Section 4.10.

The distribution graphs were generated by counting the number of individuals within a bounded region of the search space for various locations across the whole surface. The Pareto regions are again clearly defined. In the ideal case the distribution of Pareto solutions over the whole region should be even and all peaks should be the same height. This is not the case as some regions have been sampled more than others. This is largely dependent on the initial distribution of the population. The MCGA works by taking the good members of the initial population and using these to generate more members by moving outwards from these points. This process occurs because of the local mating generating different but similar individuals from two parents. If there are many parents in one area this is likely to generate more children in that area.



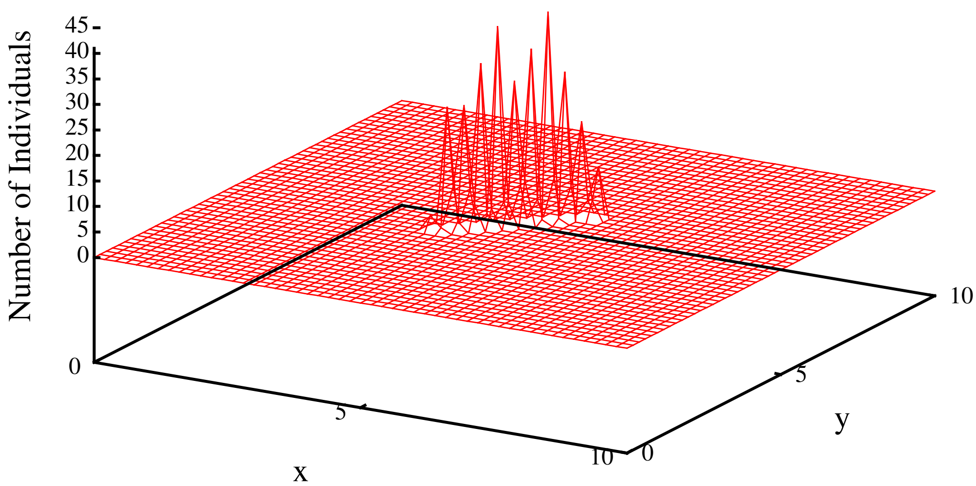


Figure 4.16 - Function 1 X-Y and Distribution Graphs (Generation 100)

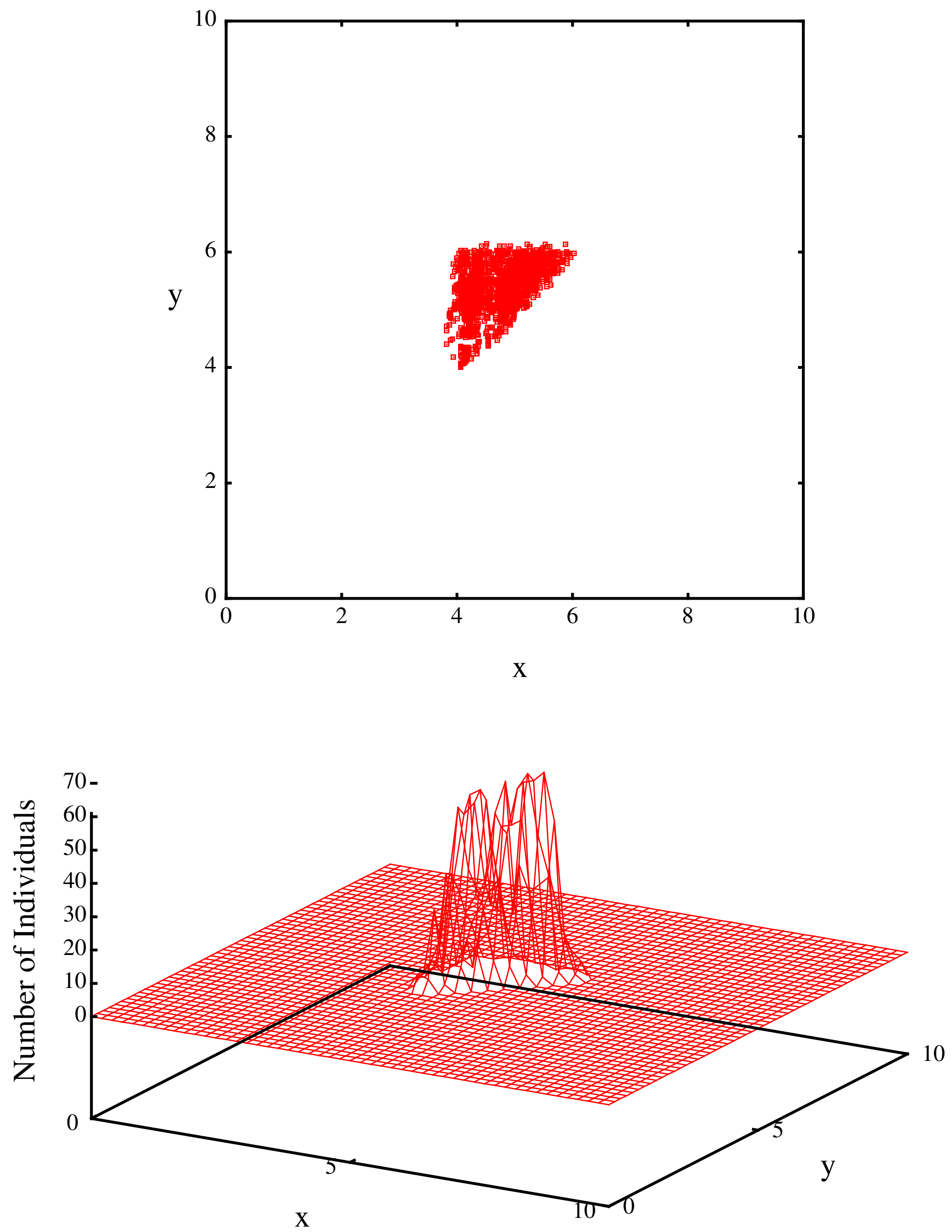


Figure 4.17 - Function 2 X-Y and Distribution Graphs (Generation 100)

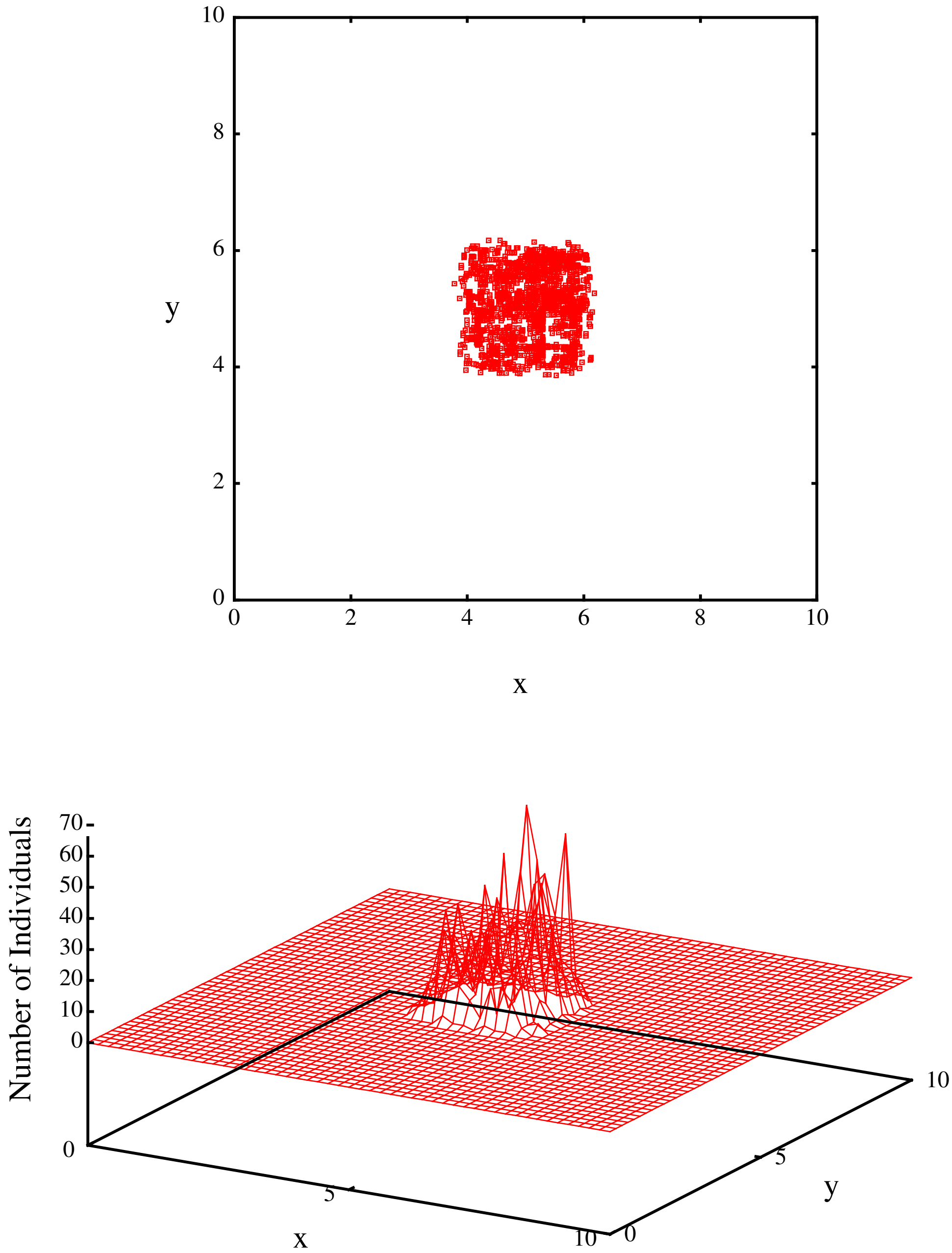


Figure 4.18 - Function 3 X-Y and Distribution Graphs (Generation 100)

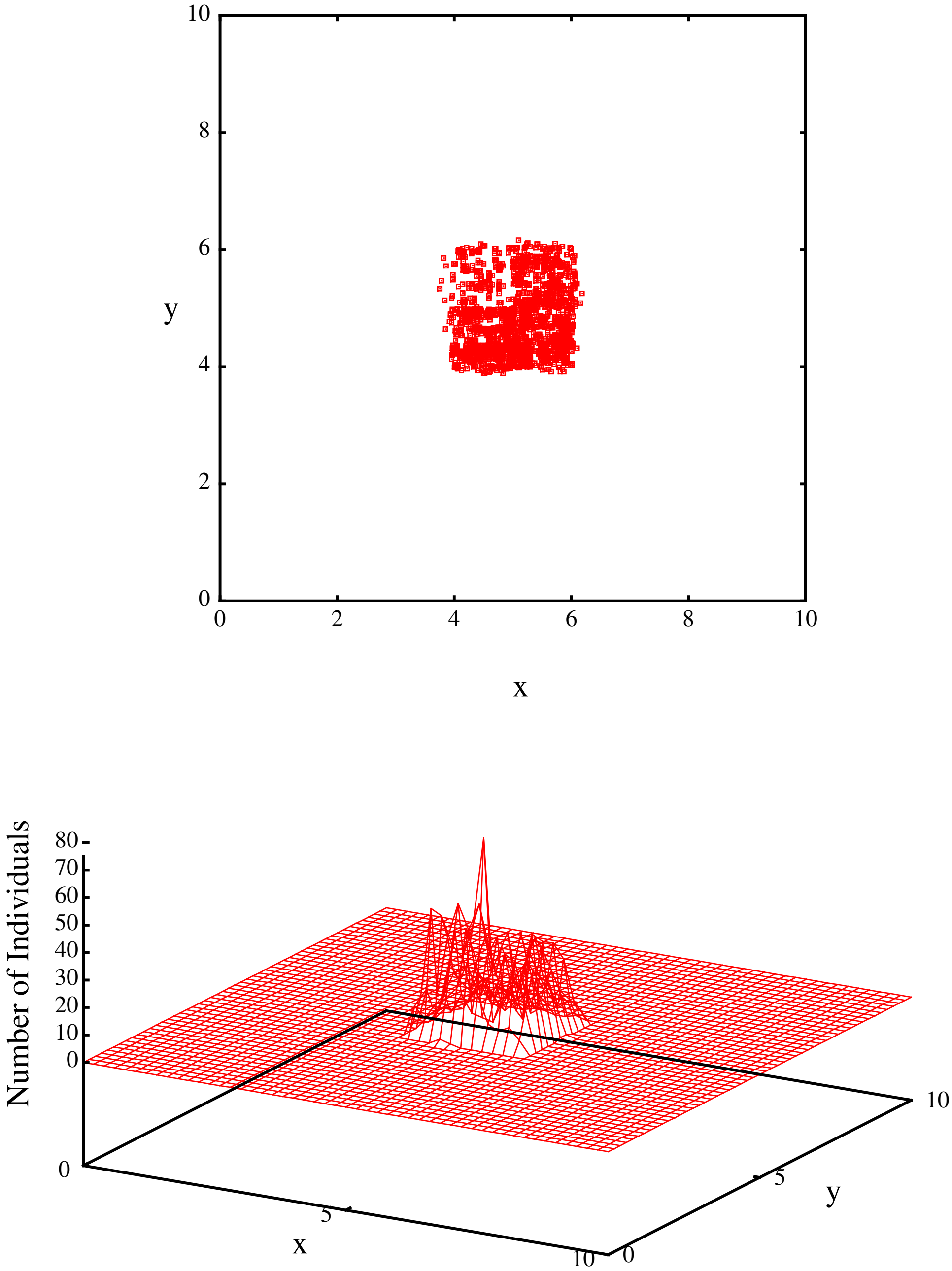


Figure 4.19 - Function 4 X-Y and Distribution Graphs (Generation 100)

Fitness sharing forces movement of the normal population outwards from these points into other areas of the search space. This is shown by the fact that points along the whole front have been discovered. The plots demonstrating this effect in the criteria domain for Function 1 are shown in Figure 4.20.

In the initial population the individuals in the normal population are spread over criteria space with only a few individuals near the Pareto front. Most of the individuals are gathered in the low fitness region at the bottom left of the plot. By generation 10 the population has increasingly utilised and propagated the better solutions through genetic search. The definition of the Pareto front is beginning to emerge, however gaps still exist within the front which need to be explored and filled. It can also be seen that in certain regions noticeable clusters have formed (indicated by the circles) where like individuals have generated others with similar properties through restricted crossover. These areas appear as high peaks on the dstribution graph (Figure4.21).

The final plot at generation 50 shows how fitness sharing and mating restriction have forced the population to spread over the whole Pareto front. There are very few individuals in the population not in the Pareto set, this means that the adaptive niche sizing has performed correctly and set the niche size to allow a diverse population to form and maintain its stability. As crossover takes place there is the possibility of losing good Pareto solutions from the population, but the Pareto Population ensures that although solutions may be lost from the normal population a copy will be maintained permanently.

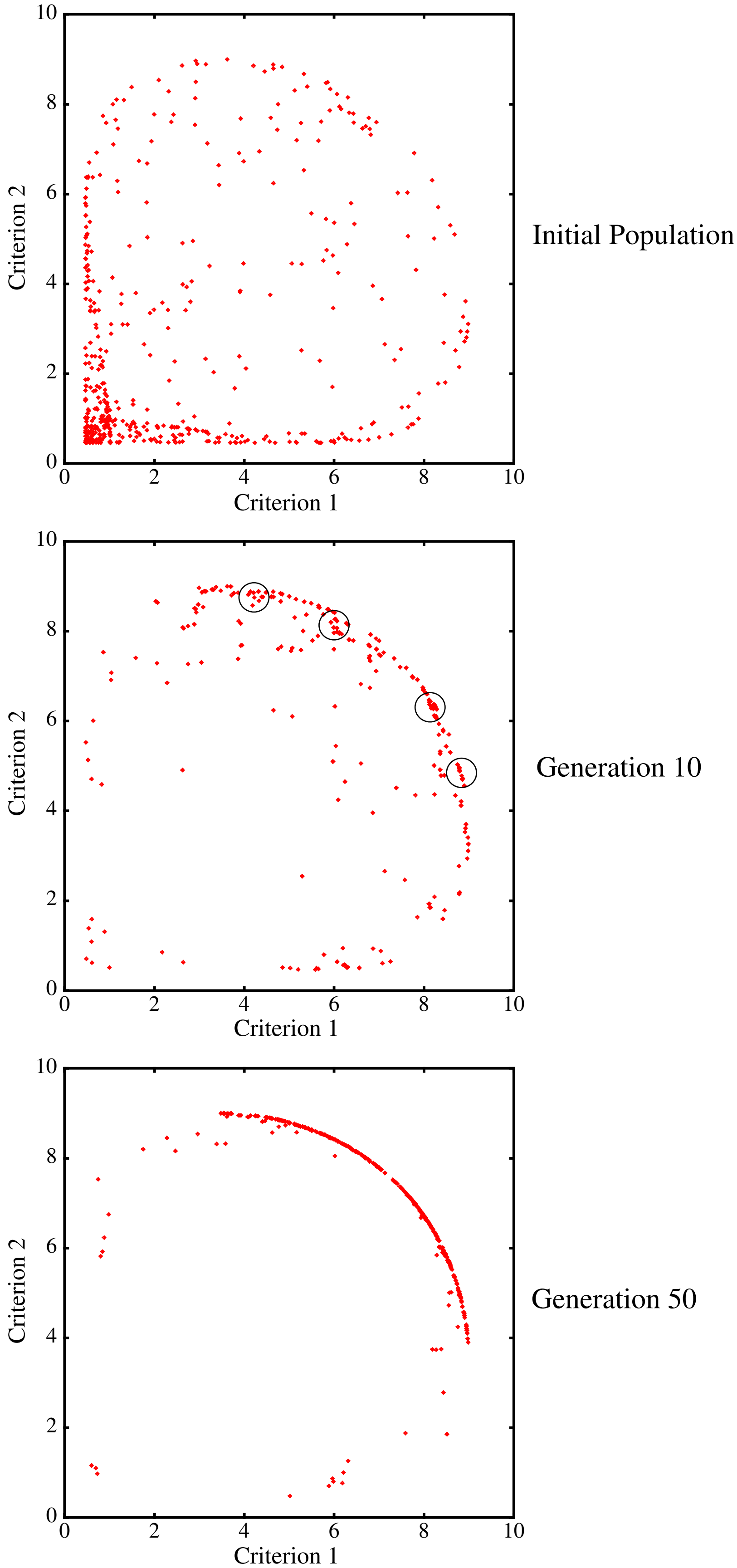


Figure 4.20 - Development of Normal Population in Criteria Space for Function 1: 2 Criteria Problem

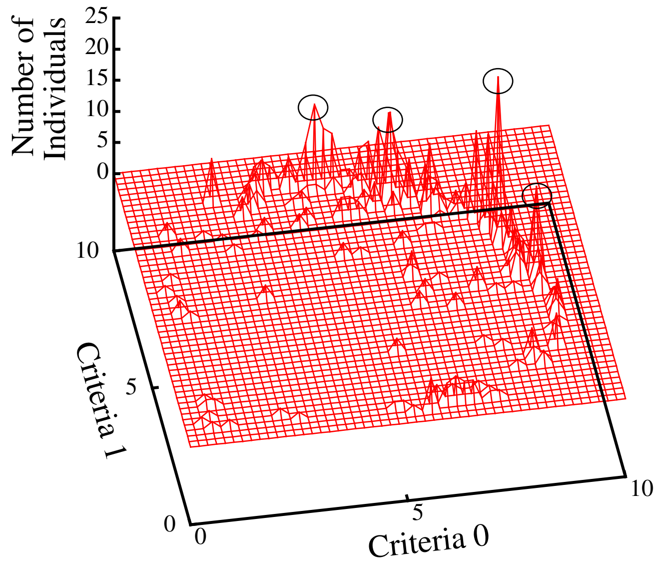


Figure 4.21 - Function 1 - Distribution in Criteria Space (10 Generations)

Figure 4.22 shows, the development of the Pareto Population over 50 Generations in distribution plot form for Function 2. The plots are a magnified version of the area 3 < x,y < 7. The initial population contains about 20 individuals within this area but they are not particularly concentrated. However, as the search progresses the number and quality of Pareto solutions increases. By generation 10 the normal population is beginning to migrate towards the Pareto surface so more Pareto individuals are being generated. By generation 20 the region of search is clearly defined and most of the members of the Pareto Population are within the Pareto region. From this point onwards the aim is to further populate the space with solutions. The final plot shows (50 Generations) that the MCGA has generated up to 35 solutions for some of the samples. The search was continued up to 100 generations (shown in Figure 4.17) in the test and the MCGA populated the space further with some regions being populated by up to 60 solutions.

The total number of rank 1 solutions was also measured for both normal and Pareto Populations for all functions. In all cases the number in the Pareto Population exceeded that maintained in the normal population. This highlights the usefulness of having such a store to prevent the loss of useful solutions. Figure 4.23 shows the number in the normal population along with the number in the Pareto Population for Function 3.

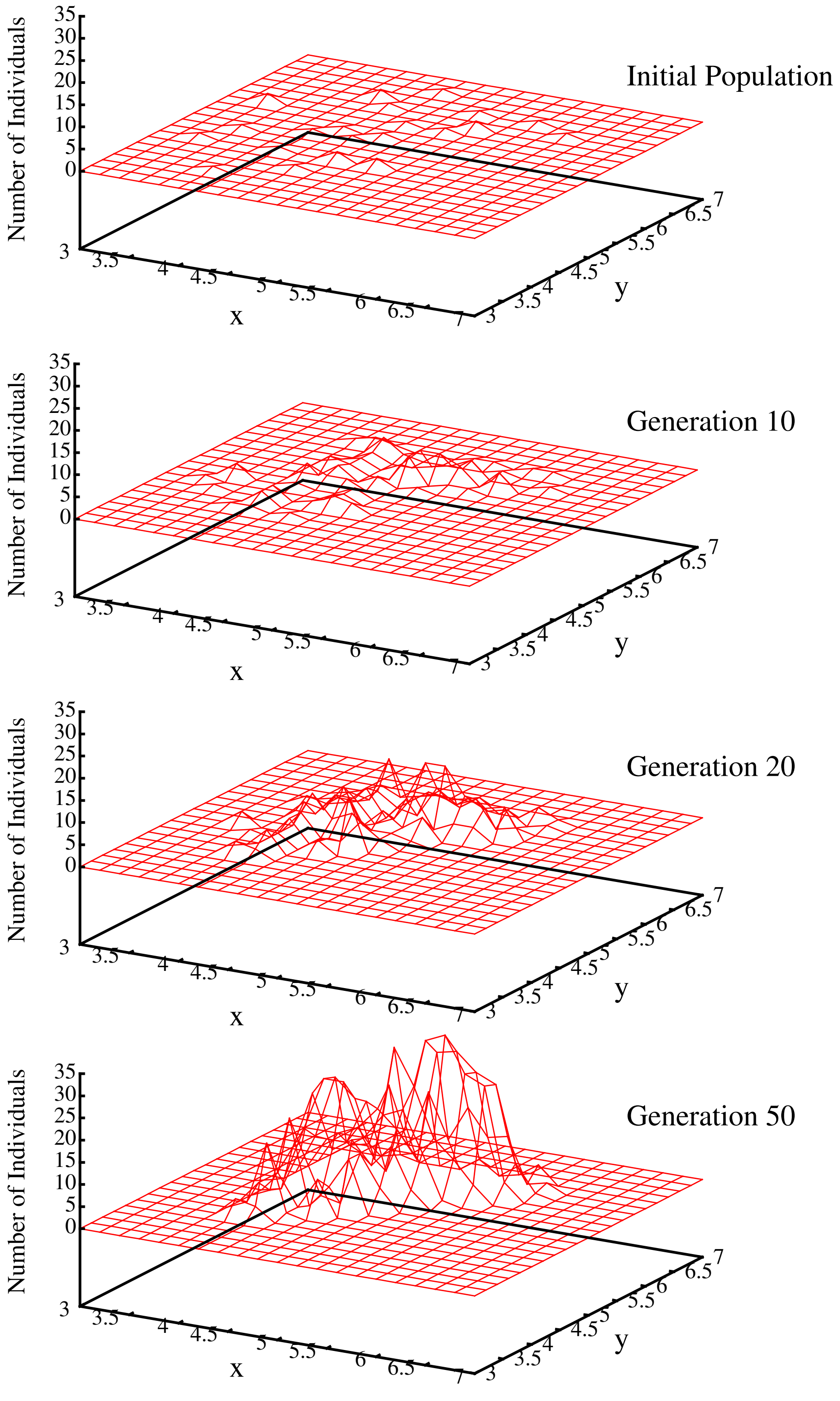


Figure 4.22 - Development of Distribution of Pareto Population (Function 2)

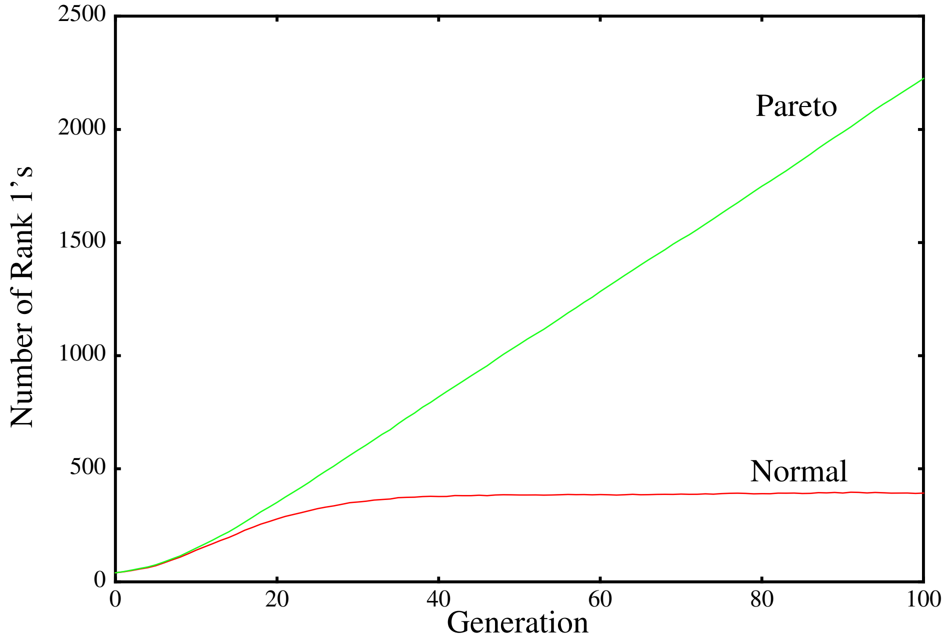


Figure 4.23 - Number of Individuals in Normal/Pareto Populations (Function 3)

4.10 Population Sizing and Niches

In section 4.9 it was noted that when the MCGA was applied to Function 4 the population would sometimes under sample certain regions of the search space. It was thought that this was affected by the population and niche sizing and the relationship between them. Initially, the population size was decreased by half to 250 to see if the effect could be induced. Some of the resulting plots are shown in Figure 4.24.

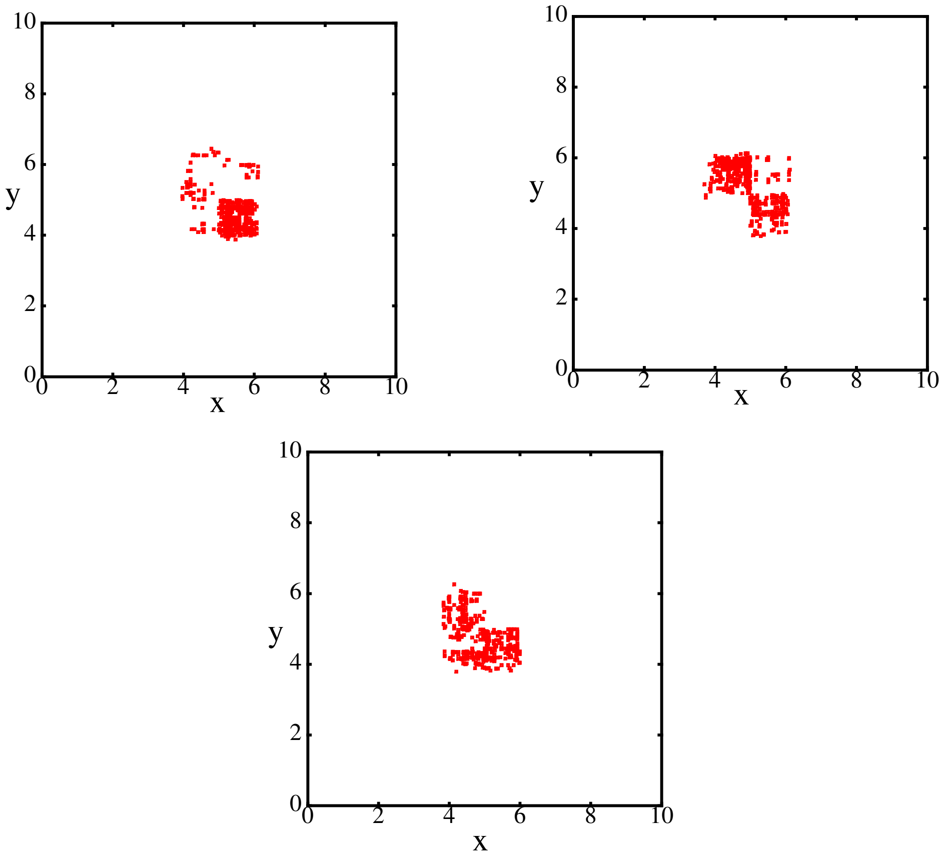


Figure 4.24 - Under Sampling of x-y Domain (Function 4)

It can be clearly seen that the MCGA has problems when trying to solve problems with small population sizes and many criteria. There are two principal factors contributing to this. Firstly a small population size will mean a reduced diversity in the initial population. This would cause the GA to move to fit regions of the search space which have been sampled in the initial population. The other reason is more complex and depends on the relationship between population size, niche size, resolution and the number of criteria.

From the equation for estimating the number of solutions required to reasonably represent a surface (derived earlier in Section 4.3.2):

$$N = ir^{i-1}$$

For a fixed population size N, as the number of criteria increases, the value of r must decrease in order to sample the whole space, and this increases the size of the niche.

In terms of fitness sharing this implies:

1) If niche size tends to 0 (high resolution), there will only be one solution in each niche. Hence there is no sharing and therefore diversification does not occur.

2) If niche size tends to infinity (low resolution), every member of the population is in each individual's niche. Hence every individual's fitness is divided by the population size and this means the same relative fitnesses as before the sharing occurred. This again prevents diversification occurring.

In test Function 4 the second condition is true. The population size(N) was 500, the number of criteria(i) was 5 so the value of r used was 3.17. This value therefore forms niches which are too large and hence fitness averaging occurs thus destroying diversity. The experiment was re-run 20 times using a population size of 1000 (resolution 3.8) for 50 generations. A typical plot is shown in Figure 4.25. For the same number of evaluations the MCGA consistently samples across the whole Pareto region.

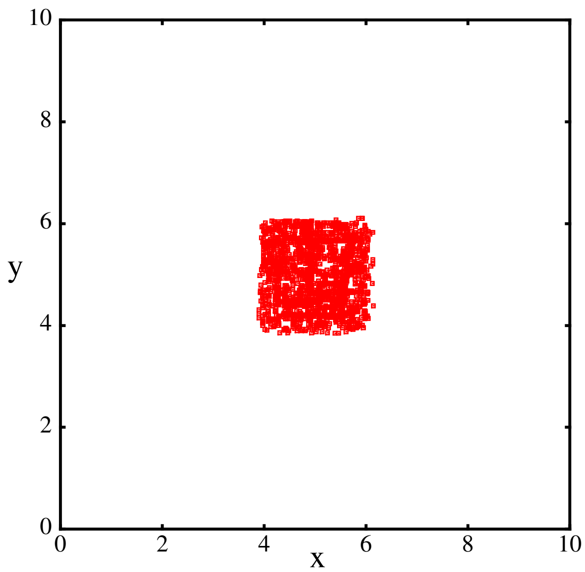


Figure 4.25 - Evenly Sampled x-y Domain (Function 4)

4.11 Run Time and Convergence Rate Measurements

Two measurements were taken in relation to the effect of increasing the number of criteria on the processing time of the MCGA. The first was the time taken by the GA running 10 runs of 30 generations and 500 individuals. This was to see if the manipulation and evaluation of additional criteria added significantly to the total processing time of the system. Figure 4.26 shows the results of this test. As the number of criteria is increased the system shows a non-linear increase in run time. This is due to the extra evaluations being performed (one for each member of the population) and the additional time in performing comparisons during ranking. It is also due to the size of the Pareto Population becoming larger as more solutions are added, resulting in increased processing time.

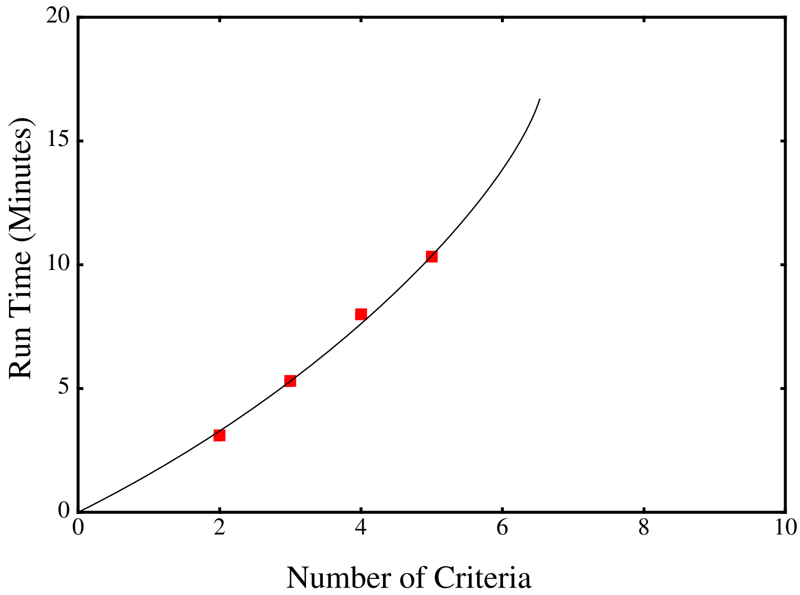


Figure 4.26 - The Multiple Criteria Genetic Algorithm

The other measurement carried out was of MCGA convergence time. This was measured when all criteria maximum, minimum and average graphs in the Pareto Population had settled. This is the point at which the MCGA had generated points reasonably evenly spread across the whole surface and exploratory search has terminated. After this point the MCGA simply utilises these points to further populate the space. This information was taken from the 50 run experiments and the number of generations required to converge was estimated and plotted on a graph. The max/ave/min plots for all three criteria in Function 2 are plotted in Figure 4.27.

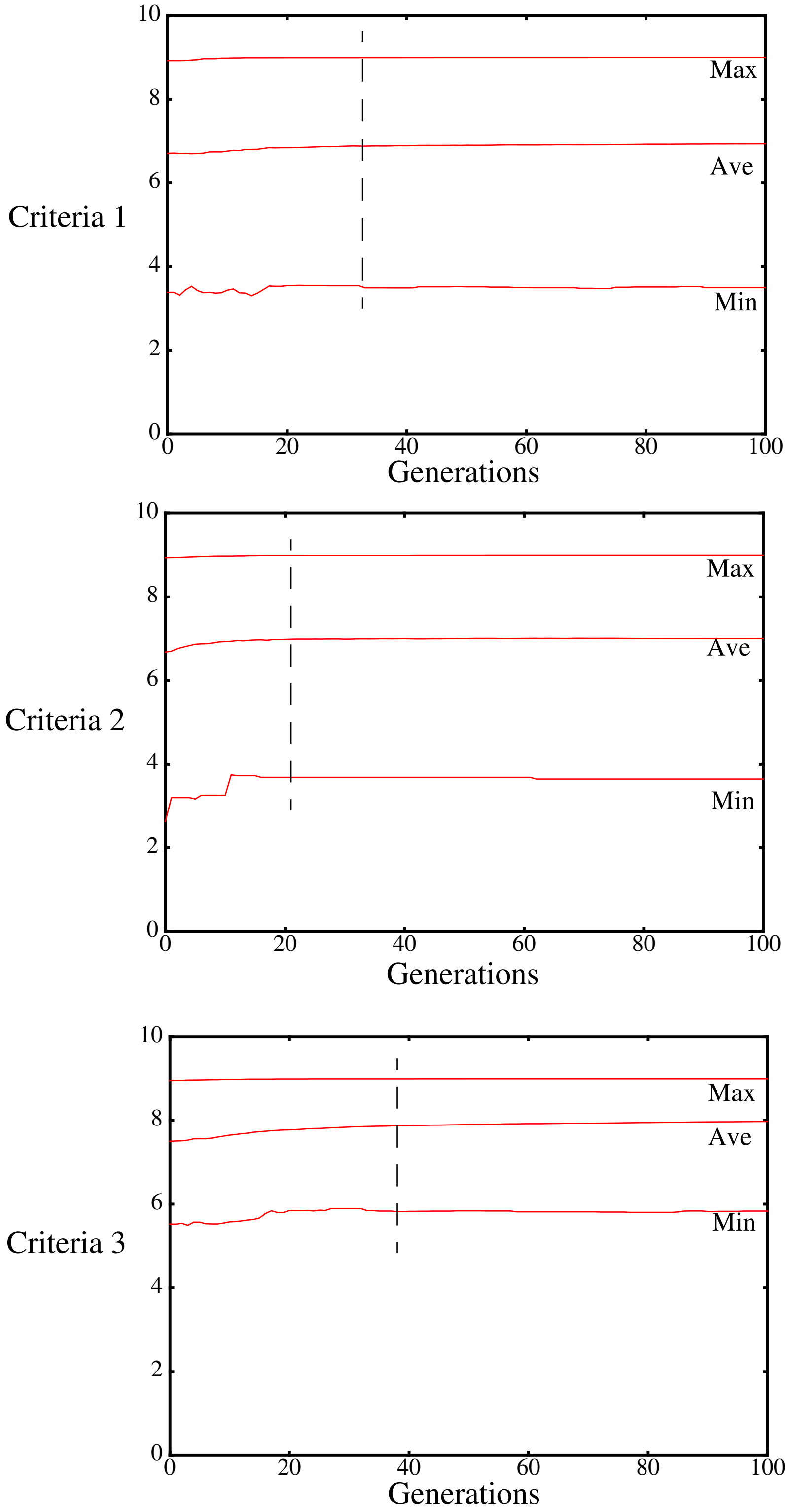


Figure 4.27 - MCGA Convergence

The convergence times were estimated from the convergence graphs and plotted in Figure 4.28. Once again the MCGA indicated a non linear increase in the time taken to converge to the Pareto region. It can also be seen that the 5 criteria problem does not lie on the curve. Its convergence rate is very similar to that of the criteria 4 case. This suggests that the convergence rate may be linked to the size of the Pareto region. Both Functions 3 (4 criteria) and 4 (5 criteria) have the same area of interest in their test functions and Functions 1 and 2 have correspondingly smaller Pareto regions. In general, the results would indicate that in order to get good coverage of a large Pareto region more generations are needed in order to allow the fitness sharing to force diversification to the edges of the region.

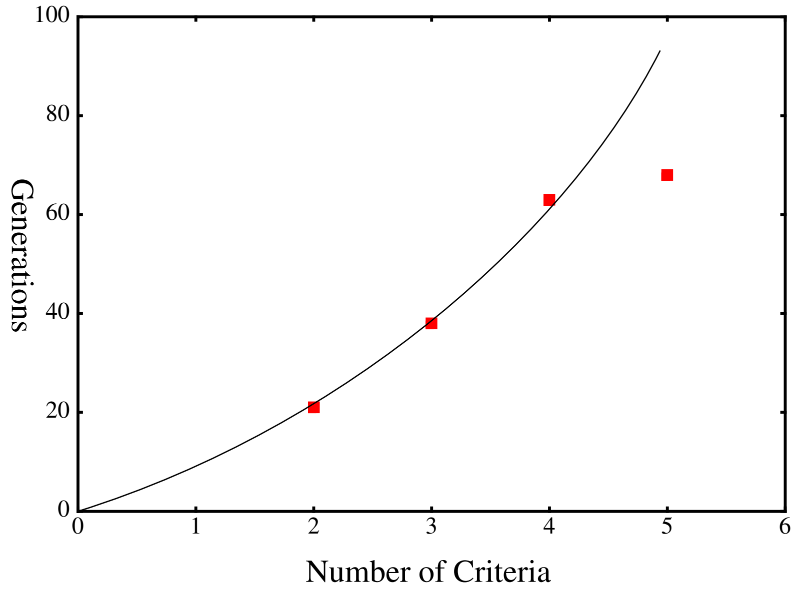


Figure 4.28 - Convergence Times

4.12 Chapter Summary

In this chapter the background theory of multiple criteria problems and how genetic algorithms have been constructed for such problems discussed. Developing from previous work a Multiple Criteria Genetic Algorithm (MCGA) was constructed but with the additional incorporation of a Pareto Population. The Pareto Population was shown to preserve individuals which may normally be lost during genetic processing.

Niche sizing was discussed along with a new method of estimating the number of individuals required to effectively sample the search space for a given resolution. Additionally an adaptive niching method was introduced which could vary the niche size automatically during search, depending on the current dimensions of the Pareto front. This removes the need to specify a niche size at the start of search from approximate and incomplete data.

The MCGA was then tested on a set of four test functions with increasing numbers of criteria. Results show that the algorithm is able to identify and effectively sample the Pareto regions of the test functions. The adaptive niche sizing method also proved to be effective in maintaining a broad population spread over the criteria space as search progresses. The Pareto Population also clearly demonstrated that it was invaluable in maintaining large numbers of solutions far in excess of the population size.

Finally, it was observed that as the number of criteria increases, a sufficiently large population size is needed in order to ensure a good initial distribution of solutions and to allow the sharing functions to perform effectively. It was also indicated that as the number of criteria grew the time for computation for a fixed number of generations and the number of generations both increased super-linearly.

In the following two chapters the MCGA will be applied to a couple of practical engineering problems of a combinatorial nature. The first example is an application in concurrent engineering involved with partitioning of information flow matrices, the second a complex scheduling problem.

5 MCGA for Process Analysis and Optimisation

5.1 Concurrent Engineering

The aim of Concurrent Engineering(CE) is to aid in the development of high quality, often complex products and deliver them to the marketplace at a reduced price and in as short a time as possible (PARSAEI 93). In any complex system there are a large number of interacting parts. The behaviour of this system is not just a function of the individual parts but also of the relationships between them (STEWARD 81). When designing a large system it is common to decompose it down into smaller parts or subsystems. The more sub-systems there are the easier they are to comprehend at the elemental level. However, as the number of relationships between sub-systems increases so management becomes more difficult. A good understanding of these relationships at the design stage of a project, both preliminary and detailed, becomes extremely important. This is due to the fact that the decisions taken during the early stages have a significant impact on all aspects of the product and making the wrong decisions early on can lead to high costs if corrections need to be made at later stages. Therefore, it can be concluded that the main purpose of CE is to shorten the product development time whilst promoting improved quality and avoiding rework in the later stages.

5.2 System Representation

To help the development of complex systems it is common to use some form of model which aids understanding whilst still providing an accurate representation of the behaviour of the system. The form of the model will be dictated by the information that needs to be represented. Several models may be required to represent the problem in different abstractions.

A graph is a common method used to represent the structure of a system (STEWARD 81). A graph is a set of vertices (points) and a set of arcs (directed lines) between certain sets of vertices. The vertices are the parts of a system and the arcs indicate which vertices affect others. For example Figure 5.1 shows a graph with 6 vertices. If vertex C is considered, it is affected by vertex B and affects vertices D and F. A path is a sequence of 1 or more vertices (e.g. A,B,C). Paths with a single arc show a direct effect (e.g. path B,C). If there is a path from A to B, A is a predecessor of B and B is a successor of A. A path with more than one arc shows an indirect effect (e.g path A, B, C, D). The length of the path is the number of arcs within it. A major concern in such graphs are circuits, these are paths of more than one arc which return to the starting vertex of the path. An example of a circuit is B, C, D, B.

.

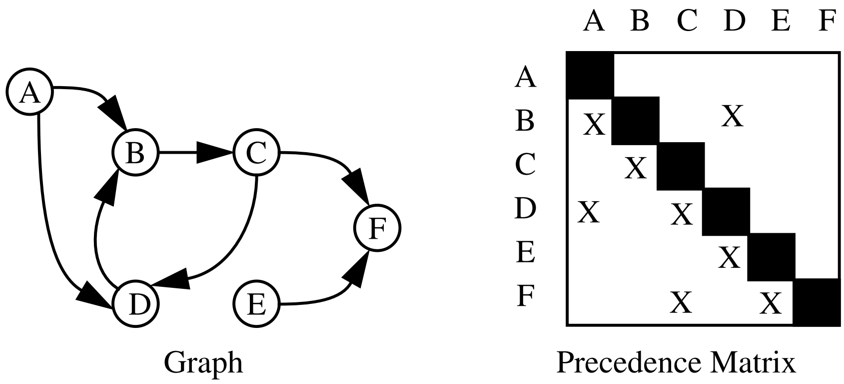


Figure 5.1 - Graphs and Matrices

The structure of a system can also be represented in a precedence matrix the same way as in a graph. A matrix is formed from i rows and j columns. Commonly i=j=number of vertices. Figure 5.1 shows the precedence matrix for the graph. Each of the X’s represents an effect. For example in the graph A affects B and D. It can also be seen that circuits are represented by marks above the leading diagonal; in this case D affects B.

It is clear that a graph in matrix form is far more amenable to computer manipulation and overcomes the size and complexity limitations of many other methods (GEBALA 91). This is particularly the case when the number of vertices and arcs increase. The matrix representation can be used either in the study of the behaviour of a system or the design of systems with specified behaviour.

5.3 The Design Structure Matrix

If the precedence matrix is used in the context of design of a product it becomes a design structure matrix. Each vertex now represents a particular stage or activity within the design process and each arc is an information flow of design knowledge to or from a given stage. When read down or across (left to right) the matrix gives an ordering of the design tasks. Ideally it would be preferable for a matrix to have as few X's above the diagonal as possible as these marks represent information which has to be fed back (feedback arcs). For example activity 1 in Figure 5.2 requires information from activities 5 and 9. In the first instance this information must be estimated for activity 1. This introduces the idea of design iterations, and each iterative block is shown by a thick black border. After the first pass through the whole process 1-9, and on returning to activity 1, a designer will have a better understanding of the information provided by activities 5 and 9. This will mean that activity 1 will involve a refinement of the previous results and these refinements will then be passed on downstream. Similar comments can be made about the other iterative loops in Figure 5.2

	1	2	3	4	5	6	7	8	9
1					X				X
2									X
3						X		X	
4							X	X	
5	X		X					X	
6									
7									X
8	X		X						
9		X							

Figure 5.2 - A Design Structure Matrix

It is clear that such iterations will increase the time it takes to complete the project. This is especially the case here when the whole design process has to be repeated. Circuits cannot be removed as they are essential information flows but by using a process called partitioning the efficiency of the process can be increased.

Partitioning involves reordering the activities and then assigning subsets of activities to blocks such that the following three conditions hold:

1) Tasks within a block are assigned contiguous integers within the matrix.
2) No arc goes from an activity at the end of one block to one at the start of another block.
3) Given any two tasks in the same block, x_i and x_j, there is a path from x_i to x_j or from x_j to x_i.

Partitioning has two consequences. Firstly, the number of X's above the diagonal is reduced and the size of iterative blocks is also reduced. The results of partitioning performed on the matrix in Figure 5.2 are shown in Figure 5.3. The resulting matrix is said to be block triangular. It can be clearly seen that the sizes of the iterative blocks are reduced with the number of feedback paths being reduced from 9 to 3. However, all the information flows have been maintained.

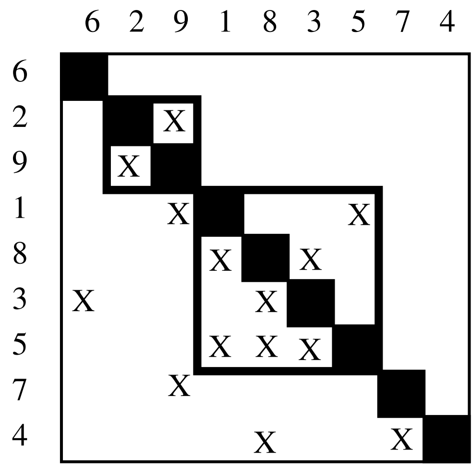


Figure 5.3 - A Partitioned Matrix

Partitioning can be carried out using several methods ranging over simple algorithms (STEWARD 81b), branch and bound (KUSIAK 91) methods, path searching (GEBALA 91) and powers of the adjacency matrix (LEDET 70). Partitioning is then followed by a process called tearing which focuses on ordering tasks within each of the identified blocks in order to resolve interdependencies. This is commonly carried out using shunt diagrams (STEWARD 81) or heuristics (KEHAT 73). However, the combinatorial nature of the partitioning problem lends itself well to being tackled by a genetic algorithm. This is a new approach and examined in more detail below.

5.4 GA For Partitioning

To operate the GA on the above problem a fitness function needs to be defined. Gebala (GEBALA 91) implemented a matrix with X's replaced by numerical values between 0 and 1 indicating the level of reliance on information. Using these numerical values it is possible to score the matrices in different ways. In the experiments carried out here the information flow is either set to 0(no X) or 1(X present) to allow comparison to other non-numerical techniques for partitioning.

A good measure of the quality of a matrix is the distance of 1's away from the diagonal and above it. If we try to minimise this value good matrices should be formed. The following score measure is calculated for a matrix of size n by n with v(i,j) being the numerical value in matrix location i,j:

$$\text{Score} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (j-i) \times v(i,j)$$

This calculation is shown Figure 5.4 for four simple matrices. It should be noted that the values of i and j are not related to the activity numbers; they are merely an indexing system for the score evaluation. The diagram clearly shows that matrices with diagonal numbers above and further away from the diagonal will have higher scores than those with the numbers close to the diagonal. It also shows that if there are no numbers above the diagonal the score is zero. So the aim of the GA search is to minimise this score in an attempt to reduce the number of feedback paths and reduce the size of blocks.

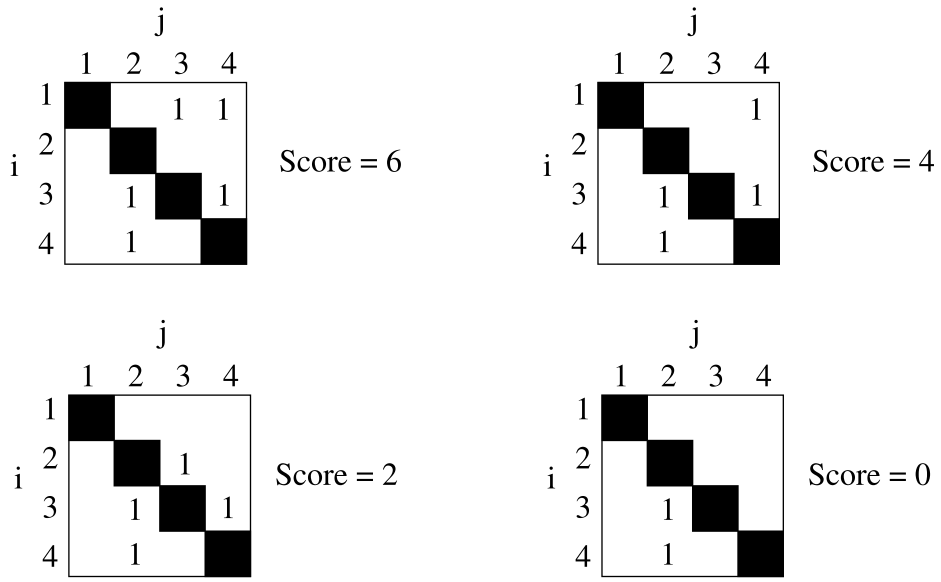


Figure 5.4 - Fitness Evaluation

In this problem the relationships between the individual activities are not as clearly defined as the relationship between cities in the TSP (Travelling Salesman Problem) though the structure of the problem is similar. Activities affect the overall ordering both locally and globally. This means that as well as being concerned with what activities are adjacent to each other (like the TSP) the absolute ordering is also very significant (unlike the TSP). For example the order 1 2 3 4 5 in the TSP is the same as 3 4 5 1 2 or 4 3 2 1 5. However, 1 2 3 4 5 is not the same as 3 4 5 1 2 for the partitioning problem.

The lack of good adjacency heuristics means that a general combinatorial operator must be used. As demonstrated in Chapter 3 the best operator combination was the Enhanced Edge crossover / 2-City Adjacent Swap mutation partnership. This operator will always build from one of the parents' starting genes, hence it will preserve overall ordering as the first activity is maintained. From there it will build up child strings based on which activities are adjacent to each other in the parent strings. As the search progresses the best first city will be propagated and the orderings will be built from this same first city.

Chapter 3 also highlighted the problem of the simple combinatorial operators struggling with larger problems. This was overcome using the 2-opt local search method. As no local measures of score exist a new local search must be implemented based on the total score. The operator works by swapping two activities and re-evaluating the matrix. If the score is lower than it was before the swap is maintained. If this is not the case the string is returned to its original configuration and the search for a good swap is continued. As in Chapter 3 the initial search is carried out by the GA and then the local search is applied in later generations to further optimise the matrix.

5.5 Test Problems, Validation and Results

The GA was tested on three test problems of size 12, 20 and 51. The problems are all taken from literature, where previously only non-GA methods of partitioning were employed. For each problem the starting matrix, partitioned matrix (as presented in the papers) and the GA optimised matrix is shown. A population size of 100 over 100 generations was used in all experiments. The small matrix was used to validate the code and the matrix models were score checked to ensure they were correctly defined.

5.5.1 Problem 1 - 12 Activity (KUSIAK 91)

This example was used by Kusiak to demonstrate a triangulation algorithm. The results of this method are shown in Figure 5.6.

	1	2	3	4	5	6	7	8	9	10	11	12
1	■											
2		■										
3		1	■									
4			1	■	1	1						1
5					■	1		1			1	
6		1				■						1
7		1					■				1	
8	1			1				■	1		1	
9			1			1			■	1		
10		1	1			1				■	1	1
11		1	1								■	
12	1								1	1	1	■

Score = 39

Figure 5.5 - Unpartitioned Matrix

	2	3	1	11	12	6	10	9	5	4	8	7
2												
3	1											
1												
11	1	1										
12			1	1			1	1				
6	1	1		1	1							
10	1				1	1						
9		1				1	1					
5				1			1				1	
4		1			1		1		1			
8			1	1				1		1		
7	1			1								

Score = 7

Figure 5.6 - Kusiak Partitioned Matrix

	2	3	11	1	6	12	10	9	4	5	8	7
2												
3	1											
11	1	1										
1												
6	1					1						
12			1	1			1	1				
10	1	1	1		1	1						
9		1			1		1					
4		1			1	1				1		
5			1		1						1	
8			1	1				1	1			
7	1		1									

Score = 6

Figure 5.7 - GA Partitioned Matrix

5.5.2 Problem 2 - 20 Activity (STEWARD 81)

Steward used a simple algorithm to tackle this matrix. The matrix represents the design process of building an electric car. Stewards matrix is shown in Figure 5.9.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1			1			1				1							1			
2																				
3				1																
4																	1			
5																1			1	
6					1											1				
7					1											1			1	
8							1													
9						1				1								1	1	
10			1			1					1						1	1		
11																		1		
12			1			1				1				1			1			
13						1									1			1		
14																				
15																				
16	1	1			1				1											
17						1				1						1				
18						1	1	1												
19		1			1															
20			1																	

Score = 159

Figure 5.8 -Unpartitioned Matrix

	2	1	3	4	5	6	7	8	9	10	11	16	17	18	19	14	12	15	13	20
2																				
1			1			1				1			1							
3				1																
4													1							
5												1			1					
6					1							1								
7					1							1			1					
8							1													
9						1				1				1	1					
10			1			1					1		1	1						
11														1						
16	1	1			1				1											
17						1				1		1								
18						1	1	1												
19	1				1															
14																				
12			1			1				1			1			1				
15																				
13						1								1				1		
20			1																	

Score = 93

Figure 5.9 - Steward Partitioned Matrix

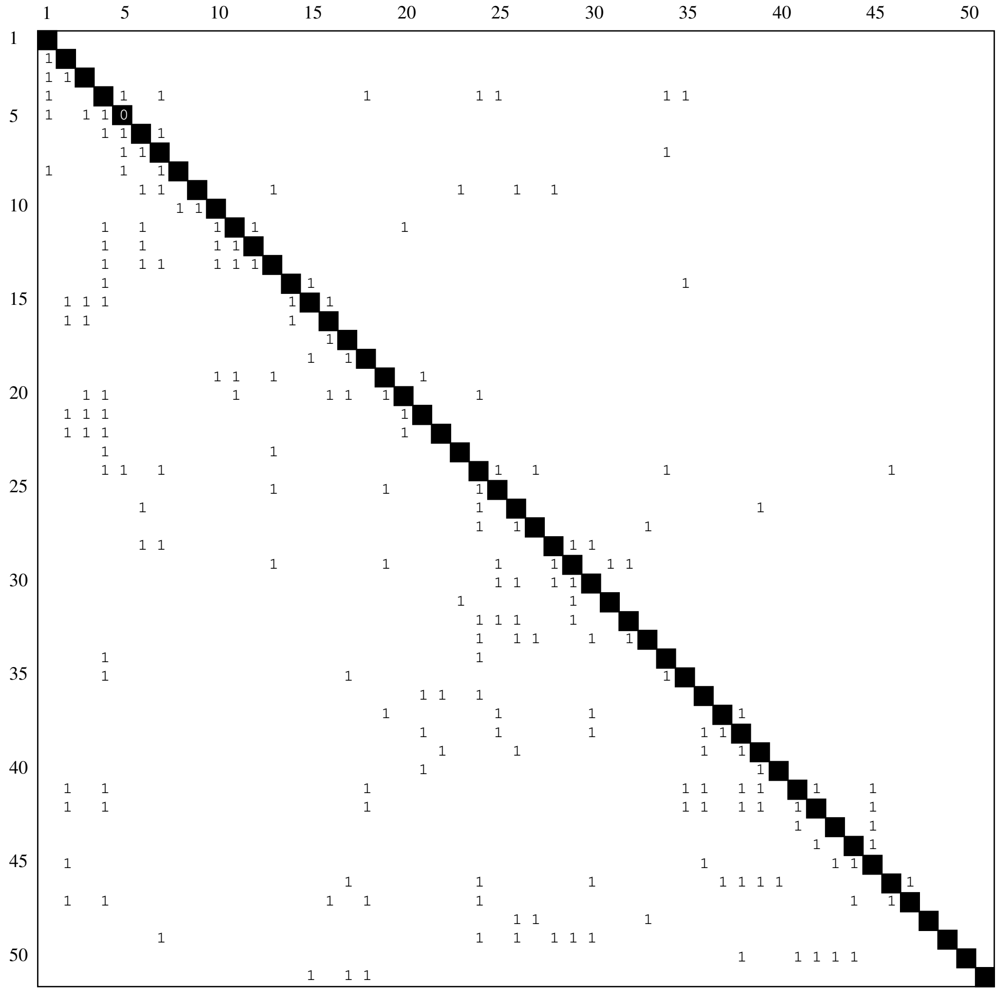
	2	19	5	7	16	6	8	18	11	9	10	17	4	3	1	14	15	12	20	13
2																				
19	1		1																	
5		1			1															
7		1	1		1															
16	1		1							1					1					
6			1		1															
8				1																
18				1		1	1													
11								1												
9		1				1		1			1									
10						1		1	1			1		1						
17					1	1					1									
4												1								
3													1							
1						1					1	1		1						
14																				
15																				
12						1					1	1		1		1				
20														1						
13						1		1									1			

Score = 24

Figure 5.10 - GA Partitioned Matrix

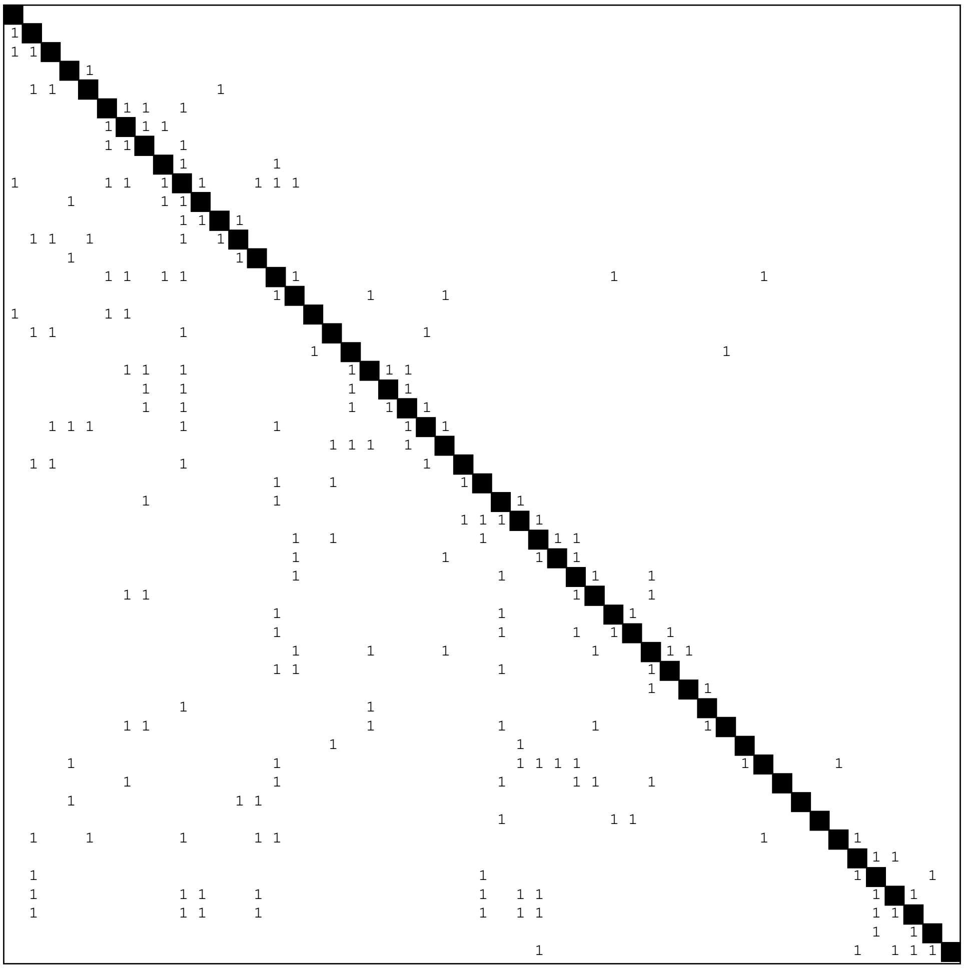
5.5.3 Problem 3 - 51 Activity (AUSTIN 96)

In this test case, of building design, the starting matrix was not provided. However the given partitioned matrix was used for the optimization.



Score = 320

Figure 5.11 - Austin Partitioned Matrix



Score = 158

Best Sequence:
1 2 3 5 7 6 34 4 35 16 14 15 17 18 24 25 8 10 13 12 11 19 20 21 23 26 31 29 32 28 30 27 33 9 22 36 39 38 37 40 46 44 45 42 41 43 47 50 49 51 48

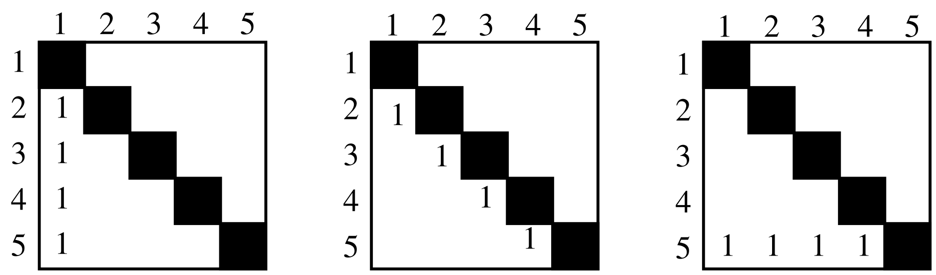
Figure 5.12 - GA Partitioned Matrix

5.6 MCGA For Partitioning

In all cases the GA method outperformed conventional partitioning techniques. As the matrix became larger the potential performance gains from the GA also increased. From these experiments it was observed that often several different matrices would yield the same or similar score but only the best was saved. If several process plans are available this would allow the planner to choose the most appropriate configuration on the basis of other less objective considerations.

The partitioning problem was extended to a three criteria problem by adding two new fitness scores. The aim of this being to generate a range of possible process plans with a trade-off between good properties.

The fitness function of the experiments carried out so far have been based on simply trying to minimise the number of 1's above the diagonal. The objective was to minimise the number and size of iterations needed in order to reduce total design time. Another way of reducing the overall time is by increasing the possibility for concurrency. This is done by moving the 1's as close to the left and bottom of the matrix as possible. This has significant impact on the level of parallelisation which is possible. Figure 5.13 below shows three cases. In the first case activities 2 to 5 can be performed concurrently if there are no resource constraints, however this processing can only start when task 1 has been completed. In case (ii) the process is purely sequential hence activities must take place in the order 1 2 3 4 5 with no parallelisation being possible. Case (iii) will allow the parallel execution of tasks 1 to 4 and after all of these have been completed activity 5 can be started.



Case (i) Case (ii) Case (iii)

Figure 5.13 - Potential For Parallelisation

In examples taken from the real world, the feedback paths make the system more complex. There is a trade-off between the level of parallelism which can be employed and the number of iterations that have to be made during the design. The MCGA aims to provide some insight into this trade-off by generating a range of solutions from which the options with the greatest potential for reducing total design time can be chosen.

One way in which the choice of matrices generated by the MCGA can be unambiguously resolved is by assigning times to activities, deciding which tasks will be performed concurrently and how many iterations will be required within the blocks. This requires the system to be clearly defined and hence clearly understood. If this is possible the total processing time can be found and minimised.

The MCGA was run on the '51 activity problem' (AUSTIN 96). A population of 100 was used and the GA run for 200 generations over 10 trials. The three criteria were:

Minimise Criterion 0 - Distance of 1's from left of matrix.

$$\text{Criteria } 0 = \sum_{i=1}^{n} \sum_{j=1}^{n} (j) \times v(i, j)$$

Minimise Criterion 1 - Distance of 1's from bottom of matrix.

$$\text{Criteria } 1 = \sum_{i=1}^{n} \sum_{j=1}^{n} (n-i) \times v(i, j)$$

Minimise Criterion 2 - The Number of 1's above diagonal.

$$\text{Criteria } 2 = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (j-i) \times v(i, j)$$

where n=number of activities and i and j are the vertical and horizontal matrix coordinates. The MCGA described and constructed in Chapter 4 was used. Additionally the 2-opt type operator described earlier in this chapter was implemented as before. Since the main aim is still to reduce the number of iterations the 2-opt

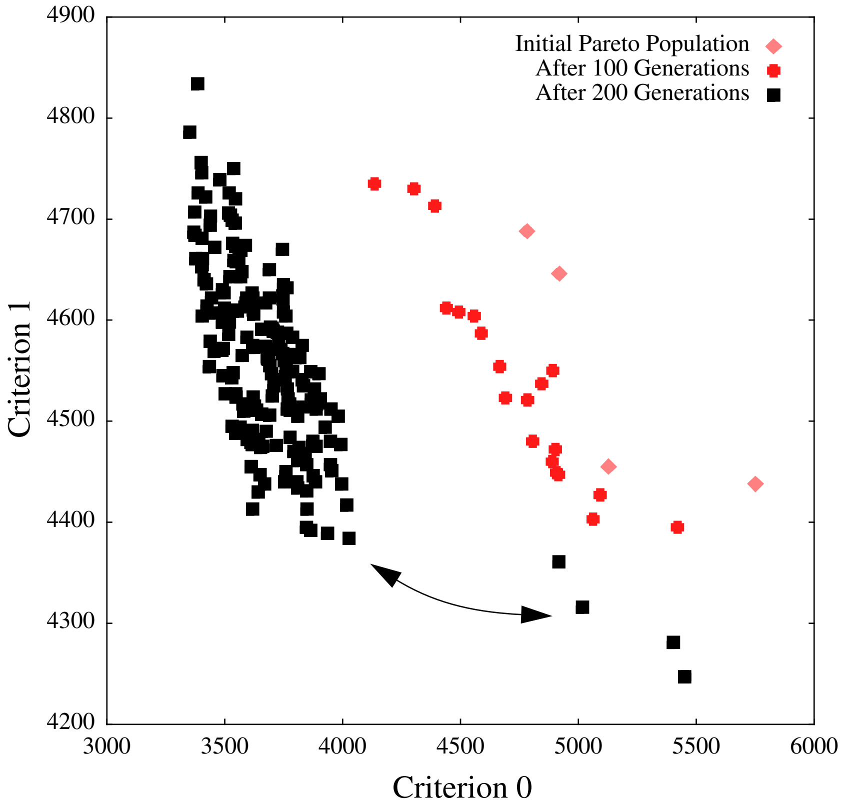


Figure 5.14 - Development of the Pareto Population

operator was left unchanged. The range of points generated is shown in Figure 5.14. The plot is shown projected onto the plane of criterion 0 and criterion 1. It is clear that significant improvement in terms of both quality and quantity of solutions has taken place over the 200 generations. In this run the Pareto Population held 192 members at the end of the search. Each solution in this population can be displayed as a partitioned matrix if required. The bias of the 2-opt search can also be seen from this plot. The 2-opt operator's only goal is to improve criterion 2. This is illustrated by the fact that exploration into criteria 0 and 1 has been neglected. This is very clear in the case of criterion 0 where there is a large gap (indicated by the arrow) in the coverage of the points. This gap does however indicate that very few points in this range have good values of criterion 2 which is still the main goal of the optimisation.

The graph was re-plotted with the Pareto Population cut down by removing all solutions with criterion 2 less than 180. There are still 9 solutions which fall into this category, these are shown in Figure 5.15. The solutions are also marked with their values for criterion 2. They represent a broad range of values for criterion 0 and criterion 1 with only a small change in the value of criterion 2.

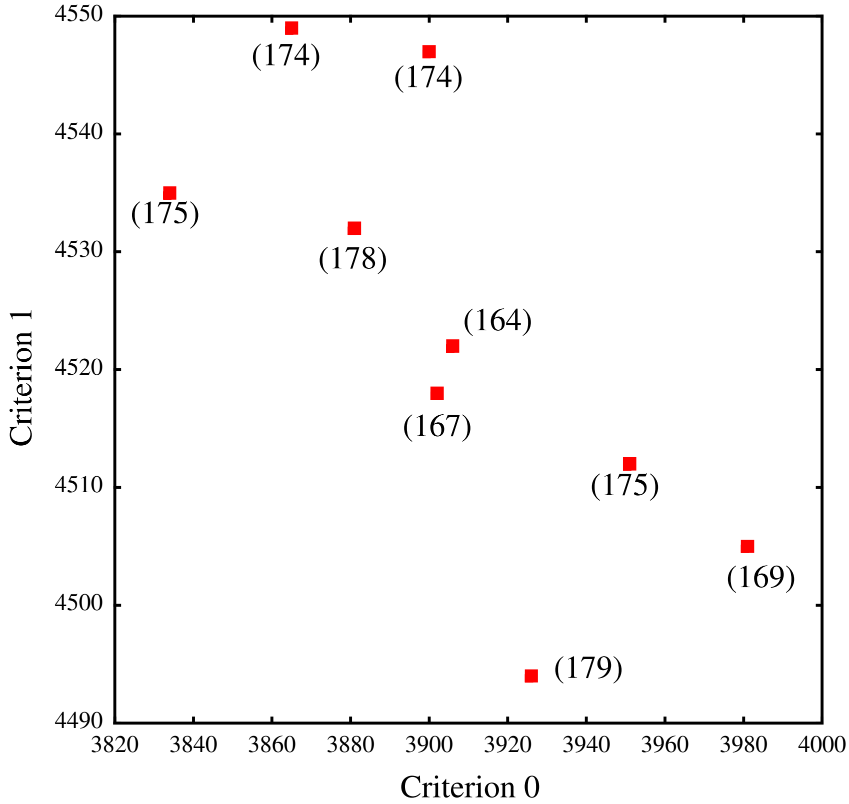
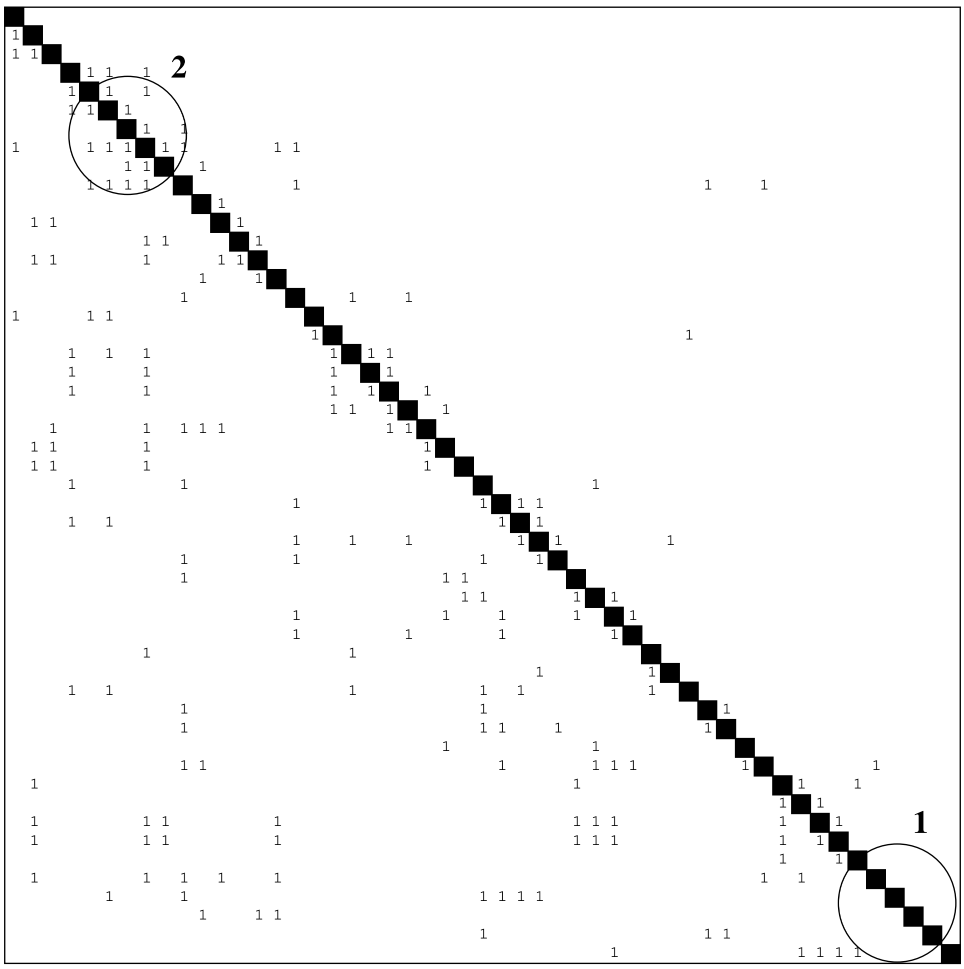


Figure 5.15 - Reduced Solution Set (Criterion 2<180)

An example of a typical matrix from this reduced set is shown in Figure 5.16. The matrix shown is the one with the minimum value of criterion 0. From this diagram it can be seen that the MCGA has attempted to move 1's away from the right side of the matrix. Comparing this with the matrix shown in Figure 5.12 it can be seen that the MCGA has moved points from location 1 to location 2 (marked on Figure 5.16) where there is an increased clustering of 1's. This has led to a decreased overall criterion 0 fitness. However some of the solutions in location 2 have moved above the diagonal increasing the value of criteria 1 and 2.

The best value of criterion 2 discovered in this Figure 5.15 is 164 which is slightly short of the single global optimization case best value of 158 for the same criterion. However, on two occasions in the 10 runs, a matrix with a value of 156 was found The increased diversity of search due to the sharing and crossover functions allows the MCGA to discover solutions which may be lost in the search process.

As mentioned before this set of 9 solutions could be further analysed in terms of the time of activities and iterations. This would give the overall time for each solution. This allows the trade-off between the Pareto points to be undertaken giving a single ‘best’ solution.



Criterion 0=3834   Criterion 1=4535   Criterion 2=175

Best Sequence:
1 2 3 6 5 7 34 4 35 24 17 16 14 15 18 25 8 10 13 12 11 19 20 21 22 26 30 28 29 32 36 39 38 37 23 31 9 27 33 40 46 45 44 42 41 43 47 49 51 48 50

Figure 5.16 - MCGA Generated Matrix

5.7 Chapter Summary

In this chapter a novel GA routine for partitioning design matrices, used in Concurrent Engineering, has been demonstrated. A single criteria GA with enhanced edge and local search was shown to out perform standard methods on 12, 20 and 51 activity problems.

The partitioning problem was then expanded to a multiple criteria one where additional objectives were tackled in order to provide a selection of good solutions. From these solutions the decision maker can select the configuration from which the most beneficial solution can be isolated.

Due to the MCGA's mechanisms for maintaining diversity it was able to find a solution with better fitness than the single criteria GA in two of the test cases. It also highlighted the fact that if local search techniques are used they tend to bias the multiple criteria search towards a preferred direction. This may be desirable if one criterion has clear priority over the others, such as in this example. This form of biased multiple criteria search clearly indicates that any local search incorporated into the MCGA must be carefully constructed so as to reduce this bias as much as possible.

In the next chapter the principles of the simple job shop are extended to allow more complex and realistic scheduling problems to be dealt with using the GA. A real world example of scheduling in a shipyard steel shop is also tackled where the use of the MCGA is demonstrated in the area of process and feasibility analysis.

6 MCGA For Scheduling in a Manufacturing Environment

6.1 Scheduling Using Genetic Algorithms

Scheduling is a combinatorial problem with important impact on both industry and commerce. If it is performed well it will give significant time and efficiency benefits and hence reduce costs. In a manufacturing environment there are several areas where scheduling is used.

Flow Shop Scheduling occurs where sets of different jobs must pass through a set of machines in the same order, often with the use of automated handling equipment. The aim is to maximise the processing rate of the jobs through the system. Cleveland (CLEVELAND 89) tackled this problem with the additional complexity of testing stations where failed jobs are returned to the previous workstation.

Resource / Staff Scheduling is concerned with allocating resources, in terms of operations and quantity, to a set of tasks. For example, in the case of staff scheduling, the number of employees possessing a certain skill must not exceed the number available within the company otherwise external staff must be brought in at extra cost. Hence the aim is to schedule activities to minimise costs (FILIPIC 95, CHO 96). In the case of machines the aim could be to regulate power consumption or some other resource attribute (EASTON 93, TANOMARU 95).

The Assembly Line Balancing Problem involves a set of n stations. At each stations a set of operations is carried out on the manufactured item and then passed onto the next station. The problem is to assign the operations to the stations in such a way that the flow of items through the line is free of bottlenecks (ANDERSON 91, KIM 96).

However, *the Job-shop Scheduling Problem* (JSP), first attempted with GAs by Davis, (DAVIS 85) is one of the hardest (TSUJIMURA 97) and most commonly encountered scheduling problems. Since Davis' work there have been many attempts at the simple JSP with a single criterion (NAKANO 91, SYSWERDA 91, BRUNS 93, FANG 93, GEN 94, REEVES 95, CHENG 96). More complex approaches by Liang (LIANG 94) and Sridhar (SRIDHAR 96) used a weighted sum approach to combine disparate metrics to give a single fitness measure. Murata (MURATA 96) used a random weight system to generate a range of trade-off solutions in a flowshop problem, while Hamada (HAMADA 95) addressed optimization with constraints. However, none of these approaches combine multiple criteria considerations and the increased job shop complexity which is considered in this chapter.

6.1.1 Job-shop Scheduling Problem Definition

A classical JSP is defined as follows:

i) There are k machines and n different *jobs*.

ii) Each job is composed of a set of m *stages*.

iii) Each stage requires a specific machine.

iv) Each stage has a fixed processing time.

v) Each stage must be carried out in a specific order.

vi) There are no precedences between different jobs.

vii) Operations cannot be interrupted.

viii) Each machine can process only one job at a time.

ix) Neither release times nor due dates are specified.

The aim is to determine the operation sequences on the machines in order to minimise the makespan, the time from starting the first task to finishing the last one. To demonstrate this problem a 4 job 3 machine example is used (Table 6.1). M1 denotes that machine 1 is needed to perform the operation and the following number indicates the time required to perform it.

Table 6.1 - Job Shop Problem

Machines M0,M1,M2	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4
Job 0	M1 ,5	M0, 3	M1, 4		
Job 1	M2, 3	M1, 6	M0, 5	M1, 3	M2, 4
Job 2	M0, 4	M2, 3	M1, 6		
Job 3	M1, 4	M2, 5	M1, 3	M0, 5	

The ordering of the jobs on each machine is very significant. This is demonstrated in Figure 6.2. The first case is randomly generated and has a makespan of 38 time units. The effect of optimisation can be clearly seen in the second example. The makespan has been reduced to 31. This effect becomes more significant as problem size increases. The bottom example shows what can be achieved if the flexibility of the machines is increased, in that the same stage of a job can be performed on more than one machine, an extension which is discussed later in this chapter. In this case machine 2 has been modified to allow it to perform machine 1's jobs as well as its own. This reduces the makespan to 24.

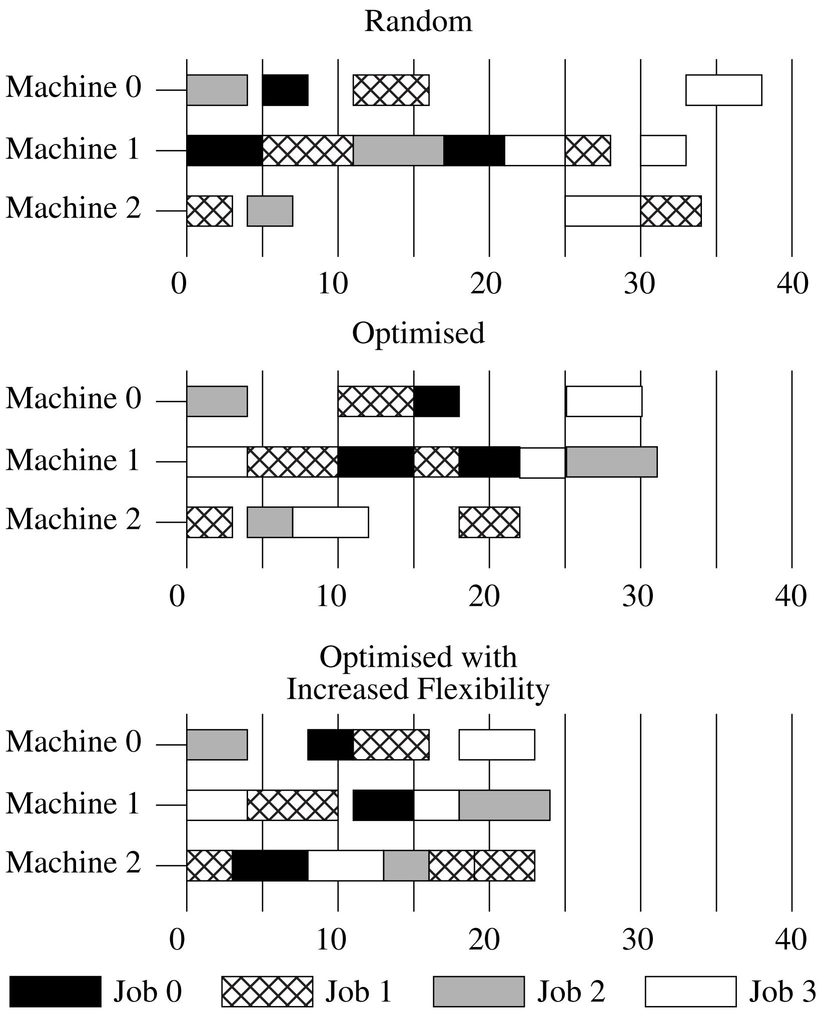


Figure 6.2 - The Effects of Job Ordering

6.1.2 Representation / Encoding

There are several representations for the JSP when tackled using GAs, these can be classified into direct or indirect encodings. Direct encodings use strings which represent the schedule directly and optimise this. Indirect encodings encode dispatching rules in a string and use these rules to create schedules, hence evolving better rules.

Direct Encoding Representations:

1.Operation - Each gene in the string represents one stage. The ordering of each stage on a machine is determined by the string order.

2.Job - Each gene represents a job. The first job in the string is scheduled, then the second is inserted into the schedule to give the best available processing time and so on.

3.Job Pair Relation - This uses a binary matrix to determine the precedence relationships of pairs of jobs on a given machine.

4.Completion Time - This uses an ordered list of completion times for each stage of each job on each machine. This method generates a large proportion of illegal schedules as overlapping stages occur.

5.Random Keys - This encodes the solution as a random number. The number will consist of two parts, an integer part representing the machine number and the fractional part which provides the job sequence on that machine.

Indirect Encoding Representations:

1.Preference List - The string is formed from m sub-chromosomes with one for each machine. Each sub-chromosome is of length n and represents the preference list for a machine. This preference list is used within a simulation to determine which job to choose from a waiting list for a given machine.

2.Priority Rule - Each chromosome is encoded as a sequence of dispatching rules for job assignment. A schedule is then constructed using a priority dispatching heuristic based on this sequence. The problem is to identify effective priority rules.

3.Disjunctive Graph - This method views the JSP as a disjunctive graph. The chromosome indicates the preferred direction of an arc between pairs of nodes i and j, assuming that no cyclic graphs will be created by doing so.

4.Machine - The chromosome is encoded as a sequence of machines and a schedule is constructed with a shifting bottleneck heuristic based on this sequence. The GA here is used to create a sequence of machines which provides the shortest schedule using the bottleneck heuristic.

6.2 Complex Scheduling

6.2.1 Scheduling for Made To Order Products

Scheduling plays an important role in most manufacturing environments. However it is particularly important in the field of Made-To-Order products. The use of scheduling in the manufacture of large one-off products such as offshore platforms, ships or chemical plants, can give great benefits. It can lead to reduced lead times and inventory levels as well as improving integration between different functions within an organisation such as manufacturing and assembly (Figure 6.3) . This increasing level of interaction facilitates the use of Concurrent Engineering techniques to increase process efficiency (see Chapter 5). However integration requires a systematic consideration of manufacturing to deadlines. Deadlines will occur internally but often a company will have to pay compensation for every day or week a product is delivered late to the customer.

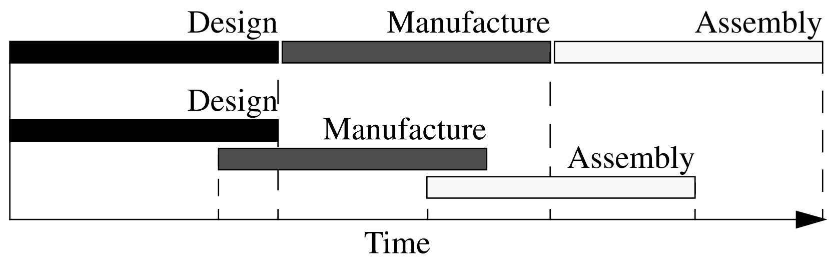


Figure 6.3 - Increased Integration Leading to Reduced Lead Times.

It is clear from the above description that the standard JSP does not realistically represent these problems and hence is very limited in its application. At this point a specification for the Complex JSP was developed to allow for these additional requirements.

The Complex JSP is defined as follows:

i) There are k machines and n different *jobs*.

ii) Each job is composed of a set of m or less *stages*.

iii) Each stage requires a specific operation.

iv) Each stage has a standard processing time.

v) Each stage must be carried out in a specific order.

vi) A job can visit the same machine one or more times.

vii) There are interdependencies between different jobs.

viii) Operations cannot be interrupted.

ix) Each machine can process only one job at a time.

x) Release times and due dates are specified.

xi) Machines can perform several operations.

xii) Machines perform operations with different efficiencies.

xiii) Machine maintenance times can be specified.

xiv) Machine setup times for operations can be specified.

xv) Due date penalties can be specified.

xvi) Multiple scheduling criteria can be optimised.

The Complex JSP model has a much greater level of complexity and range of possible applicability. There are still n-jobs to be tackled. Each of these jobs has m stages but instead of specifying a specific machine an operation is specified for each stage. The motivation for operation based scheduling is outlined more fully in section 6.2.2. Each stage is also given a standard processing time which is the time it takes to perform that operation on a ‘standard’ machine. The stages must be processed in a specific order as before but a stage can involve going on a specific machine more than once, consecutively if necessary. Jobs are not given precedences but interdependencies can be specified. For example if a job requires the completion of other jobs before it can start this can be specified within the problem. Operations cannot be interrupted unless it is for maintenance. It is possible to specify whether a job should wait until after maintenance. The jobs also have release times and due dates. The release times allow scheduling to be integrated with previous events such as the completion of a design task or the delivery of materials which prevent a job starting. Due dates can also be specified and the severity of penalties on these due dates can also be set. Hard due dates can be given large penalties but ones where slippage is allowed can be less heavily penalised. Machines can be flexible and are able to perform more than one type of operation. The time it takes the machine to process a given stage is dependent on its efficiency at performing the operation required at that stage. With the machines performing different operations machine setup times must also be considered within the schedule plans. Finally, scheduling is not just about minimising makespan, in complex scheduling it also involves the consideration of several conflicting criteria. This is discussed further in section 6.2.3.

6.2.2 Operation Based Scheduling

Schedulers have commonly used the classical job shop approach which assumes that any given stage of a job can be undertaken at only one machine and the scheduling task is to order the handling of the stages on the given machines. This is often not the case in practice as a workshop may have several machines of the same type or various machines which can carry out the same operations. Therefore the operations needed for any one stage of a job could be carried out on a number of different machines. There are also other complications involved in this type of model. If a machine is set up for one type of operation and then switched to another type of operation there is some setting up involved. Additionally different types of machines performing the same operation may also take different amounts of time (e.g newer machines will operate faster than older ones). These factors indicate that the scheduler required must be capable of handling operations and not jobs.

6.2.3 Multiple Criteria Scheduling

Scheduling has commonly been tackled using Linear Programming, Monte-Carlo and Heuristic methods (MUTH 63). The aim of such procedures has been to minimise makespan, the total time from the start of the first task to the end of the last. However, other important factors exist within a scheduling task. Regulated allocation of resources, limits on the allowed number of tasks in progress, the completion of tasks by specific deadlines, interrelationships between tasks and conflicts for shared resources are just a few examples. Solutions are sought which strike a balance between these potentially competing requirements.

As a simple example, a workshop with limited space may be considered, limiting the amount of work in progress. The objective is to minimise the time taken to complete a set of tasks. Three cases are shown in Figure 6.4 demonstrating various cases in the trade-off set. If the level of work in progress (x) is specified it can be seen that the trade off solution (Case 2) gives the minimum makespan (t) for that level of work. Cases 1 and 3 represent solutions that minimise makespan and work in progress respectively.

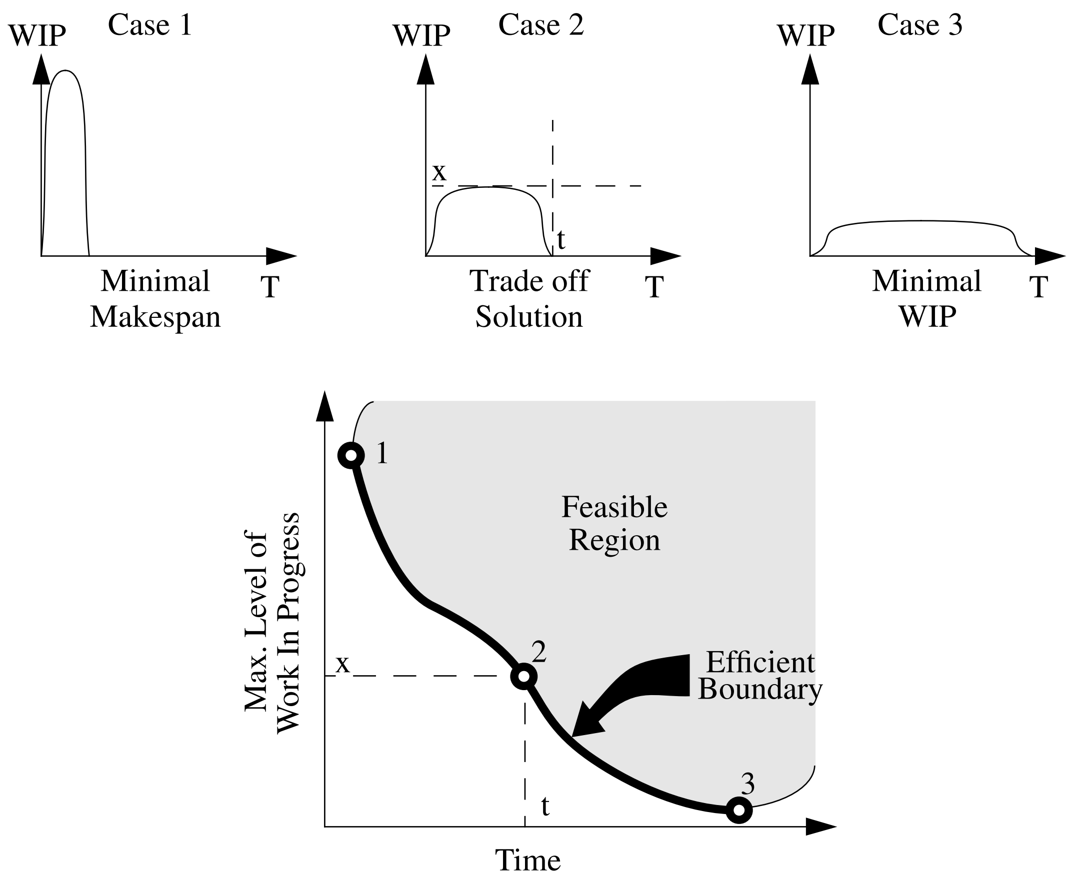


Figure 6.4 - Trade-off in Scheduling

These trade-offs can be more clearly shown on a trade-off curve. The locations of the three cases discussed above are shown on the curve. The solutions to the problem will be confined within a feasible region which is defined by system limitations and imposed constraints. Hence, the aim of scheduling in a multi-criteria context is to generate the trade-off boundary. This boundary can then be used to choose the solution that has the most desirable balance of criteria for the given set of circumstances. The most desirable balance is obviously dictated by the priorities in question.

6.3 Construction of the MCGA Scheduler

6.3.1 Data Representation

The complexity of the scheduling problem can no longer be represented by a single table showing jobs, stages, machines and times as in Table 6.1. Several tables relating to different aspects of system specification are needed to fully specify the problem and these are explained below. A standard input data file which is used with the scheduler can be found in Appendix D. The system uses the user defined clock tick as a time unit. The clock tick can be of any real-time duration (1 minute, 10 minutes, 1 hour, etc.). However, the higher the definition (smaller the clock time) the longer the scheduler will take to run for a given length schedule as there will necessarily be more clock cycles.

The machine has several standard problem variables: Number of Machines, Number of Operations, Number of Jobs, Maximum Number of Stages, Maximum Number of Dependencies and Maximum Number of Maintenance Periods. It also has a variable which indicates whether job splitting is allowed. Job Splitting occurs when a job is on a machine and scheduled maintenance has to take place before the job is complete. This results in the job being stopped and restarted after maintenance. If job splitting is not allowed the scheduled job will not be started until after maintenance has ceased.

The first table used is the machine operating efficiency table (Table 6.5). This is used to show the efficiency of machines when performing the different operations. Those with an efficiency of 1.0 perform operations in the standard specified time. Those with an efficiency greater than 1 perform it faster whilst those with less perform it slower. Machines with efficiency 0 cannot perform the operation. For example machine 0 can perform Op0 with 100% efficiency and Op3 with 50% efficiency but cannot perform any other operations.

Table 6.5 - Machine Operating Efficiency Table

Machine	Op0	Op1	Op2	Op3	Op4	Op5
M0	1.0	0	0	0.5	0	0
M1	0	0.5	0	0	0.8	0
M2	0	1.0	0.8	0	0	0.8
M3	0.5	0.8	0	0.3	0	1.0

The machine maintenance times are shown in Table 6.6. If the machine requires new parts to be fitted or to be stripped down and cleaned routine maintenance times can be allocated. For each machine the number of periods is noted and for each of these periods a time range is specified. For example, machine 3 has 2 periods where it is not available, 20-35 clock ticks and 60-64 clock ticks.

Table 6.6 - Machine Maintenance Times

Machine	Num	N/A 0	N/A 1	N/A 2	N/A 3
M0	2	50-60	100-110		
M1	1	20-25			
M2	4	10-15	25-30	34-37	50-52
M3	2	20-35	60-64		

If a machine is able to perform several different types of operation the transition between operation types may require some setup time. These times are specified in Table 6.7. The current operation is on the left and the next operation reads down from the top. If a transition from Op3 to Op4 takes place 2 clock units will be required to complete this transition. Inclusion of this facility allows the MCGA to experiment with performing all of one type of operation on one machine, hence no setup time would be needed, or changing machine operation types. The effects of this can only be found through simulation, since it is case dependent.

Table 6.7 - Operation Setup Times

	Op0	Op1	Op2	Op3	Op4	Op5
Op0	0	4	2	2	3	1
Op1	3	0	1	5	4	2
Op2	3	2	0	2	3	1
Op3	4	3	2	0	2	1
Op4	3	5	2	1	0	2
Op5	5	4	4	1	2	0

In Table 6.8 each job has a number of stages each of which is specified in terms of two numbers. The first is the operation type which has to be performed at that stage. The second is the standard time required at that stage.

Table 6.8 - Job-Stage Table

Job	Num	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4
Job 0	2	2,4	4,10			
Job 1	4	1,6	2,3	5,8	0,3	
Job 2	3	4,5	1,3	2,8		
Job 3	4	0,7	2,7	4,3	5,1	
Job 4	5	1,4	0,5	1,7	3,9	4,6
....	....	....	....	....	....	....

If Table 6.8 and Table 6.5 are brought together the effect of efficiency can be demonstrated. For an example let job 3-stage 0 be chosen from Table 6.8. This stage requires operation 0 to be performed. Using Table 6.5 it can be seen that machine 0 and machine 3 can perform this operation. If machine 3 is chosen this machine performs this operation with efficiency 0.5. The total time taken for this operation is given by:

$$\text{Time} = \frac{\text{Standard Time}}{\text{Efficiency}}$$

Hence, on machine 3, job 3-stage 0 will take:

$$\text{Time} = \frac{7}{0.5} = 14 \text{ clock ticks.}$$

Similarly, job 3-stage 0 will only take 7 clock ticks on machine 1, because the efficiency is 1. If the result of this calculation is a decimal it is rounded up to the nearest clock tick. This means that over estimates will be provided. If a shorter clock tick unit is used this error becomes smaller but as mentioned earlier the simulation will take longer.

The final table (Table 6.9) used specifies the earliest start times, dependencies and deadlines for each job. The column labelled ‘start’ shows the earliest time that job can commence. This may be due to a late delivery date of raw material or the date for completion of a part design. The next column specifies the number of dependencies that this job has. For example job 3 has 2 dependencies, job 7 and 8 (from the following columns). This means that before job 3 can start both job 7 and job 8 must be completed. Finally, the last two columns specify the deadlines and penalties for the jobs. The deadline is the clock time at which the job is due. After this point penalty costs are incurred. Job 3 has a stricter deadline that the other jobs because it has a 5 unit cost penalty. Job 4 incurs no penalties whatever time it finishes.

Table 6.9 - Job Data

Job	Start	Num	Dep 0	Dep 1	Dep 2	Deadline	Penalty
Job 0	10	3	6	8	9	100	1
Job 1	0	0				50	1
Job 2	5	0				90	1
Job 3	0	2	7	8		100	5
Job 4	0	1	16			0	0
....	....	....	....	....	....	....	....

6.3.2 String Configuration

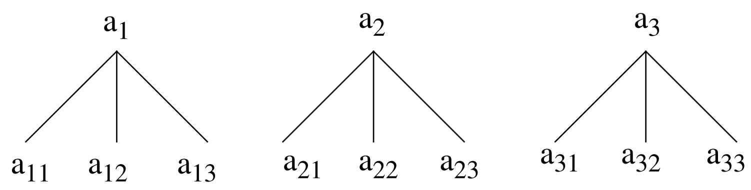
Due to the nature of this scheduling problem it is not possible to use a normal string which is partitioned into machines. This is because the possible interchange of stages between machines would mean that although strings would remain the same length overall, the boundaries between machine partitions would change as genes move around in the string. This could also cause the GA to introduce duplicates during crossover. The system used here still partitions the string into machines but it holds multiple copies of stages which occur on several machines. By using a group of activation genes inserted at the start of the string it activates/disables the duplicated stages within the string. This technique was developed from the ideas of the Structured GA (DASGUPTA 92).

6.3.3 Structured GA

Instead of using a single level string, structured GAs use a multi-level structure of strings which are activated and deactivated by genes at higher levels. Figure 6.10 shows the structure of a simple structured string.

Each string is manipulated by crossover and mutation as in a simple GA. The main difference occurs when fitness is determined. Instead of applying a fitness function to

Structure



String Encoding

a_1 a_2 a_3 a_{11} a_{12} a_{13} a_{21} a_{22} a_{23} a_{31} a_{32} a_{33}

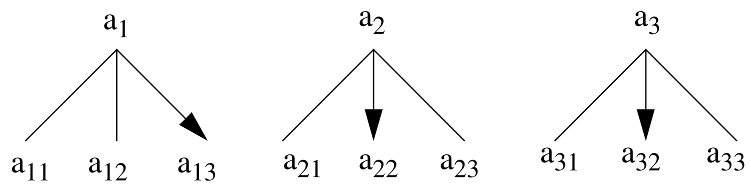
Figure 6.10 - The Structured GA

the whole string, only those parts of the string which are activated by genes at higher levels in the hierarchy are used. For example if a_1 and a_2 were activated, genes a_{11} a_{12} a_{13} a_{21} a_{22} a_{23} would be used in the evaluation of fitness. The passive parts of the string are neutral and are carried along in the redundancy of the genetic material. During mutation and crossover any changes which alter high level genes cause switches in the active sets. This means effectively that multi-bit changes can occur within the active string in one cycle. This permits faster adaptation to environmental changes. Structured GAs are not easily trapped within local optima as they have the ability of being able to carry out multiple bit changes with the modification of a single high level bit. They are also able to act rapidly to the selective pressures of a changing environment.

6.3.4 Selective Gene Activation

The structured GA techniques are based on a binary string. However if the concept is extended to use an integer string a different type of activation can be initiated. The string is the same as before but instead of being able to turn on and off sets of genes the upper level genes can now choose which genes they activate (Figure 6.11).

Structure



String Encoding

a_1 a_2 a_3 a_{11} a_{12} a_{13} a_{21} a_{22} a_{23} a_{31} a_{32} a_{33}

Decoded String (arrows indicate activation)

a_{13} a_{22} a_{32}

Figure 6.11 - Gene Activation

6.3.5 String Construction using Selective Gene Activation

In the context of the scheduler, selective gene activation can be used to choose which jobs are performed on which machines. This technique is demonstrated here using a simple example. Table 6.12 is a table of the machine operating efficiencies and Table 6.13 shows the Job-Stage data. Each stage of a job is now defined in terms of the operation that needs to be performed at that stage. Operations can be thought of as specific tasks (e.g drilling or milling in a machine shop) and it is usually the case that more than one machine can perform this operation. Operations can still be restricted to a single machine (Op0, Op1, Op2 in this example) so the simple JSP can still be tackled.

Table 6.12 - Machine Operating Efficiency Table

Machine	Op0	Op1	Op2	Op3	Op4	Op5
M0	1.0	0	0	1.0	1.0	0
M1	0	1.0	0	1.0	0	1.0
M2	0	0	1.0	0	1.0	1.0

The data can be interpreted as follows: Job 0-Stage 1 involves Op4 which can be performed on either Machine 0 or Machine 2.

Table 6.13 - Job-Stage Table

Job	Num	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4
Job 0	3	1,5	4,3	1,4		
Job 1	5	2,3	1,6	3,5	1,3	4,4
Job 2	3	0,4	2,3	1,6		
Job 3	4	5,4	2,5	1,3	0,5	

The string is made up of n+1 distinct segments, where n is the number of machines and the extra segment is made up of a string of activation genes. Each of these activation genes will correspond to one of the machine choices (i.e. operations which can be performed by more than one machine) so in this case there will be 4 activation genes. This is because only operation types 3,4 and 5 can be performed on more than one machine, the other operations being limited to a single machine, as Table 6.12 shows. In each of the machine segments there is a gene for each of the stages which are to be performed on that machine, including those which could appear on two or more machines. Hence the string is a concatenation of these segments:

Machine Choices - Machine 0 - Machine 1 - ..... - Machine i - ..... - Machine n

In the case of the example we have:

Machine Choices = j0s1, j1s2, j1s4, j3s0

Jobs on Machine 0 = j0s1, j1s2, j1s4,j2s0, j3s3

Jobs on Machine 1 = j0s0, j0s2, j1s1, j1s2, j1s3, j2s2, j3s0, j3s2

Jobs on Machine 2 = j0s1, j1s0, j1s4, j2s1, j3s0, j3s1

To translate this into a string which can be manipulated by the MCGA several encoding steps take place. First the machine choice section is constructed by concatenating the machine choices in ascending job-stage order. In this example there are four machine choices: j0s1, j1s2, j1s4, j3s0. This means that there are four integer activation genes in the machine choice section. Taking the first gene, j0s1, this is of type Op4. Looking in Table 6.12 this can go either on machine 0 or on machine 2. Hence the value of the gene will be restricted to either 0 or 2. For j1s2, this can go on machines 0 or 1, hence this gene will be restricted to either 0 or 1 and so on for the other machine choices. Following this each machine section is constructed. These sections are similar to the Travelling Salesman Problem (Chapter 3) in construction. Taking Machine 0 as an example there are 5 job stages so this string section will be 5 genes long. Each job is simply represented by its job number and its stage number is not used. For this example a sample string would be 0,1,1,2,3. Any permutation of these five genes is legal; for example 1,2,0,1,3 or 3,1,2,1,0. The order of the jobs form a preference list for that machine with the first job being attempted given highest priority. The stage numbers do not need to be included due to the implied ordering of jobs. In the case of machine 1 there are three 1's, but in order for j1s2 to be started j1s1 must already have been completed, hence stage 1 must come first. This encoding is repeated for all machines until the string is complete. The initial process of constructing the string is done automatically from the input datafile. The user does not need to specify machine choice restrictions, segment lengths and job contents or machine sections.

As the system uses an indirect notation a schedule builder is used. This was done to allow the extra functionality required. The schedule builder requires a preference list for each machine in order to carry out processing. Within these lists each job stage should only appear once.

To demonstrate the decoding process a sample string from the above example is decoded. The string is first separated into machine choice and machine sections.

Machine Choice	Machine 0	Machine 1	Machine 2
0, 0, 0, 2	3, 1, 2, 1, 0	2, 1, 1, 0, 3, 3, 0, 1	3, 1, 0, 1, 3, 2

The machine choice string is now used to activate the corresponding jobs in each of the machine lists. In all other lists these jobs will be removed. For example, the first gene in the machine choice list is for job 0 stage 1. This job can either go on machine 0 or machine 2. In this case machine 0 has been selected hence job 0 stage 1 must be removed from machine 2. Looking at the Machine 2 there is only one job 0 so this is removed. If there is more than one stage of the same job on the machine the implicit job ordering is used. For example, the second machine choice gene belongs to job 1 stage 2. Machine 0 has been chosen so the other machine location for this job, machine

1, has to be removed. It can be seen that machine 1 has three job 1 genes. By referring to Table 6.12 and 6.13, job 1 stage 2 is the second possible job 1 stage to appear on machine 1, the first being job1 stage 1. Hence the second 1 is removed from the machine 1 list. If this is continued for all machine choice sections the following machine preference lists result:

Machine 0 Preference List:	3, 1, 2, 1, 0
Machine 1 Preference List:	2, 1, 0, 3, 0, 1
Machine 2 Preference List:	3, 1, 3, 2

All the extra jobs have been removed from the machine sections and no duplicates exist. Using these lists the schedule builder can now be invoked to build a schedule and collect statistics on the data.

6.3.6 Crossover and Mutation Operators

A complete string operator cannot be used on the string because it is separated into several distinct sections. Instead crossover is applied to each of the individual sections in the undecoded string. The machine choice section is crossed using a simple two point crossover. This means that two random points are selected within the machine choice string section and the genes between these points are swapped between partners. This is valid as this section is not order based.

Within each of the machine job list sections a new operator is employed based on a modification of the enhanced edge crossover operator (STARKWEATHER 91) used in Chapters 3 and 5. In these sections of the string the two-point crossover operator cannot be used because it would introduce repeated jobs. The modified operator is slightly more complex, but it ensures that repeated jobs are not created and jobs are not lost during crossover. When two partners are to be combined the first job of parent 1 is chosen first. Then the job directly following this in the strings of both parents is found. The next job is chosen randomly to be one of these two jobs and this is then used to find the subsequent job in the same way and so on. If the chosen job is found to be a duplicate then the second of the two jobs is chosen. If this job is also a duplicate then the next job is chosen as the first job in parent 1's machine job list not yet used. This process is repeated until all the genes have been used. The whole procedure is then repeated for parent 2 using its first job. This procedure is shown in Figure 6.14.

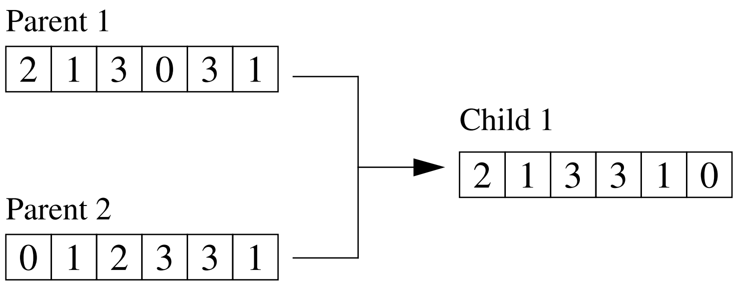


Figure 6.14 - Crossover in Machine Sections

Using parent 1 the process is started from job 2. Then the following jobs in both parents, 1 and 3, are found and a choice is made between them at random. If 1 is chosen the next jobs are 3 and 2. Job 2 only appears once and has already been used so job 3 must be used. Following job 3 in the parent strings are jobs 0 and 3, and 3 is chosen at random. Following the second job 3 is job 1 in both cases so this is chosen. Job 0 is the first job in parent 1 unused so is inserted at the end. As explained before, this process is repeated from the first job in parent 2 to create another child. This operator tries to consider the immediately following job and also the overall ordering by trying to select the earliest jobs in cases of conflict. If two identical strings are present the children will be the same as the parents.

Mutation is a much simpler operator. Again mutation is performed on separated sections of the string. In the machine choice section, if a gene is chosen it is randomly reset to another machine choice. For example if there is a choice of three machines 0, 2 and 3 for a specific job and the current machine selected is 2 it would change to either 0 or 3. In the machine job list sections a gene selected for crossover will simply be swapped with another randomly selected gene in that section.

Due to the combinatorial complexity of the problem an exhaustive 2-opt search is not possible. Instead a small search is initiated which behaves like an intelligent mutation operator. This search chooses a first job and then a second is picked randomly and the two are swapped. If this swap provides improvement in at least one of the criteria without loss in any of the others then the swap is accepted. If this is not the case the jobs are swapped back and another job is picked randomly to swap with the first. This process is repeated up to 5 times if a good swap is not found then the string is left untouched. These swaps occur within each machine section.

6.3.7 Schedule Builder

The schedule builder takes the strings provided by the MCGA and decodes them into machine preference lists as outlined before. These lists are then used to create the schedule. The builder will always create the same schedule for a given input string. The builder starts by resetting the clock to zero. Then the system enters a loop which is terminated when all jobs have finished. This loop starts by looking at each machine to determine its current state, either setup, maintenance, working or idle. If the machine has finished a job, the job is marked complete, its finishing time is noted and the machine set to idle. If the machine is either working, in setup or in maintenance the time to completion of this task is decremented by 1 clock tick. If the machine is working, then the time of utilisation of that machine is incremented.

Following this the preference list of each machine which is idle is examined to find the highest priority job which is available and has not yet been performed. Availability is defined in terms of several factors. A job is available if:

i) All previous stages of that job are complete.
ii) All dependencies of that job are satisfied.
iii) The machine is set up for that job.
iv) If job splitting is not permitted the job must be completed before any maintenance begins.

In the cases of i), ii) and iv) the next job in the preference list of the machine is examined to see if it is available. If iii) is the case the machine begins its setup routine for the operation of that job. The time required for setup is determined by the current machine operation and the operation required by the new job and is found in the operation setup time table (Table 6.7). If no job on the list is available the machine is left in its idle state. If a job is found then the time required to complete that job is calculated and it is placed on the machine. The clock is then incremented and the process returns to the start of the loop.

When all jobs are complete the code exits the loop and various statistics can be calculated such as makespan, penalty costs, average job times, machine utilisations etc.

These are passed back to the MCGA for analysis. The code will also generate plans showing each machine, what jobs and stages are being performed, any setups, maintenance times, idle times and also statistics from the run.

The diagram of the MCGA Scheduling System is shown in Figure 6.15.

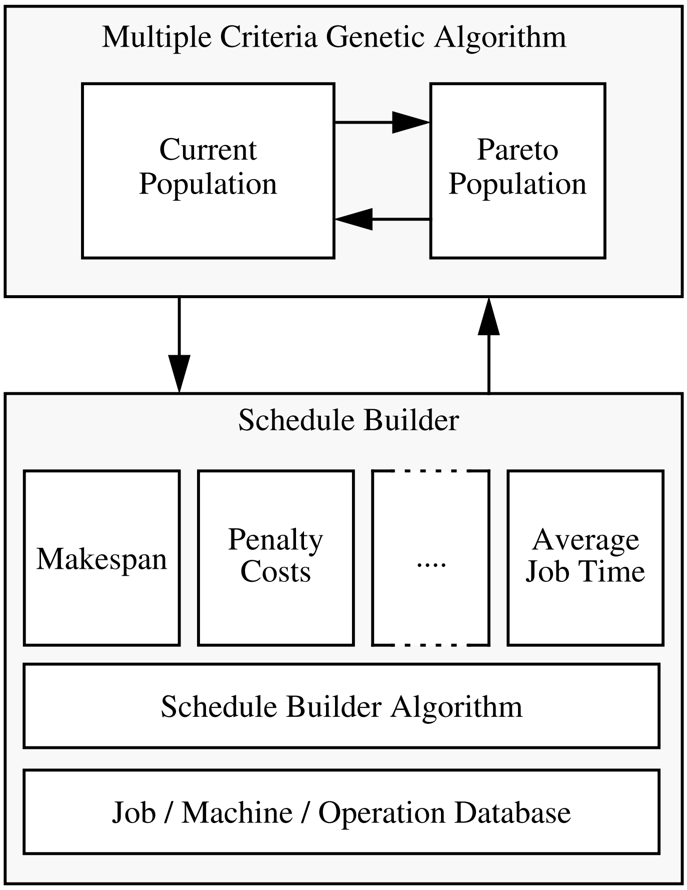


Figure 6.15 - The MCGA Scheduling System

6.4 Scheduler Validation and Testing

The MCGA Scheduler was tested on four examples. In the first instance the scheduler was tested to check for correct operation by examining the results of the output of the MCGA. The schedule builder automatically verifies that all jobs have been completed when a schedule has been built. The scheduler was then used to investigate robustness. Thirdly, a practical example involving a shipyard steel shop with shared resources was tackled. Finally the shipyard problem was re-examined to show how the MCGA could be used to analyse the processes involved in terms of evaluating possible refinements.

6.4.1 A Complex Scheduler Example

The first example uses the data file shown in Appendix D. There are 4 machines, 20 jobs with a maximum of 5 stages and 6 types of operations. It includes all of the complex features discussed earlier. The search space contains at most 3^{12}x2^{23}x16!x20!x35!x35! possible combinations. This is calculated by finding the number of machine choices, there are 12 stages with 3 choices (3^{12}) and 23 stages with 2 choices (2^{23}). This is then multiplied by the number of possible combinations in each of the machine sections (16!, 20!, 35!, 35!). The search space will actually be smaller than this due to the fact that some of the jobs are repeated in the machine sections. This is the same as having the same city occurring twice or more times in the Travelling Salesman Problem. This test was run using two criteria, makespan and average job time, the latter being defined as the sum of time taken by each job divided by the number of jobs. The aim of the average job time criterion is to reduce the number of jobs which are in progress at one time. This criterion may be used in small workshops where storage space is at a premium.

For the tests a population of 500 was run over 100 generations. A typical run on this example takes about 25 minutes. The graphs showing the development of the Pareto Population over the 100 generations are shown in Figure 6.16. The effect of job splitting was also tested. Figure 6.17 shows the final population of two runs, with and without job splitting. Additionally, schedule plans are also shown in Appendix D for minimum makespan, minimum average job time and compromise cases.

Firstly the plans were examined for validity. The job list was checked to ensure all jobs were completed. The timings of the jobs were checked as were the setups when required. The MCGA Scheduler generates an average of 72 solutions per test over a 20 test run. It was found that multiple solutions would generate the same trade-off values even though they were genotypically different. This is why the graphs do not appear to have all of the 72 solutions present. Although, the graphs show the trade-off between the scheduling criteria these are not as well defined as those in the test examples in Chapter 4. The schedule plans reveal that, in general, in order to reduce makespan the average job time would increase. This is because the scheduler has to be more efficient with resources, so job stages need to be moved around so that they fit into gaps within the schedule. To minimise average job times, the schedules which group several job stages together were favoured. This leads to large gaps in the schedule where jobs will not commence until necessary resources are free. In the extreme case minimising average job time is equivalent to a 'no-wait' job shop where no gaps are allowed between job stages.

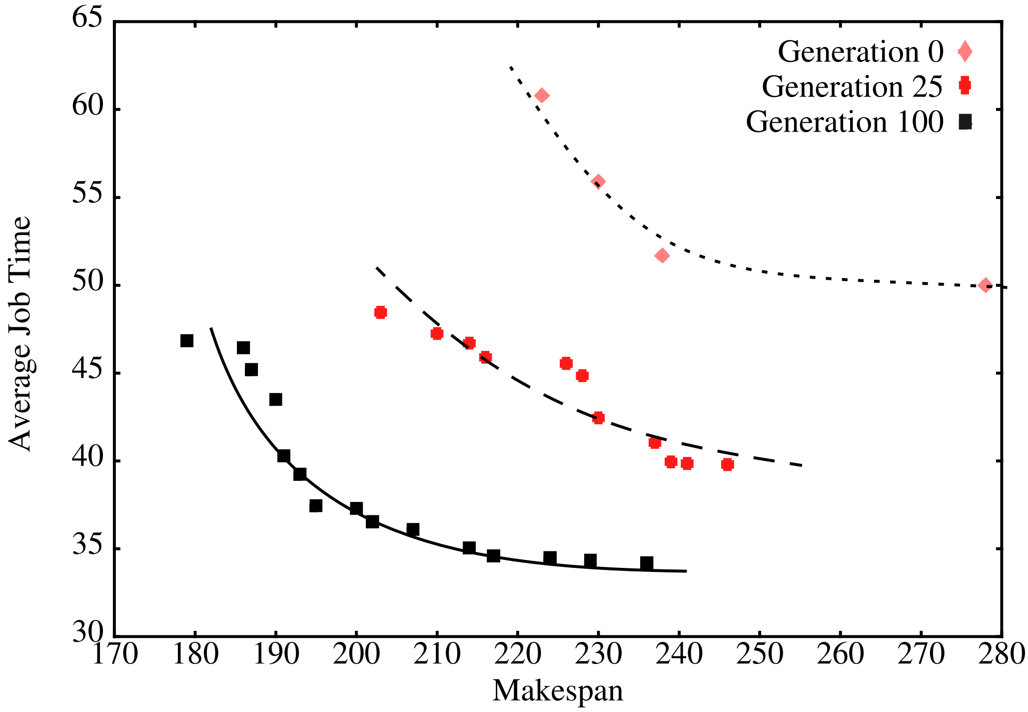


Figure 6.16 - Generation of Pareto Solutions

As mentioned earlier Figure 6.17 shows the effect of job splitting. Clearly using job splitting has a positive effect on both makespan and Average Job Time. If job splitting is not permitted then jobs are held until such time as they can be completed without any interruption. However if there are short jobs which can be completed these will be selected to fill in the gaps. The problem arises when there are long jobs and short gaps between maintenance periods. The jobs are continually put back until large enough windows exist. This sometimes leads to long delays especially on machines with lots of scheduled maintenance. In this case the best schedules are those where as many jobs as possible are placed onto other machines effectively bypassing the troublesome machine. The current algorithm does not allow the jobs to be removed from machines by other jobs of higher priority. This is a possible extension for future work.

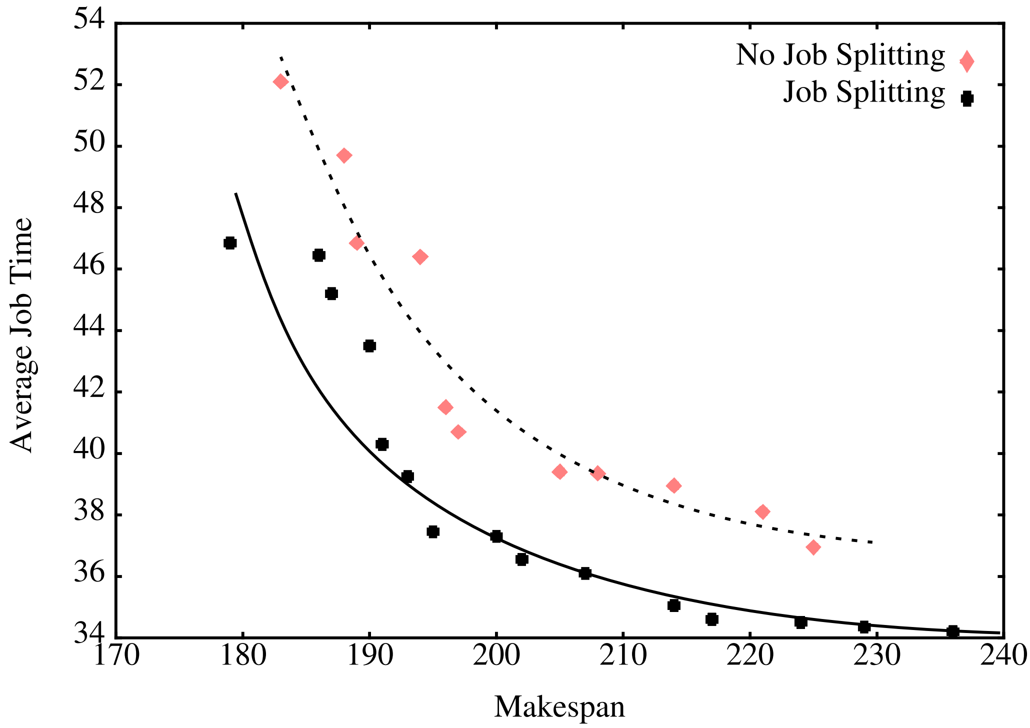


Figure 6.17 - Job Splitting vs No Job Splitting

6.4.2 Scheduling for Robustness

In real working environments schedules often get disrupted. Machines can break down, new jobs can arrive unexpectedly or raw materials arrive late, to name but a few possible occurrences. These disruptions can mean that a schedule collapses and is no longer workable. The schedule will then have to be modified or completely re-worked. The aim of robust scheduling is to try to develop schedules which are workable even when such events occur. This may be done by using a new random element. The program has been modified to introduce periods when machines become unavailable due to breakdown. The effects of such breakdowns are measured in terms of the scheduling criteria, each averaged over several runs of the schedule builder. The schedule which is the most robust will then be the one with the best average values of the criteria for the various breakdown scenarios.

For this test the MCGA was set up as before. Each machine was given a probability of breaking down for 5 clock ticks in 100. Each of the schedules was evaluated 10 times to give an average makespan and variance in makespans was calculated. The number of evaluations was limited to this relatively small number, so the runtime was approximately five hours on a Sun Ultrasparc machine. Job splitting was allowed during the runs.

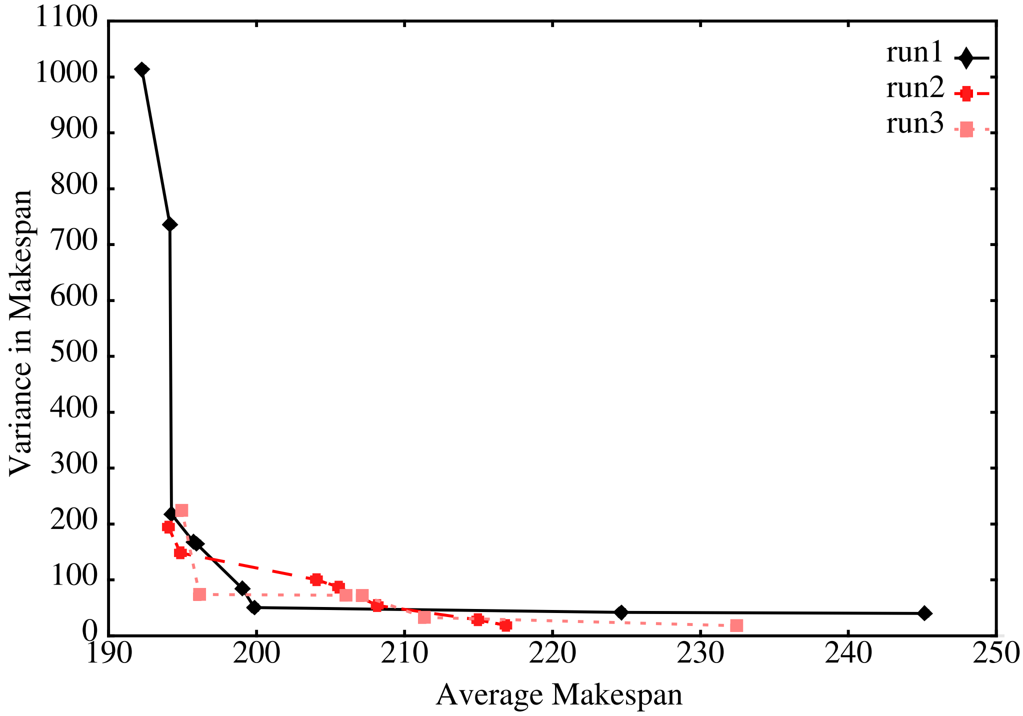


Figure 6.18 - Variability in Robust Scheduling

Although the MCGA was able to find good solutions to the problem, the Pareto set was small (Figure 6.18). The solutions produced by the various runs were not consistent with each other. In the previous two tests the MCGA was able to generate results over comparable criteria ranges and similar distributions in every run. The variability in this example is probably due to the random nature of the testing. Because there are only a small number of evaluations of the schedule per solution, the system will present only an approximate and probably biased indication of robustness. The greater the number of evaluations which are made per solution, the more accurate the results. This is due to the fact that every schedule would be subjected to a greater range of variations and hence the results would be more repeatable from run to run. However the time required to run a 50 evaluations / solution test would be in the region of 25 hours for the population size and number of generations used here. These effects aside it can still be seen that the variance drops as average makespan increases. This is an obvious result since longer makespans can accommodate bigger fluctuations because they have more slack.

6.4.3 Shipyard Scheduling With Shared Resources

The aim of this example* is to schedule work within a shipyard steel shop environment (TODD 97b). There are three machines, a Flame Planer (FP - cuts rectangular plates with weld preparation), a Shell Plate Cutting Machine (PCS - cuts curved plates with weld preparation) and a Dual Plate Cutting Machine (PC2 - cuts profiled plate parts with no preparation). The goals are to reduce the total time taken and comply as fully as possible with a specific assembly schedule. The steel shop is to produce all of the plates for a 24m section of a general cargo ship. This comprises a total of 144 plates of various sizes and cut patterns. The parts to be cut are deck/side/bulkhead plates, shell plates, longitudinal girders, transverse floors and brackets/small parts shown in Figure 6.19 below. The approximate time in minutes to process each piece of raw material is shown in Table 6.20 The complete loading-marking-cutting-unloading process is simulated. The example assumes the components which are cut into more parts will take longer to mark and unload as well as longer to cut.

.

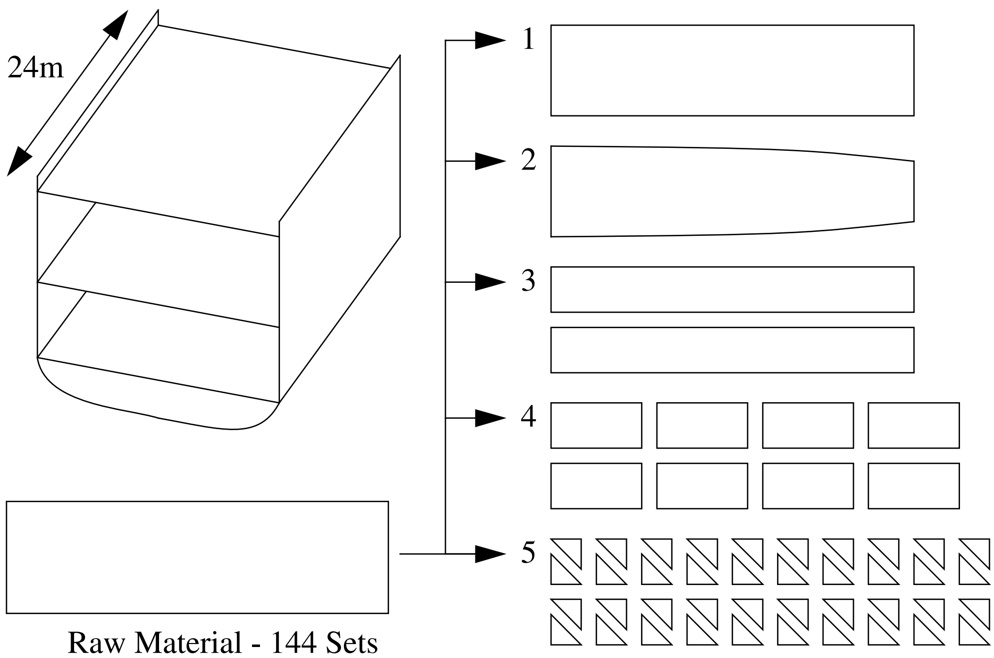


Figure 6.19 - The MCGA Scheduling System

FP and PC2 operations can also be performed on the PCS hence most of the jobs can be placed on one of a choice of machines. The PC2 can cut two plates simultaneously. The machines also run at different cutting speeds. The FP, PCS and PC2 run at 400,

* Supplied by George Bruce of The Department of Marine Technology, University of Newcastle

300 and 450mm/min respectively. This means that jobs can be done faster on some machines than others. The cutting times in the table are shown for a machine cutting at 500mm/min. A crane is needed to load and unload the plates/cut parts. As the crane is a shared resource some jobs may have to wait before they can be placed onto machines.

Table 6.20 - Part Production Table

Type	Quantity	Loading	Marking	Cutting	Unloading	Machine
1	64	15	5	60	5	FP/PCS
2	20	15	5	60	5	PCS
3	24	15	5	110	10	PCS/PC2
4	12	15	10	145	25	PCS/PC2
5	24	15	40	185	150	PCS/PC2

The closer the cutting can be matched to the subsequent assembly schedule, the less work in progress will be accumulated. Every manufactured part is given a due date obtained from the assembly schedule and an associated cost penalty for missing this due-date. This is proportional to the length of time that it is late. The penalties can be increased for critical parts whose delay will have a greater impact on the time of the assembly process.

The above scenario was run using the MCGA Scheduler. A typical run (Population Size 500, Number of Generations 250) takes just over 4 hours on an Sun UltraSparc. The aim of the exercise was to minimise makespan (estimated at around 8500 minutes) and total penalty costs. Figure 6.21 shows a graph of the two criteria for various numbers of generations. It can be seen from the initial trade off set at generation 0 that the quality and number of solutions has increased. It can also be seen that if makespan alone had been minimised without the constraints, high penalty costs would be incurred. One interesting point to note is that the specified assembly schedule cannot be met within the constraints of the system due to the fact that zero penalty has not been achieved. The assembly schedule could be modified but it is better to specify an ideal case assembly schedule and optimise towards it. By the same reasoning if an assembly schedule can be met by the MCGA tool then it would be sensible for the demands on the scheduler to be tightened in order to achieve further process gains.

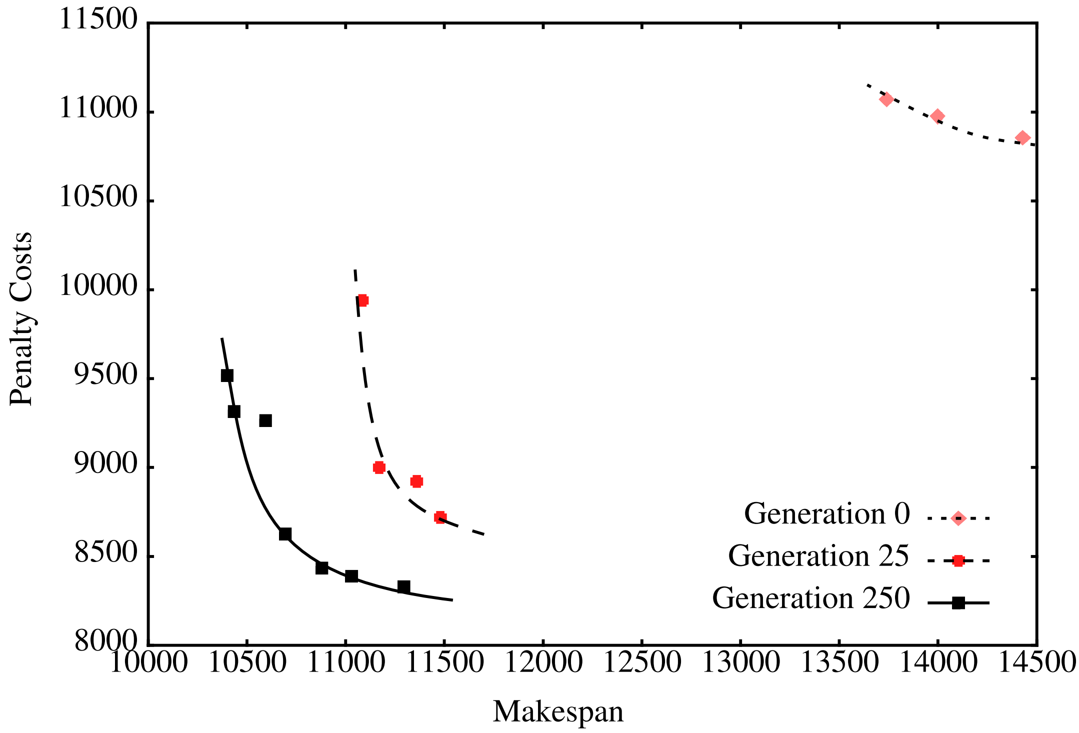


Figure 6.21 - The MCGA Scheduling System

The process plans for several of the schedules were examined. From these plans further insight into the process was obtained. The MCGA performed well in achieving a good balance of solutions which tried to optimise on both criteria. It was found that as makespan was minimised deadlines for jobs were discarded in favour of increasing utilisation. The use of the crane was also more staggered as the scheduler tried to prevent it being required by two jobs simultaneously thereby reducing delays. In circumstances where penalties were minimised, one of two things would occur. Either the scheduler would accept a small slippage in a large number of jobs or it would try to meet as many deadlines as possible at the expense of heavy delays on a small number of jobs. For the purposes of assembly, the first case is more favourable since it would still allow the same assembly plan but slightly delayed. In the latter case changes to the plan would be needed because a small number of components are delayed significantly therefore disrupting the assembly ordering.

6.4.4 Process Analysis of the Shipyard Problem

A significant benefit of the MCGA Scheduler is that it allows experimentation with various process setups. Three further runs were carried out, one with the addition of a

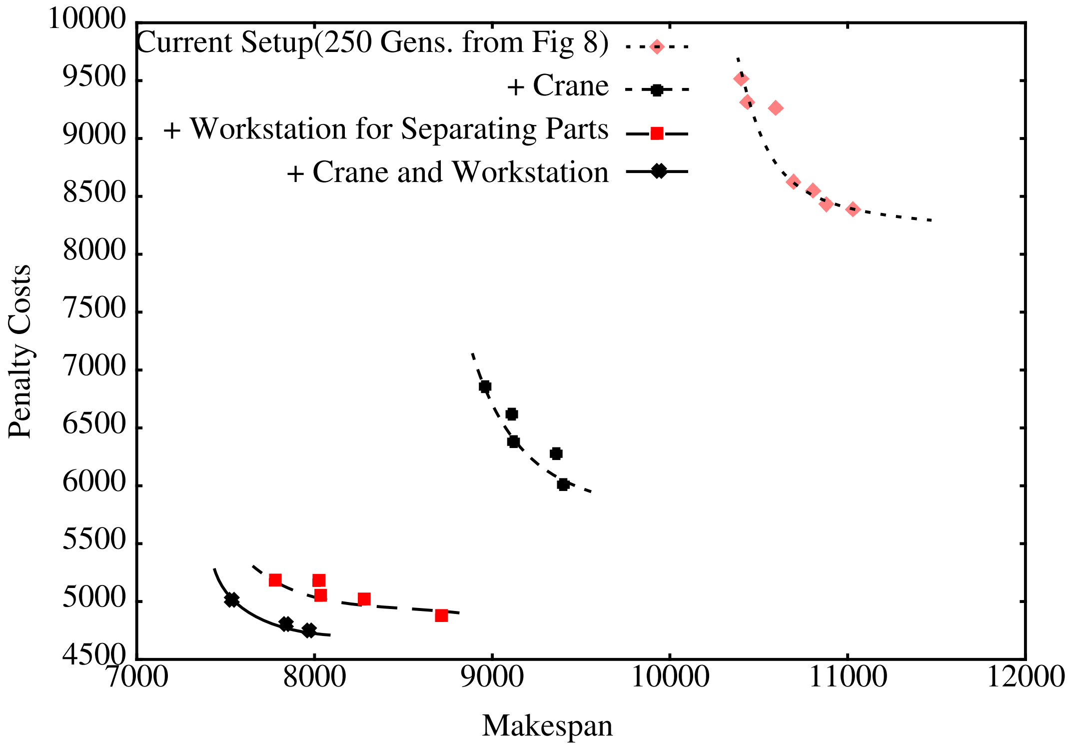


Figure 6.22 - The MCGA Scheduling System

second crane, another with an added workstation for manual separation of plates and one with both crane and workstation present. In environments with shared equipment such as cranes, it is often the case that bottlenecks form around these resources. It can be seen from Figure 6.22 that the benefits of adding a second crane are large. This is also demonstrated in the machine utilisation plots of Figure 6.23. In the first case with only a single crane there are many times when machines are waiting for a loading or unloading operation. When a second crane is added the occurrence of waiting machines is vastly reduced thus reducing total time for completion. On both graphs the number of machines in use drops to two part way through the schedule. This is because the Flame Planer runs out of jobs to perform. This free time could be used to start another project or build up a store of ‘ready-cut’ parts in anticipation of future work.

Returning to Figure 6.22 further performance gains can be achieved by adding a workstation for separating parts. When this is done parts are not fully cut on the machines. Instead they are separated on an additional workstation. This saves a lot of crane time as only one plate needs to be unloaded from the machine rather than each of the separated plates. The added flexibility and speed allows the tasks to be completed in much less time with lower penalty costs. This kind of experimentation enables the user to assess the viability of possible machine upgrades or the addition of extra machines.

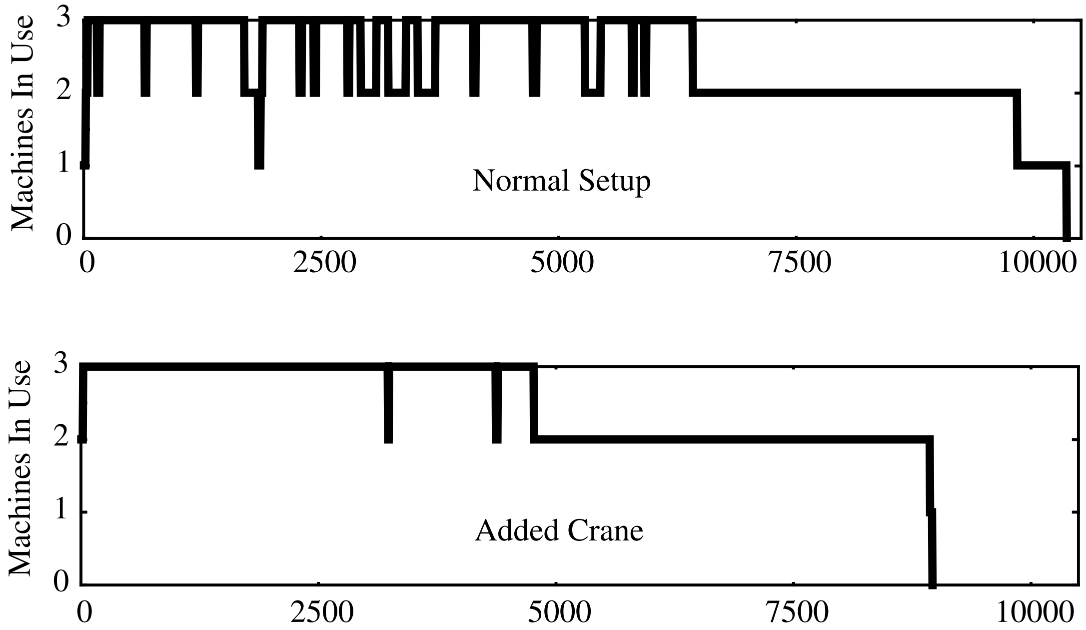


Figure 6.23 - The MCGA Scheduling System

Further work needs to be done to take into account realistic operating conditions. In this example the cranes are assumed to take no time to traverse the steel shop, and this is obviously not true. It has also been assumed that the machines work continuously, whereas manning considerations must also be taken into account. The schedules will have to reflect working hours and holidays to allow efficient use of personnel. In general, the requirement for additional features will depend on the priorities of individual applications. A deterministic scheduler generator of this kind can often be more convenient in judging competing actions than a full-scale Monte Carlo Simulator.

6.5 Chapter Summary

This chapter has investigated the use of evolutionary techniques to solve scheduling problems. The simple job shop model was thought not to be detailed enough for application to many scheduling problems. For this reason a complex scheduling definition was drawn up incorporating many new features, such as the use of multi operation machines, job dependencies and setup periods.

Following this a description of operation based scheduling along with the data required to perform it was given. The construction of the MCGA Scheduler was outlined and the string representation, crossover and mutation operators and the schedule builder were also presented.

The MCGA Scheduler was tested on four problems. The problems variously looked at the performance of the scheduler and also how it could be used to perform process analysis and robust design. A shipyard steel shop example was tackled using the MCGA. This example looked at the effect of sharing a resource and the affect this has on processing time. It also addressed the issue of scheduling to deadlines with various cost penalties.

In the next chapter the MCGA is further enhanced by including additional functionality in an attempt to make the search more controllable from a user point of view. Various techniques are developed, implemented and tested. These techniques allow the user to tailor the MCGA search to their needs.

7 Diversity and Convergence

7.1 Introduction

Design is a process of divergence and convergence. Diversity during the initial design stages is important to explore as many solutions as possible. Convergence is necessary in order to settle on a preferred design. Both of these terms are widely used in the GA community and the link between the action of the GA and the design process means that the GA is a potentially powerful design tool.

In GAs the trade off between diversity and convergence, as with the design process, is an important one. If the maintained level of diversity is too great then the GA will never converge. If it is too low the GA will converge to give poor solutions. However in multiple criteria GAs this relationship is slightly different. The aim is to maintain a diverse range of solutions whilst trying to converge these solutions onto the trade off surface. In this chapter some methods to modify the MCGA search are investigated. These modifications may be to increase the diversity of the Pareto front or to concentrate on one area of interest of the front. In both cases the aim is to enhance search and increase functionality.

7.2 Utilising the Pareto Population

The Pareto Population holds a list of the current set of Pareto individuals. As such it holds invaluable information on the current progress of search. When the MCGA with the Pareto Population was formulated and tested (Chapter 4) it became apparent that this feature could be utilised further to enhance the search process, prompting investigations to move in this direction. Six methods were developed and implemented, three for promoting diversity, two for encouraging convergence and another for providing interactive search. All of these are developed and tested in this chapter.

7.3 Diversification

The aim of the three diversification methods is to encourage the search to quickly spread out and identify the full extent of the Pareto surface. The idea was to obtain a reasonable coverage over the whole space avoiding large ‘holes’ anywhere in the trade off surface. The strategies were examined for the two criteria case against the other divergence strategies and the elitist strategy introduced in Chapter 4. The methods are designed to be applicable to the n-criteria case and are explained as such.

7.3.1 Pareto Injection

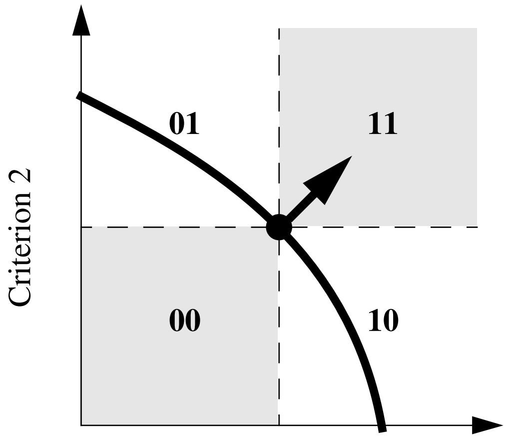
This method, as with the others, uses the Pareto Population to re-introduce individuals to the normal population. The method of picking individuals employs an additional parameter called *proportion*. This parameter is set between 0 and 1 and defines the percentage of individuals that are inserted into the population using the method. For the purposes of these tests proportion was set to 0.5. The greater this value the greater the effect of the selection. However, high values of proportion would reduce the population diversity in the early generations of the search leading to sub optimal solutions being generated. The injection method chooses individuals at random from the Pareto Population and places them back into the normal population. After this the remainder of the population is filled using the roulette wheel method from individuals in the normal population.

7.3.2 Niche Counts

Niches are currently used to share fitness among the individuals within a local region. The more dense the population within a region the more heavily penalised individuals in that region become. This effect causes the population to spread outwards. As the niche size is already defined this can also be used to count the number of individuals within each niche for the Pareto Population. This provides information about the density of each Pareto region. In order to promote diversity the aim is to populate the sparser regions of the surface. In this method the Pareto niche count is used to select individuals with low numbers in their niche. A second variable is defined called *closeness* which ranges between 0 and 1. It was used to provide a threshold at which individuals are selected to be re-inserted into the population. For these experiments closeness was set to 0.2. In this case the individuals in the Pareto Population with a niche count in the lowest 20% band will have a random chance of being selected and inserted. This operation fills a 50% proportion of the normal population. The aim of the process being to equalise the number of Pareto individuals within each niche by encouraging growth in areas which are underpopulated compared to others.

7.3.3 Adjacent Distance

This is the most complex of the three divergence operators. This operator finds, for each Pareto solution, all of the closest adjacent individuals in each direction. This procedure will be explained for 3-criteria space but is generally applicable to n criteria.



Maximise C1 and C2

Figure 7.1 - General Notation

Figure 7.1 shows the general notation used. For each Pareto point lines may be drawn through the point parallel to the criteria axes. Both criteria are maximising ones. The direction of search can be denoted in binary format by using a bit for each criteria. For example, binary 10 means maximise criterion 1 and minimise criterion 2. This notation is used in the MCGA software to define which criteria are to be maximised or minimised. Returning to the case above the binary direction of search is 11 so the search is in the direction of the arrow. As all members of the Pareto set are non-dominated, and they do not dominate any other member in the set they must lie in the non-shaded areas. If this is extended to three dimensions the spaces become cubes then after that hypercubes. The three dimensional case is shown in Figure 7.2. The search area is now defined by three binary digits, with the direction of search in the direction of the arrow. With respect to the current point all members of the Pareto Population must lie in the non-shaded cuboids. There are 2^n cuboids in total for a n-criteria space. All of these cuboids will contain other points of the Pareto Population apart from the cuboid that defines the direction of search and its inverse. For example if criteria 1 and

2 are to be maximised and criterion 3 is to be minimised this is defined by the binary value 110. Its inverse is 001 which is minimise criteria 1 and 2 and maximise criterion 3. All other binary combinations between 000 and 111 define a direction for which an adjacent point can be found to the point in question. By finding the distance to the

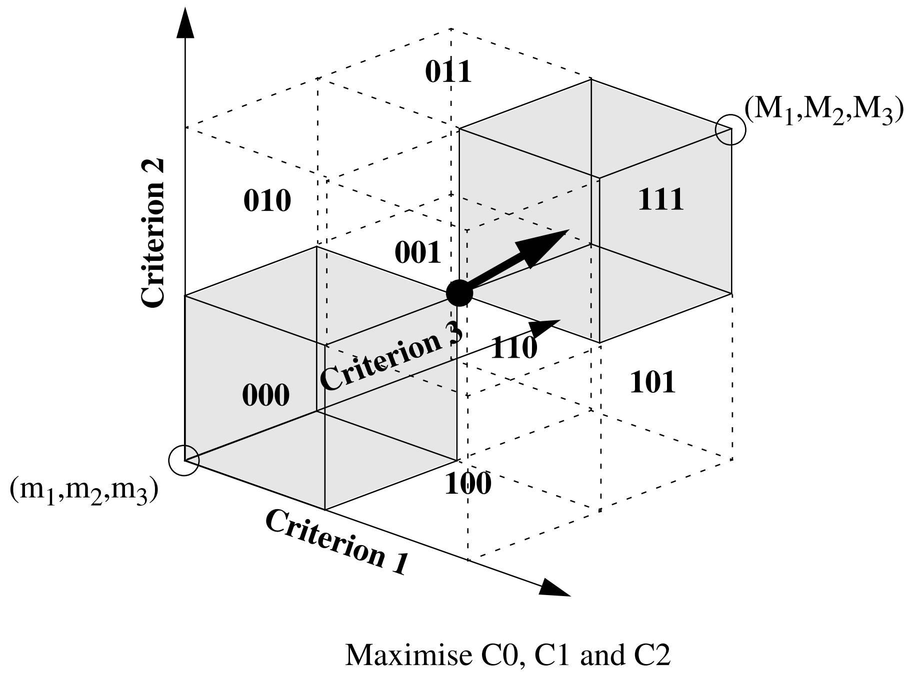


Figure 7.2 - 3 Criteria Case

nearest points in each direction gaps in the Pareto surface can be identified. These distance values can be used like the niche count values in the last routine. However, instead of having one distance value there is a value for each of the hyper-cuboids. The individuals are selected by again using the closeness threshold. The distance of adjacent points for all hyper-cuboids is examined and any with a distance falling in the 20% longest has a chance of being selected. Then instead of inserting just the chosen individual the adjacent individual which shares the distance is also inserted. This is shown in case (i) in Figure 7.3 for a 2-criteria space. In this case there is a large gap between the two highlighted individuals. Hence, if one of these is selected for inclusion it automatically selects its adjacent partner in the appropriate hyper-cuboid. This will hopefully promote a filling effect from both ends of the gap as shown by the arrows. Case (ii) shows a special occurrence at the limits of the Pareto front. When points at the extremes of the Pareto surface have no adjacent individuals in certain cuboids. In these circumstances points are given the opportunity to be selected for re-insertion since this would encourage more solutions at the extremes to be created and possibly increase the range of the Pareto front.

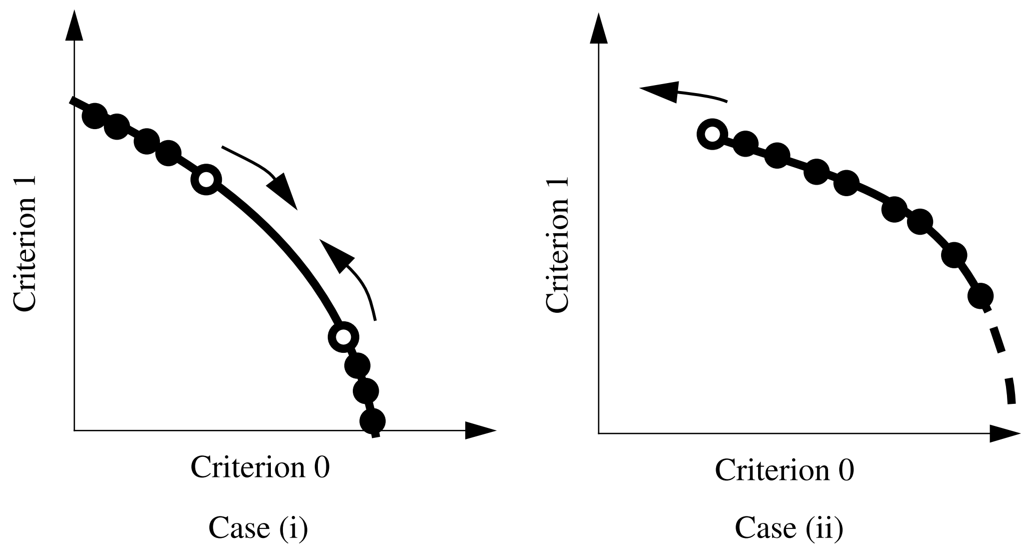


Figure 7.3 - Cases for Selection of Individuals

7.3.4 Testing of Divergence Methods

All of the tests in this chapter were run over 50 generations of 500 individuals with each of these runs being repeated 50 times to give average performances. They were tested on the two criteria case and the graphs plotted. As stated earlier the parameters, *proportion*=0.5 and *closeness*=0.2, were set for all experiments. The three methods were compared to the original Pareto elitist strategy used in Chapters 4,5 and 6.

The aim of the divergence methods was to maximise the explorative properties of the MCGA and try to reach the full extent of the Pareto region as rapidly as possible and also to try and generate a good spread of points over the whole surface. Figure 7.4 shows the progress of the four selection methods in trying to maximise criterion 0 and criterion 1. The greater the criterion value the greater the Pareto region discovered. From the diagram it can be seen that the random injection method quickly begins to

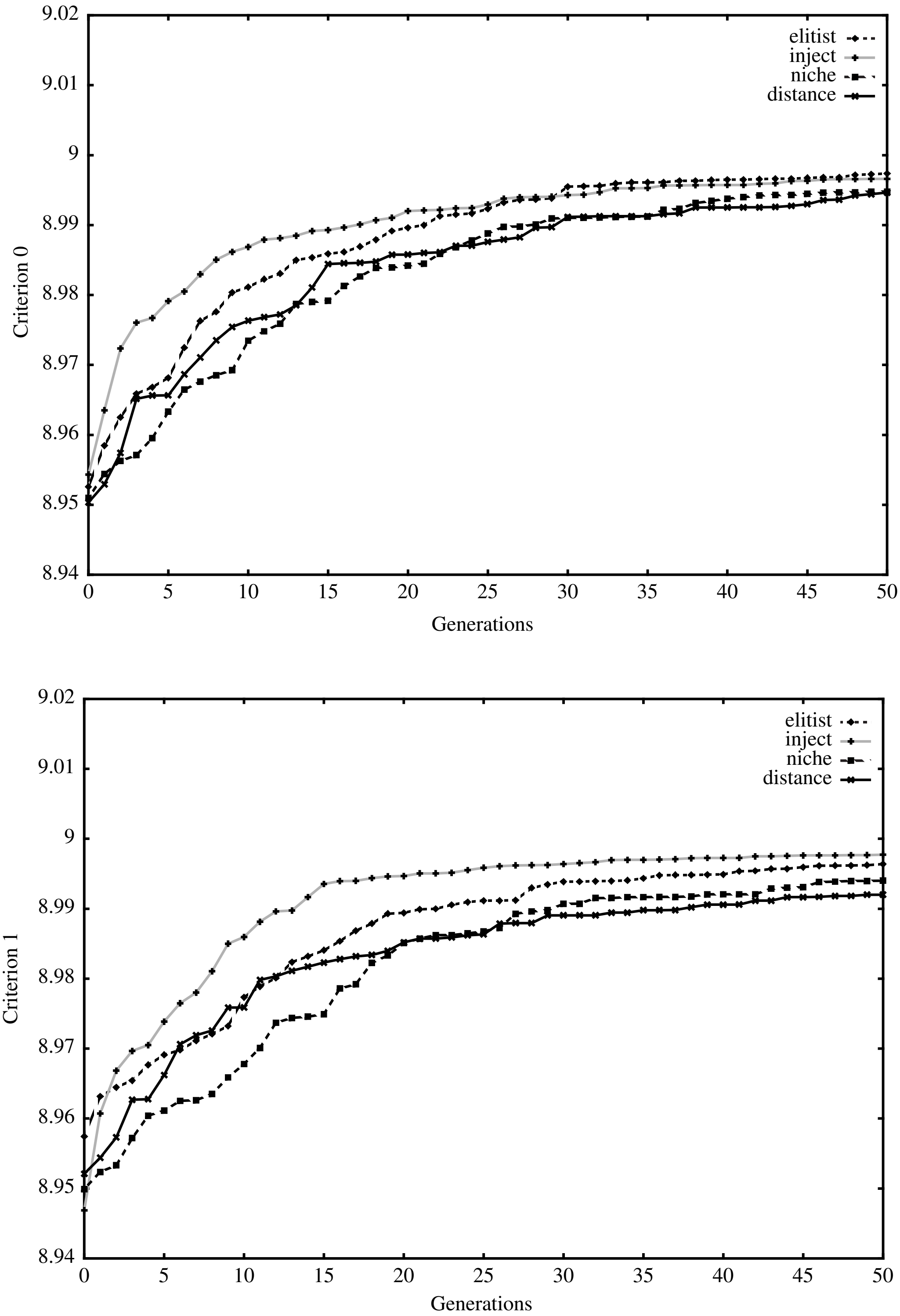


Figure 7.4 - Maximum Value of Criteria

explore the extremities more effectively than the other methods. The elitist method also performs well compared to the niche and distance methods. In the criterion 1 graph the elitist method is actually able to achieve a better final value during the last few generations. The niche and distance methods do not perform as well as the others when seeking extremes. Performance was also investigated by measuring the spread of points by counting the number of individuals in the niche of each Pareto individual and finding the variance of this value. A low variance would indicate about the same number of individuals in every niche and therefore a good distribution. Figure 7.5 shows the variance in the number of individuals in each niche. It is seen that both the niche and distance operators perform very well in regulating the number of individuals. The variance in the niches for the injection method is far greater than the others, indicating that clustering was taking place.

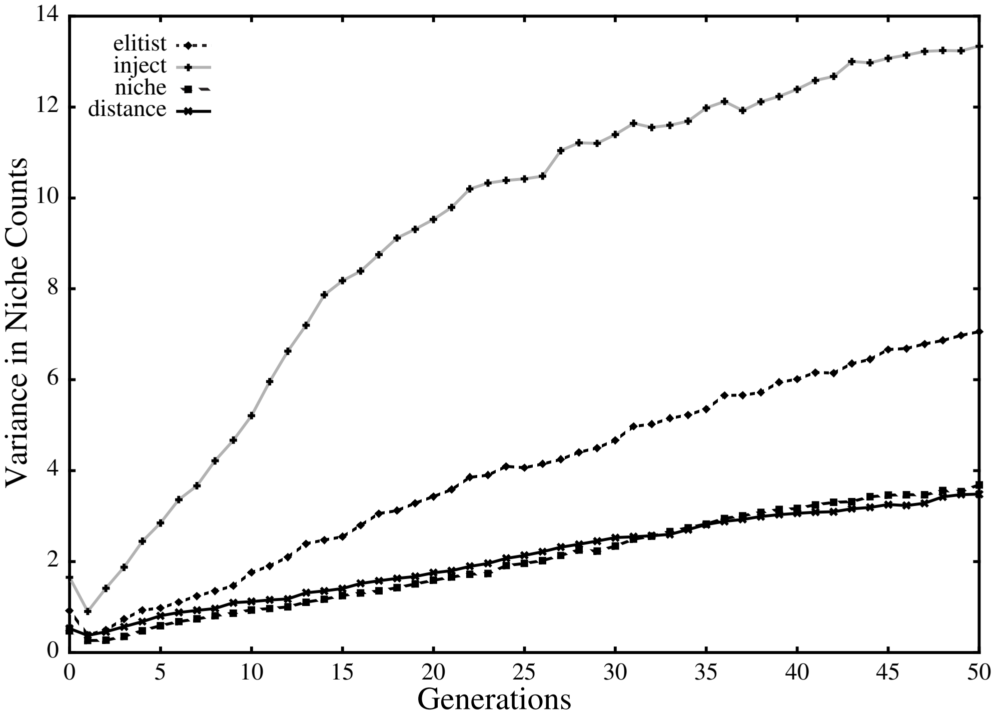


Figure 7.5 - Variance in Niche Counts

The reduction in niche variance in the niche and distance methods is not due to the selective powers of the operators. Figure 7.6 shows the average number of Pareto solutions generated per generation over the 50 runs. From this graph it is clear that the reason the variance in niche count is so low in the niche and distance methods is because the number of Pareto individuals overall is much lower. The variance is more likely to be small as there are less individuals on average in each niche. Looking more closely at which individuals are actually selected to be re-inserted into the population using the niche and distance methods gives indications on why this is the case. When evaluated on niche count or distance very few individuals in the Pareto Population are

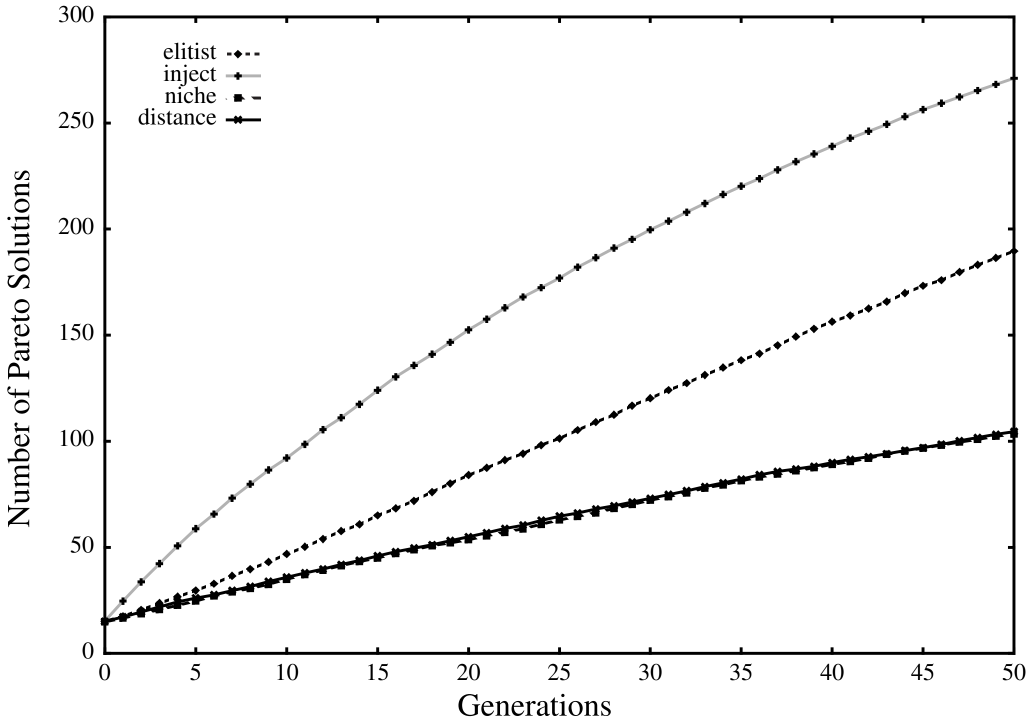


Figure 7.6 - Average Number of Pareto Solutions

actually within the 20% threshold. This means that multiple copies of a few individuals are inserted into the population. When crossover occurs the niche restriction forces a large proportion of the multiple copies of individuals to mate with each other so offspring are created with no genetic changes. Some spreading effect on the population will still occur due to occasional mating between different strings leading to creation of more diverse offspring.

Figure 7.7 showing the final population Pareto solutions highlights the differences between the operators. The elitist method gives good coverage; however a gap can be clearly seen on the plot (highlighted by the circle). The random injection method fares much better and gives good coverage over the whole Pareto front with no gaps. It also produces the most Pareto solutions but not in equal distribution over the Pareto front. Both the niche and distance methods perform reasonably well at spreading the population over the whole Pareto region. However the problem with these methods is that the selection procedure selects too many like individuals thus preventing sufficient diversity to enable large quantities of solutions to be generated during the crossover and mutation procedures. A modification to these selection procedures to allow for a more diverse level of selection could be implemented to improve search, although as the diversity is increased the outcome would tend towards the random Pareto injection method.

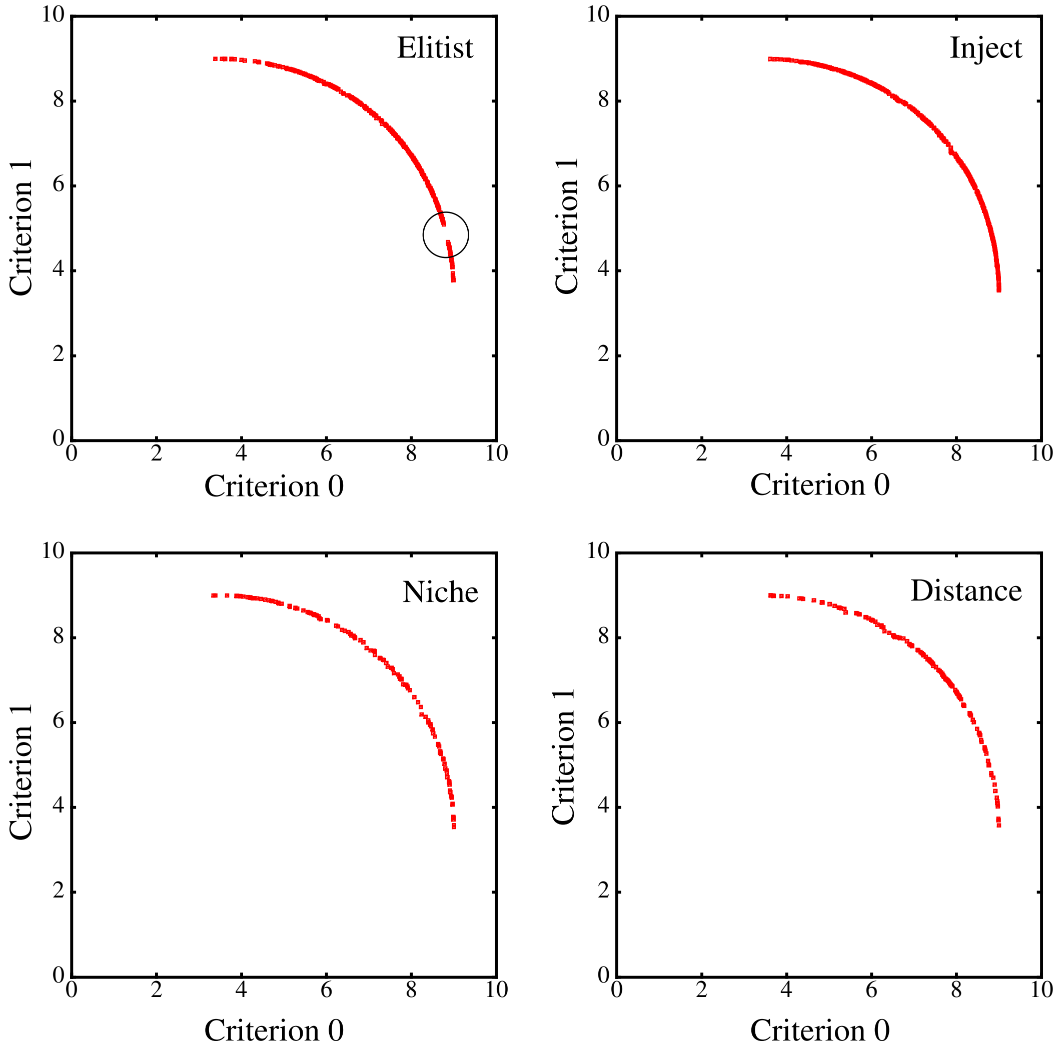


Figure 7.7 - Comparison of Pareto Solutions

7.4 Convergence

The aim of the divergence routines was to encourage the quick development of a broad spread of individuals to form the Pareto surface as quickly as possible. For example, if the random injection method is used for only 10 generations a reasonable idea of the form of the Pareto front (Figure 7.8) may be obtained for the two criteria problem of Chapter 4. From this point the decision maker can choose an area to concentrate further search. This requires convergence methods that can concentrate effort on the most interesting regions of the search space. Two methods are described in the following sections which can be used if the designer has a good idea of what type of solution or balance of criteria he seeks. These methods work in the same way as the divergence

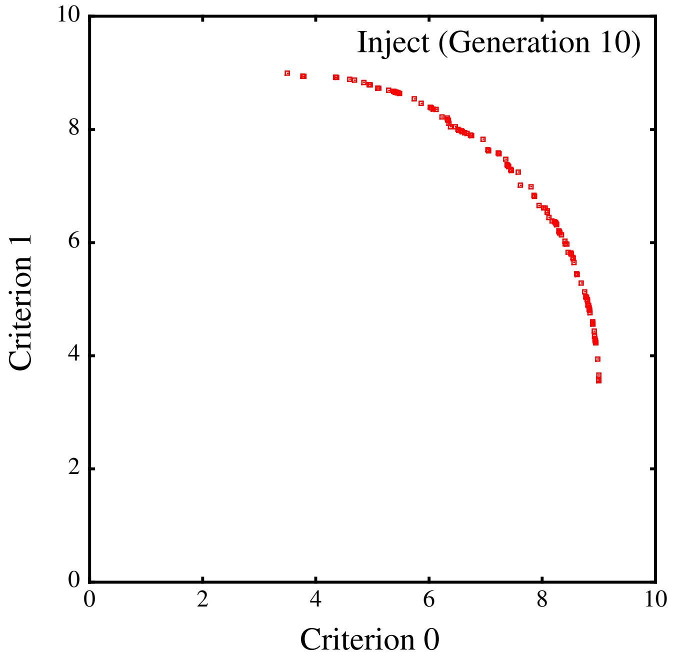


Figure 7.8 - Approximation of Pareto Set

methods by re-inserting individuals selected by a suitable metric. Again these were tested on the 2 criteria example over 50 generations of 500 individuals for 50 trials.

7.4.1 Target Values

If the decision maker has seen the decision surface or is reasonably clear on the properties he would like in a design he can usually specify the approximate criteria values he would like to achieve. This method allows the decision maker to specify these values as a target or goal.

The distance of each individual of the Pareto Population from the target position is calculated. This is then used to randomly select individuals which are in the closest 20% of the whole set. This distance is the Euclidian distance between the normalised value of the target point (T_i) and the normalised value of the current point (C_i). In the case of maximisation, normalisation is performed between the target value and the minimum value in the Pareto Population of that criteria. Minimisation is performed between the target value and the maximum value in the Pareto Population of that same criteria. In the tests the target value of criterion 0 is 9, while that of criterion 1 is 5.

7.4.2 Target Ratio

The target ratio works in a similar way to the value method. The decision maker decides on the approximate ratios of the criteria in the final solution for each successive pair of criteria. These ratios may be regarded as trade off values, representing the relative importance of the criteria. For example, for a 3 criteria problem two ratios are given between criteria 0 and 1 and criteria 1 and 2. The MCGA then tries to generate Pareto solutions with ratios near the ones specified. The selection is performed by measuring the square of the difference between the target ratio (T_r) and the current solution ratio (C_r) for each pair of criteria. This means that if there are n criteria, n-1 ratio distance values need to be calculated. The solution has a chance of being selected on any one of these ratio distance values. Obviously, if more ratios within a solution are below the 20% closeness value the greater the likelihood of selection. The target ratio for the two criteria case used here was set at 9:5.

7.4.3 Testing of Convergence Methods

The methods were tested in the same way as the divergence methods (Section 7.3.4). The graphs showing the Pareto solutions generated and their distributions in criteria space are shown in Figures 7.9 and 7.10. Both methods show their capabilities for concentrating on the specified regions of the search space. Few outlying points were investigated during the search. Those which did occur were discovered in the first few generations. On the Pareto plots the target points/ratios along with the 0.2 closeness lines are plotted. Many of the points found especially in the value method do not fall within this range. These points are created as a result of Pareto solutions mating with other Pareto solutions and normally selected solutions. The target ratio method gives a smaller number of points outside these boundaries. This method, because it uses ratio, does not generate an equal balance of points either side of the target ratio. This is why the +0.2 band appears smaller than the -0.2. The points range from a ratio of just over 6:5 to nearly 12:5 about the specified ratio of 9:5. This bound can be easily identified as the number of points decreases quite rapidly outside it.

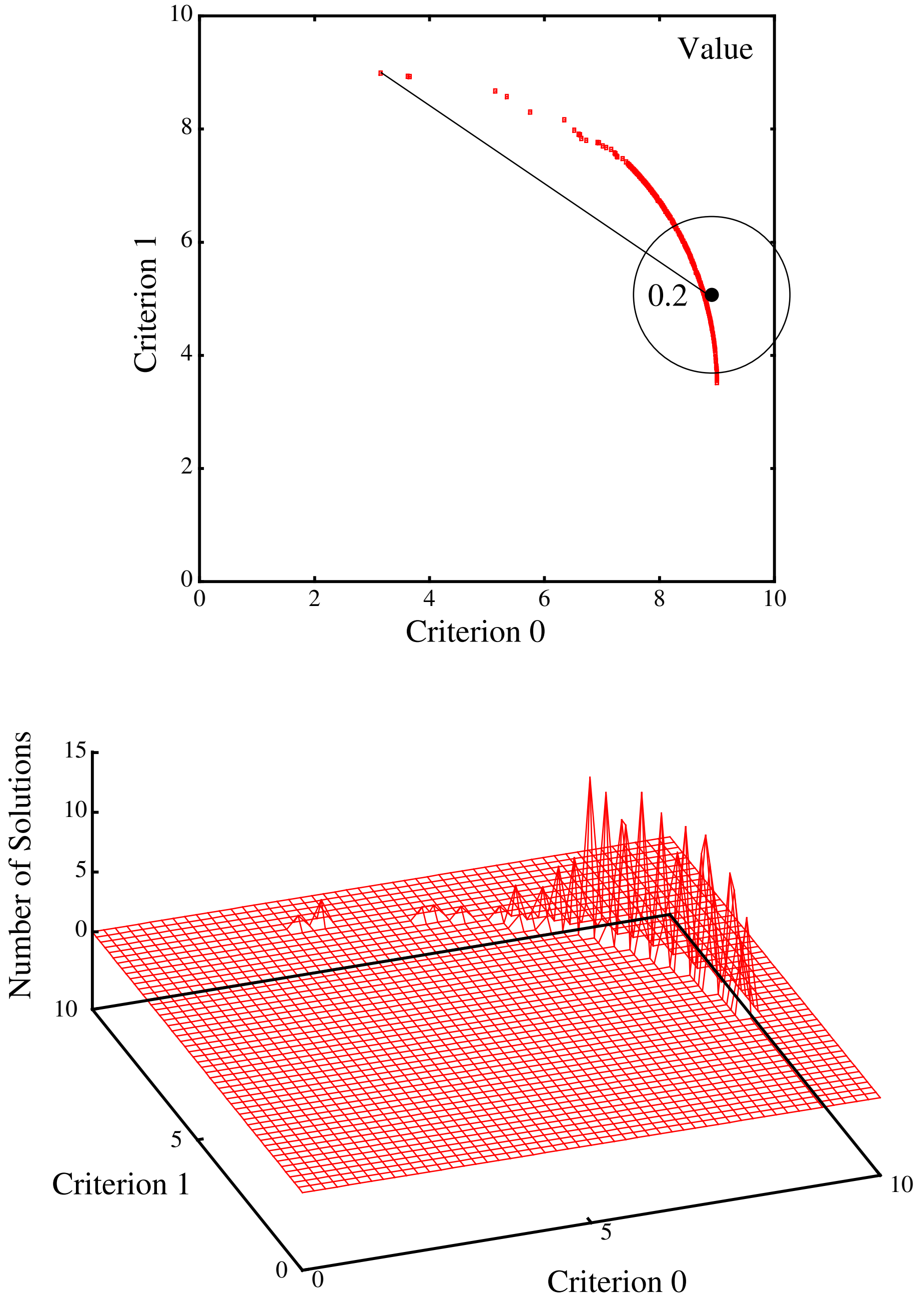


Figure 7.9 - Value Method - Pareto Set and Distribution

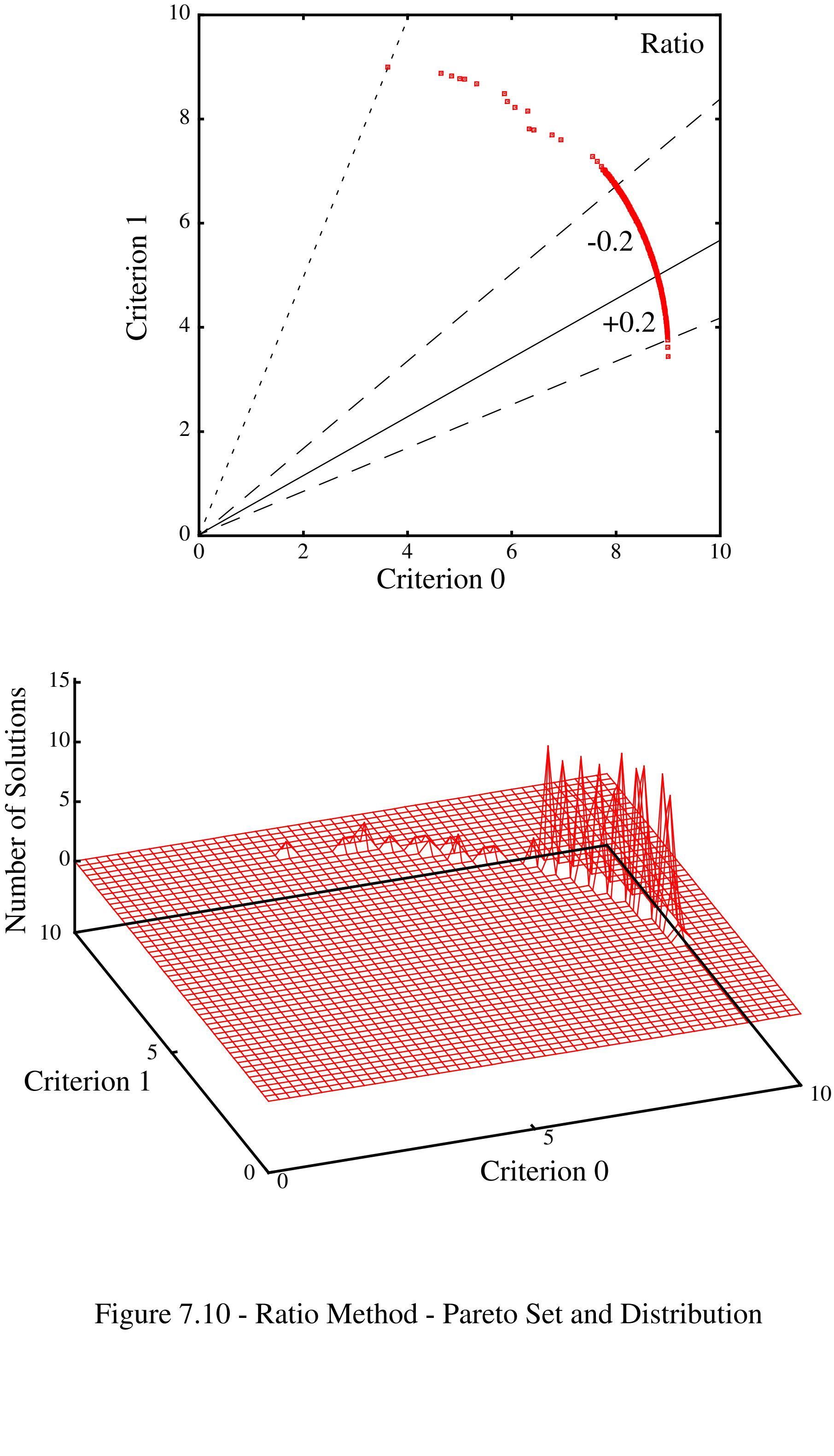


Figure 7.10 - Ratio Method - Pareto Set and Distribution

The accompanying distribution graphs in Figures 7.9 and 7.10 show the concentration of Pareto solutions predominantly clustered around the target area. Both methods give a reasonably even distribution over the area of interest, putting little effort into searching other areas of the search space. Both methods generated similar numbers of Pareto solutions over the 50 generations (Figure 7.11). The target value method was able to generate slightly more solutions than the ratio method probably due to the fact that this method produced more solutions outside the specified closeness of 0.2.

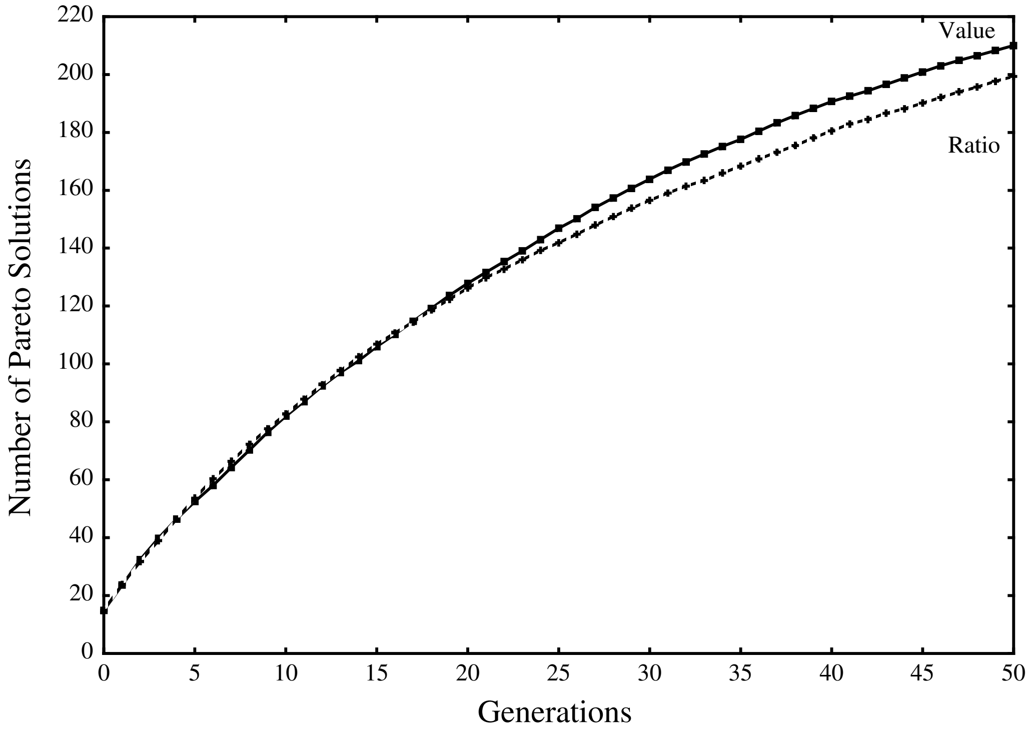


Figure 7.11 - Number of Pareto Solutions Generated

Although both methods are equally effective, of the two the target value method is more easily defined by the user. The user can specify his approximate goal and the MCGA will generate solutions as near to it as possible. The system will still operate if the goal is inside the feasible region. However a better performance is obtained when the goal is just outside the Pareto front in infeasible space, as this naturally represents aspirations.

7.5 Interactive: Neural Network Preference

The essential element in any decision making process is obviously the decision maker. In most of the examples in this thesis the decision maker/designer defines the problem model. The MCGA attempts to synthesise a range of solutions for that model and then the decision maker selects one. The decision maker may also use some form of tool to select from the solutions (see Chapter 9). Increasingly design tools are attempting to obtain preferences from the designer earlier in the design process. The designer in turn wants to investigate the available solution space before stating his preferences. In this way the designer can shape the design and discard any design directions which are inappropriate or impractical.

The aim here is to investigate a way of allowing the decision maker to interactively shape the MCGA search. This will lead to the MCGA concentrating its effort on those areas which are preferred by the decision maker. This is an important consideration in the case where evaluation times are high so that unnecessary computations are avoided.

As before the method utilises the Pareto Population and encourages the growth of preferred individuals by placing them back into the population. In order to select these individuals a preference model has to be generated, modified and maintained. For this purpose a neural network is used. This approach was developed entirely independently from recent work by Grierson (GRIERSON 96) and Bull (BULL 97) who use Neural Networks to generate fitness functions based on designer information.

7.5.1 Neural Networks

Neural networks are a computational technique which mimics the computational abilities of biological systems (ZHANG 95). A neural network has three key characteristics:

i) It consists of a number of processing elements (neurons).
ii) Each neuron is connected to other elements through weighted links.
iii) The functionality is determined by modifying these weights during a learning phase.

The first neural model, The Perceptron (Figure 7.12), was developed by Rosenblatt (ROSENBLATT 58) nearly forty years ago. The Perceptron takes a weighted sum of inputs and gives an output of -1 or +1, depending on whether the value of the weighted sum has a value larger than a given threshold. Learning is performed by adjusting the weights. Since the advent of the Perceptron over 50 different neural network models have been developed. Neural networks have in the past 10 years undergone a renaissance after a subdued period in the seventies following investigations carried out by Minsky (MINSKY 69). Minsky demonstrated mathematically that a Perceptron was not capable of representing an exclusive OR logic function. This caused many

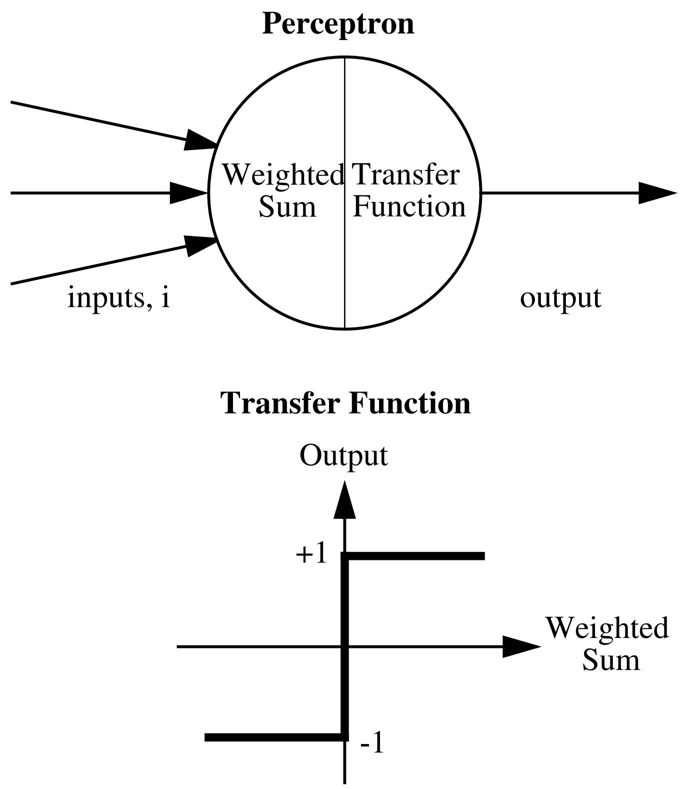


Figure 7.12 - The Perceptron

practitioners to believe that neural networks could not progress further, hence research activity declined. This trend was dramatically changed by Hopfield (HOPFIELD 84) who reawakened interest by introducing some new concepts such as the sigmoidal transfer function which replaced the threshold function of the Perceptron. Later the Parallel Distributed Processing (PDP) Research Group published two volumes (RUMELHART 86) and later a third (McCLELLAND 88) which has had an explosive effect on neural network research. One of the most significant and widely quoted contributions is the derivation of the back propagation algorithm for multilayer

Perceptrons, published in Nature (RUMELHART 86b). It was later uncovered that this work was first investigated by Werbos (WERBOS 74) in his PhD thesis which remained unknown for over a decade. It is the back propagation neural network that is the most widely known and this is the architecture which is applied here to capture preferences.

7.5.2 Back Propagation Neural Networks

The back-propagation neural network is a multilayer Perceptron network with a different transfer function within the neurons. The network is classified as supervised in that it is used in a two stage process. The first stage of this process is learning and the second stage is prediction. During the learning process the neural network is 'taught' to recognise a given set of input and output conditions. The learning rule used, called back-propagation, is more robust and capable than the one used with the Perceptron. Additionally the neurons use an enhanced transfer function, a sigmoid, and they are usually arranged into three or more layers (Figure 7.13). Only feed forward connections are allowed and connections must be between adjacent layers.

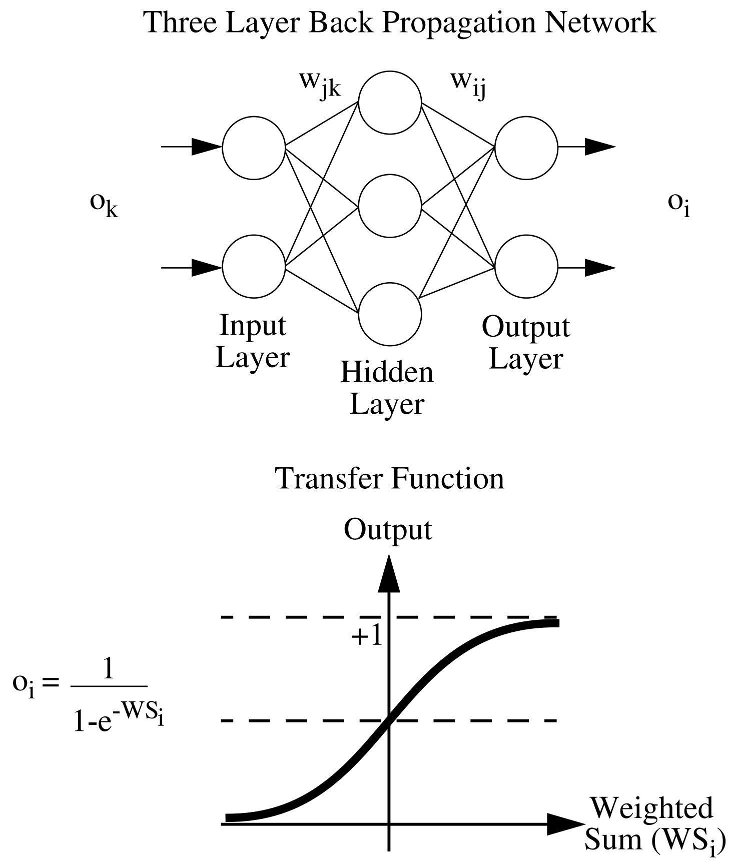


Figure 7.13 - The Back Propagation Neural Network

The back propagation neural network is taught to create a mapping between input and output patterns. During the training process the input-output pairs are known. However, instead of using a delta rule (difference between input and output) a squared error rule is used. The error is given by:

$$E = \frac{1}{2}\sum_{i=1}^{n}[t_i - o_i]^2 \qquad 7.1$$

Where t_i is the target value and o_i is the actual value at output i for all n neurons in the output layer. The aim of back propagation is to perform weight adjustments between layers to reduce E to a minimum.

$$\begin{aligned} w_{ij(new)} &= w_{ij(old)} + \Delta w_{ij} \\ w_{jk(new)} &= w_{jk(old)} + \Delta w_{jk} \end{aligned} \qquad 7.2$$

The change of weights is given by:

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} \qquad 7.3$$

$$\Delta w_{jk} = -\eta \frac{\delta E}{\delta w_{jk}} \qquad 7.4$$

Where η is referred to as the learning rate. This basically means that the changes in weights are in the direction of decreasing error. It is therefore necessary to find $\frac{\delta E}{\delta w_{ij}}$ and $\frac{\delta E}{\delta w_{jk}}$, the rate of change of error relative to change in weights.

$\frac{\delta E}{\delta w_{ij}}$ is difficult to compute, however the value can be calculated by using chaining rules. This is shown in Figure 7.14 for the relationship between the output and hidden layer.

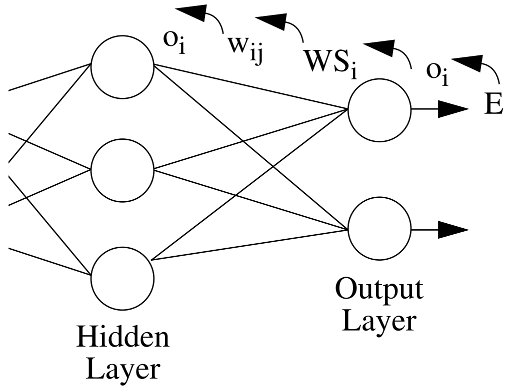


Figure 7.14 - Chain Rules

Chaining gives:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta o_i} \cdot \frac{\delta o_i}{\delta WS_i} \cdot \frac{\delta WS_i}{\delta w_{ij}} \qquad 7.5$$

$$\frac{\delta E}{\delta o_i} = -(t_i - o_i) \qquad \frac{\delta o_i}{\delta WS_i} = o_i(1 - o_i) \qquad \frac{\delta WS_i}{\delta w_{ij}} = o_j \qquad 7.6$$

Therefore:

$$\frac{\delta E}{\delta w_{ij}} = -(t_i - o_i)o_i(1 - o_i)o_j \qquad 7.7$$

Moving backwards through the network the relationship $\frac{\delta E}{\delta w_{jk}}$ can be derived and is given by:

$$\frac{\delta E}{\delta w_{jk}} = \left(\sum_{i=1}^{n} \frac{\delta E}{\delta w_{ij}}\right)(o_j(1 - o_j)o_k) \qquad 7.8$$

The summation occurs because the layer j has more than 1 output hence propagating backwards all of them must be taken into account when finding $\frac{\delta E}{\delta w_{jk}}$.

The adjustment of weights via back propagation is continued until the squared error term is reduced below a certain threshold over the complete teaching data set. When this has been achieved the trained network can be used as a predictor giving an output pattern based on given unseen inputs.

7.5.3 Integration of MCGA and Neural Network

The aim of this technique is to use the neural network to generate a preference surface based on preference data collected from the user. This surface will then be used to select the preferred members of the Pareto Population for re-introduction into the normal population. This will hopefully promote further investigation into the preferred areas of the search space in turn generating more individuals. The process is defined in Figure 7.15 in the form of a flow chart.

The preferencing process will take place at regular intervals through the MCGA process with the first preference occurring after ten generations. Prior to this the

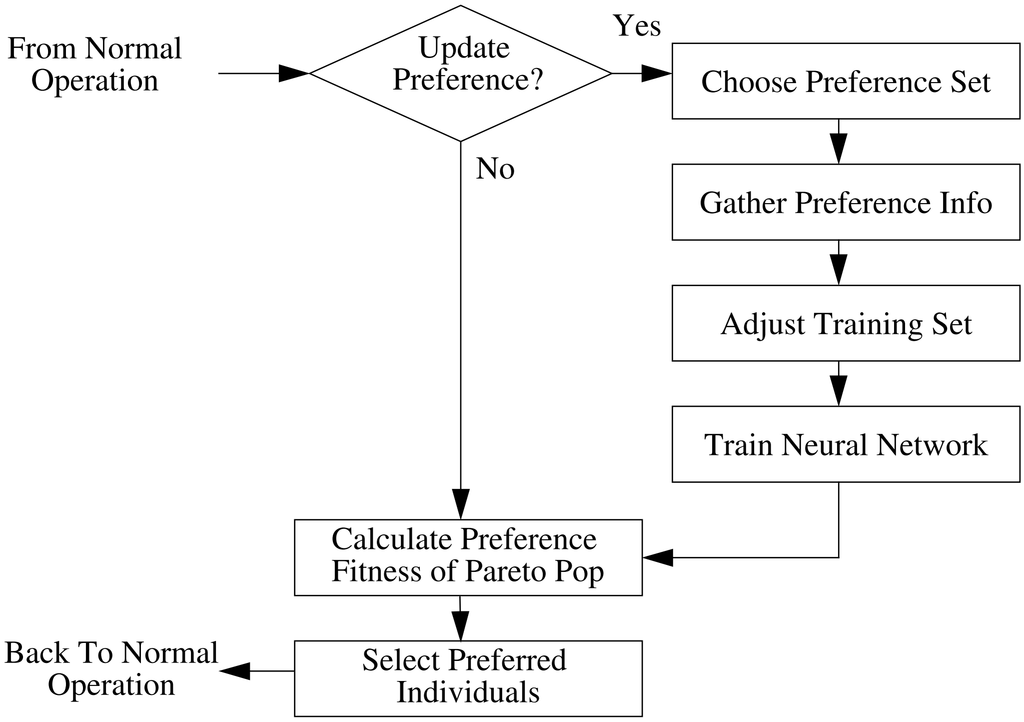


Figure 7.15 - Neural Network Preferencing

standard elitist/roulette procedure is used, then the neural preference/roulette procedure is employed. If preferencing is to take place the preference set has to be chosen first. This preference set is a selection of ten individuals from the normal population. In the first set chosen this is done randomly. After this the system tries to select a broad cross section of individuals in terms of their neural preference score. The aim is to generate a more general picture of the surface. Two control individuals are also inserted into this set. These have the largest and smallest preference values for the current training set. The new set is displayed to the designer/decision maker in a random order. The system then gathers preference information by asking for a score between 0 and 1 for each of the preference set. When complete the new scores for the control individuals are used to adjust the scores from the previous preference sets. Ten previous sets are held giving a total training set size of 100 points when full. The adjusted training set is then used to train the neural network using the back propagation method. The newly formed preference surface can then be used to score the Pareto individuals between 0 and 1. This score is then used to select a set of individuals from the Pareto Population which are re-inserted into the population in order to promote search in the preferred regions of the search space.

7.5.4 Neural Network Testing

The neural network was implemented and integrated into the MCGA. The user initially specifies two variables similar to those used in the convergence methods. The proportion of the population that is selected through re-insertion and a threshold (closeness) at which individuals will be accepted, based on their preference score. A population of 500 was run over 50 generations on the two criteria problem from Chapter 4. Proportion and Closeness are set to 0.5 and 0.2 respectively.

The neural network layout used was a 3 layer system with 2 input, 5 hidden and 1 output neurons. The diagram of this is shown below in Figure 7.16. The inputs take in

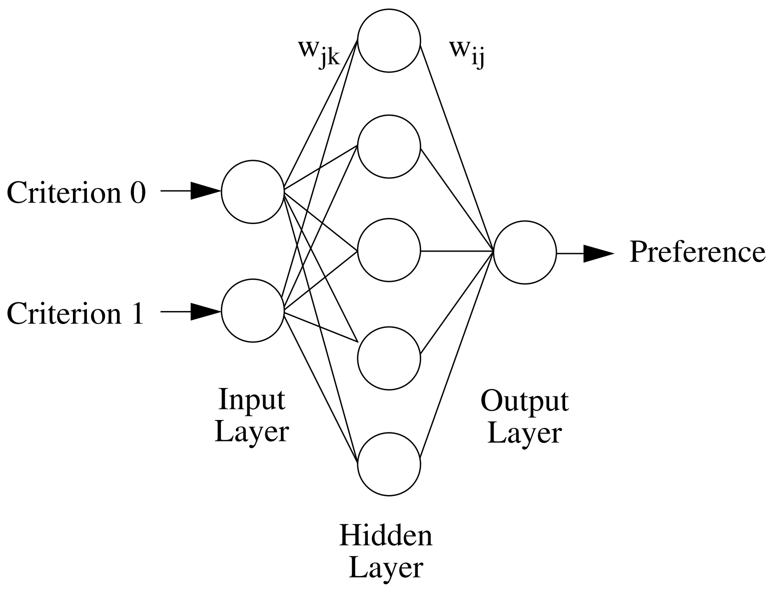


Figure 7.16 - Implemented BP Neural Network

the two criteria values. They are processed through the hidden layer via w_{jk} and through w_{ij} to the output layer. The number in the hidden layer was chosen as 5 arbitrarily and could be increased or decreased if required.

After 10 generations of search the decision maker is presented with a table of randomly chosen designs an example of which is shown in Figure 7.17. The table shows each design with its criteria values.

```
*** Interactive Preference ***

Design 0 :        2.52965     7.46026

Design 1 :        7.07147     5.87281

Design 2 :       0.478745     5.34876

Design 3 :        7.48098      7.0025

Design 4 :        3.81956     7.80841

Design 5 :         4.0185    0.469328

Design 6 :          0.619     6.84764

Design 7 :        3.10982     8.14307

Design 8 :        8.42858     5.96905

Design 9 :       0.465795     5.86715

Please Score the above Designs on scale of 0-1 :
(Base your judgements on this set of designs only, not any previous ones)
```

Figure 7.17 - List of Designs for Preference Scoring

The decision maker is asked to score the designs relative to each other and not to any previous sets. He can also score two designs the same if he so wishes. In this test it was decided that ideally, designs with a ratio criterion 0:criterion 1 of around 9:5, and both values as large as possible, were preferred. In practice the judgement criteria would be much more involved and complex and the decision maker would have to base his judgements on heuristics and design experience. Here the design set was scored as shown in Figure 7.18.

```
Score for Design 0 :   0.0

Score for Design 1 :   0.8

Score for Design 2 :   0.0

Score for Design 3 :   0.7

Score for Design 4 :   0.1

Score for Design 5 :   0.1

Score for Design 6 :   0.0

Score for Design 7 :   0.1

Score for Design 8 :   1.0

Score for Design 9 :   0.0

Thank you - returning to processing
```

Figure 7.18 - Decision Maker Scored Designs

From this preference set design 8 is clearly seen as the best, followed by design 1 and then design 3. Most of the other designs were poor and given low scores. When the scores have been obtained, the code enters the neural network learning phase. The generated preference surface can then be used to score the Pareto Population. This score overwrites the fitness score in each string. The fitness is then used to re-insert the most preferred individuals into the population with the aim of promoting more individuals in this area. The population is then filled up using roulette wheel selection on the current members of the population. The search then continues for another 10 generations as normal.

Following this another preference set is chosen, this time with the aim of presenting a broader set of individuals based on their preferred values. This set is shown in Figure 7.19. It can be seen that two designs, design 6 (8.42858, 5.96905) and design 3 (0.478745, 5.34876) appeared in the previous preference set (Figure 7.17) as designs 8 and 2 respectively. These are the two control individuals used to scale the previous sets. The decision maker again gives his preference scores between 0 and 1.

```
*** Interactive Preference ***

Design 0 :        8.34814     6.14678

Design 1 :       0.564551     1.40331

Design 2 :        4.39327     8.86197

Design 3 :       0.478745     5.34876

Design 4 :        7.71528     7.08282

Design 5 :        8.68926     5.28533

Design 6 :        8.42858     5.96905

Design 7 :        8.39937     2.91864

Design 8 :         8.3139     2.16067

Design 9 :         8.8383     4.80271

Please Score the above Designs on scale of 0-1 :
(Base your judgements on this set of designs only not any previous ones)
```

Figure 7.19 - List of Designs for Preference Scoring

This decision maker scoring process was repeated 5 times and the Pareto Solutions are displayed to the user with their preference scores, generated from the final round of preference scoring. Figure 7.20 shows the distribution of Pareto solutions after 50 generations. It is clear that the bulk of the discovered solutions are towards the preferred values of criteria 0 and 1. There are still some other solutions dotted around due to the fact that the MCGA does not discard any Pareto solutions. These were probably generated early in the search process. As the preferences of the designer become apparent the search shifts most of its effort into generating new preferred solutions. This can be seen in Figure 7.21 which shows the individuals of the normal population plotted in criteria space. The initial population shows a broad spread as expected. At generation 20, ten generations after the first preference session the population has redirected its effort towards the preferred region. After several more preference rounds the population is dictated by expressed preferences and the population clusters close to the 9,5 point. By changing the value of closeness the efforts of the MCGA can be focused even more precisely.

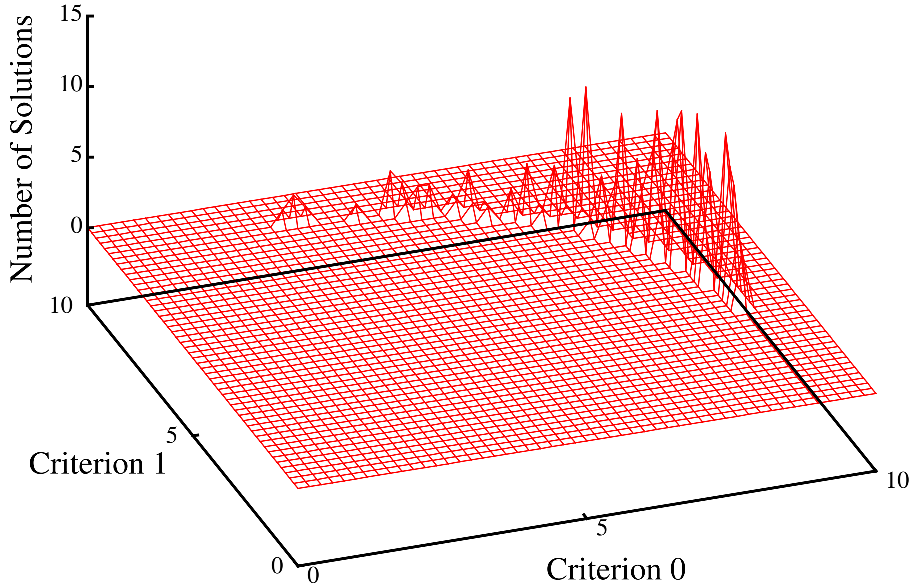


Figure 7.20 - Final Distribution of Pareto Population

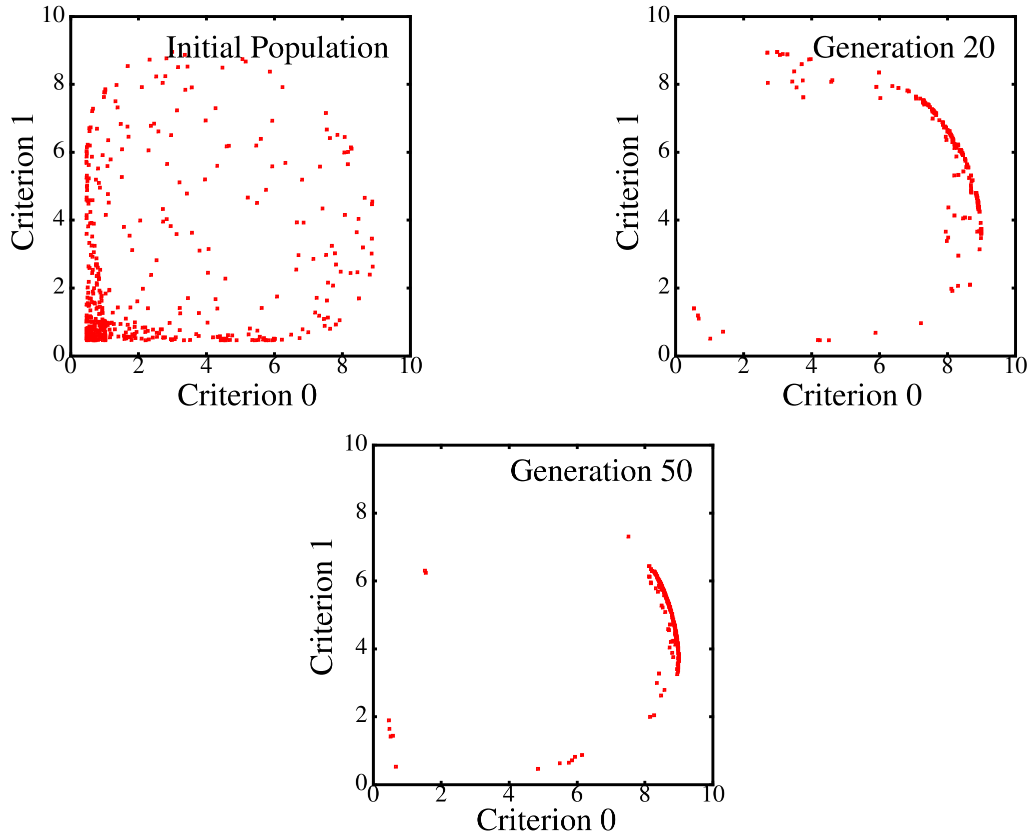


Figure 7.21 - Migration of the Normal Population

An interesting result to look at is the preference surface. Three of the preference surfaces for this run are shown in Figure 7.22. The initial surface is quite smooth as only a few points are specified. As more points are obtained the preference surface becomes more defined and extra features begin to emerge. In the final plot the surface is strongly structured with the preference being greatest in the correct area for this example. Away from this area the surface quickly falls away thus reducing the likelihood of selection.

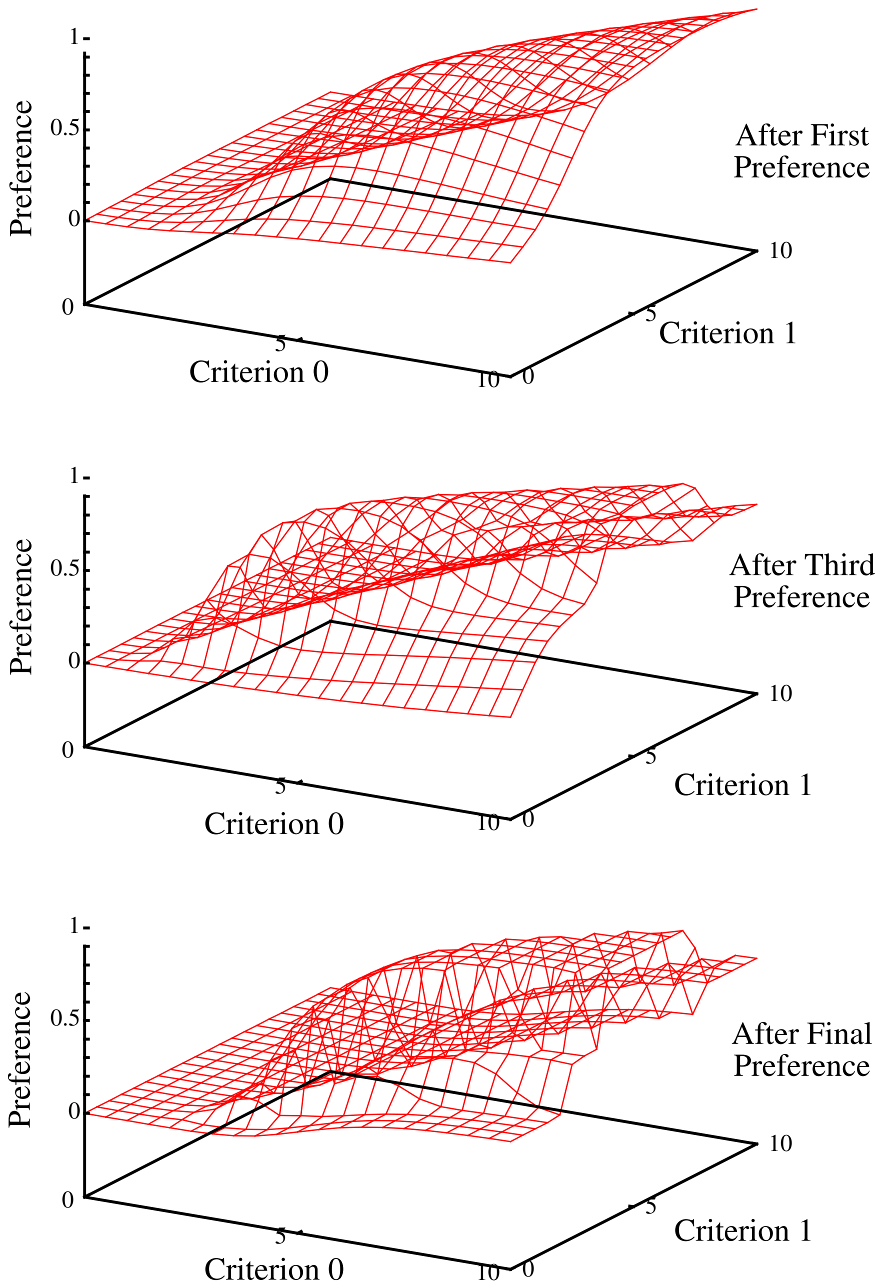


Figure 7.22 - The Developing Preference Surface

Although the technique performs well when handled correctly there are several problems which were highlighted during the tests. Firstly, the neural network, because it is started from random points sometimes gets stuck in local minima during the back propagation procedure. This leads to preference surfaces which do not truly reflect the preferences of the decision maker. This was overcome by allowing the neural network to be re-run if the mean squared error was too great. This can be done several times until a low enough value is found.

The biggest problem is checking for consistency. There is no consistency checking within the neural network. This means that if the same or similar points are given greatly different values of preference this can cause the MCGA to struggle when trying to fit surfaces. In such cases effects like the one mentioned above can occur and poor surfaces are generated. Some form of pre-processing is therefore required before back-propagation to increase the robustness of the curve fitting. A method of identifying and removing inconsistent scores is required. If necessary the decision maker could be questioned about inconsistencies and asked to reconcile them.

7.6 Chapter Summary

This chapter introduced several methods for controlling the evolution of MCGA search. These methods allow the decision maker to tailor the type of search performed to his own requirements.

The first three methods described try to promote an increased level of diversity within the population. This can be used as an enhancement to normal search or for performing quick investigation into the approximate shape of the Pareto Surface. Of the methods tested the random injection method proved to be the most effective.

The two convergence methods described are used to concentrate search effort on the regions of the Pareto surface of greatest interest to the decision maker. The decision maker, with some knowledge of the form of the Pareto surface, is allowed to specify a target value or target ratio that indicates the type of solutions desired. The MCGA will then put maximum effort into finding solutions with the specified properties.

Finally, a complex interactive process is described that uses a back-propagation neural network to collect preference information from the user. The system asks the decision maker to score the quality of solutions during search. These comments are then formed into a preference surface which can be used to direct search into regions of the search space which are more suited to the users requirements.

Further work is still required to increase the stability of the method particularly when dealing with inconsistent judgements.

In the next chapter the MCGA, in association with the injection divergence and the target value convergence methods, is applied to an extremely large combinatorial problem, namely that of loading a containership with containers with the aim of optimising several criteria simultaneously.

8 MCGA for Containership Loading

8.1 Introduction

In this chapter the MCGA will be applied to the problem of containership loading. Containership loading has commonly been tackled by skilled personnel drawing on many years of handling experience. More recently, attempts have been made to aid the job of the stevedore by storing the details of containers loaded on board and calculating the current ship status. However, most attempts at automation have been restricted to purely heuristic methods producing a feasible, and not necessarily optimal, configuration under the supervision of a human planner. This chapter applies the MCGA with heuristic operators, to this large scale combinatorial problem.

8.2 Containership Loading

During the past 30 years the container handling revolution has increased the efficiency of worldwide trade. The container is a universal transportation method capable of being moved by sea, road or rail with relative ease. With the increasing level of world trade and the corresponding increase in size of containerships (up to around 8000 containers by the year 2000) one of the main areas for cost and efficiency gains now occurs in port whilst performing container loading, unloading and arrangement. The shorter the turnaround time the more money is saved. Thus any methods to improve the process would be significant.

An early attempt at such a system in the late 60's and early 70's (WEBSTER 69, WEBSTER 70) failed to be attractive enough to gain wide appeal but the importance of these attempts were acknowledged by the Maritime Administration of the USA. Since then the falling costs and increasing speeds of computer hardware have seen several papers introducing computerization of container loading with varying degrees of success. The first major breakthrough came with the adoption of the Computer-Aided Preplanning System (CAPS) by American President Lines(APL) in 1981. This system, explained by Shields (SHIELDS 84), uses a combination of simulation using Monte-Carlo techniques and human interaction to provide possible vessel plans. Since then further investigations have been carried out (SHA 85, RATCLIFFE 87, SEN 88, SAGINAW 89) using expert systems and rule based techniques to aid the stevedore in finding suitable configurations. Several of the papers (RATCLIFFE 87, SEN 88) also have automated on-deck lashing algorithms built in to further aid the planner in his task. The aim of this chapter is to apply the MCGA to the containership problem in an attempt to produce an integrated optimisation oriented system (TODD 97).

8.3 Containership Terminology

Before going into more detail it would be helpful to outline the terminology used in this chapter. Figure 8.1 shows a plan of a containership used in this part of the work, with some common terminology shown on it.

The vessel holds 1958 containers in total, 753 on deck, 1205 in holds. The containers are held in a number of bays along the length of the ship, 23 in total. Bays are numbered from the bow and are usually grouped in pairs e.g. 0-1, 2-3, 4-5 etc. At deck level hatches are placed over pairs of bays. The hatches, numbered from the bow, are usually split; In this case into three sections covering a certain range of cells. This is done so that the whole hatch does not have to be removed to get to containers in the hold. For a hatch to be removed all containers placed on it also have to be removed.

The containers in each bay are arranged in tiers, 12 in this ship. Each of the tiers are made up of a number of cells, numbered across the ship from port to starboard. The position of a container within the ship is entirely specified by a bay-tier-cell value. In order for a lower tier container to be removed all containers above it with the same cell number must be lifted off. The number of cells varies along the ship depending on the hull shape. At the midships position the number of cells is greater than at the bow or stern. This effect can be seen on the bay plans shown in Appendix E where the bay shape follows the ships lines.

The layout of bays, cells and tiers differs greatly from ship to ship. The number of containers, the position of the accommodation block and engine rooms, and hull shape will all define the final plans. The system was designed to accommodate variation by providing routines which were not ship dependent.

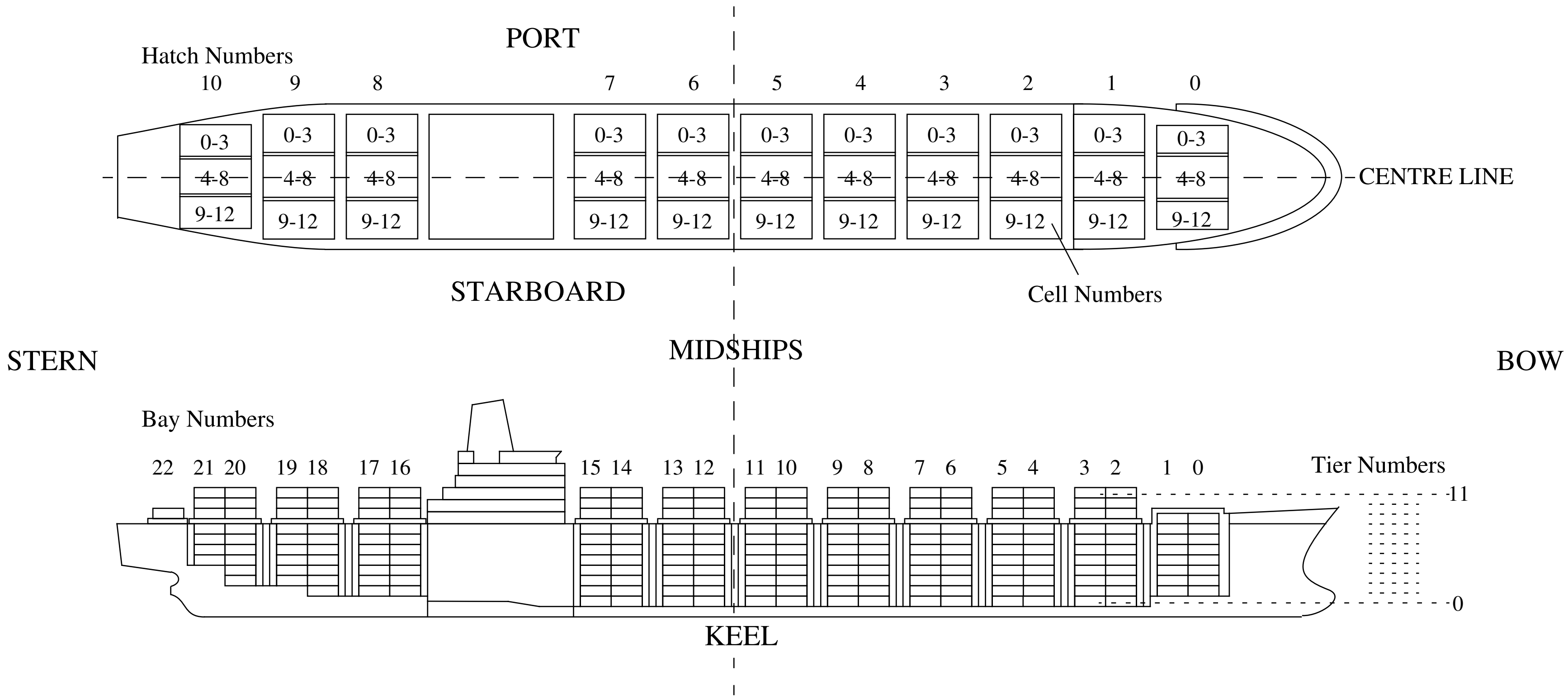


Figure 8.1 - Containership Terminology

8.4 The Containership Loading Problem

The preplanning of containership layouts is a large scale combinatorial problem similar to that of the Travelling Salesman Problem in Chapter 3. A fixed number of individual containers are to be placed into a fixed number of available locations. However, unlike the TSP problem the number of containers is usually less than the number of possible locations and the relationships between entities is more complex. This complexity precludes the use of known TSP operators. The scale of the problem is enormous. For a 2000TEU (Twenty-foot Equivalent Unit) containership, such as the one tackled here, the number of possible stowage configurations is approximately 3.3×10^{5735} (DILLINGHAM 86) and as the size of containerships increases the reliance on manual systems becomes increasingly difficult and more costly (SAGINAW 89). The problem is also a multi-criteria one, in that the stevedore will have to juggle with several objectives in order to find a suitable solution. These objectives are briefly examined below, those marked with an asterisk are further explained in Appendix E:

Overstows - Commonly a vessel will call at several ports on a circular journey. At each port containers for that port are unloaded, and this may involve removing containers bound for later ports which have been placed on top of others bound for the current port. The number of these overstows should be minimised in order to save port time and hence costs. This is shown in Figure 8.2. All of the black containers need to be

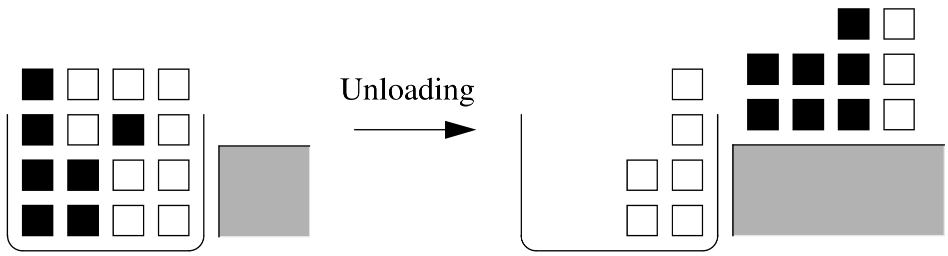


Figure 8.2 - Overstows

unloaded but in order to do this three white ones are also removed. These unloaded white containers are the overstows.

Container Requirements - Containers may require to be in a specific hold on the ship. In the case of refrigerated containers location must be close to power supply points.

Moments:

i) Longitudinal* - The trim of the vessel, the difference in the height of the waterline between bow and stern, must be minimised. This will sometimes give better ship performance by reducing the amount of power needed to drive the vessel through water. It also ensures the propeller is properly immersed*. However, an even trim is helpful for stowing containers in holds within vertical cell guides.

ii) Transverse*- The heel of the vessel, the tilt towards port or starboard, must be minimised.

iii) Vertical* - This must also be minimised below a certain value which is determined by intact and damage stability considerations.

iv) Bending - The structural stresses along the ship must be minimised in compliance with given specifications. This effectively means a generally uniform weight distribution.

Container Groupings - As well as grouping containers for the same port it may also be useful to group containers with some common characteristics. Conversely, containers which are classified as hazardous usually have to be located in compliance with specific stowage restrictions such as minimum separation requirements.

Deck Strength - The total weight of containers on a given deck or support structure must not exceed a specified maximum value.

Draft Limitations - Certain ports on route may have limits on the maximum allowed draft (depth of vessel below water). This essentially means that vessels may not be allowed to load fully.

Ballast* - The amount of ballast must be reduced, as this affects both the draft and performance of the ship. Any ballast carried is additional non-productive mass that must be propelled through the water.

Crane Limitations - Ports may also have restrictions on the reach and lifting capacity of cranes. In this case containers for certain ports may have to be placed higher up in the vessel or heavy containers may have to be placed nearer the sides. This in turn can create problems of stability.

Shift Costs - It may be more expensive to move containers in certain ports. This means that in ports with low associated costs containers may be reorganised to reduce overstows in expensive ports.

Lashing Strength - There are limits on the strength of container lashings, hence containers on deck must not apply excessive loading to the lashings.

8.5 Exploratory Test Problem

Initially a small scale problem was set up to examine the feasibility of tackling the full size problem. It was also used to test the functionality of the unloading code. This preliminary investigation used a simplified layout of 132 containers shown in Figure 8.3. The containers were numbered and given a destination either A, B or C and no weights were specified. The code was run using the MCGA for a two criteria case. The aim of criterion 0 was to minimise unloads and criterion 1 was to maximise a proximity score. The string was encoded bay by bay. Each of these bay sections was then divided into tiers from the keel up and then the position of individual container numbers denotes which cells, from port to starboard, containers are placed in. This is shown with a simple example in Figure 8.4.

Unloads are calculated for the ship over several unloads at ports A, B and C. The ship enters port A and all containers unloaded including overstows are counted. The overstows are replaced back into their original positions with port A containers replaced by empty spaces. The ship is then unloaded in port B in the same way and again in port C. Criteria 0 is defined as the total number of unloads counted, the minimum value being the total number of containers overall. The proximity score is defined as the number of like containers adjacent to a given container. This number was summed for every container on the ship. The maximum value being calculated as 600 over the whole ship if it holds all identical containers.

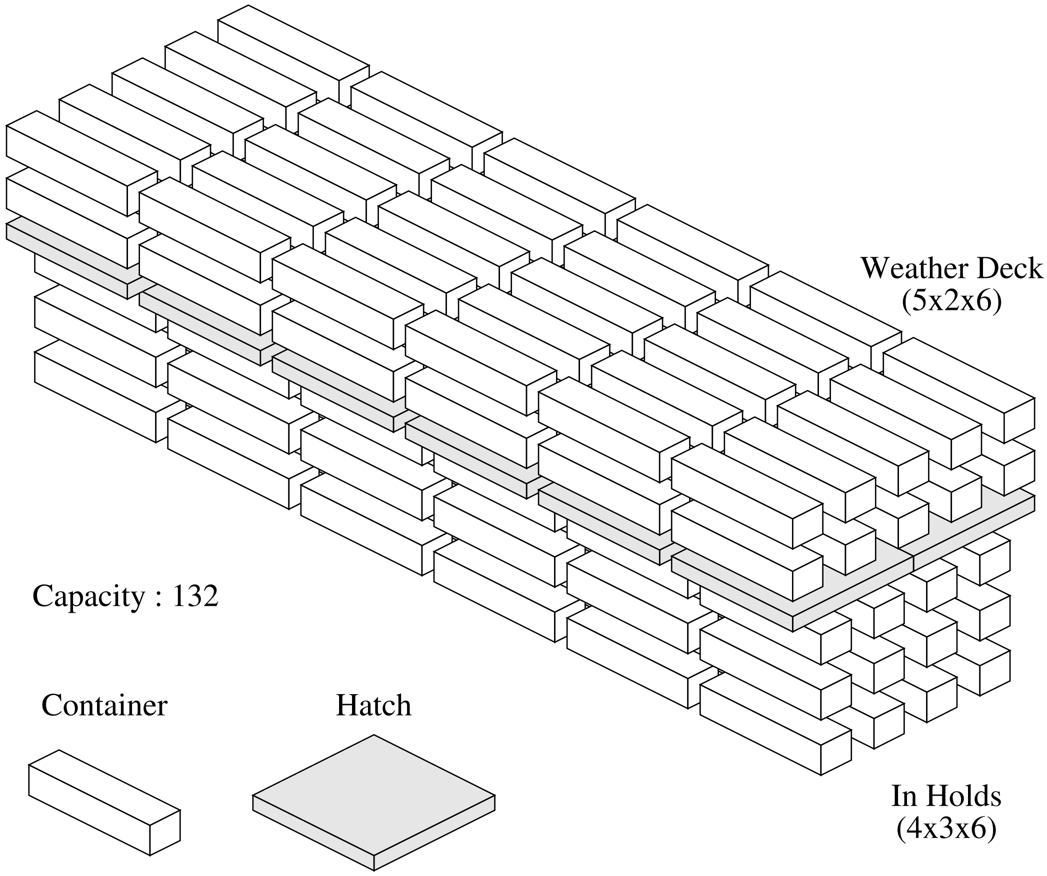


Figure 8.3 - Test Problem

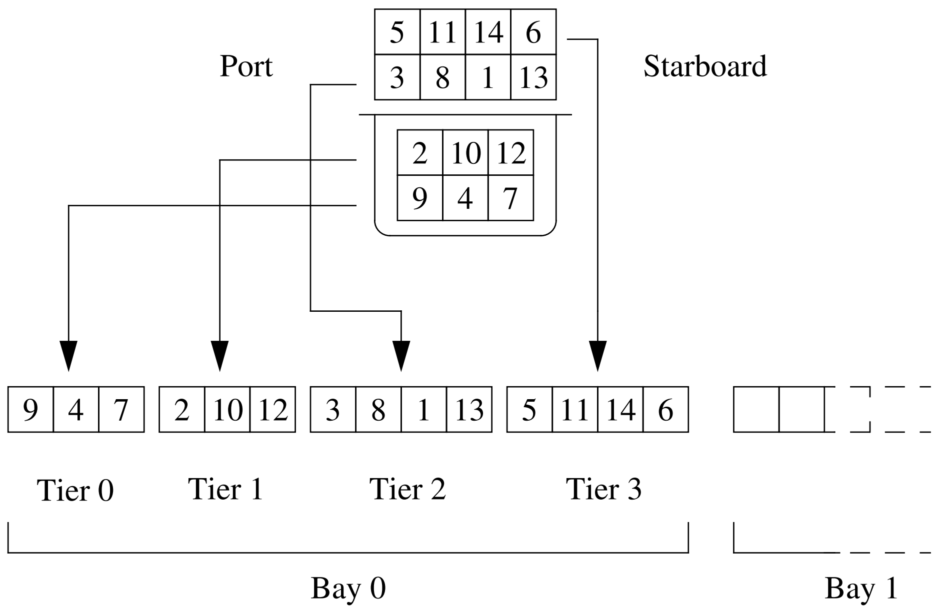


Figure 8.4 - String Encoding (Simplified)

The code was run with a population size of 100 over 500 generations. The unloading and proximity routines were checked and found to be functioning correctly. The unloading was tested by looking at the step by step container unloading plans produced by the code, and checking the right containers were being removed. The average curves for criterion 0 - unloads and criterion 1 - proximity are shown in Figure 8.5.

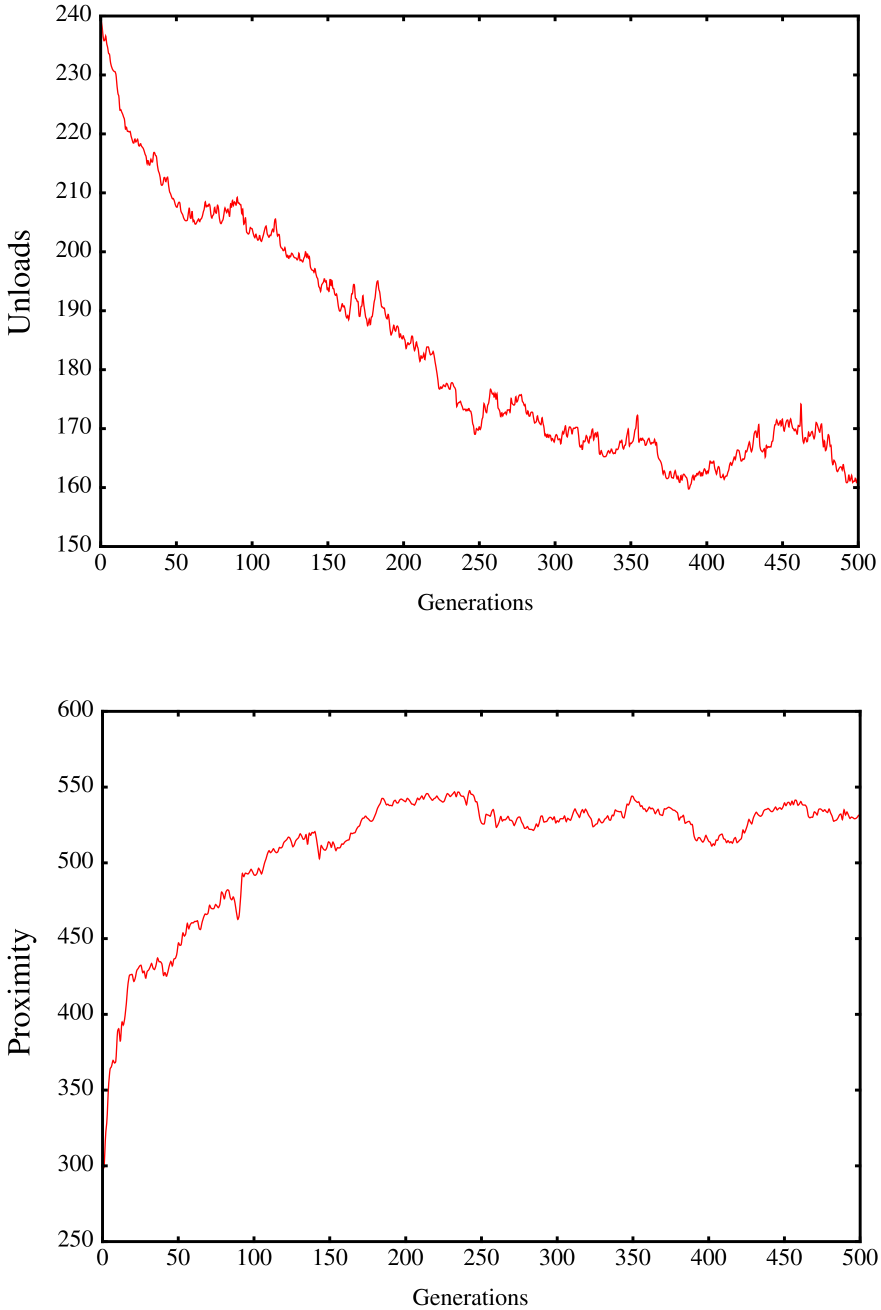


Figure 8.5 - Criteria Average of Pareto Population

From the graphs it can be seen that the MCGA is able to optimise most of the way towards the goals. Additional processing would be needed to ‘clean-up’ the layouts. The size of this problem is small compared to a full size containership. The time taken to optimise a 2000 container vessel would be far greater and also require a greater population size. As shown before in Chapter 3, local search and heuristics can aid in the search process. In order to tackle a larger problem such techniques would need to be employed in order to create reasonable designs.

8.6 Full Size Containership with Validation

For the full size containership several more criteria were added. Some of the objectives were combined to form a single criteria. The ship carries containers bound for the next five ports (A-E). The system is designed such that it is used at each port en-route using updated data. The current configuration (entering port A) is taken and using information on the containers which are to be loaded for forthcoming ports and data on the ship, such as centres of gravity, it generates new configurations. These configurations are evaluated on four criteria. These criteria are:

1.Unloads - The number of unloads is the sum of the number of containers, including overstows, which have to be removed in each of the ports (A-E) with containers from earlier ports removed. This value is minimised to reduce time and effort in port.

2.Proximity - This looks at each container and evaluates its location based on the containers grouped around it. In addition unlike containers are given minus scores. For example a container bound for port A surrounded by port B containers would be given a negative proximity score. This score is summed over the whole ship and maximised over a range of candidate configurations. This function could also be used to separate hazardous containers or place refrigerated containers in the proximity of power supplies.

3.Transverse Centre of Gravity - The transverse moments of each container about the ship centre line are summed over the whole ship and the magnitude is minimised ensuring the ship is as upright as possible.

4.Vertical Centre of Gravity - The vertical moments about the keel are summed and minimised to ensure maximum stability. This usually leads to the heavier containers being moved to the bottom hence removing them from the deck

The MCGA for Containership Loading is constructed as shown Figure 8.6. The containership model holds information on ship layout, horizontal, vertical and longitudinal container centres for ship moments, data on masses and destinations of containers and three evaluation routines for calculating the proximity, performing unloading and calculating moments. The MCGA is implemented as in Chapter 4 using Pareto elitist selection. Additionally, extra runs were carried out using the random injection and target value methods described in Chapter 7.

The model was checked systematically by examining each evaluation mechanism individually on a fixed container layout. A procedure was also incorporated into the code which checked that no duplicate containers had been introduced or containers lost from the string during processing.

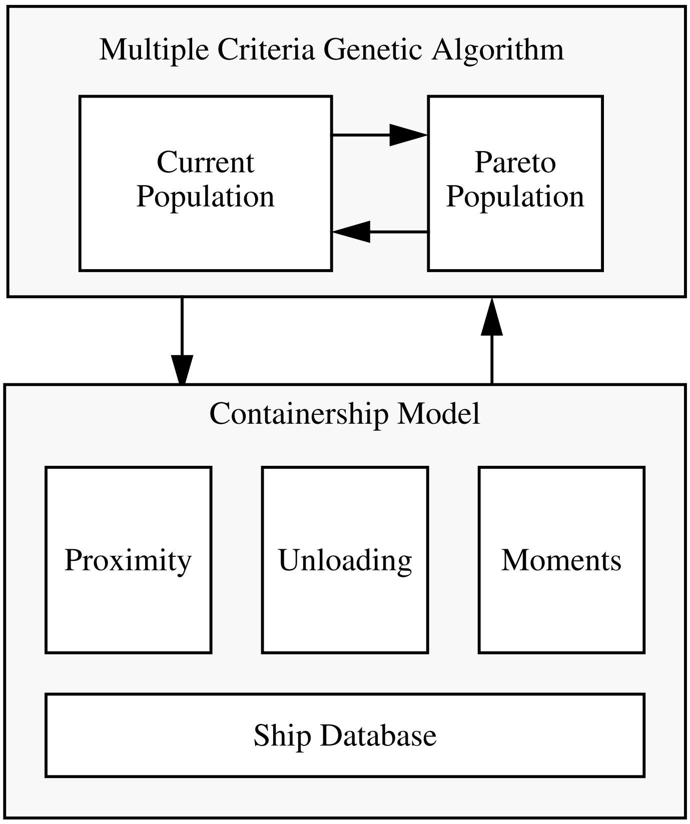


Figure 8.6 - Criteria Average of Pareto Population

8.7 Heuristic Operators

8.7.1 Crossover

The system uses a standard crossover routine with up to 50 randomly selected cross points. This operator is restricted using the same technique described in Chapter 4. Any string can only mate with strings in the confines of the surrounding hyper-cuboid. This restriction is used to prevent individuals which are a great distance apart in criteria space from crossing together. Performing such mating can destroy the characteristics of both strings. Once again the sides of the cuboid are dictated by the maximum and minimum values of the criteria in the Pareto Population.

Even with this restriction the crossover operator can cause errors to occur in the strings. A repair routine is used to identify any omissions or duplicates and to return or remove them from the string. The removal of duplicates is done at random. However, replacement is carried out in a manner such that containers are placed near other containers of the same type if possible.

8.7.2 Mutation

It was discovered in the earlier tests that a simple mutation operator (i.e. direct swapping of two randomly chosen container locations within the string) did not provide sufficient diversity with the search converging too early to give non-optimal solutions. This led to the investigation of a heuristic mutation operator whose aim was to perform a more 'intelligent' mutation. This operator utilises four general heuristics that a stevedore uses when loading a vessel.

> Heuristic 1: Place heavier containers near the bottom in order to reduce the vertical moments hence increasing stability.
>
> Heuristic 2: Place heavier containers near the centre line of the ship in order to reduce the transverse moment i.e. no tilt to port to starboard.
>
> Heuristic 3: Try to place containers of similar destinations together. This will help to simplify the unloading process and hopefully reduce overstows.

> Heuristic 4: Try to spread the containers throughout the ship longitudinally. This will promote even trim and will also minimise shear forces along the vessel.

When a container is randomly selected by the mutation operator it first decides whether the container should perform *short jump* (within 50 containers either way in the string, i.e. usually within the same or adjacent bays) or *long jump* mutation (anywhere in ship). Short jump mutation is carried out if the container's current bay is not overfilled and the container being moved is amongst other containers of the same type. If these two conditions are not met it should be moved, as the bay the container is in is either too full or it is surrounded by containers from other destinations. Having chosen short jump or long jump mutation, 20 possible locations for the container are randomly selected within the range of the jump. For each of these locations the suitability of the swap is assessed in terms of the heuristics described above. Of these 20 locations, those which lead to a reduction in horizontal and vertical moments and abide by the proximity and overfilling heuristics are marked. From these marked locations a swap position is randomly selected and the swap takes place.

8.8 Results

The code with the new heuristic operators was implemented and compiled for the elitist+roulette, random injection+roulette and target value+roulette methods for selection. Each of the methods was run with a population size of 500 over 500 generations. The methods were run 5 times and compared to check for consistency. The results shown here are for a typical run.

Figures 8.7 to 8.10 show the maximum/minimum values achieved in each criteria over 500 generations. A curve is plotted for each method of selection used. In three of the four graphs the injection method performs the best followed by the elitist method. In Figure 8.9 both the inject and target methods perform poorly compared to the elitist method. The elitist method is able to maintain more diversity than the injection and target methods. In the case of the target method this is due to the selection routine using a distance metric. The distance metric takes the normalised distances of each point

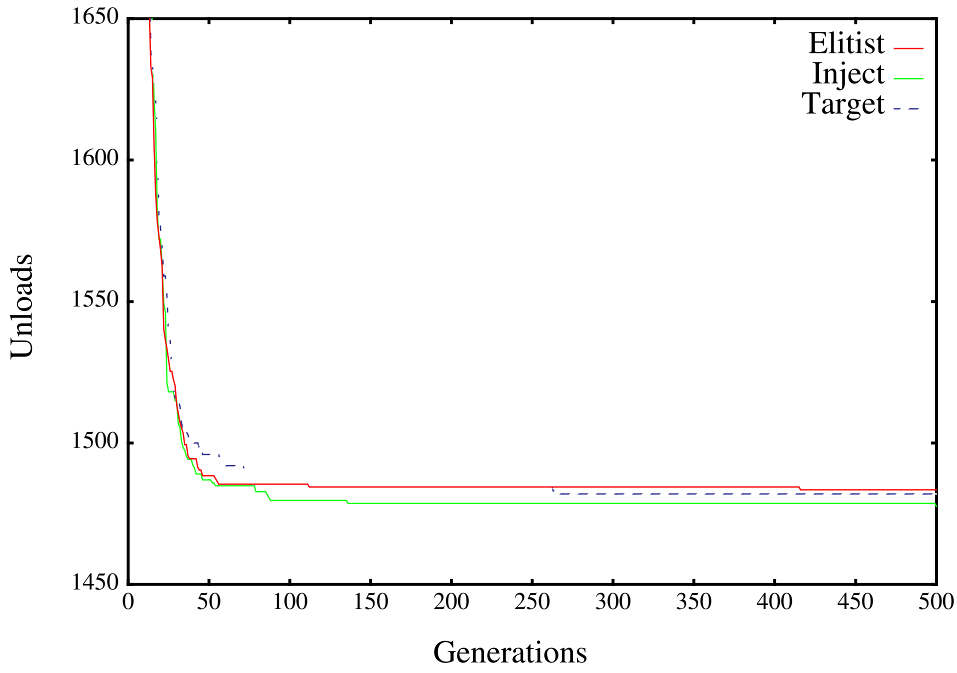


Figure 8.7 - Unloads Against Generations

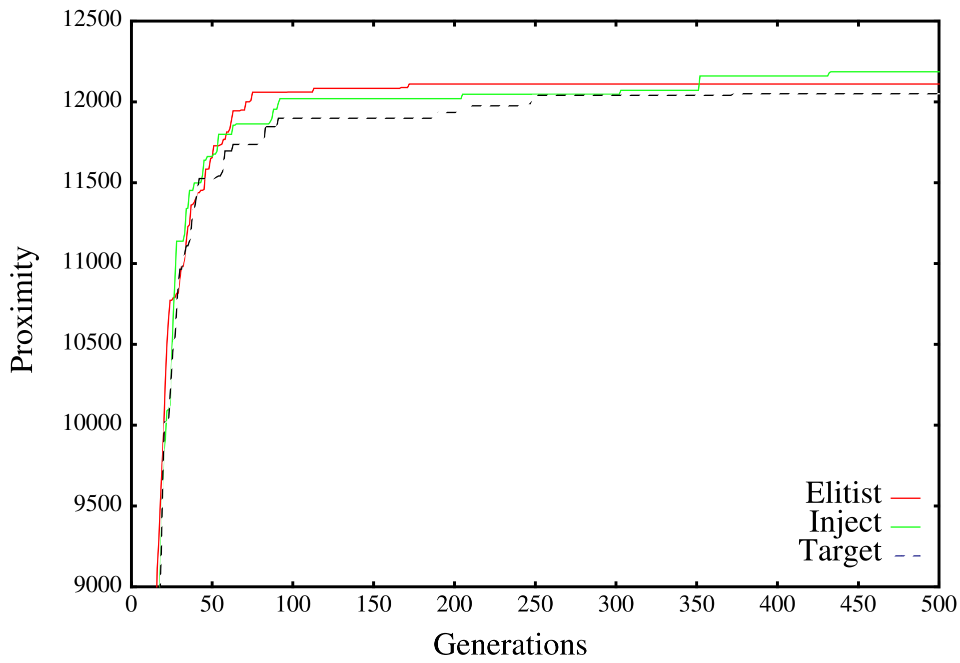


Figure 8.8 - Proximity Against Generations

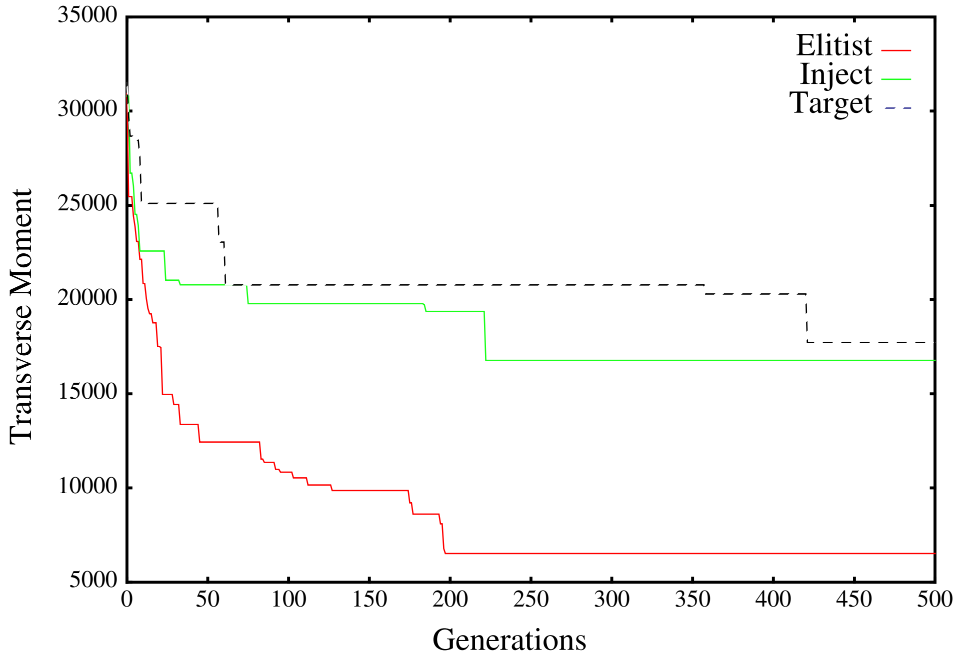


Figure 8.9 - Transverse Moment Against Generations

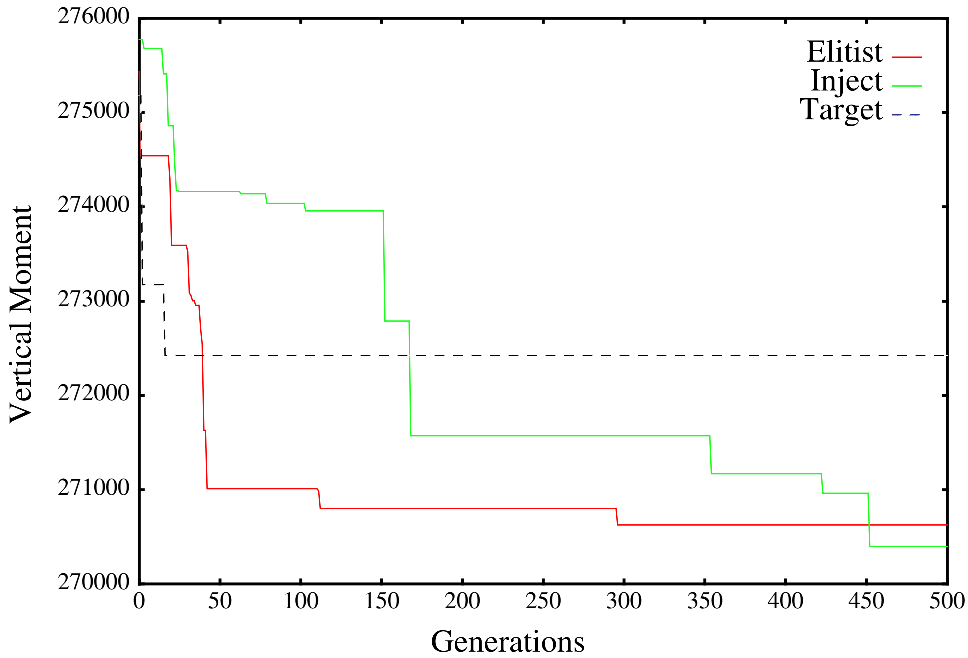


Figure 8.10 - Vertical Moment Against Generations

from the ideal point. Due to the convex shape of the pareto surface it will not select individuals at the extremes of the surface and will always favour compromise solutions. This is shown in Figure 8.11. The target value is set at the maximum of each of the criteria and the distance lines radiate outwards from this point. The region of the Pareto surface which is favoured is highlighted

In the containership model the target value was unloads=1400, proximity=15000, transverse moment=0, vertical moment=26500. The values of proximity and vertical moment were estimated.

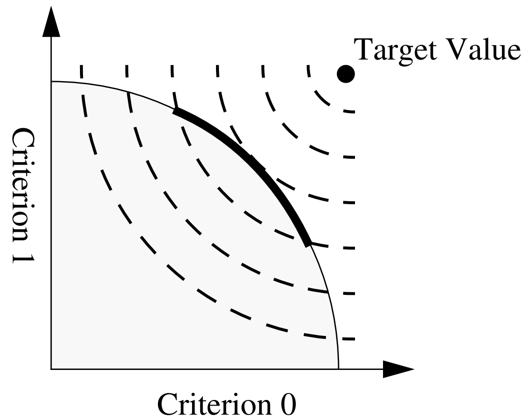


Figure 8.11 - Target Values and Convex Surfaces

This bias towards middling individuals is more clearly shown using the plots of the final Pareto Populations. Figure 8.12 shows the final Pareto Populations for all three selection methods. The graphs show the Pareto points projected onto the unloads and proximity axes. In the case of the elitist method the MCGA has found two main clusters of points. The smaller cluster contains points which have low values of transverse moment. The other methods perform poorly in this criteria and hence do not detect this region of efficient solutions. The target method clearly favours middling individuals identifying points near the centre region of the elitist methods' large cluster. It however produces a few points with good unload scores. The dotted line is drawn at unloads = 1480. This is used to show that the injection method achieves better values than the elitist method in both unloads and proximity scores. The fact that no second cluster has been produced suggests that this method is biased towards the optimisation of proximity and unloads criteria. Looking back at Figures 8.7 - 8.10 the proximity and unloads curves converge far faster than the moment curves. This suggests that lots of solutions with high values of proximity and low values of unloads

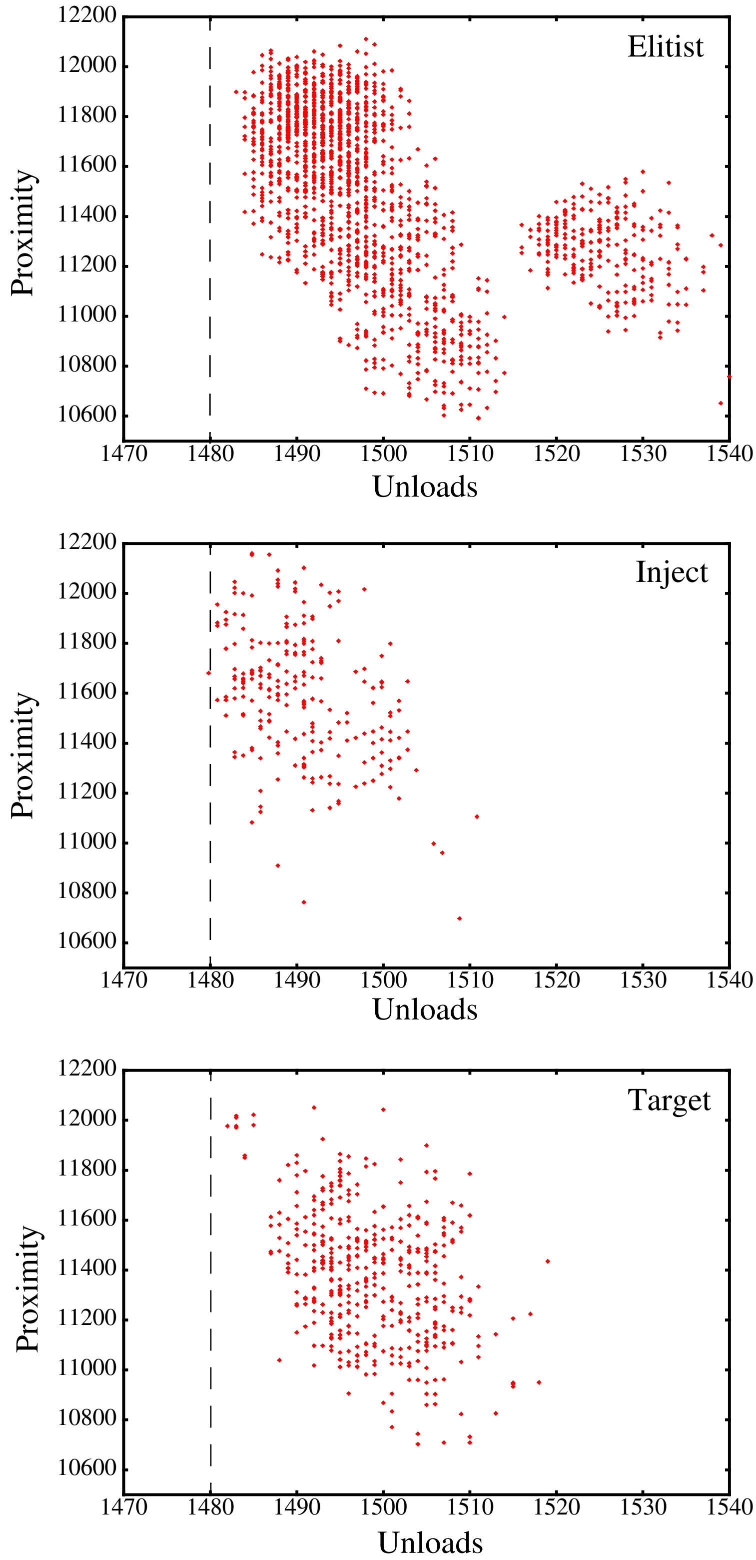


Figure 8.12 - Final Pareto Populations

will be produced at the start of the search. The random injection algorithm picks a certain number of individuals at random from the Pareto Population and re-inserts them into the normal population. The selected solutions will reflect the bias in the Pareto Population so there will be more solutions with high proximity and low unloads scores inserted into the population. This will mean that the new solutions generated during genetic processing will have a similar bias, hence more biased solutions will be inserted into the Pareto Population. This perpetuates itself, creating more and more biased solutions. This problem was not encountered in Chapter 7 since all criteria had exactly the same fitness landscape and no heuristics were used.

The execution for all three cases was relatively time consuming. The elitist+roulette routine was the slowest at around 10 hours. The other two methods took approximately 7 hours. This is because the elitist method generated much greater numbers of Pareto solutions over the 500 generations. The random injection method generates around 300, the target value around 450 and the elitist in the region of 1700 solutions. This difference is due to the low bias generating a broader set of solutions. This means that more processing of the Pareto solutions needs to be carried out when using the elitist method.

8.9 Combined Elitist and Random Injection

The previous tests showed that the different methods produced differing results, the elitist method provided better values of transverse moment while the injection and target value methods provided better values in unloads and proximity. This led to the further implementation of the Containership MCGA using both Elitist and Random Injection combined. The aim being to combine the diversity of the elitist method with the more directed effort of injection. It was hoped that the elitist method would prevent the search from having a bias towards any one criteria whilst still improving the speed and solution quality afforded by the injection method. The MCGA was run under the same conditions as before. The elitist strategy was used followed by the random injection method which filled to 50% of the population. The remainder was filled using roulette wheel selection from the normal population.

This system produced far more solutions, between 3500 and 4000 over the 500 generations. It would not be possible to hold this many points within a normal GA unless the population size was set to 4000. This list could be reduced by specifying some sensible ranges on the criteria, for example by only considering designs with unloads under 1600 and vertical moments less than 279000. Some methods for selecting individuals are described in Chapter 9. Typically, the total run time was just over 14 hours on an Ultra-Sparc 1 due to the increased Pareto Population size. This was reduced to around 8 hours by only processing the data in the Pareto Population every 10 generations. In comparison, a port using conventional methods would need several days to compile plans. The configurations produced are sensible and generally good. However, there are instances, occurring quite regularly, where heavy containers are placed near the top or out to the sides of the ship and also examples of empty spaces in the holds which could be filled by containers on deck. These occur because of the nature of the GA, which is capable of finding good solutions quickly but cannot guarantee totally optimal ones. It has trouble identifying single containers which do not affect any of the criteria significantly even with the heuristics present.

The four graphs (Figures 8.13-8.16) show the population average and the Pareto Population average for all four criteria. As in the single criterion case, the use of criteria average for all four criteria is a rational method for deciding when a useful search has ceased. However, this does not provide a good indication of how the pareto front develops. Snap-shots of the Pareto Population in two criteria at 0, 20, 100 and 500 generations were therefore generated and projected onto a two criteria axis. The two chosen criteria are indicated. Examples of these graphs are shown in Figures 8.17 - 8.19. As stated before, ideally we would like a solution with Unloads=1400, Proximity=15000, VM=26500, TM=0 but because this solution is not feasible a solution from the trade-off set must be chosen.

It can be seen from Figures 8.13 and 8.14 that both values of average unloads and proximity converge quickly within the population. This is due to the GA operators quickly acting to move similar containers together thus decreasing the number of unloads. Figure 8.17 shows that the search has not been biased by the use of the injection method. The correlation between unloads and proximity is clearly shown in this figure. The movement of the population is along a tight band. At 500 generations two groupings(circled) can also be seen just to the right of the main group. These points have been maintained because they have good values of transverse moment. This was

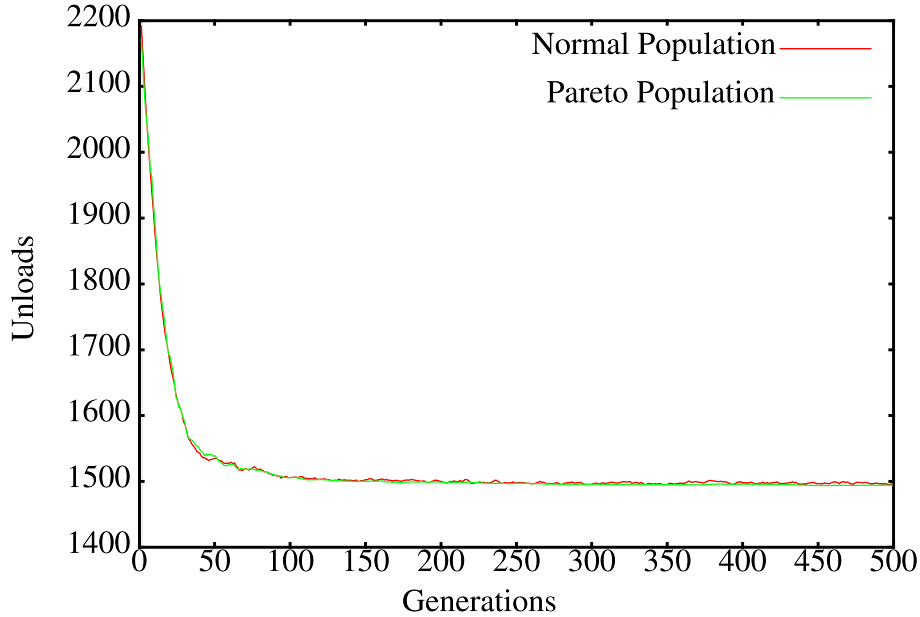


Figure 8.13 - Normal and Pareto Population Average Fitness (Unloads)

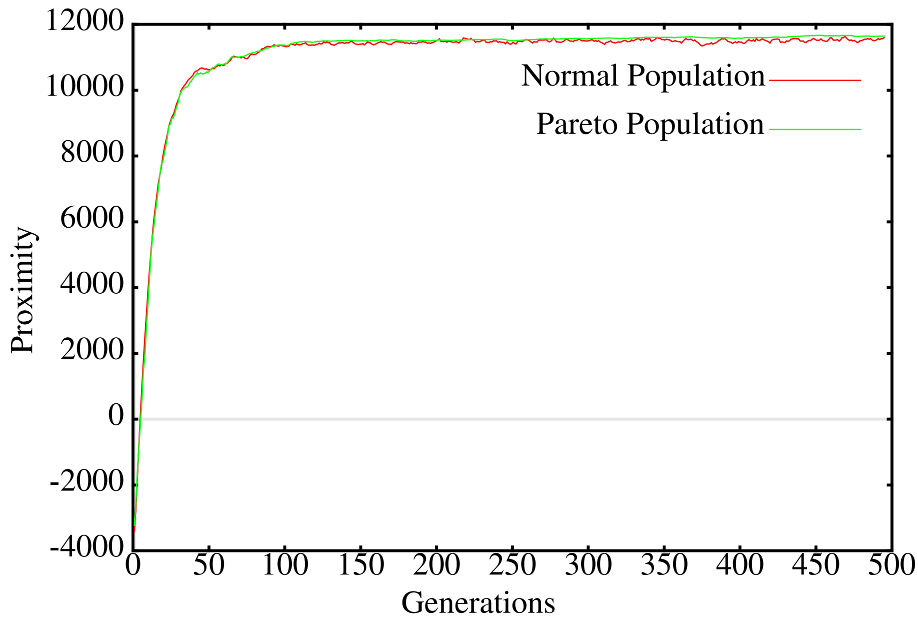


Figure 8.14 - Normal and Pareto Population Average Fitness (Proximity)

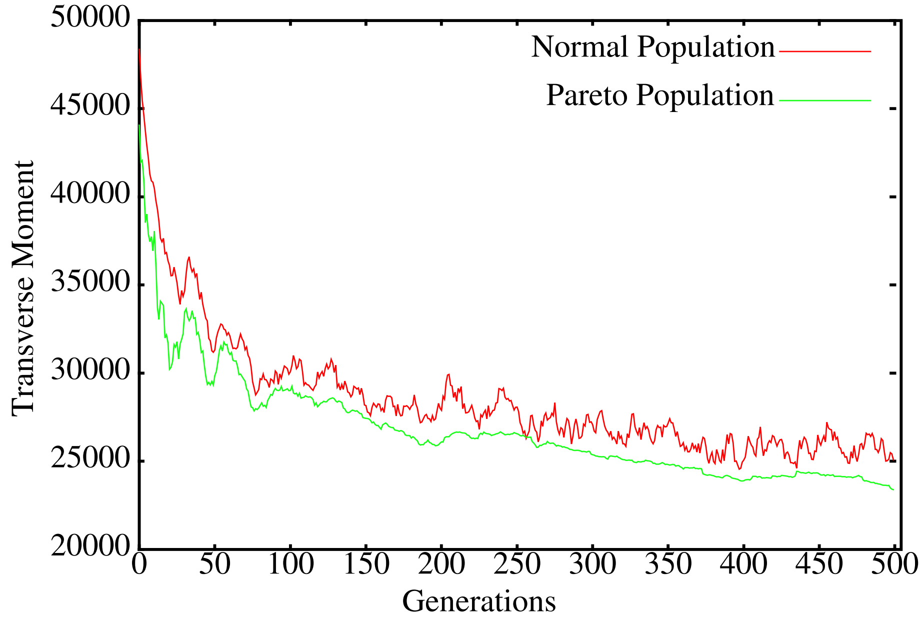


Figure 8.15 - Normal and Pareto Population Average Fitness

(Transverse Moment)

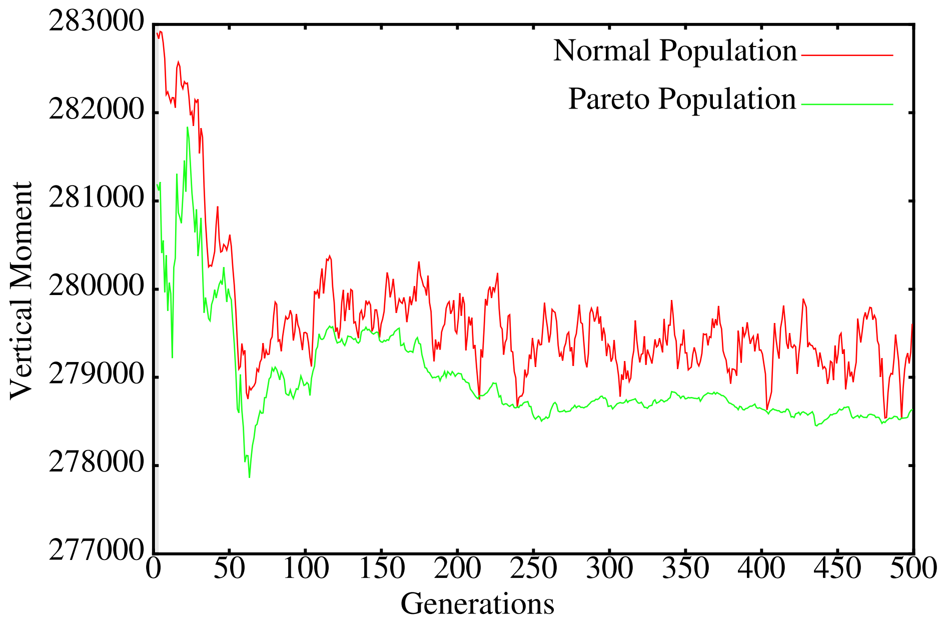


Figure 8.16 - Normal and Pareto Population Average Fitness

(Vertical Moment)

identified as a single grouping in the previous example but the search has surpassed some individuals and identified two separate groups.

Transverse (TM) and vertical (VM) moments have less regular convergence curves (Figures 8.15 and 8.16). The average values of these fluctuate until the unloads and proximity settle, at this point (~150 Generations) the TM and VM averages also start to converge. This fluctuation occurs because proximity and unloads converge much faster than TM or VM. This means that most rank one solutions generated and replaced in the first 150 generations are due to previous rank one solutions being surpassed in the unload and proximity criteria. Hence the normal and Pareto Populations of the early generations are mainly dependent on increasing unloads and proximity and not on the progress of the VM and TM values. Furthermore, VM and unloads are not highly correlated as illustrated by Figure 8.18, which shows unloads against VM for 0, 20, 100 and 500 generations. The vertical bands of Pareto points show that a wide range of VM values exist for any given unload value. However, near the bottom of the graph, around 1500 unloads(circled), it can be seen that it becomes harder to find configurations with small numbers of unloads and low VM.

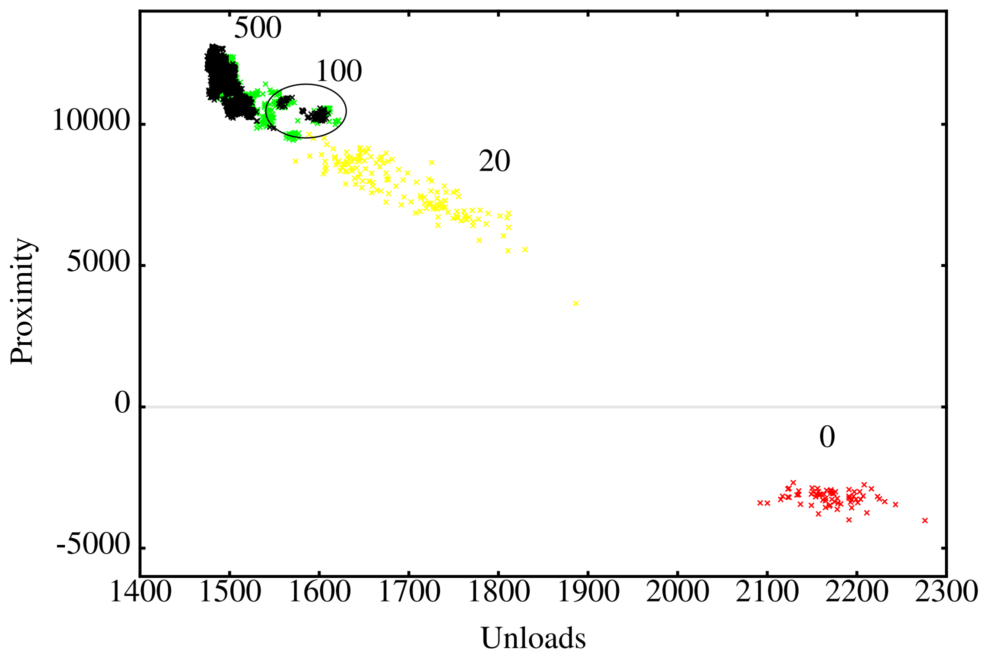


Figure 8.17 - Movement of the Pareto Population

Over 500 Generations (Proximity against Unloads)

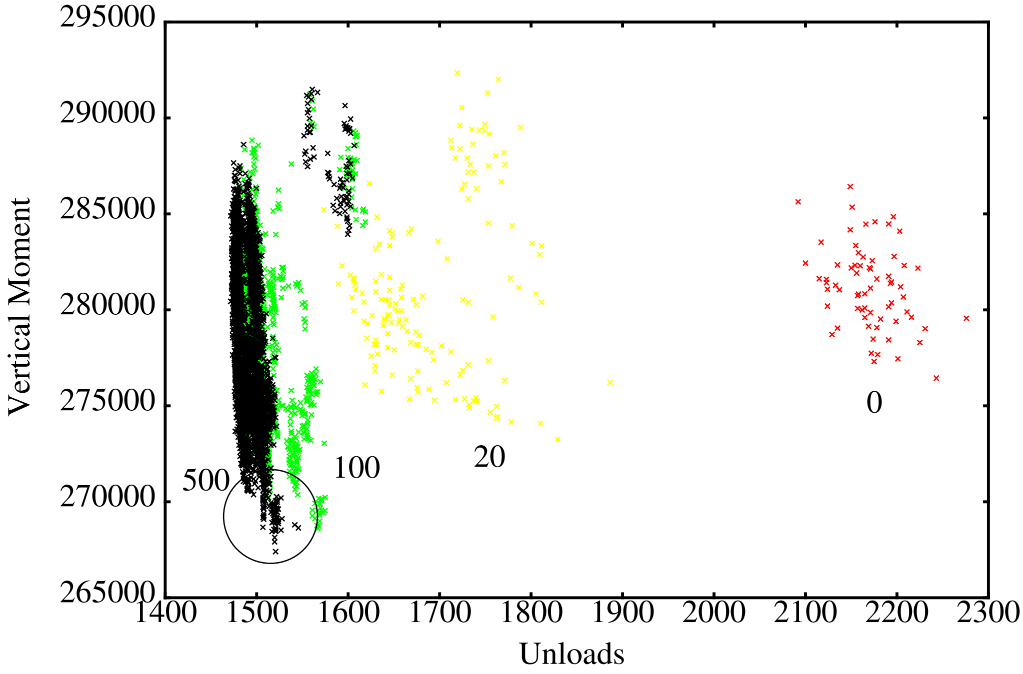


Figure 8.18 - Movement of the Pareto Population

Over 500 Generations (VM against Unloads)

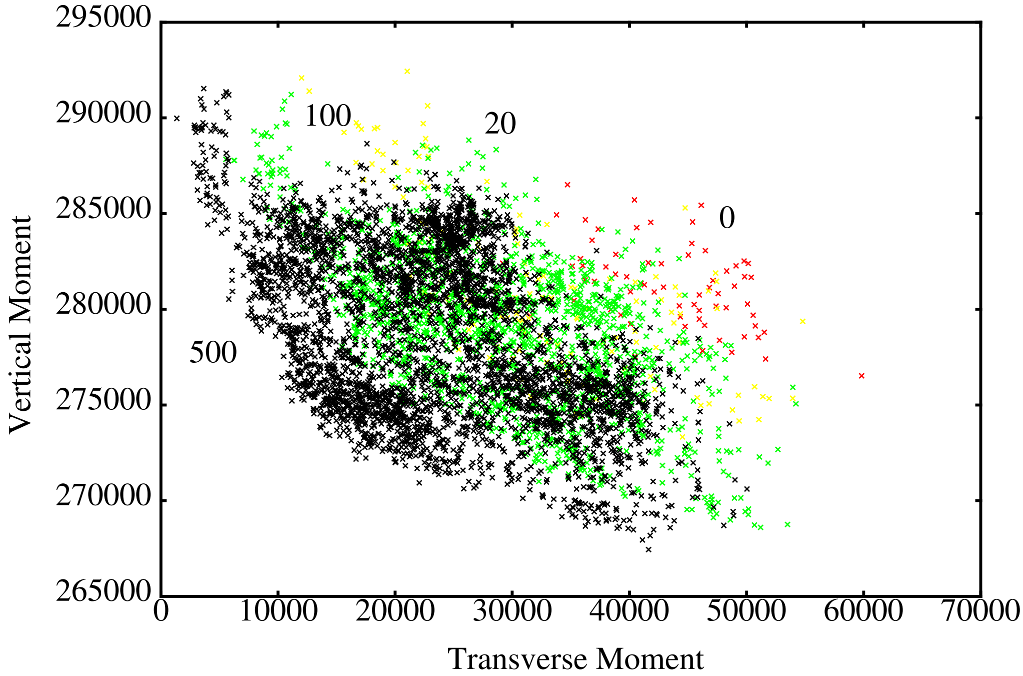


Figure 8.19 - Movement of the Pareto Population

Over 500 Generations (VM against TM)

Figure 8.19 shows that there is a weak correlation between VM and TM, it is visibly less pronounced than that in figure 8.17. This relationship is derived from the configuration of containers within the ship. The lower in the vessel containers are placed, the more likely they are to be placed nearer the centre, as the number of cells across each bay reduces as you get closer to the keel.

In general the MCGA is able to produce a range of configurations with generally good properties. However the nature of the GA prevents it from being able to deal with single mispositioned containers. A ‘clean-up’ routine was written to be applied after the search had terminated and is outlined in section 8.11. The time taken to run the code is too long to be used in most real time contexts. For the system to be implemented in a port the run time would need to be less than 2hrs. This could be achieved with more efficient programming and the implementation of parallel algorithms. Expansion of the program to include more factors in the search such as considerations of deck strength, crane restrictions and shear strength analysis could also be undertaken. The proximity routine could be refined to include grouping refrigerated containers or separating hazardous ones.

8.10 GA, Heuristic and GA+Heuristic

To examine the relative effectiveness of the GA, Heuristic and GA+Heuristic search methods separately the three algorithms were run against each other. The convergence graphs of the three methods for the four criteria are shown in Figures 8.20-8.23. The heuristic method repetitively applies the rules used in the mutation operator. This method performs well in the first two criteria (unloads and proximity); however it is very poor at the other two (vertical and horizontal moment) although it was designed to address all four. This highlights the difficulty of devising and incorporating heuristics into multiple criteria search. Ideally any heuristics or local search would possess no bias whatsoever as this may preclude a truly diverse search. From the graphs the GA method can be seen to perform badly on the first two criteria. However it does perform much better on the moment calculations than the heuristic search. Drawing the two techniques together creates a synergy between the methods. The Heuristic+GA method performs better than the other methods in three of the four criteria creating improved solutions with good values across all criteria.

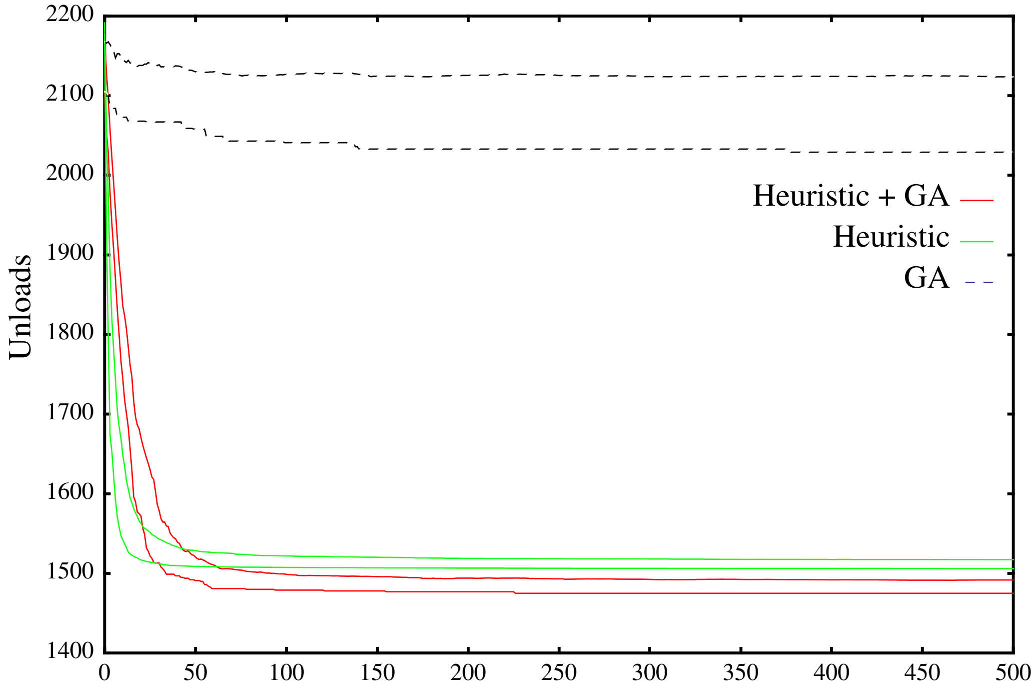


Figure 8.20 - Unloads Against Generations

(Minimise Unloads - Ave and Min Curves Shown)

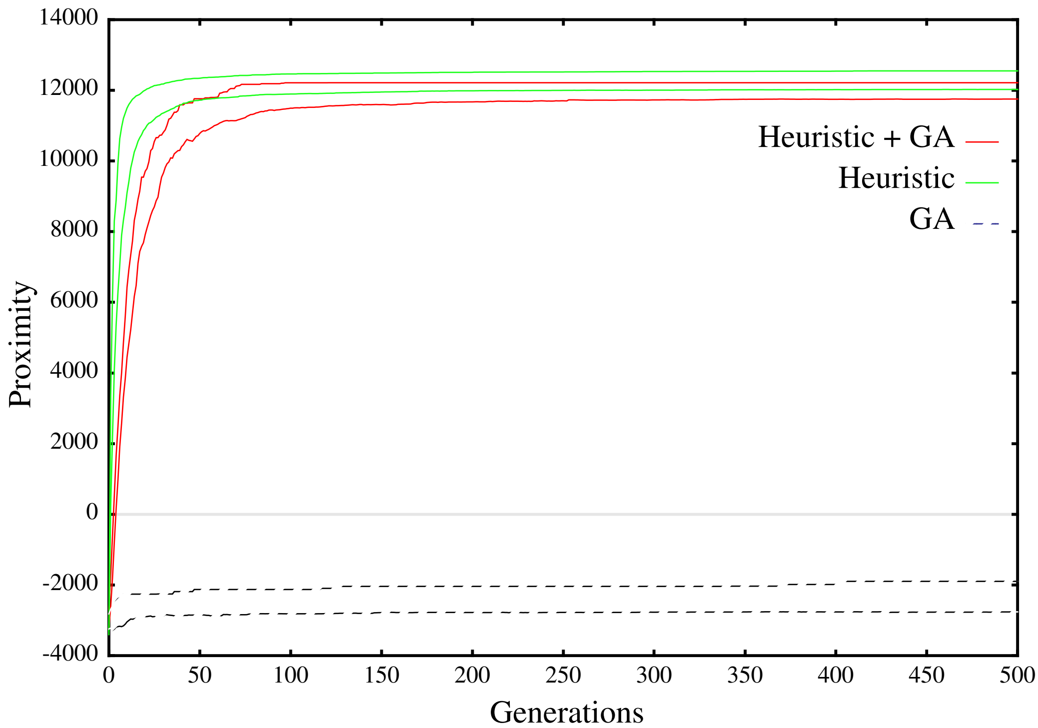


Figure 8.21 - Proximity Against Generations

(Maximise Proximity - Ave and Max Curves Shown)

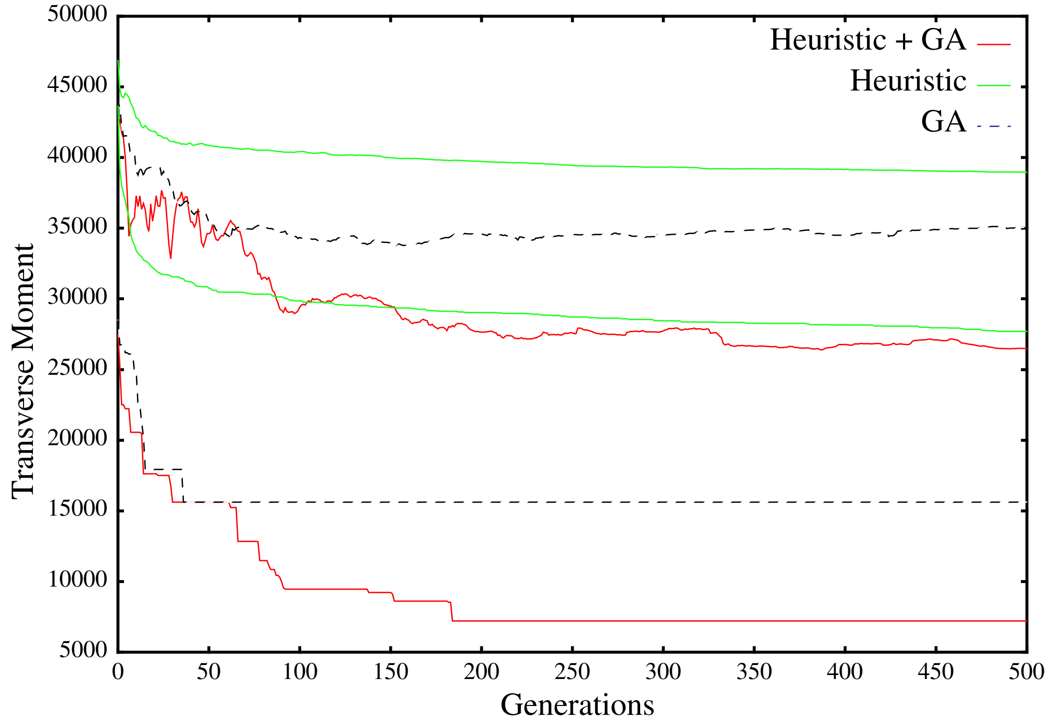


Figure 8.22 - Transverse Moment Against Generations

(Minimise Transverse Moment - Ave and Min Curves Shown)

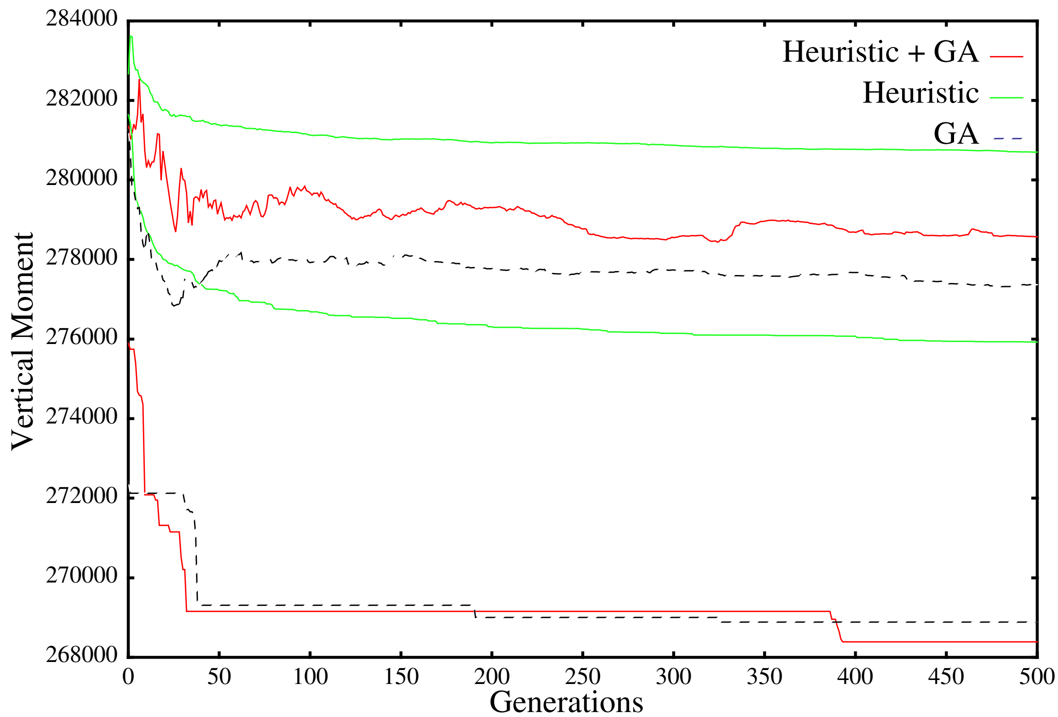


Figure 8.23 - Vertical Moment Against Generations

(Minimise Vertical Moment - Ave and Min Curves Shown)

8.11 Container ‘Clean Up’ Routine

A routine was implemented for processing the container plans after MCGA search. The method uses a local search similar to the 2-opt method discussed in earlier chapters. The process is too time consuming to be applied to every one of the designs generated by the MCGA so a few designs must be selected by the decision maker. Initially the input configuration and the MCGA generated design are read in. The former is needed to show which containers must remain in their original positions. The generated design undergoes a 2-opt search where containers are picked and swapped with other containers only if the vertical or horizontal moment is improved. Figure 8.24 shows the kind of improvements possible. It may be better to terminate MCGA search earlier and then do the rest of search on a bay by bay basis using this method. This would cut down the amount of MCGA processing required. Unfortunately the diverse range of solutions provided by the MCGA would not be maintained as the time for processing them all is too great. A typical output plan of a whole vessel is shown in appendix E.

```
   0X 0    0X 0  682B13  610B16    0X 0  260B 8    0X 0    0X 0    0X 0    0X 0   21B 6    0X 0    0X 0
 200B 4    0X 0  659B20  701B15  126B 6  223B 8    0X 0    0X 0    0X 0 1024C 7  175B 7  580B18  613B16
 100B 6  576B14 1376E 7 1236D17  640B10  556B12  681B18    0X 0  554B20  377B 5  102B 8  192B 5  199B 7

                   0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0  624B15  544B 4
                   0X 0    0X 0    0X 0  609B 6    0X 0    0X 0    0X 0  158B 6  525B 8
                   0X 0  366B 5  671B12  645B19  652B 6    0X 0    0X 0  474B 4  527B 8
                 426B 6   60B 6  558B17  194B 4  572B18  261B 6  598B20  447B 5  531B10
                 674B 4  602B15  381B 7  415B 4  262B 8  280B 5  405B 4  457B 5  529B11
                 603B18   80B 6  258B 6  608B15  578B 4  648B11  635B13  495B 4  518B12
                1064C18 1214D15  586B18  616B11  679B14  146B 7  611B11  524B 4  521B14
                 574B12  112B 5  530B16  522B17  523B19  520B18  526B17  519B 4  528B15
```

Transverse Moment: 51   Vertical Moment: 4730

Before Clean Up

```
   0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0
   0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0    0X 0
   0X 0  556B12    0X 0  377B 5  192B 5  200B 4  544B 4    0X 0    0X 0  474B 4   21B 6  682B13  640B10

                 194B 4  366B 5  126B 6  674B 4  447B 5  415B 4  405B 4  578B 4  659B20
                 100B 6  609B 6 1024C 7  158B 6  652B 6  280B 5  495B 4  524B 4  260B 8
                 426B 6   60B 6  262B 8  175B 7 1376E 7  261B 6  199B 7  701B15  223B 8
                 525B 8   80B 6  671B12  529B11  527B 8  381B 7  648B11  576B14  531B10
                 518B12  102B 8  610B16  624B15  679B14  146B 7  635B13  258B 6  616B11
                 580B18  602B15  613B16 1236D17  558B17  611B11  522B17  457B 5  574B12
                 681B18  608B15  526B17  572B18  603B18 1214D15 1064C18  112B 5  521B14
                 554B20  530B16  586B18  645B19  523B19  520B18  598B20  519B 4  528B15
```

Transverse Moment: 0 Vertical Moment: 3595

After Clean Up

Figure 8.24 - Container ‘Clean Up’ Routine

8.12 Chapter Summary

In this chapter a very large combinatorial problem was tackled using the MCGA. The containership problem is a TSP-like problem involving loading containers into large vessels. The aim of this process was to reduce handling in port and improve various ship characteristics.

The problem is a complex one with many container combinations possible. In order to tackle such a problem special heuristic operators were required in order to aid in search. Additionally two selection methods, random injection and target value, from Chapter 7 were tested on this real life problem. It was found that these methods did modify the type of search carried out, however this modification was not always helpful.

A combined random injection/elitist method was also implemented based on previous experiments and found to be reasonably successful at tackling the problem. However, the results generated still require further processing using a 'clean up' routine in order to make essential improvements.

The use of heuristics proved to be vital in tackling this problem. The synergy between the MCGA and the heuristics was demonstrated. It was also highlighted that such rules or local search methods must also be as unbiased as possible when dealing with multiple criteria search. Biased operators may mislead search creating poor results.

The next chapter looks at several techniques which can be used to help the decision maker select a single solution from a set of solutions generated by the MCGA.

9 Multi-Attribute Decision Making for Solution Selection

9.1 Introduction

Having used the MCGA to sythesise a range of solutions, a single solution must be selected by the decision maker. This chapter explains several Multiple Attribute Decision Making(MADM) methods from the literature which can be used to reduce the solution set to a single design. These processes usually involve some method for expressing user preference. Three methods, UTA, TOPSIS and CODASID, are outlined. These methods are part of a MCDM Decision Support Environment (SEN 95) developed within the Newcastle Engineering Design Centre (EDC).

9.2 Reducing The Decision Set

After running an MCGA search a set of candidate solutions usually exists. In some cases, like the containership example, these sets may be very large. Often the decision maker can discard solutions straight away. Cost is usually a factor which has very clear bounds restricting it. Looking at the trade-off between cost and efficiency in Figure 9.1, the aim is to minimise cost and maximise efficiency. Consider that a solution set has been found, possibly by the MCGA, denoted by the thick line (case(i)). If an upper bound for cost is specified the solution set may be reduced to that shown in case(ii). Furthermore, if there is a minimum required efficiency that can be specified, then the solution set can be reduced further to that shown in case(iii). Possible problems arise if the maximum cost and minimum efficiency bounds are too strict as in case(iv) and are not satisfied by any of the generated solutions. In this case the constraints must be relaxed by the decision maker to allow some solutions to satisfy the conditions.

Another way of reducing the set is to specify preferred values for all criteria and choose the nearest solutions to these values over a given threshold as illustrated in Figure 9.2. Again if the threshold is set too high an empty solution set may exist. This will mean the threshold level on the system will have to be relaxed. This is similar to the principle used in the target value method of Chapter 7 and the TOPSIS method.

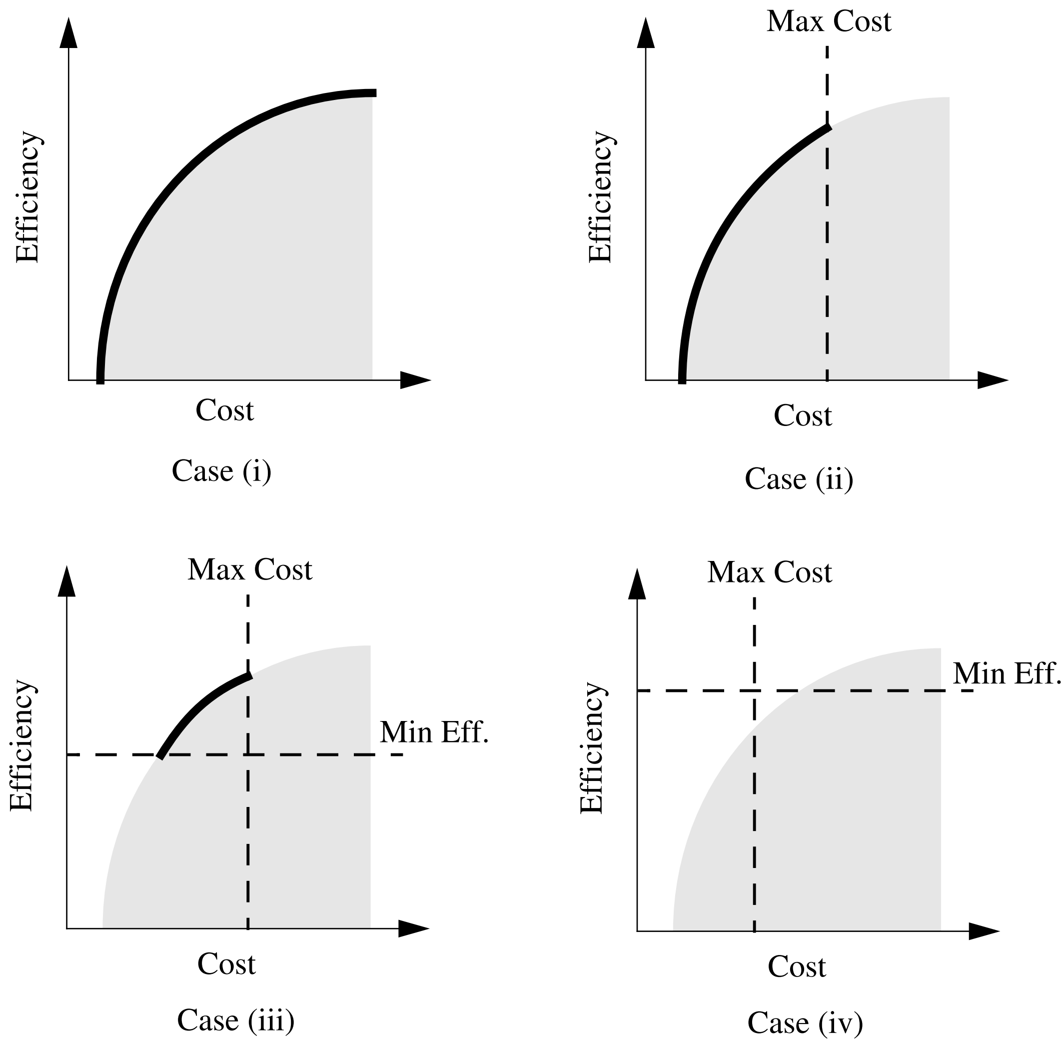


Figure 9.1 - Criteria Constraints

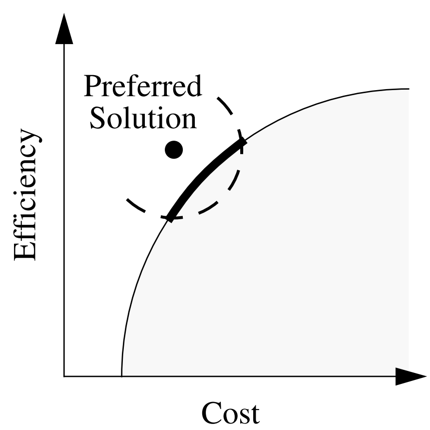


Figure 9.2 - Preferred Solutions

Both of these methods can be used to reduce the solution set to a single solution which can be implemented or further developed. However, it is more usual that a small set of solutions will exist from which a single solution must be selected. This reduced set can be then processed further using an MADM method.

9.3 MADM Methods

In order to perform MADM a set of alternative designs and a set of performance attributes for each must be given. The aim is to choose the most favourable solution or rank solutions in order, with regard to all attributes. The attributes can be either qualitative or quantitative. In the case of the MCGA they are mostly quantitative. In this case the MADM problem can be represented as a decision matrix, such as the one shown in Figure 9.3. This matrix contains 6 alternative containership loading strategies for use in the examples developed in this chapter.

$$
\begin{array}{cc}
 & \text{Attributes} \\
 & \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \\ \text{Unloads} & \text{Proximity} & \text{T.Moment} & \text{V.Moment} \end{array} \\
\text{Designs} \begin{array}{c} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} &
\left(\begin{array}{cccc}
1514 & 11027 & 40830 & 275652 \\
1490 & 11544 & 32857 & 282047 \\
1488 & 11729 & 31900 & 289674 \\
1485 & 11524 & 47254 & 286813 \\
1499 & 11509 & 22775 & 290021 \\
1491 & 11738 & 32948 & 284712
\end{array} \right)
\end{array}
$$

Figure 9.3 - Containership Decision Matrix

There are many methods with which to resolve such a selection problem. However three methods - UTA, TOPSIS and CODASID are explained as these are the ones that are generally in harmony with the problem at hand implemented within the MCDM Decision Support Environment. These methods use differing ways of expressing preference and deciding on the final ranking of solutions. All three operate on a decision matrix of the form shown in Figure 9.3.

9.3.1 UTA Method

The Additive UTility (UTA) method (JACQUET-LAGREZE 82) uses subjective ranking of alternative designs to assess a set of additive utility functions based on these rankings. Linear programming estimates the parameters of the utility functions and sensitivity analysis is performed to generate a mean utility function. The steps involved are as follows:

i) Select a subset of strings. Form a decision matrix of these designs.

ii) Define an additive utility function as:

$$U(y) = \sum_{i=1}^{k} u_i(y_i)$$

The maximum and minimum values for each attribute y_i are defined. This interval is then cut into $\alpha_i - 1$ equal sub intervals. The marginal utility of a design is then approximated by a linear interpolation. The utility of a design a_l is now given by:

$$u[y(a_l)] = \sum_{i=1}^{k} \hat{u}_i[y(a_l)] + \sigma(a_l)$$

where $\sigma(a_l)$ is an estimation error for the estimated utility function $\hat{u}$.

iii) Preference information can then be acquired. These preferences can be strict(P) where one design is better than another, or indifferent(I) where two designs are about the same in terms of acceptability. A small positive real number, δ, must be chosen such that two solutions a_l and a_h can be separated significantly. This leads to some general properties :

$$u[y(a_l)] - u[y(a_h)] \geq \delta \Leftrightarrow a_l \text{ is Prefered to } a_h$$

$$u[y(a_l)] - u[y(a_h)] = 0 \Leftrightarrow a_l \text{ is Indifferent to } a_h$$

iv) All marginal utility functions are defined as monotonic, either non-increasing or non-decreasing.

v) The problem is then constructed as a linear programming one:

$$\text{minimise } F = \sum_{l=1}^{n} \sigma(a_l)$$

Subject to:
All preferences and indifferences being satisfied.
All marginal utility functions being either non-increasing or non-decreasing.
Maximum utility = 1, Minimum Utility = 0 and $\sigma(a) \geq 0$.

Thus the aim here is to minimise the total deviation subject to monotonicity and preference relations. Solving the above problem gives an estimation of the marginal utilities. However, there is usually more than one solution.

vi) The mean utilities are then generated by exploring other solutions around the initially generated optimum to give the mean marginal utility functions.

vii) The mean utility functions can then be used to rank the designs. A better design has a larger utility value.

Example:
Using the Decision Support Environment much of the UTA process is automated leaving the decision maker to simply specify some preferences.

i) The designs in the Containership Decision Matrix (Figure 9.3) were selected from the Pareto Population.

ii) The maximum values were defined, the number of intervals was set to 5 and δ was set to 0.01. The utilities and errors can then be calculated.

iii) The partial preference information was supplied as follows:

$a_4 \; P \; a_3 \qquad a_3 \; P \; a_2 \qquad a_2 \; P \; a_6 \qquad a_6 \; P \; a_5 \qquad a_5 \; P \; a_l$

Linear Programming is carried out to minimise the errors in the marginal utilities and the mean marginal utilities are found.

vii) The final ranking is then determined based on these utilities and the final preference order displayed.

Design 4 : 0.487317
Design 3 : 0.381109
Design 2 : 0.378485
Design 6 : 0.377219
Design 5 : 0.372703
Design 1 : 0.296712

9.3.2 TOPSIS

The Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) is based on the concept that the chosen alternative should be as close to the ideal solution as possible and as far away from the negative ideal (HUANG 81). This method again requires a decision matrix but also needs relative weights to represent preference information. TOPSIS also assumes that each attribute is monotonically increasing or decreasing. TOPSIS operates as follows:

i) From the decision matrix a normalised matrix is constructed element by element giving each attribute the same unit vector.

$$z_{ij} = \frac{y_{ij}}{\sqrt{\sum_{i=1}^{n} y_{ij}^2}}$$

ii) A weighted normalised decision matrix is then formed whose elements are given by:

$$z_{ij} = w_j z_{ij}$$

where w_j is the weight of the jth attribute.

iii) The ideal point a* and the negative ideal a^- are then defined as the collection of best and worst values of the attributes.

iv) The closeness of each design to the ideal and negative ideal points is calculated.

$$S_i^* = \sqrt{\sum_{j=1}^{k} (x_{ij} - x_j^*)^2}$$

$$S_i^- = \sqrt{\sum_{j=1}^{k} (x_{ij} - x_j^-)^2}$$

v) The relative closeness to the ideal is then found for each design.

$$C_i^* = \frac{S_i^-}{(S_i^- + S_i^*)}$$

vi) The alternatives are then ranked based on the magnitude of closeness, with preferred solutions having greater values.

Unfortunately TOPSIS suffers from two weaknesses. Firstly, the definition of separation between alternatives and ideal and negative ideal points is done via a Euclidean distance measurement. This is highly sensitive to the subjective weights used to give the weighted normalised decision matrix. Sensitivity is increased further for higher dimension decision spaces. The second problem concerns the fact that the distance definition automatically assumes that the attributes can be directly compensated by each other in a simple manner. This may lead to the method selecting designs with strange balances between criteria possibly leading to extreme solutions. The following example illustrates this method.

Example:

i) The Containership Decision Matrix (Figure 9.3) was first normalised. It should be noted that the minimisation values have been made negative in order that all attributes are maximised.

The normalized decision matrix

-0.4136	0.3910	-0.4683	-0.3950
-0.4070	0.4093	-0.3769	-0.4042
-0.4065	0.4159	-0.3659	-0.4151
-0.4056	0.4086	-0.5420	-0.4110
-0.4095	0.4081	-0.2612	-0.4156
-0.4073	0.4162	-0.3779	-0.4080

ii) Weights were then provided, in this case unloads were given priority followed by vertical moment. The normalised decision matrix was then weighted.

Weights = 0.4500 0.1000 0.2000 0.2500

The weighted normalized decision matrix

-0.1861	0.0391	-0.0937	-0.0988
-0.1832	0.0409	-0.0754	-0.1011
-0.1829	0.0416	-0.0732	-0.1038
-0.1825	0.0409	-0.1084	-0.1028
-0.1843	0.0408	-0.0522	-0.1039
-0.1833	0.0416	-0.0756	-0.1020

iii) The ideal point and negative ideal were then calculated:

Ideal

-0.1825	0.0416	-0.0522	-0.0988

Negative Ideal

-0.1861	0.0391	-0.1084	-0.1039

v) The relative closeness to the ideal was found for each design:
alternative 1 = 0.272633, alternative 2 = 0.588965, alternative 3 = 0.622184
alternative 4 = 0.068485, alternative 5 = 0.911041, alternative 6 = 0.583980

vi) The alternative designs can then be ranked based on the closeness.

$$5 > 3 > 2 > 6 > 1 > 4$$

9.3.3 CODASID

COncordance and Discordance Analyses by Similarity to Ideal Designs again uses the decision matrix to generate a preference order for a set of designs (SEN 94). CODASID only allows limited compensation between attributes thus overcoming the problem encountered in TOPSIS. Three new indices, a preference concordance index, an evaluation concordance index and a discordance index provide independent measures for ranking designs.

A concordance and discordance analysis is carried out based on the weights, original attribute values (O-values), normalised values (N-values) and weighted normalised values (WN-values).

i) Concordance Analysis - If an attribute in a design (a_k) is better than the same attribute in another design (a_l) it is said to be preferred in terms of that attribute. If a_k is preferred to a_l (a_kPa_l) any normalised attribute value which supports this assumption is said to be concordant with a_kPa_l. When taking into account all attributes it is also possible to ask to what extent multiple attributes are concordant with a_kPa_l. Two indices are calculated, an evaluation concordance(EC) and a preference concordance(PC) index for pairs of designs.

$$EC(a_kPa_l) = \frac{\text{largest absolute difference between N-Values on concordance attributes}}{\text{largest absolute difference between N-Values on all attributes}}$$

$$PC(a_kPa_l) = \text{sum of the normalised weights of the concordance attributes}$$

ii) Discordance - If a_k is preferred to a_l (a_kPa_l) any normalised attribute value which does not support this assumption is said to be discordant with a_kPa_l. A combined discordance(CD) is defined.

$$CD(a_kPa_l) = \frac{\text{largest absolute difference between WN-Values on discordance attributes}}{\text{largest absolute difference between WN-Values on all attributes}}$$

WN-values are used in this case so that information in the decision matrix and the weights are both used.

iii) Pairwise comparisons are then carried out in order to assess preferences and the indices can then be calculated.

iv) These indices can be combined to form measures of the total dominance, total importance and total inferiority of a design over all others.
Net preference concordance dominance index, $p(a_k)$:

$$p(a_k) = \text{sum of differences between } PC(a_kPa_l) \text{ and } PC(a_lPa_k) \text{ for all other designs, } a_l$$

Net evaluation concordance dominance index, $e(a_k)$:

$$e(a_k) = \text{sum of differences between } EC(a_kPa_l) \text{ and } EC(a_lPa_k) \text{ for all other designs, } a_l$$

Net discordance dominance index, $d(a_k)$:

$$d(a_k) = \text{sum of differences between } CD(a_kPa_l) \text{ and } CD(a_lPa_k) \text{ for all other designs, } a_l$$

v) Each design has three new indices which are formed into a Judgement Evaluation Matrix which becomes the new decision matrix.

vi) The Judgement Evaluation Matrix is then processed using the TOPSIS method, first to give the normalised and weighted normalised values and then the final ranking of alternatives.

Example:

The Containership Decision Matrix (Figure 9.3) was formed into its normalised and weighted normalised forms using the weights.

The normalized decision matrix

0.0000	0.0000	0.2624	1.0000
0.8276	0.7271	0.5881	0.5549
0.8966	0.9873	0.6272	0.0241
1.0000	0.6990	0.0000	0.2232
0.5172	0.6779	1.0000	0.0000
0.7931	1.0000	0.5844	0.3694

The weighted normalized decision matrix

0.0000	0.0000	0.0525	0.2500
0.3724	0.0727	0.1176	0.1387
0.4034	0.0987	0.1254	0.0060
0.4500	0.0699	0.0000	0.0558
0.2328	0.0678	0.2000	0.0000
0.3569	0.1000	0.1169	0.0924

i) The evaluation concordance and preference concordance was then calculated for all designs.

The evaluation concordance index matrix

0.0000	0.1113	0.2440	0.2598	0.2500	0.1576
0.4701	0.0000	0.1327	0.2370	0.2286	0.0559
0.5622	0.0921	0.0000	0.2289	0.1782	0.0366
0.4248	0.0431	0.0756	0.0000	0.1818	0.0517
0.4832	0.1030	0.0932	0.2500	0.0000	0.1039
0.5288	0.0682	0.0895	0.2579	0.2418	0.0000

The preference concordance index matrix

0.0000	0.2500	0.2500	0.4500	0.2500	0.2500
0.7500	0.0000	0.2500	0.5500	0.8000	0.9000
0.7500	0.7500	0.0000	0.3000	0.8000	0.6500
0.5500	0.4500	0.7000	0.0000	0.8000	0.4500
0.7500	0.2000	0.2000	0.2000	0.0000	0.2000
0.7500	0.1000	0.3500	0.5500	0.8000	0.0000

ii) This was followed by a calculation of combined discordance:

The combined discordance index matrix

0.0000	0.5103	0.5751	0.5199	0.4481	0.5213
0.1113	0.0000	0.0649	0.0776	0.0824	0.0273
0.2440	0.1327	0.0000	0.0963	0.0746	0.0876
0.2467	0.2034	0.1543	0.0000	0.2000	0.1835
0.2500	0.2833	0.2077	0.2752	0.0000	0.2487
0.1576	0.0626	0.0551	0.0931	0.0831	0.0000

iii) Pairwise comparisons were then carried out.

iv) The above three tables were then reduced into three indices, $p(a_k)$, $e(a_k)$ and $d(a_k)$, for each design.

v) These indices form a judgement-evaluation matrix:

The judgement-evaluation matrix

-2.1000	-1.4463	1.5651
1.5000	0.7067	-0.8288
1.5000	0.4629	-0.4219
0.9000	-0.4566	-0.0742
-1.9000	-0.0472	0.3767
0.1000	0.7805	-0.6168

vi) Finally the judgement evaluation matrix was processed using TOPSIS to give a final ranking of:

2 > 6 > 3 > 4 > 5 >1

9.4 Discussion

The three methods provided different rankings:

UTA : 4 > 3 > 2 > 6 > 5 > 1

TOPSIS : 5 > 3 > 2 > 6 > 1 > 4

CODASID : 2 > 6 > 3 > 4 > 5 > 1

Looking at the three methods, 2 and 3 are the most consistently good solutions. 1 is obviously a poor solution since is performs badly in all three methods. Solutions 4 and 5 are the most interesting. In the first case(UTA), 5 is nearly at the bottom and 4 is at the top, in the second case(TOPSIS) it is the other way round, and in the third case(CODASID) they are both at the bottom. Looking at the criteria values in Figure 9.3, 4 has the best value of unloads but a poor value of transverse moment. 5 on the other hand has a very good value of transverse moment and a poor value of unloads and vertical moment. Using the UTA method unloads were heavily favoured and using TOPSIS transverse moment is the dominating factor, especially as design 5 is much lower in value of this attribute than the other designs. Using CODASID the preference has favoured alternative 2, which is not outstanding on any criteria but represents a compromise design.

It is clear that expressing preference information is very important when using MADM methods. It is important that the designer understands what his priorities are. It is also important that the designer has an idea of the workings behind the methods in order to understand the processes which are occurring and the reasons why certain ordered rankings are being made.

MADM tools can often be a way of discovering preferences. The methods can be executed several times in order to experiment with weights. This can lead to greater understanding and an awareness of the conflicts involved in the problem.

9.5 Chapter Summary

The MCGA is part of a range of tools available for multiple criteria investigation and analysis. The MCGA can be used to generate solutions to complex problems. These solutions must then be further processed to obtain a preferred solution.

Three methods for performing selection from a range of possible alternatives were investigated. UTA, TOPSIS and CODASID were assessed by executing the selection procedures to select a preferred containership layout from six possible candidates. The systems produced ordered ranking of the designs indicating which solutions were preferred over others.

MADM tools act on the preference information which is provided by the designer or decision maker. The methods are very sensitive to changes in this information and as such it is important that the decision maker clearly understands the methods, the problem and his preferences in order to use the tools properly.

The final chapter draws conclusions on the work presented and also discusses possible future directions for research arising from the topics covered in this thesis.

10 Conclusions and Further Work

10.1 Conclusions

The aim of this thesis was to develop a general purpose GA based tool for handling large combinatorial multiple criteria problems. Many practical problems of interest in engineering design and operation fall into this category and a systematic investigation of an adaptive algorithm like a GA seemed an appropriate way to proceed. The developed tool was to be applied to a range of engineering based problems in design and operation to test its usefulness.

Initially the basic mechanics and theory of GAs was introduced in order to provide a basis for the work. An experiment to compare different encoding schemes was also carried out as a familiarisation exercise. Performing this type of study is an ideal way to understand more of the workings of any tool or algorithm. In this experiment it was found that real coded GAs can offer some advantages over normal binary GAs. However the real GA did suffer from early convergence problems due to the compact encoding.

The Travelling Salesman Problem (TSP) was used to test a set of crossover and mutation operators to assess their ability in tackling combinatorial problems. Various domain independent operators were explained for reference purposes and applied to the travelling salesman problem in order to asses their relative effectiveness. The aim was to find a good general operator for combinatorial type problems. Enhanced Edge Recombination Crossover and Adjacent Two City Mutation performed the best during the tests. However, these operators still struggled when problem size increased. These tests provided an essential insight into the recombinative effects which must be present to provide constructive crossover in combinatorial problems. Such effects were used later in the thesis when defining new operators and representations.

Using domain specific heuristics and local search did improve performance significantly. Using such techniques it is possible to gain near optimal and sometimes optimal solutions for large TSPs. However, the use of local search and heuristics does prevent the general applicability of the search. This means that for new problems new operators and local search techniques have to be implemented. The kind of efficiency gains that can be sought by using optimization methods on combinatorial engineering problems was also demonstrated using an electronic component placement problem.

The background theory of multiple criteria problems and how genetic algorithms have been constructed for such problems was discussed. Developing from previous work a Multiple Criteria Genetic Algorithm (MCGA) was constructed but with the additional incorporation of a Pareto Population. The Pareto Population was shown to preserve individuals which may normally be lost during genetic processing.

Niche sizing was discussed along with a new method of estimating the number of individuals required to effectively sample the search space for a given resolution. Additionally an adaptive niching method was introduced which could vary the niche size automatically during search, depending on the current dimensions of the Pareto front. This removes the need to specify a niche size at the start of search from approximate and incomplete data.

The MCGA was tested on a set of four test functions with increasing numbers of criteria. Results showed that the algorithm was able to identify and effectively sample the Pareto regions of the test functions. The adaptive niche sizing method also proved to be effective in maintaining a broad population spread over the criteria space as search progresses. The Pareto Population also clearly demonstrated that it was invaluable in maintaining large numbers of solutions far in excess of the population size. It was observed that as the number of criteria increases, a sufficiently large population size is needed in order to ensure a good initial distribution of solutions and to allow the sharing functions to perform effectively. It was also indicated that as the number of criteria grew the time for computation for a fixed number of generations and the number of generations both increased super-linearly.

A novel GA routine for partitioning design matrices, used in Concurrent Engineering, was demonstrated. A single criteria GA with enhanced edge and local search was shown to out perform standard methods on 12, 20 and 51 activity problems.

The partitioning problem was then expanded to a multiple criteria one where additional objectives were tackled in order to provide a selection of good solutions. From these solutions the decision maker can select the configuration from which the most beneficial solution can be isolated.

Due to the MCGA's mechanisms for maintaining diversity it was able to find solutions with better fitness than the single criteria GA in the test cases. It also highlighted the fact that if local search techniques are used they tend to bias the multiple criteria search towards a preferred direction. This may be desirable if one criterion has clear priority over the others. This form of biased multiple criteria search clearly indicates that any local search incorporated into the MCGA must be carefully constructed so as to reduce this bias as much as possible.

The use of evolutionary techniques to solve scheduling problems was investigated. The simple job shop model used currently was thought not to be detailed enough for application to many scheduling problems. For this reason a complex scheduling definition was drawn up incorporating many new features, such as the use of multi operation machines, job dependencies and setup periods.

Following this a description of operation based scheduling along with the data required to perform it was given. The construction of the MCGA Scheduler was outlined and the string representation, crossover and mutation operators and the schedule builder were also presented.

The MCGA Scheduler was tested on four problems. The problems variously looked at the performance of the scheduler and also how it could be used to perform process analysis and robust design. A shipyard steel shop example was tackled using the MCGA. This example looked at the effect of sharing a resource and the affect this has on processing time. It also addressed the issue of scheduling to deadlines with various cost penalties.

The MCGA was further enhanced by including additional functionality in an attempt to make the search more controllable from a user point of view. Various techniques were developed, implemented and tested. These techniques allow the user to tailor the MCGA search to their needs.

The first three methods described try to promote an increased level of diversity within the population. This can be used as an enhancement to normal search or for performing quick investigation into the approximate shape of the Pareto Surface. Of the methods tested the random injection method proved to be the most effective.

The two convergence methods described are used to concentrate search effort on the regions of the Pareto surface of greatest interest to the decision maker. The decision maker, with some knowledge of the form of the Pareto surface, is allowed to specify a target value or target ratio that indicates the type of solutions desired. The MCGA will then put maximum effort into finding solutions with the specified properties.

A complex interactive process was also implemented that uses a back-propagation neural network to collect preference information from the user. The system asks the decision maker to score the quality of solutions during search. These comments are then formed into a preference surface which can be used to direct search into regions of the search space which are more suited to the users requirements. Further work is still required to increase the stability of the method particularly when dealing with inconsistent judgements.

In the final example the MCGA, in association with the injection divergence and the target value convergence methods, was applied to an extremely large combinatorial problem, namely that of loading a containership with containers with the aim of optimising several criteria simultaneously. The containership problem is a TSP-like problem involving loading containers into large vessels. The aim of this process was to reduce handling in port and improve various ship characteristics.

The problem is a complex one with many container combinations possible. In order to tackle such a problem special heuristic operators were required to aid in search. It was found that the divergence and convergence methods did modify the type of search carried out, however this modification was not always helpful.

A combined random injection/elitist method was also implemented based on previous experiments and found to be reasonably successful at tackling the problem. However, the results generated still require further processing using a 'clean up' routine in order to make essential improvements.

The use of heuristics proved to be vital in tackling this problem. The synergy between the MCGA and the heuristics was demonstrated. It was also highlighted that such rules or local search methods must also be as unbiased as possible when dealing with multiple criteria search. Biased operators may mislead search creating poor results.

The MCGA is part of a range of tools available for multiple criteria investigation and analysis. The MCGA can be used to generate solutions to complex problems. These solutions must then be further processed to obtain a preferred solution.

Three methods for performing selection from a range of possible alternatives were investigated. UTA, TOPSIS and CODASID were assessed by executing the selection procedures to choose a preferred containership layout from six possible candidates. The systems produced ordered ranking of the designs indicating which solutions were preferred over others.

MADM tools act on the preference information which is provided by the designer or decision maker. The methods are very sensitive to changes in this information and as such it is important that the decision maker clearly understands the methods, the problem and his preferences in order to use the tools properly.

Having tackled several problems during the course of this research some general points ermerged. Firstly, it is important to fully understand the problem being tackled. A good knowledge of the problem will aid in the construction of a good model, adequate representation, suitable operators and heuristics. It is also important to clearly understand the goals of the problem. Certain criteria may be unimportant and others may have a greater effect on the search than envisaged. This is sometimes only discovered after the problem has been tackled for the first time. Thus a systematic validation of any code and models used is crucial.

The MCGA has provided insights into how GAs can be used in both multiple criteria and combinatorial type problems. Possible difficulties with such problems have been identified and in many cases addressed using novel techniques. The developed MCGA has proved to be a flexible and useful tool for both engineering design and operation. It has successfully been applied to several combinatorial problems in various engineering fields and as such it has achieved the goals of this research.

10.2 Further Work

Several possible areas for further work have arisen from this research concerning GAs, multiple criteria analysis and the various engineering problems tackled.

The MCGA has many possible areas for future enhancement. The code itself could be further optimised and made more efficient. This is an important factor when dealing with large problems. Performance can also be improved by further investigation of the various GA parameters. This thesis has been mainly concerned with developing a general method and solving problems rather than fine tuning of parameters. Niche sizing is one of the areas which can have a profound effect on performance since both solution fitness and the crossover operator are affected. The convergence and divergence methods can be also experimented on further. New methods of this type need to be investigated and a full analysis of performance and effects can be carried out. Parallelisation of the code could provide the additional performance required to allow the MCGA to tackle larger and more complex problems. Breakthroughs in these areas are badly needed since time consuming evaluation mechanisms are now being used with GAs leading to an explosion in the time taken to solve problems.

Interaction is an increasingly important factor in developing engineering tools. The Neural Network Preference method proved to be a potentially useful way of directing MCGA search towards a preferred area of the search space. Further development is needed to address the issues of selecting good comparison sets, presenting such sets to the user and handling decision maker inconsistency. The presentation of solutions is probably of greatest importance because it dictates how the designer assigns preference. What information is presented, how much information is necessary, and in what format solutions are put to the user are all questions which need to be addressed. The gathered information also needs to reflect what is important to the designer and this information must be acted upon by the MCGA.

Presentation of results is another area of research. When the number of dimensions exceeds 3 the difficulty of showing a trade-off surface becomes apparent. Even in three dimensions it is often hard to see the full shape of the surface without the ability to rotate it. Visualisation of high dimensional spaces is an interesting area for investigation. What is needed is a simple and quickly readable diagrammatic form which does not loose any criteria information and demonstrates the trade-offs involved.

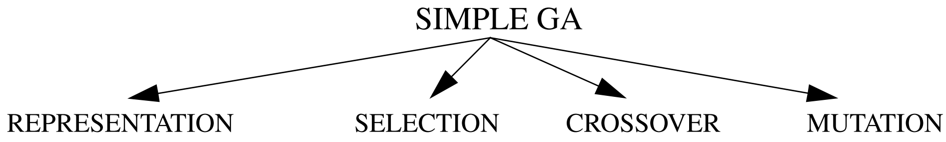
Each of the applications tackled with the MCGA offer other possibilities for further work. The process analysis tool can be used to tackle larger problems and ones of increased complexity. The tool could also be used to look at how resources are to be assigned to each of the design tasks. This work is currently being undertaken within the Centre.

The applicability of the scheduler can be further increased by adding greater functionality. A mechanism allowing multiple machines to tackle the same job simultaneously could be implemented, as could a mechanism to allow jobs to be stopped during processing if another more urgent job comes along. This moves towards the idea of real-time scheduling using the MCGA. Other types of scheduling problem could also be tackled such as staff scheduling, flow shop and resource planning. This work is being carried forward into an industrial based project again within the Newcastle EDC.

Finally, the containership problem still requires improvement of the heuristics and selection methods in order to improve solution quality and to remove the need for a clean up routine. The MCGA could also be applied to other packing and layout problems.

Evolutionary Techniques, such as GAs, are a very important area of investigation and have personally provided a great deal of interesting work and stimulation. The increase in the complexity and diversity of applications in the last 3 years has been enormous. Industry is now adopting these new ideas and techniques which should, over the next few years, provide potentially large benefits in the field of engineering design and operation.

Appendix A- GA Reference



When applying GAs four important issues arise:

1. How to represent the problem to be solved in a suitable form.
2. How to perform selection of strings for the next generation.
3. How to perform the crossover of two strings.
4. How to perform mutation within the population.

The Simple GA is the most basic form of genetic algorithm. It uses binary representation, roulette wheel selection and simple crossover and mutation. However, many modifications of simple GAs have been developed and investigated, some of these are described below.

A.1 Representation Methods

The representation used will be dependent on what type of problem is being tackled. There are several possible representation methods which are outlined below.

Binary (GOLDBERG 89) - The simplest form of encoding is the use of a concatenated binary string. Each parameter in the fitness function is represented as a binary number. These are then concatenated to form a string which can be manipulated by the GA.

Gray Coding (GOLDBERG 89) - This is a method which reduces the Hamming Distance between two consecutive numbers to 1. The advantage of this over binary is that in order to increment the value of a parameter the GA has to change 1 bit, whereas in binary all of the bits may have to be changed. e.g. in Gray Code 7 is 0100 and 8 is 1100 - Hamming Distance = 1, in Binary 7 is 0111 and 8 is 1000, a Hamming Distance of 4. This is useful in situations where the GA is searching around a small area of the search space as it reduces the number of bit changes needed.

Fixed Point (SOUTH 93)- Fixed point representations have decimal points in the same places on every string. They can use either binary or Gray representation.

Floating Point (SOUTH 93) -The position of the decimal points can vary from string to string. This means that each string must be encoded with information on the positions of the decimal points.

Ordered Representations (GREFFENSTETTE 85b)- All of the above representations are for standard parameter type problems. GAs however are also applied to ordered problems such as the travelling salesman problem or TSP (GOLDBERG 85, GREFFENSTETTE 85b). The TSP involves finding the shortest route between n cities, the salesman must visit all of the cities once and return to his home city. For example, if there are five cities, a, b, c, d, and e, starting from a, valid tours are: acdbe or aecbd, however abcce and abbdc are not valid. The requirement is for a representation that does not repeat cities, their are two such representations, Ordinal and Adjacency.

Ordinal (GREFFENSTETTE 85b) - In this case the tour is described by N integers in which the ith element can range from 1 to (N-i+1). It uses two lists, a TourList (The ordinal representation) and a FreeList (An ordered list of the cities). For example (a d c e b) is represented by (1 3 2 2 1), the starting city is fixed.

TourList	FreeList	Actual Tour
()	(a b c d e)	()
(1)	(b c d e)	(a)
(1 3)	(b c e)	(a d)
(1 3 2)	(b e)	(a d c)
(1 3 2 2)	(b)	(a d c e)
(1 3 2 2 1)	()	(a d c e b)

The advantage of the ordinal representation is that the normal crossover operator may be applied to it and it will always give valid tours, the generated tours may not however resemble their parents. The part of the tour to the right of the crossover boundary tends to be affected more by the effect of crossover, so if the crossover point is near the start of the string it causes a huge disruption on the offspring. For this reason the ordinal representation, in many cases, does no better than random search when applied to TSP.

Adjacency (GREFFENSTETTE 85b) - In adjacency representation the tour is described by a list of cities. A path from city i to city j is present if the allele in position i is j. So for the adjacency tour (3 1 5 2 4):

	Path	Actual Tour
i=1, j=3	1>3	(1 3)
i=2, j=1	2>1	(2 1 3)
i=3, j=5	3>5	(2 1 3 5)
i=4, j=2	4>2	(4 2 1 3 5)
i=5, j=4	5>4	Correct

Starting at city 1 the tour is (1 3 5 4 2). The problem with this implementation is that the standard crossover operators cannot be applied, hence modified crossover methods must be used. These are discussed later in the Crossover Methods section.

A.2 Selection Methods

The selection procedure is used to chose suitable candidates for mating. This is done by biasing the selection towards fitter candidates in the population. However some selection methods do have a random element in order that convergence of the population does not occur too quickly by ensuring that a few weaker members survive.

Roulette Wheel - This is the simplest selection method. The idea is that a ‘roulette wheel’ is divided up between each string in the population with the string holding a segment proportional in size to its fitness. The wheel is then spun and depending on where the ball lands the particular string holding that segment of the wheel is chosen and put forward for mating. This system works on the principle that the higher the fitness of a string, the bigger the segment of the wheel it holds and the more likely the ball will land in that segment. This process is repeated until the desired number of strings are obtained.

Proportionate - This scheme chooses the number of occurrences of a particular string in proportion with their fitness, compared to the populations' average fitness value. So above average strings will provide more of the crossover candidates. This method can lead to premature convergence, as below average strings stand a very small chance of being selected. The opposite effect also occurs in later generations when all strings have similar high fitnesses and there is insufficient discrimination between strings to drive the GA towards a suitable optimum.

Elitist - This scheme is usually implemented along with another method. The elitist strategy simply ensures that the best or the n best strings in the population are taken into the next generation. The strings chosen are not crossed in any way, they are simply copied forward.

Linear Rank and Scaled Fitness - The linear rank method ranks the population in order of fitness, assigns a linear fitness value to all strings and then performs some selection mechanism on the newly scaled fitness values. e.g. in a population of 10 strings the top string would receive 10 as a fitness, the second would receive 9 and so on. Similar methods simply scale the fitness value of the strings in order to increase or decrease the discrimination between good and bad strings.

Sigma Truncation - This methods uses standard deviations of strings from the average to remove the weakest strings in the population. Any strings that are more than c standard deviations below the average are removed. Scaling and selection are then performed. The value of c is typically between 1 and 3.

Stochastic Remainder - This techniques assigns the number of strings based on the integer part of the expected number of offspring. It allocates the fractional parts to the process of roulette wheel selection in order to fill any remaining places.

Tournament - The candidates for the next generation are selected via a tournament in which k randomly selected strings compete on fitness. The winners of the tournament are placed into the mating pool.

A.3 Crossover Methods

The aim of crossover is to try and achieve new strings of high fitness via the combination of two other strings of lower fitness. This is usually done by swapping segments of the parent strings.

Single Point - This involves choosing a single crossover point and swapping the tails of the strings.e.g. Crossing Strings 1 and 2 at position 4 gives NewStrings 1 and 2.

String 1 - 010111	GIVES	NewString 1 - 010101
String 2 - 110001	>>>>>	NewString 2 - 110011

Two Point - Similar to one point crossover, this involves choosing two points and swapping the segments between the two points. e.g crossing Strings 1 and 2 between positions 3 and 7 gives NewStrings 1 and 2.

String 1 - 0100011000	GIVES	NewString 1 - 0101001000
String 2 - 1111001110	>>>>>	NewString 2 - 1110011110

Multiple Point - Multiple point crossover is performed in one of two ways depending on whether odd or even numbers of points are used. For even numbers of crossing points the strings are assumed to wrap around, so a four point cross at points 2,4,7,9 would give.

0000000000	GIVES	0011000110
1111111111	>>>>>	1100111001

However for odd numbers 0 is assumed to be a default crossing point so it behaves in the same way as even numbers. e.g. in 3-point crossover with cross points at positions 2,4 and 7 would give.

0000000000	GIVES	0011000111
1111111111	>>>>>	1100111000

A.4 Mutation

Mutation in standard GAs is used mainly as a background operator preventing either 1's or 0's dying out at any string positions. However, value of the mutation probability needs to be considered very carefully. Too low and the diversity of the search is reduced, too high and the GA will not converge at a fast enough rate

A.5 Inversion

Inversion is a special operator which is not used widely in simple GA's. Inversion is performed by randomly choosing two points on a string and then flipping the order between these two points. e.g. for points 1 and 6 in the string below.

10110100 GIVES 11011000

The inversion operator is not widely used as it is quite disruptive and is only useful in certain applications.

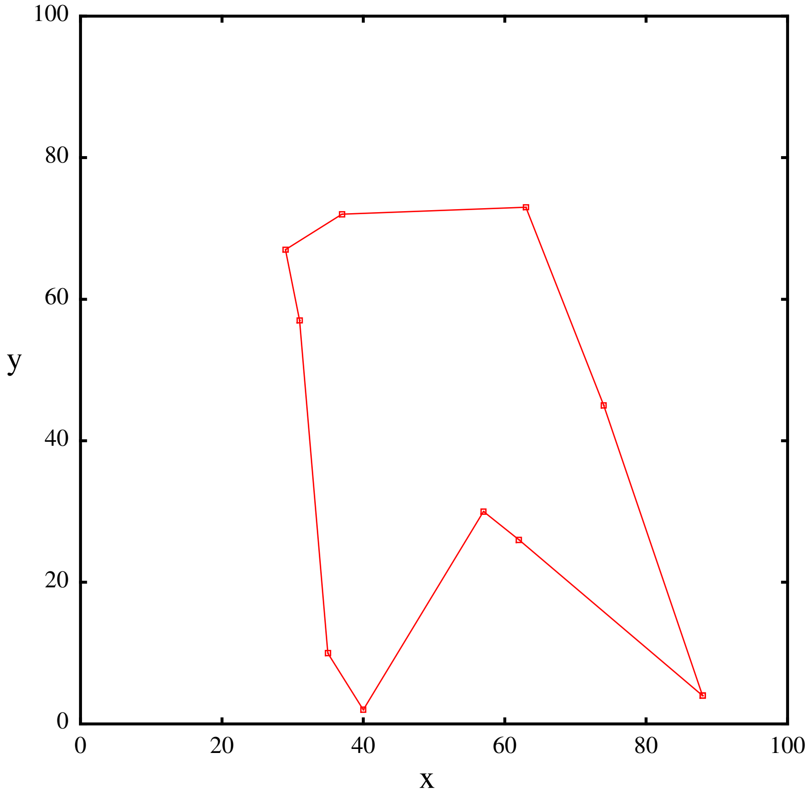
Appendix B- TSP Experimental Graphs

B.1 TSP Problems and Solutions

10 City

1:57,30	2:62,26	3:88,4	4:37,72	5:74,45	6:31,57	7:63,73
8:29,67	9:35,10	10:40,2				

Near Optimal Route:



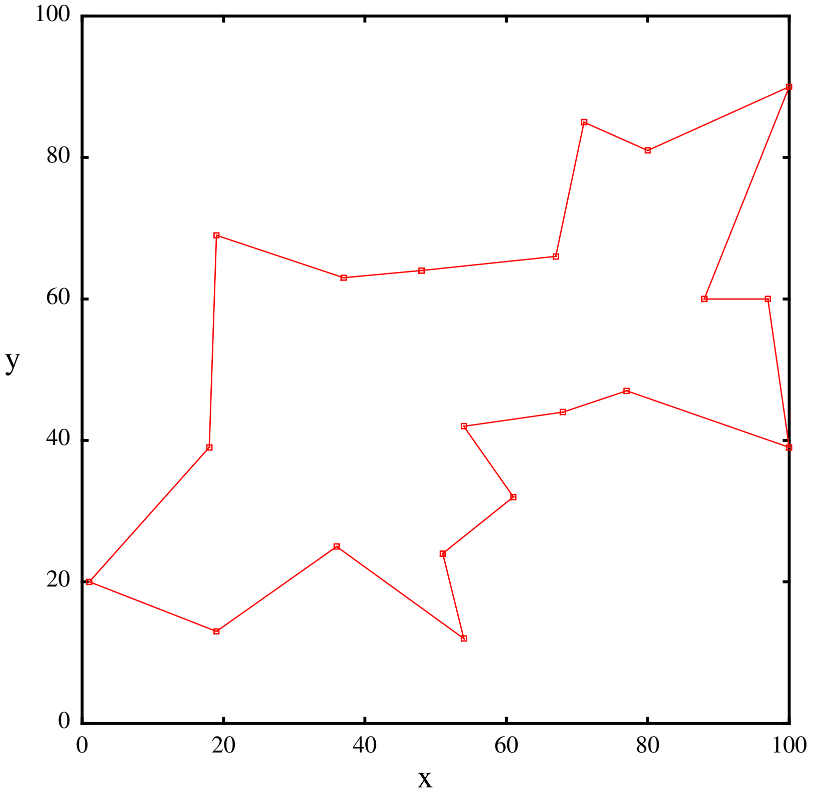
3 > 5 > 7 > 4 > 8 > 6 > 9 > 10 > 1 > 2

Distance : 221.226

20 City

1:51,24	2:68,44	3:54,42	4:67,66	5:100,39	6:77,47	7:37,63
8:19,13	9:1,20	10:54,12	11:19,69	12:71,85	13:80,81	14:88,60
15:36,25	16:100,90	17:97,60	18:48,64	19:61,32	20:18,39	

Near Optimal Route:



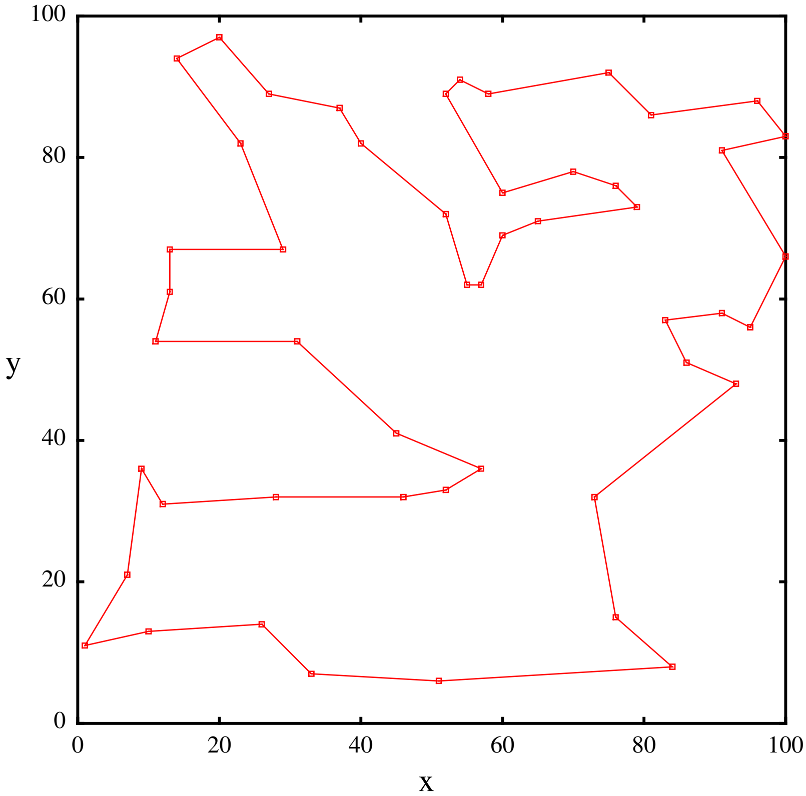
1 > 19 > 3 > 2 > 6 > 5 > 17 > 14 > 16 > 13 > 12 > 4 > 18 > 7 > 11 > 20 > 9 > 8 > 15 > 10 >

Distance : 366.045

50 City

1:95,56	2:28,32	3:26,14	4:91,58	5:100,83	6:65,71	7:29,67
8:83,57	9:93,48	10:46,32	11:7,21	12:79,73	13:60,69	14:52,72
15:20,97	16:100,66	17:73,32	18:96,88	19:57,36	20:54,91	21:1,11
22:27,89	23:40,82	24:70,78	25:12,31	26:76,76	27:14,94	28:91,81
29:9,36	30:31,54	31:13,67	32:11,54	33:51,6	34:13,61	35:52,89
36:33,7	37:60,75	38:52,33	39:84,8	40:58,89	41:23,82	42:37,87
43:86,51	44:45,41	45:75,92	46:10,13	47:81,86	48:57,62	49:55,62
50:76,15						

Near Optimal Route:



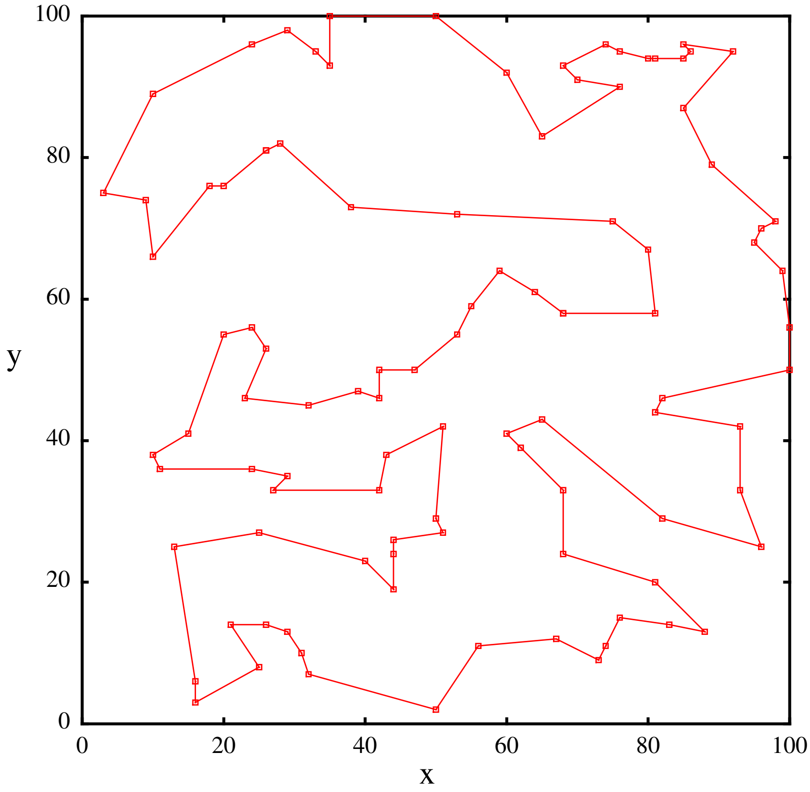
1>4>8>43>9>17>50>39>33>36>3>46>21>11>29>25>2>10>38>19>44>30>32>34>31>7>41>27>1
5>22>42>23>14>49>48>13>6>12>26>24>37>35>20>40>45>47>18>5>28>16>

Distance : 575.894

100 City

1:68,58	2:92,95	3:18,76	4:95,68	5:80,94	6:26,81	7:21,14
8:100,56	9:85,87	10:86,95	11:44,19	12:51,27	13:40,23	14:96,70
15:25,27	16:81,44	17:9,74	18:16,6	19:33,95	20:42,50	21:81,58
22:20,55	23:68,24	24:35,100	25:73,9	26:76,95	27:50,100	28:24,96
29:82,46	30:15,41	31:42,33	32:44,24	33:60,92	34:29,35	35:20,76
36:13,25	37:64,61	38:53,55	39:76,15	40:11,36	41:99,64	42:10,66
43:42,46	44:70,91	45:55,59	46:3,75	47:82,29	48:26,53	49:35,93
50:76,90	51:75,71	52:56,11	53:89,79	54:96,25	55:85,96	56:28,82
57:65,43	58:67,12	59:24,56	60:62,39	61:25,8	62:23,46	63:100,50
64:43,38	65:32,7	66:60,41	67:29,13	68:31,10	69:68,33	70:50,2
71:50,29	72:74,11	73:38,73	74:83,14	75:80,67	76:85,94	77:51,42
78:27,33	79:53,72	80:59,64	81:98,71	82:10,38	83:47,50	84:26,14
85:10,89	86:66,57	87:44,26	88:16,3	89:74,96	90:39,47	91:93,42
92:81,94	93:29,98	94:68,93	95:24,36	96:65,83	97:93,33	98:81,20
99:88,13	100:32,45					

Near Optimal Route:



1>21>75>51>79>73>56>6>6>35>3>42>17>46>85>28>93>19>49>24>27>33>96>50>44>94>89>2
6>5>92>76>10>55>2>9>53>81>14>4>41>8>63>29>16>91>97>54>47>57>66>60>69>23>98>99>
74>39>72>25>58>52>70>65>68>67>84>7>61>88>18>36>15>13>11>32>87>12>71>77>64>31>7
8>34>95>40>82>30>22>59>48>62>100>90>43>20>83>38>45>80>37>86

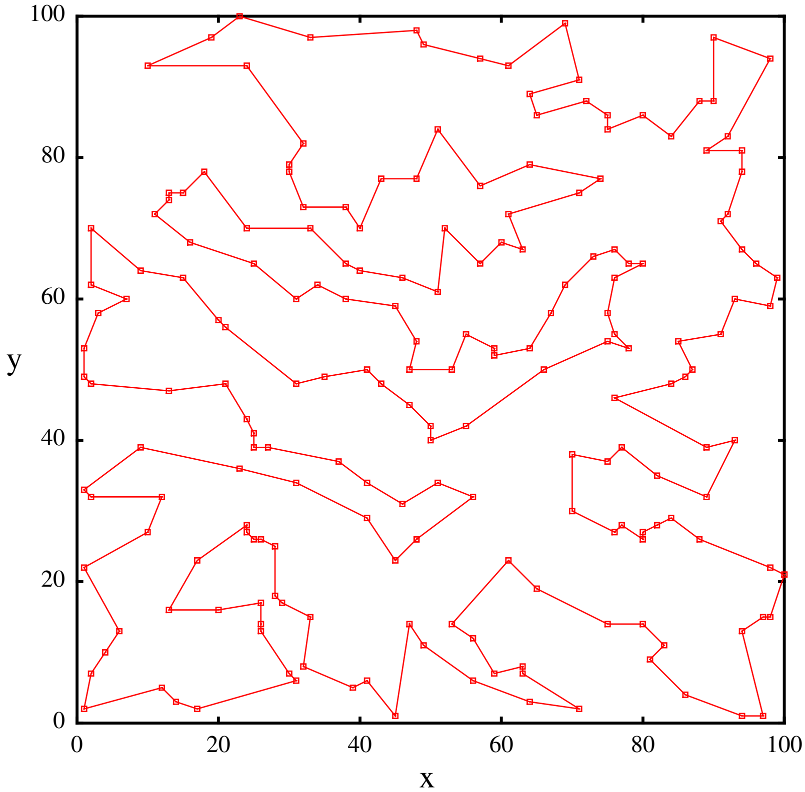
Distance : 779.234

200 City

1:48,26	2:76,27	3:38,60	4:23,100	5:80,26	6:26,17	7:41,6
8:72,88	9:65,19	10:30,79	11:92,83	12:91,71	13:84,83	14:80,86
15:25,39	16:21,56	17:65,86	18:48,98	19:33,15	20:34,62	21:1,2
22:24,43	23:60,68	24:75,84	25:45,1	26:76,63	27:90,88	28:16,68
29:66,50	30:43,77	31:74,77	32:24,28	33:12,32	34:89,39	35:84,48
36:97,1	37:64,53	38:77,39	39:76,55	40:11,72	41:23,36	42:70,38
43:50,42	44:46,63	45:99,63	46:59,7	47:94,1	48:90,97	49:75,37
50:80,14	51:63,67	52:64,79	53:49,11	54:84,29	55:49,96	56:32,82
57:69,99	58:59,52	59:88,88	60:98,59	61:13,16	62:3,58	63:48,54
64:75,14	65:64,3	66:32,73	67:1,33	68:75,58	69:64,89	70:98,94
71:78,53	72:98,15	73:38,73	74:71,2	75:24,27	76:73,66	77:51,34

78:19,97	79:21,48	80:51,84	81:2,70	82:87,50	83:26,26	84:94,13
85:94,81	86:4,10	87:76,67	88:2,32	89:63,7	90:53,14	91:57,94
92:17,2	93:24,93	94:56,32	95:45,59	96:41,29	97:57,76	98:28,25
99:80,65	100:1,53	101:88,26	102:2,7	103:28,18	104:61,23	105:14,3
106:31,6	107:47,45	108:75,54	109:89,81	110:47,50	111:67,58	112:41,34
113:89,32	114:53,50	115:39,5	116:27,39	117:29,17	118:13,74	119:17,23
120:76,46	121:13,75	122:24,70	123:15,75	124:61,72	125:51,61	126:12,5
127:56,6	128:31,48	129:35,49	130:30,7	131:45,23	132:80,27	133:1,22
134:82,28	135:33,97	136:31,34	137:82,35	138:52,70	139:55,42	140:37,37
141:47,14	142:43,48	143:93,60	144:10,27	145:38,65	146:71,75	147:1,49
148:46,31	149:57,65	150:13,47	151:86,4	152:40,64	153:78,65	154:92,72
155:98,22	156:59,53	157:50,40	158:81,9	159:97,15	160:75,86	161:94,78
162:86,49	163:61,93	164:10,93	165:91,55	166:20,16	167:26,14	168:25,65
169:15,63	170:70,30	171:85,54	172:2,62	173:55,55	174:31,60	175:94,67
176:25,26	177:30,78	178:20,57	179:71,91	180:41,50	181:96,65	182:48,77
183:25,41	184:33,70	185:6,13	186:9,64	187:56,12	188:9,39	189:26,13
190:7,60	191:40,70	192:32,8	193:93,40	194:18,78	195:100,21	196:2,48
197:83,11	198:77,28	199:63,8	200:69,62			

Near Optimal Route:



1>94>77>148>112>140>116>15>183>22>79>150>196>147>100>62>190>172>81>186>169>178>
16>128>129>180>142>107>43>157>139>29>108>71>39>68>26>99>153>87>76>200>111>37>58

>156>173>114>110>63>95>3>20>174>168>28>40>118>121>123>194>122>184>145>152>44>125>138>149>23>51>124>146>31>52>97>80>182>30>191>73>66>177>10>56>93>164>78>4>135>18>55>91>163>57>179>69>17>8>160>24>14>13>59>27>48>70>11>109>85>161>154>12>175>181>45>60>143>165>171>82>162>35>120>34>193>113>137>38>49>42>170>2>198>5>132>134>54>101>155>195>72>159>84>36>47>151>158>197>50>64>9>104>90>187>46>199>89>74>65>127>53>141>25>7>115>192>19>117>103>98>83>176>75>32>119>61>166>6>167>189>130>106>92>105>126>21>102>86>185>133>144>33>88>67>188>41>136>96>131>

Distance : 1169.11

B.2 Notation of Results

The graphs are labelled in the following form:

bsf.2.5.2

Graph Type Problem Code Crossover Code Mutation Code

Graph Type

bsf - best so far
ave - average

Problem Code

1	10 City
2	20 City
3	50 City
4	100 City
5	200 City

Crossover Code

1	Inversion
2	One Point
3	Two Point End
4	Two Point Centre
5	3 and 4 with Equal Probability
6	Position
7	Independent Position
8	Partially Mapped
9	Ordered
10	Cycle

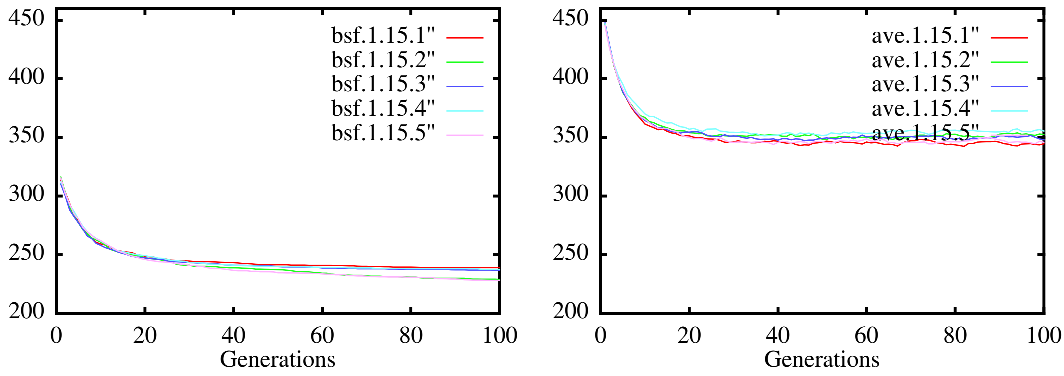
11 Edge Recombination
12 Enhanced Edge Recombination
13 Subtour Chunks
14 Alternating Edges

Mutation Code

1 Adjacent 2-City Swap
2 Random 2-City Swap
3 Adjacent 3-City Swap
4 Random 3-City Swap
5 Shift Operator

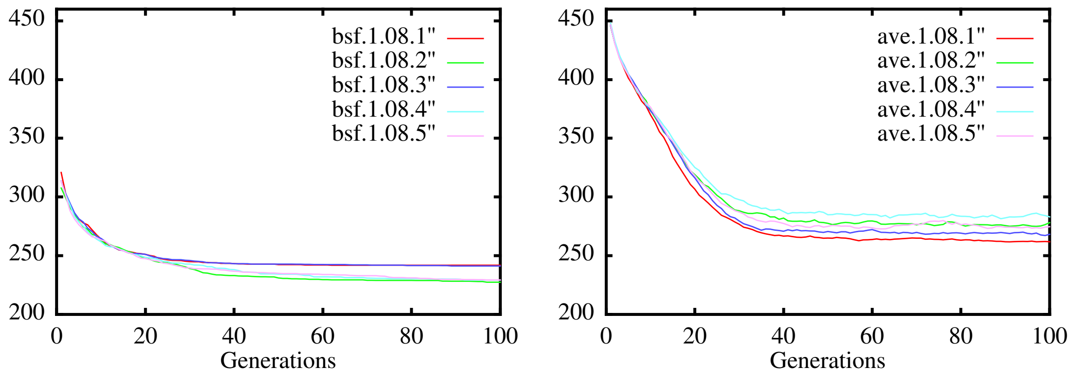
B.3 Sample Graphs

GRAPH 1 - BSF and AVE, 10 City, Crossover-1 , Mutation-1,2,3,4,5



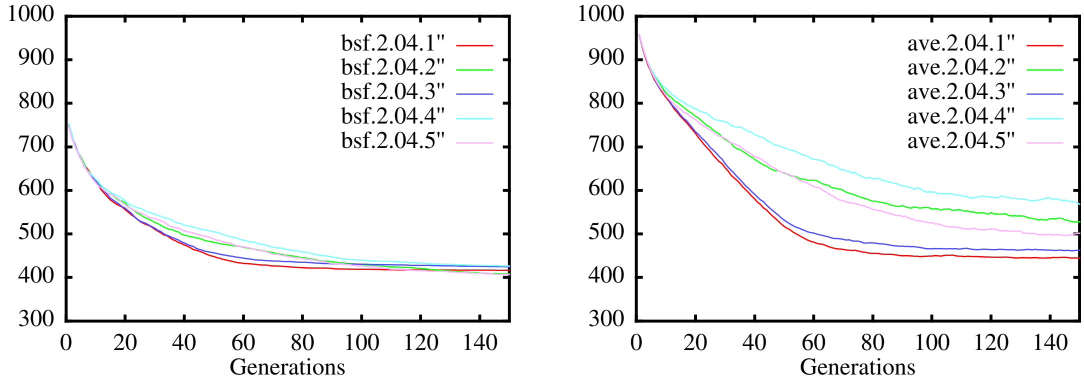
3 Points-5 , 2 Points-2 , 1 Point-3.

GRAPH 2 - BSF and AVE, 10 City, Crossover-8 , Mutation-1,2,3,4,5



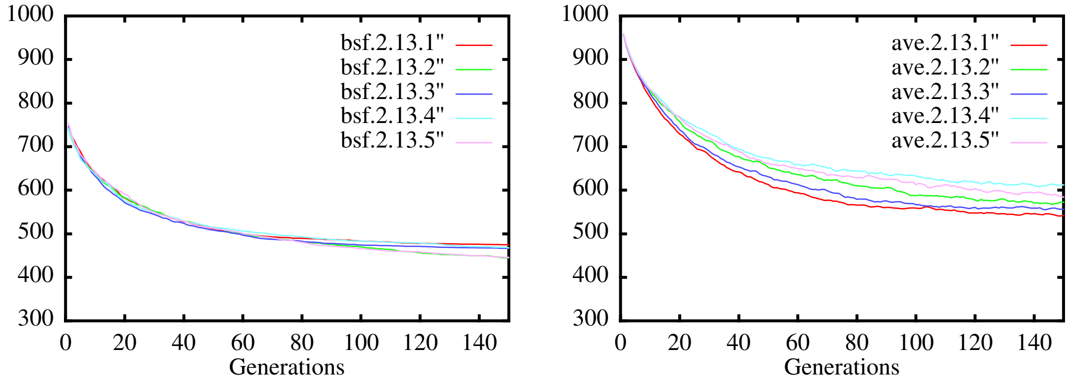
3 Points-2 , 2 Points-4 , 1 Point-5.

GRAPH 3 - BSF and AVE, 20 City, Crossover-4, Mutation-1,2,3,4,5



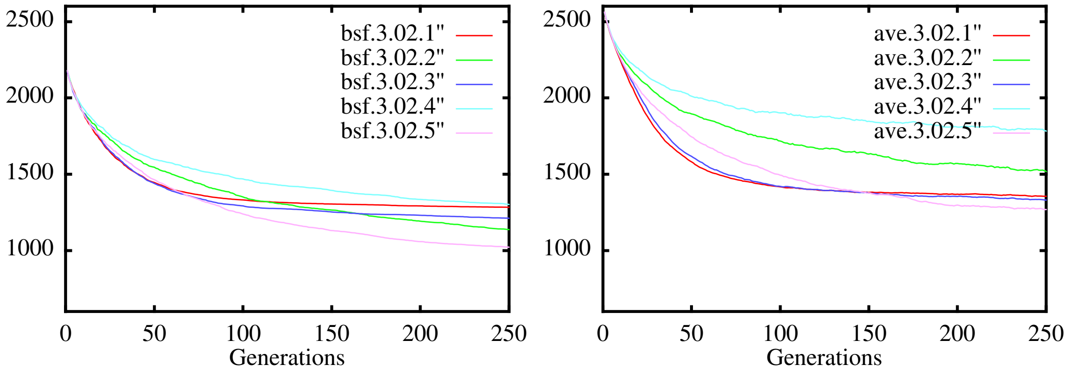
3 Points-5 , 2 Points-2 , 1 Point-1.

GRAPH 4 - BSF and AVE, 20 City, Crossover-13, Mutation-1,2,3,4,5



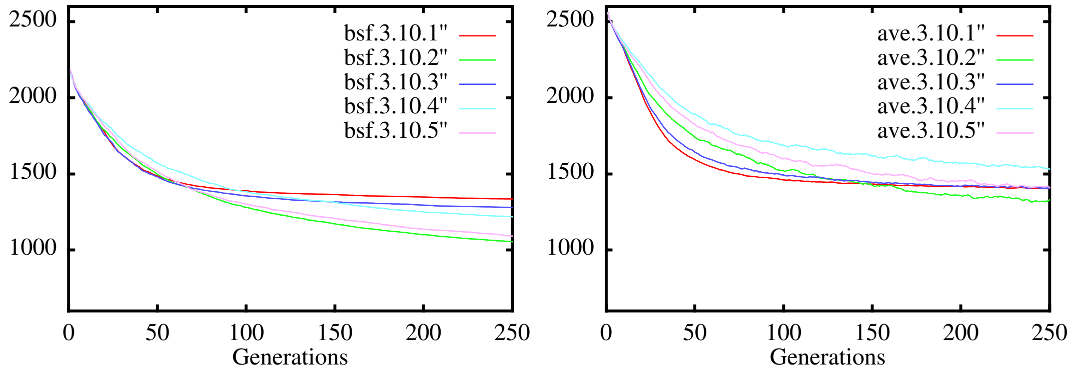
3 Points-2 , 2 Points-5 , 1 Point-3.

GRAPH 5 - BSF and AVE, 50 City, Crossover-2, Mutation-1,2,3,4,5



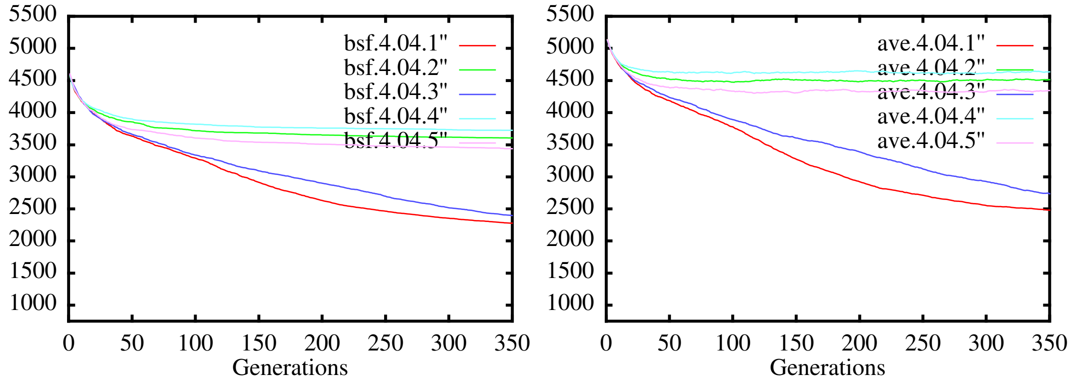
3 Points-5 , 2 Points-2 , 1 Point-3.

GRAPH 6 - BSF and AVE, 50 City, Crossover-10, Mutation-1,2,3,4,5



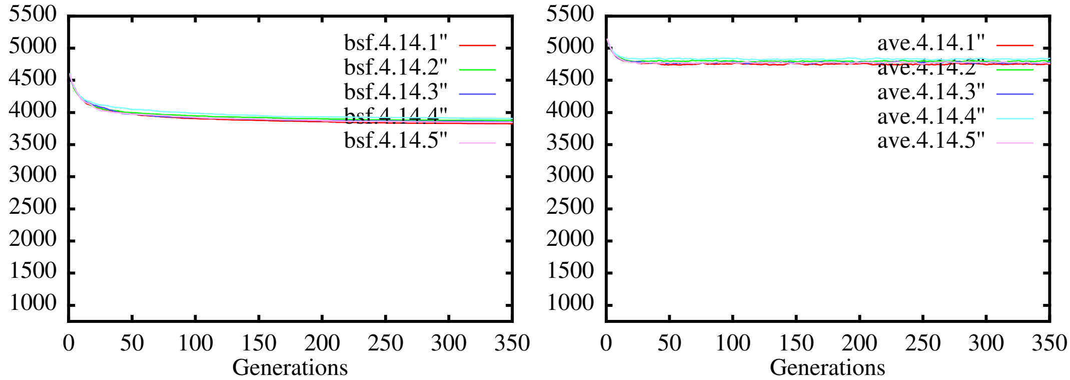
3 Points-2 , 2 Points-5 , 1 Point-4.

GRAPH 7 - BSF and AVE, 100 City, Crossover-4, Mutation-1,2,3,4,5



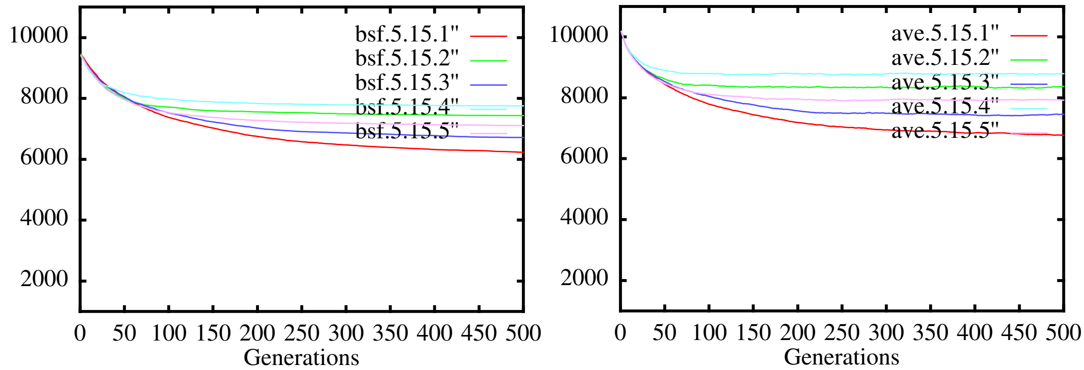
3 Points-1 , 2 Points-3 , 1 Point-5.

GRAPH 8 - BSF and AVE, 100 City, Crossover-14, Mutation-1,2,3,4,5



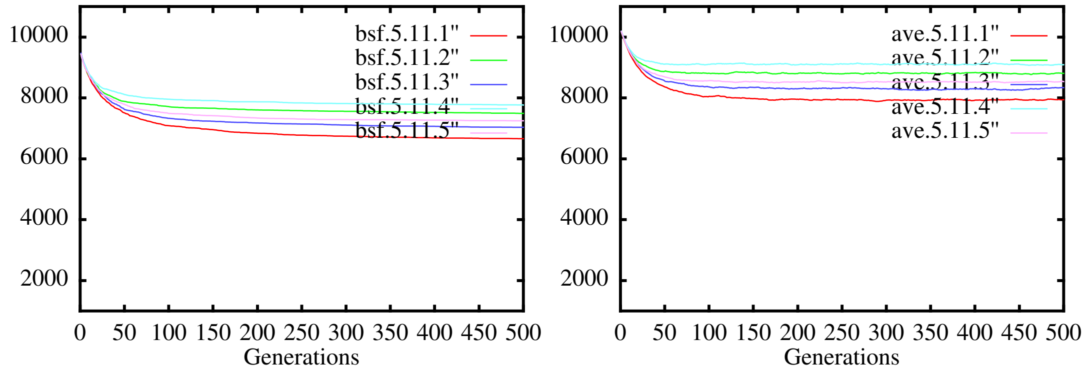
3 Points-1 , 2 Points-5 , 1 Point-3.

GRAPH 9 - BSF and AVE, 200 City, Crossover-1, Mutation-1,2,3,4,5



3 Points-1 , 2 Points-3 , 1 Point-5.

GRAPH 10 - BSF and AVE, 200 City, Crossover-11, Mutation-1,2,3,4,5



3 Points-1 , 2 Points-3 , 1 Point-5.

Appendix C- MCGA Code Structure

C.1 Main Structure

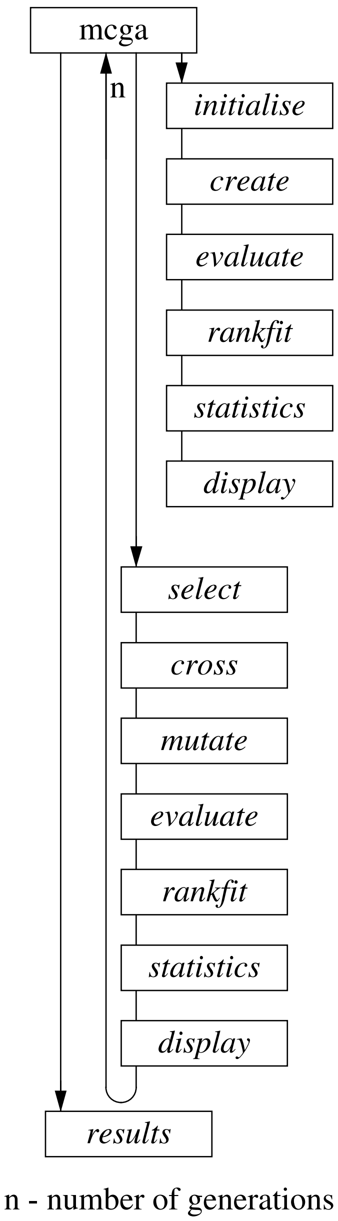


Figure C.1 - *mcga.cc*

The MCGA structure is generally based on the simple GA. The code uses a class called *gastring* containing data and some functions for setting and getting variables in strings. MCGA also utilises two data stores called *constant* and *data*. *Constant* holds all constants for the system and *data* holds the arrays, variables and structure definitions. The system uses a normal and a Pareto Population. The normal population is a fixed array of type *gastring* and the Pareto Population is constructed as a linked list. There is also a second array the same size as the normal population which is used for copying purposes and a further linked list used to store the rank one individuals of the current population. Figure C.1 shows the basic algorithm. The system first goes through initialisation and evaluation stage and then enters the MCGA's main loop. When finished the MCGA outputs the results of the search.

Any names in boxes which are italicised indicate the name of a further lower level of file decomposition. Each of these lower level blocks are explained on the following pages.

C.2 Initialise

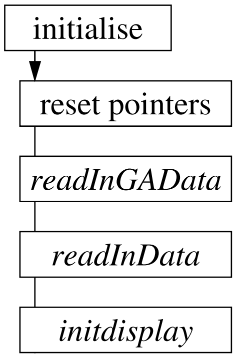
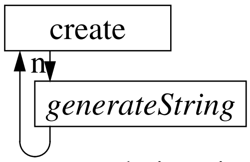


Figure C.2 -
initialise.cc

Initialise resets the GA's pointers and then proceeds by reading in the values for the GA(e.g. Pop Size, Probability of Crossover and Mutation etc.) from *gadata* in the *user* directory. The program then reads in any problem specific data from the *problemData* file. The *readInData* code must be inserted by the user depending on the application. The data read in is then displayed to the screen.

C.3 Create

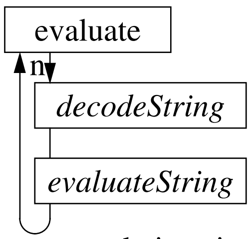


n - population size

Figure C.3 -
create.cc

This procedure randomly generates the initial string configurations for every individual.

C.4 Evaluate



n - population size

Figure C.4 -
evaluate.cc

Every member of the population is first decoded then evaluated on all criteria. This evaluation can be done within the *evaluateString* routine or through a separate evaluation routine of external program. Both of these routines are problem dependent so must be edited by the user.

C.5 RankFit

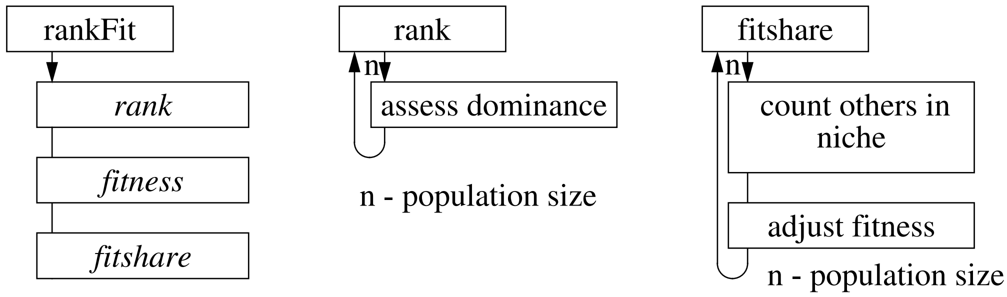


Figure C.5 - *rankFit.cc, rank.cc, fitshare.cc*

The rank, fitness and fitshare procedures convert the individual criteria values into an adjusted rank using dominance comparisons. This rank is then changed into a fitness value by scaling relative to the total population size. The fitness of each string is then adjusted based on the number of other strings within that string's niche. The greater the number the greater the penalty on the string.

C.6 Statistics

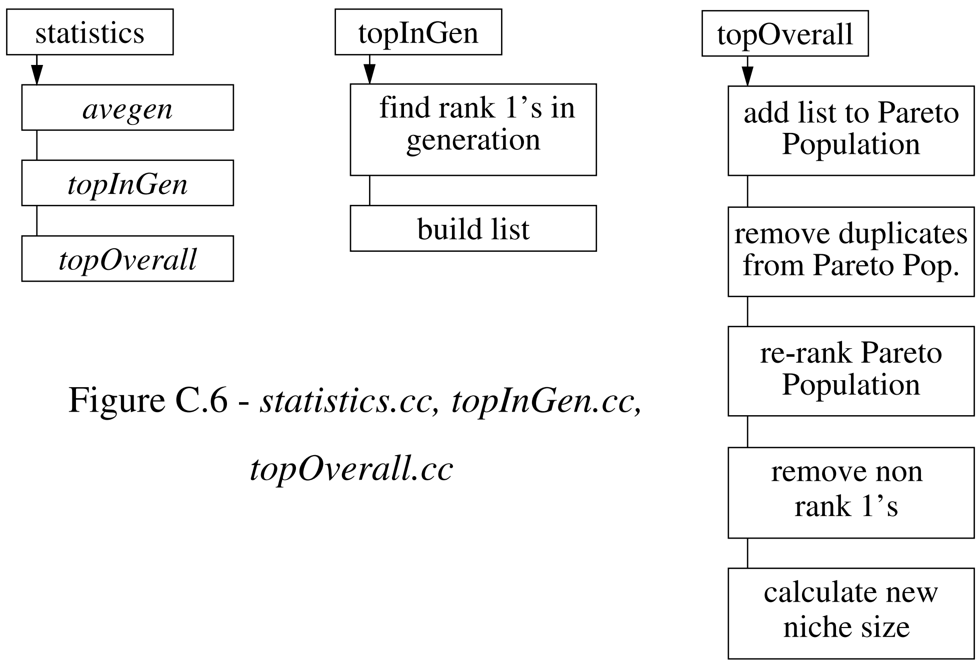


Figure C.6 - *statistics.cc, topInGen.cc, topOverall.cc*

The statistics routines perform functions to find population averages and to compile the Pareto Population. The first routine finds the averages for all criteria in the population. The next routine moves through the current generation picking out all rank one solutions and compiles them into a list. In the following routine this list is sorted to remove duplicates and then appended to the end of the Pareto Population. The Pareto Population is then re-ranked and all non-rank one individuals are removed. Finally the new niche size is calculated from the Pareto Population.

C.7 Display

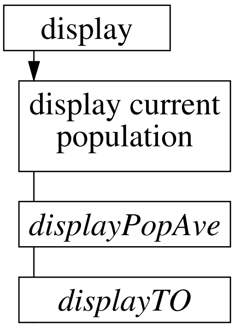


Figure C.7 - *display.cc*

The display routine is only invoked if the display flag is set at the start of the run. The routine first displays the current population and then population averages. It then displays the whole of the Pareto Population with overall averages.

C.8 Select

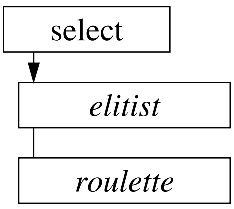


Figure C.8 - *initialise.cc*

The selection procedure is a two stage strategy utilising two routines. The first passes all rank one individuals currently in the population through into the mating pool. The final routine fills the rest of the mating pool using roulette wheel selection.

C.9 Cross

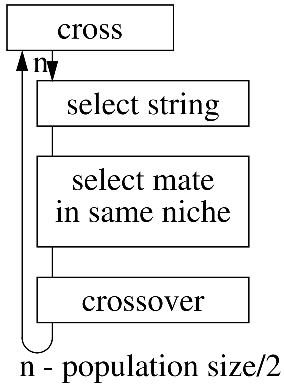


Figure C.9 - *cross.cc*

The system performs crossover by randomly selecting one individual from the mating pool, finding all the strings within its niche and then randomly selecting one of these. Crossover is then performed which may vary depending on the problem being solved.

C.10 Mutate

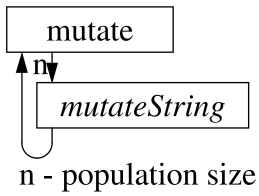


Figure C.10 - *mutate.cc*

The system steps though each string determining randomly if mutation occurs. If it does then a problem specific mutation operator is applied.

C.11 Results

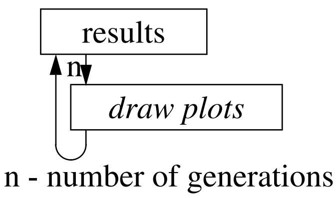


Figure C.11 - *results.cc*

Outputs results to screen in the form of gnuplot graphs showing the current population or the Pareto Population.

Appendix D- Scheduler Information

D.1 MCGA Scheduler Data File (Test Example)

```
TITLE: EXAMPLE 1 DATA FILE


Number of Machines
4
Number of Operations
6
Number of Jobs
20
Number of Stages(max)
5
Number of Dependencies(max)
3
Number of N/A(max)
4
Job Splitting(0-No, 1-Yes)
1


Machines Op Eff Table
1.0     0       0       0.5     0       0
0       0.5     0       0       0.8     0
0       1.0     0.8     0       0       0.8
0.5     0.8     0       0.3     0       1.0


List Of N/A Time
Machine--NumOfN/A--Start/Finish
0   2   50 60   100 110
1   1   20 25
2   4   10 15   25 37   50 55   100 120
2   2   20 35   60 64
```

Machine/Op Setup Table

0	1	2	3	4	5
1	0	1	2	3	4
2	1	0	1	2	3
3	2	1	0	1	2
4	3	2	1	0	1
5	4	3	2	1	0

Jobs/Stages

Job Stages--Op/Time

0	2	2 4	4 10			
1	4	1 6	2 3	5 8	0 3	
2	3	4 5	1 3	2 8		
3	4	0 7	2 7	4 3	5 1	
4	5	1 4	0 5	1 7	3 9	4 6
5	1	3 4				
6	2	2 8	5 6			
7	4	1 6	2 4	5 6	4 5	
8	5	2 6	4 8	2 4	3 4	0 5
9	3	5 4	2 9	1 4		
10	3	0 9	3 7	4 6		
11	2	1 6	2 8			
12	1	0 9				
13	1	2 8				
14	4	5 9	2 6	3 4	2 6	
15	3	2 4	3 5	5 3		
16	4	0 5	1 6	4 5	1 3	
17	2	2 6	1 8			
18	5	0 4	2 8	1 3	0 5	3 4
19	1	1 7				

Job Data

Job--Early Start--Num Deps--Deps--Deadline--Penalty

0	10	3	6 8 9	100	1
1	0	0		50	1
2	5	0		90	1
3	0	2	7 8	100	1
4	0	1	16	0	0
5	0	0		20	1

6	40	0		0	0
7	25	0		0	0
8	60	0		100	1
9	45	1	17	0	0
10	70	2	5 6	50	1
11	0	0		100	1
12	0	0		0	0
13	20	0		0	0
14	0	1	15	0	0
15	0	0		0	0
16	45	2	11 12	0	0
17	0	0		120	1
18	10	0		50	1
19	30	0		80	1

D.2 MCGA Scheduler Plan File

Notation:

```
15-> | M0: I            | M1: 17   1   15 | M2: N/A           | M3: S        2
 1   | 2   3            | 2   4    5   6  | 2    7            | 2   8        6
```

1 - Clock

2 - Machine Separator

3 - Current State : Idle

4 - Current State : Job 17

5 - Current State : Stage 1

6 - Time Left In Current State

7 - Machine Not Available (Maintenance/Breakdown)

8 - Current State : Set Up

Plans are terminated by a data table containing various statistics.

Schedule Plan for Minimum Makespan

1->	M0:	12	0	9	M1:	I			M2:	17	0	7	M3:	1	0	7
2->	M0:	12	0	8	M1:	I			M2:	17	0	6	M3:	1	0	6
3->	M0:	12	0	7	M1:	I			M2:	17	0	5	M3:	1	0	5
4->	M0:	12	0	6	M1:	I			M2:	17	0	4	M3:	1	0	4
5->	M0:	12	0	5	M1:	2	0	6	M2:	17	0	3	M3:	1	0	3
6->	M0:	12	0	4	M1:	2	0	5	M2:	17	0	2	M3:	1	0	2
7->	M0:	12	0	3	M1:	2	0	4	M2:	17	0	1	M3:	1	0	1

8->	M0:	12	0	2	M1:	2	0	3	M2:	1	1	3	M3:	11	0	7
9->	M0:	12	0	1	M1:	2	0	2	M2:	1	1	2	M3:	11	0	6
10->	M0:	18	0	4	M1:	2	0	1	M2:	N/A			M3:	11	0	5
11->	M0:	18	0	3	M1:	S		3	M2:	N/A			M3:	11	0	4
12->	M0:	18	0	2	M1:	S		2	M2:	N/A			M3:	11	0	3
13->	M0:	18	0	1	M1:	S		1	M2:	N/A			M3:	11	0	2
14->	M0:	I			M1:	17	1	16	M2:	N/A			M3:	11	0	1
15->	M0:	I			M1:	17	1	15	M2:	N/A			M3:	S		2
16->	M0:	I			M1:	17	1	14	M2:	1	1	1	M3:	S		1
17->	M0:	I			M1:	17	1	13	M2:	18	1	10	M3:	S		2
18->	M0:	I			M1:	17	1	12	M2:	18	1	9	M3:	S		1
19->	M0:	I			M1:	17	1	11	M2:	18	1	8	M3:	1	2	8
20->	M0:	I			M1:	N/A			M2:	18	1	7	M3:	N/A		
21->	M0:	I			M1:	N/A			M2:	18	1	6	M3:	N/A		
22->	M0:	I			M1:	N/A			M2:	18	1	5	M3:	N/A		
23->	M0:	I			M1:	N/A			M2:	18	1	4	M3:	N/A		
24->	M0:	I			M1:	N/A			M2:	18	1	3	M3:	N/A		
25->	M0:	I			M1:	N/A			M2:	N/A			M3:	N/A		
26->	M0:	I			M1:	17	1	10	M2:	N/A			M3:	N/A		
27->	M0:	I			M1:	17	1	9	M2:	N/A			M3:	N/A		
28->	M0:	I			M1:	17	1	8	M2:	N/A			M3:	N/A		
29->	M0:	I			M1:	17	1	7	M2:	N/A			M3:	N/A		
30->	M0:	I			M1:	17	1	6	M2:	N/A			M3:	N/A		
31->	M0:	I			M1:	17	1	5	M2:	N/A			M3:	N/A		
32->	M0:	I			M1:	17	1	4	M2:	N/A			M3:	N/A		
33->	M0:	I			M1:	17	1	3	M2:	N/A			M3:	N/A		
34->	M0:	I			M1:	17	1	2	M2:	N/A			M3:	N/A		
35->	M0:	I			M1:	17	1	1	M2:	N/A			M3:	N/A		
36->	M0:	I			M1:	7	0	12	M2:	N/A			M3:	1	2	7
37->	M0:	I			M1:	7	0	11	M2:	N/A			M3:	1	2	6
38->	M0:	I			M1:	7	0	10	M2:	18	1	2	M3:	1	2	5
39->	M0:	I			M1:	7	0	9	M2:	18	1	1	M3:	1	2	4
40->	M0:	I			M1:	7	0	8	M2:	11	1	10	M3:	1	2	3
41->	M0:	I			M1:	7	0	7	M2:	11	1	9	M3:	1	2	2
42->	M0:	I			M1:	7	0	6	M2:	11	1	8	M3:	1	2	1
43->	M0:	1	3	3	M1:	7	0	5	M2:	11	1	7	M3:	S		2
44->	M0:	1	3	2	M1:	7	0	4	M2:	11	1	6	M3:	S		1
45->	M0:	1	3	1	M1:	7	0	3	M2:	11	1	5	M3:	5	0	13
46->	M0:	I			M1:	7	0	2	M2:	11	1	4	M3:	5	0	12
47->	M0:	I			M1:	7	0	1	M2:	11	1	3	M3:	5	0	11
48->	M0:	I			M1:	18	2	6	M2:	11	1	2	M3:	5	0	10
49->	M0:	I			M1:	18	2	5	M2:	11	1	1	M3:	5	0	9
50->	M0:	N/A			M1:	18	2	4	M2:	N/A			M3:	5	0	8
51->	M0:	N/A			M1:	18	2	3	M2:	N/A			M3:	5	0	7
52->	M0:	N/A			M1:	18	2	2	M2:	N/A			M3:	5	0	6
53->	M0:	N/A			M1:	18	2	1	M2:	N/A			M3:	5	0	5
54->	M0:	N/A			M1:	19	0	14	M2:	N/A			M3:	5	0	4
55->	M0:	N/A			M1:	19	0	13	M2:	N/A			M3:	5	0	3
56->	M0:	N/A			M1:	19	0	12	M2:	7	1	5	M3:	5	0	2
57->	M0:	N/A			M1:	19	0	11	M2:	7	1	4	M3:	5	0	1
58->	M0:	N/A			M1:	19	0	10	M2:	7	1	3	M3:	S		3
59->	M0:	N/A			M1:	19	0	9	M2:	7	1	2	M3:	S		2
60->	M0:	N/A			M1:	19	0	8	M2:	7	1	1	M3:	N/A		
61->	M0:	18	3	5	M1:	19	0	7	M2:	8	0	7	M3:	N/A		
62->	M0:	18	3	4	M1:	19	0	6	M2:	8	0	6	M3:	N/A		
63->	M0:	18	3	3	M1:	19	0	5	M2:	8	0	5	M3:	N/A		
64->	M0:	18	3	2	M1:	19	0	4	M2:	8	0	4	M3:	N/A		
65->	M0:	18	3	1	M1:	19	0	3	M2:	8	0	3	M3:	S		1

66->	M0:	S		3	M1:	19	0	2	M2:	8	0	2	M3:	16	0	10
67->	M0:	S		2	M1:	19	0	1	M2:	8	0	1	M3:	16	0	9
68->	M0:	S		1	M1:	S		3	M2:	15	0	5	M3:	16	0	8
69->	M0:	18	4	8	M1:	S		2	M2:	15	0	4	M3:	16	0	7
70->	M0:	18	4	7	M1:	S		1	M2:	15	0	3	M3:	16	0	6
71->	M0:	18	4	6	M1:	8	1	10	M2:	15	0	2	M3:	16	0	5
72->	M0:	18	4	5	M1:	8	1	9	M2:	15	0	1	M3:	16	0	4
73->	M0:	18	4	4	M1:	8	1	8	M2:	6	0	10	M3:	16	0	3
74->	M0:	18	4	3	M1:	8	1	7	M2:	6	0	9	M3:	16	0	2
75->	M0:	18	4	2	M1:	8	1	6	M2:	6	0	8	M3:	16	0	1
76->	M0:	18	4	1	M1:	8	1	5	M2:	6	0	7	M3:	S		5
77->	M0:	15	1	10	M1:	8	1	4	M2:	6	0	6	M3:	S		4
78->	M0:	15	1	9	M1:	8	1	3	M2:	6	0	5	M3:	S		3
79->	M0:	15	1	8	M1:	8	1	2	M2:	6	0	4	M3:	S		2
80->	M0:	15	1	7	M1:	8	1	1	M2:	6	0	3	M3:	S		1
81->	M0:	15	1	6	M1:	I			M2:	6	0	2	M3:	7	2	6
82->	M0:	15	1	5	M1:	I			M2:	6	0	1	M3:	7	2	5
83->	M0:	15	1	4	M1:	I			M2:	8	2	5	M3:	7	2	4
84->	M0:	15	1	3	M1:	I			M2:	8	2	4	M3:	7	2	3
85->	M0:	15	1	2	M1:	I			M2:	8	2	3	M3:	7	2	2
86->	M0:	15	1	1	M1:	I			M2:	8	2	2	M3:	7	2	1
87->	M0:	I			M1:	7	3	6	M2:	8	2	1	M3:	9	0	4
88->	M0:	8	3	8	M1:	7	3	5	M2:	13	0	10	M3:	9	0	3
89->	M0:	8	3	7	M1:	7	3	4	M2:	13	0	9	M3:	9	0	2
90->	M0:	8	3	6	M1:	7	3	3	M2:	13	0	8	M3:	9	0	1
91->	M0:	8	3	5	M1:	7	3	2	M2:	13	0	7	M3:	S		4
92->	M0:	8	3	4	M1:	7	3	1	M2:	13	0	6	M3:	S		3
93->	M0:	8	3	3	M1:	I			M2:	13	0	5	M3:	S		2
94->	M0:	8	3	2	M1:	I			M2:	13	0	4	M3:	S		1
95->	M0:	8	3	1	M1:	I			M2:	13	0	3	M3:	16	1	7
96->	M0:	I			M1:	I			M2:	13	0	2	M3:	16	1	6
97->	M0:	I			M1:	I			M2:	13	0	1	M3:	16	1	5
98->	M0:	I			M1:	I			M2:	9	1	11	M3:	16	1	4
99->	M0:	I			M1:	I			M2:	9	1	10	M3:	16	1	3
100->	M0:	N/A			M1:	I			M2:	N/A			M3:	16	1	2
101->	M0:	N/A			M1:	I			M2:	N/A			M3:	16	1	1
102->	M0:	N/A			M1:	16	2	6	M2:	N/A			M3:	S		1
103->	M0:	N/A			M1:	16	2	5	M2:	N/A			M3:	8	4	10
104->	M0:	N/A			M1:	16	2	4	M2:	N/A			M3:	8	4	9
105->	M0:	N/A			M1:	16	2	3	M2:	N/A			M3:	8	4	8
106->	M0:	N/A			M1:	16	2	2	M2:	N/A			M3:	8	4	7
107->	M0:	N/A			M1:	16	2	1	M2:	N/A			M3:	8	4	6
108->	M0:	N/A			M1:	S		3	M2:	N/A			M3:	8	4	5
109->	M0:	N/A			M1:	S		2	M2:	N/A			M3:	8	4	4
110->	M0:	N/A			M1:	S		1	M2:	N/A			M3:	8	4	3
111->	M0:	I			M1:	16	3	6	M2:	N/A			M3:	8	4	2
112->	M0:	I			M1:	16	3	5	M2:	N/A			M3:	8	4	1
113->	M0:	S		3	M1:	16	3	4	M2:	N/A			M3:	S		5
114->	M0:	S		2	M1:	16	3	3	M2:	N/A			M3:	S		4
115->	M0:	S		1	M1:	16	3	2	M2:	N/A			M3:	S		3
116->	M0:	3	0	7	M1:	16	3	1	M2:	N/A			M3:	S		2
117->	M0:	3	0	6	M1:	4	0	8	M2:	N/A			M3:	S		1
118->	M0:	3	0	5	M1:	4	0	7	M2:	N/A			M3:	6	1	6
119->	M0:	3	0	4	M1:	4	0	6	M2:	N/A			M3:	6	1	5
120->	M0:	3	0	3	M1:	4	0	5	M2:	N/A			M3:	6	1	4
121->	M0:	3	0	2	M1:	4	0	4	M2:	9	1	9	M3:	6	1	3
122->	M0:	3	0	1	M1:	4	0	3	M2:	9	1	8	M3:	6	1	2
123->	M0:	I			M1:	4	0	2	M2:	9	1	7	M3:	6	1	1

124->	M0:	10	0	9	M1:	4	0	1	M2:	9	1	6	M3:	15	2	3
125->	M0:	10	0	8	M1:	I			M2:	9	1	5	M3:	15	2	2
126->	M0:	10	0	7	M1:	I			M2:	9	1	4	M3:	15	2	1
127->	M0:	10	0	6	M1:	I			M2:	9	1	3	M3:	14	0	9
128->	M0:	10	0	5	M1:	I			M2:	9	1	2	M3:	14	0	8
129->	M0:	10	0	4	M1:	I			M2:	9	1	1	M3:	14	0	7
130->	M0:	10	0	3	M1:	9	2	8	M2:	3	1	8	M3:	14	0	6
131->	M0:	10	0	2	M1:	9	2	7	M2:	3	1	7	M3:	14	0	5
132->	M0:	10	0	1	M1:	9	2	6	M2:	3	1	6	M3:	14	0	4
133->	M0:	4	1	5	M1:	9	2	5	M2:	3	1	5	M3:	14	0	3
134->	M0:	4	1	4	M1:	9	2	4	M2:	3	1	4	M3:	14	0	2
135->	M0:	4	1	3	M1:	9	2	3	M2:	3	1	3	M3:	14	0	1
136->	M0:	4	1	2	M1:	9	2	2	M2:	3	1	2	M3:	S		4
137->	M0:	4	1	1	M1:	9	2	1	M2:	3	1	1	M3:	S		3
138->	M0:	S		3	M1:	S		3	M2:	0	0	5	M3:	S		2
139->	M0:	S		2	M1:	S		2	M2:	0	0	4	M3:	S		1
140->	M0:	S		1	M1:	S		1	M2:	0	0	3	M3:	4	2	8
141->	M0:	10	1	14	M1:	3	2	3	M2:	0	0	2	M3:	4	2	7
142->	M0:	10	1	13	M1:	3	2	2	M2:	0	0	1	M3:	4	2	6
143->	M0:	10	1	12	M1:	3	2	1	M2:	14	1	7	M3:	4	2	5
144->	M0:	10	1	11	M1:	0	1	12	M2:	14	1	6	M3:	4	2	4
145->	M0:	10	1	10	M1:	0	1	11	M2:	14	1	5	M3:	4	2	3
146->	M0:	10	1	9	M1:	0	1	10	M2:	14	1	4	M3:	4	2	2
147->	M0:	10	1	8	M1:	0	1	9	M2:	14	1	3	M3:	4	2	1
148->	M0:	10	1	7	M1:	0	1	8	M2:	14	1	2	M3:	2	1	3
149->	M0:	10	1	6	M1:	0	1	7	M2:	14	1	1	M3:	2	1	2
150->	M0:	10	1	5	M1:	0	1	6	M2:	I			M3:	2	1	1
151->	M0:	10	1	4	M1:	0	1	5	M2:	2	2	10	M3:	S		2
152->	M0:	10	1	3	M1:	0	1	4	M2:	2	2	9	M3:	S		1
153->	M0:	10	1	2	M1:	0	1	3	M2:	2	2	8	M3:	14	2	13
154->	M0:	10	1	1	M1:	0	1	2	M2:	2	2	7	M3:	14	2	12
155->	M0:	4	3	18	M1:	0	1	1	M2:	2	2	6	M3:	14	2	11
156->	M0:	4	3	17	M1:	10	2	7	M2:	2	2	5	M3:	14	2	10
157->	M0:	4	3	16	M1:	10	2	6	M2:	2	2	4	M3:	14	2	9
158->	M0:	4	3	15	M1:	10	2	5	M2:	2	2	3	M3:	14	2	8
159->	M0:	4	3	14	M1:	10	2	4	M2:	2	2	2	M3:	14	2	7
160->	M0:	4	3	13	M1:	10	2	3	M2:	2	2	1	M3:	14	2	6
161->	M0:	4	3	12	M1:	10	2	2	M2:	I			M3:	14	2	5
162->	M0:	4	3	11	M1:	10	2	1	M2:	I			M3:	14	2	4
163->	M0:	4	3	10	M1:	I			M2:	I			M3:	14	2	3
164->	M0:	4	3	9	M1:	I			M2:	I			M3:	14	2	2
165->	M0:	4	3	8	M1:	I			M2:	I			M3:	14	2	1
166->	M0:	4	3	7	M1:	I			M2:	14	3	7	M3:	S		2
167->	M0:	4	3	6	M1:	I			M2:	14	3	6	M3:	S		1
168->	M0:	4	3	5	M1:	I			M2:	14	3	5	M3:	3	3	1
169->	M0:	4	3	4	M1:	I			M2:	14	3	4	M3:	I		
170->	M0:	4	3	3	M1:	I			M2:	14	3	3	M3:	I		
171->	M0:	4	3	2	M1:	I			M2:	14	3	2	M3:	I		
172->	M0:	4	3	1	M1:	I			M2:	14	3	1	M3:	I		
173->	M0:	I			M1:	4	4	7	M2:	I			M3:	I		
174->	M0:	I			M1:	4	4	6	M2:	I			M3:	I		
175->	M0:	I			M1:	4	4	5	M2:	I			M3:	I		
176->	M0:	I			M1:	4	4	4	M2:	I			M3:	I		
177->	M0:	I			M1:	4	4	3	M2:	I			M3:	I		
178->	M0:	I			M1:	4	4	2	M2:	I			M3:	I		
179->	M0:	I			M1:	4	4	1	M2:	I			M3:	I		
180->	M0:	I			M1:	I			M2:	I			M3:	I		

M0 Util :55.8659%   M1 Util :70.9497%   M2 Util :67.0391%   M3 Util :64.2458%

Job	0	Start:	138	Finish:	156	Deadline:	100	Penalty:	3136
Job	1	Start:	1	Finish:	46	Deadline:	50	Penalty:	0
Job	2	Start:	5	Finish:	161	Deadline:	90	Penalty:	5041
Job	3	Start:	116	Finish:	169	Deadline:	100	Penalty:	4761
Job	4	Start:	117	Finish:	180	Deadline:	0	Penalty:	0
Job	5	Start:	45	Finish:	58	Deadline:	20	Penalty:	1444
Job	6	Start:	73	Finish:	124	Deadline:	0	Penalty:	0
Job	7	Start:	36	Finish:	93	Deadline:	0	Penalty:	0
Job	8	Start:	61	Finish:	113	Deadline:	100	Penalty:	169
Job	9	Start:	87	Finish:	138	Deadline:	0	Penalty:	0
Job	10	Start:	124	Finish:	163	Deadline:	50	Penalty:	12769
Job	11	Start:	8	Finish:	56	Deadline:	100	Penalty:	0
Job	12	Start:	1	Finish:	10	Deadline:	0	Penalty:	0
Job	13	Start:	88	Finish:	98	Deadline:	0	Penalty:	0
Job	14	Start:	127	Finish:	173	Deadline:	0	Penalty:	0
Job	15	Start:	68	Finish:	127	Deadline:	0	Penalty:	0
Job	16	Start:	66	Finish:	117	Deadline:	0	Penalty:	0
Job	17	Start:	1	Finish:	36	Deadline:	120	Penalty:	0
Job	18	Start:	10	Finish:	77	Deadline:	50	Penalty:	729
Job	19	Start:	54	Finish:	68	Deadline:	80	Penalty:	0

TOTAL TIME: 179   PENALTY: 28049   AVE JOB TIME: 46.85

Schedule Plan for Minimum Average Job Time

1->	M0:	5	0	8	M1:	1	0	12	M2:	17	0	7	M3:	11	0	7
2->	M0:	5	0	7	M1:	1	0	11	M2:	17	0	6	M3:	11	0	6
3->	M0:	5	0	6	M1:	1	0	10	M2:	17	0	5	M3:	11	0	5
4->	M0:	5	0	5	M1:	1	0	9	M2:	17	0	4	M3:	11	0	4
5->	M0:	5	0	4	M1:	1	0	8	M2:	17	0	3	M3:	11	0	3
6->	M0:	5	0	3	M1:	1	0	7	M2:	17	0	2	M3:	11	0	2
7->	M0:	5	0	2	M1:	1	0	6	M2:	17	0	1	M3:	11	0	1
8->	M0:	5	0	1	M1:	1	0	5	M2:	11	1	10	M3:	17	1	10
9->	M0:	S		3	M1:	1	0	4	M2:	11	1	9	M3:	17	1	9
10->	M0:	S		2	M1:	1	0	3	M2:	N/A			M3:	17	1	8
11->	M0:	S		1	M1:	1	0	2	M2:	N/A			M3:	17	1	7
12->	M0:	12	0	9	M1:	1	0	1	M2:	N/A			M3:	17	1	6
13->	M0:	12	0	8	M1:	S		3	M2:	N/A			M3:	17	1	5
14->	M0:	12	0	7	M1:	S		2	M2:	N/A			M3:	17	1	4
15->	M0:	12	0	6	M1:	S		1	M2:	N/A			M3:	17	1	3
16->	M0:	12	0	5	M1:	2	0	6	M2:	11	1	8	M3:	17	1	2
17->	M0:	12	0	4	M1:	2	0	5	M2:	11	1	7	M3:	17	1	1
18->	M0:	12	0	3	M1:	2	0	4	M2:	11	1	6	M3:	S		1
19->	M0:	12	0	2	M1:	2	0	3	M2:	11	1	5	M3:	18	0	8
20->	M0:	12	0	1	M1:	N/A			M2:	11	1	4	M3:	N/A		
21->	M0:	I			M1:	N/A			M2:	11	1	3	M3:	N/A		
22->	M0:	I			M1:	N/A			M2:	11	1	2	M3:	N/A		
23->	M0:	I			M1:	N/A			M2:	11	1	1	M3:	N/A		
24->	M0:	I			M1:	N/A			M2:	1	1	3	M3:	N/A		
25->	M0:	I			M1:	N/A			M2:	N/A			M3:	N/A		
26->	M0:	I			M1:	2	0	2	M2:	N/A			M3:	N/A		
27->	M0:	I			M1:	2	0	1	M2:	N/A			M3:	N/A		
28->	M0:	I			M1:	S		3	M2:	N/A			M3:	N/A		
29->	M0:	I			M1:	S		2	M2:	N/A			M3:	N/A		
30->	M0:	I			M1:	S		1	M2:	N/A			M3:	N/A		

31->	M0:	I			M1:	2	1	6	M2:	N/A			M3:	N/A		
32->	M0:	I			M1:	2	1	5	M2:	N/A			M3:	N/A		
33->	M0:	I			M1:	2	1	4	M2:	N/A			M3:	N/A		
34->	M0:	I			M1:	2	1	3	M2:	N/A			M3:	N/A		
35->	M0:	I			M1:	2	1	2	M2:	N/A			M3:	N/A		
36->	M0:	I			M1:	2	1	1	M2:	N/A			M3:	18	0	7
37->	M0:	I			M1:	19	0	14	M2:	N/A			M3:	18	0	6
38->	M0:	I			M1:	19	0	13	M2:	1	1	2	M3:	18	0	5
39->	M0:	I			M1:	19	0	12	M2:	1	1	1	M3:	18	0	4
40->	M0:	I			M1:	19	0	11	M2:	2	2	10	M3:	18	0	3
41->	M0:	I			M1:	19	0	10	M2:	2	2	9	M3:	18	0	2
42->	M0:	I			M1:	19	0	9	M2:	2	2	8	M3:	18	0	1
43->	M0:	I			M1:	19	0	8	M2:	2	2	7	M3:	S		5
44->	M0:	I			M1:	19	0	7	M2:	2	2	6	M3:	S		4
45->	M0:	16	0	5	M1:	19	0	6	M2:	2	2	5	M3:	S		3
46->	M0:	16	0	4	M1:	19	0	5	M2:	2	2	4	M3:	S		2
47->	M0:	16	0	3	M1:	19	0	4	M2:	2	2	3	M3:	S		1
48->	M0:	16	0	2	M1:	19	0	3	M2:	2	2	2	M3:	1	2	8
49->	M0:	16	0	1	M1:	19	0	2	M2:	2	2	1	M3:	1	2	7
50->	M0:	N/A			M1:	19	0	1	M2:	N/A			M3:	1	2	6
51->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	5
52->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	4
53->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	3
54->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	2
55->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	1
56->	M0:	N/A			M1:	I			M2:	18	1	10	M3:	S		4
57->	M0:	N/A			M1:	I			M2:	18	1	9	M3:	S		3
58->	M0:	N/A			M1:	I			M2:	18	1	8	M3:	S		2
59->	M0:	N/A			M1:	I			M2:	18	1	7	M3:	S		1
60->	M0:	N/A			M1:	I			M2:	18	1	6	M3:	N/A		
61->	M0:	1	3	3	M1:	I			M2:	18	1	5	M3:	N/A		
62->	M0:	1	3	2	M1:	I			M2:	18	1	4	M3:	N/A		
63->	M0:	1	3	1	M1:	I			M2:	18	1	3	M3:	N/A		
64->	M0:	I			M1:	I			M2:	18	1	2	M3:	N/A		
65->	M0:	I			M1:	I			M2:	18	1	1	M3:	7	0	7
66->	M0:	I			M1:	18	2	6	M2:	S		1	M3:	7	0	6
67->	M0:	I			M1:	18	2	5	M2:	16	1	6	M3:	7	0	5
68->	M0:	I			M1:	18	2	4	M2:	16	1	5	M3:	7	0	4
69->	M0:	I			M1:	18	2	3	M2:	16	1	4	M3:	7	0	3
70->	M0:	I			M1:	18	2	2	M2:	16	1	3	M3:	7	0	2
71->	M0:	I			M1:	18	2	1	M2:	16	1	2	M3:	7	0	1
72->	M0:	18	3	5	M1:	I			M2:	16	1	1	M3:	I		
73->	M0:	18	3	4	M1:	S		3	M2:	S		1	M3:	I		
74->	M0:	18	3	3	M1:	S		2	M2:	7	1	5	M3:	I		
75->	M0:	18	3	2	M1:	S		1	M2:	7	1	4	M3:	I		
76->	M0:	18	3	1	M1:	16	2	6	M2:	7	1	3	M3:	I		
77->	M0:	I			M1:	16	2	5	M2:	7	1	2	M3:	S		2
78->	M0:	I			M1:	16	2	4	M2:	7	1	1	M3:	S		1
79->	M0:	I			M1:	16	2	3	M2:	S		3	M3:	18	4	13
80->	M0:	I			M1:	16	2	2	M2:	S		2	M3:	18	4	12
81->	M0:	I			M1:	16	2	1	M2:	S		1	M3:	18	4	11
82->	M0:	I			M1:	I			M2:	9	0	5	M3:	18	4	10
83->	M0:	I			M1:	I			M2:	9	0	4	M3:	18	4	9
84->	M0:	I			M1:	I			M2:	9	0	3	M3:	18	4	8
85->	M0:	I			M1:	I			M2:	9	0	2	M3:	18	4	7
86->	M0:	I			M1:	I			M2:	9	0	1	M3:	18	4	6
87->	M0:	I			M1:	I			M2:	S		3	M3:	18	4	5
88->	M0:	I			M1:	I			M2:	S		2	M3:	18	4	4

89->	M0:	I			M1:	I			M2:	S		1	M3:	18	4	3
90->	M0:	I			M1:	I			M2:	9	1	11	M3:	18	4	2
91->	M0:	I			M1:	I			M2:	9	1	10	M3:	18	4	1
92->	M0:	I			M1:	I			M2:	9	1	9	M3:	S		2
93->	M0:	I			M1:	I			M2:	9	1	8	M3:	S		1
94->	M0:	I			M1:	I			M2:	9	1	7	M3:	16	3	3
95->	M0:	I			M1:	I			M2:	9	1	6	M3:	16	3	2
96->	M0:	I			M1:	I			M2:	9	1	5	M3:	16	3	1
97->	M0:	I			M1:	I			M2:	9	1	4	M3:	4	0	5
98->	M0:	I			M1:	I			M2:	9	1	3	M3:	4	0	4
99->	M0:	I			M1:	I			M2:	9	1	2	M3:	4	0	3
100->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	2
101->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	1
102->	M0:	N/A			M1:	I			M2:	N/A			M3:	S		4
103->	M0:	N/A			M1:	I			M2:	N/A			M3:	S		3
104->	M0:	N/A			M1:	I			M2:	N/A			M3:	S		2
105->	M0:	N/A			M1:	I			M2:	N/A			M3:	S		1
106->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	2	6
107->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	2	5
108->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	2	4
109->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	2	3
110->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	2	2
111->	M0:	4	1	5	M1:	I			M2:	N/A			M3:	7	2	1
112->	M0:	4	1	4	M1:	7	3	6	M2:	N/A			M3:	I		
113->	M0:	4	1	3	M1:	7	3	5	M2:	N/A			M3:	I		
114->	M0:	4	1	2	M1:	7	3	4	M2:	N/A			M3:	I		
115->	M0:	4	1	1	M1:	7	3	3	M2:	N/A			M3:	I		
116->	M0:	I			M1:	7	3	2	M2:	N/A			M3:	I		
117->	M0:	I			M1:	7	3	1	M2:	N/A			M3:	I		
118->	M0:	I			M1:	S		3	M2:	N/A			M3:	I		
119->	M0:	I			M1:	S		2	M2:	N/A			M3:	I		
120->	M0:	I			M1:	S		1	M2:	N/A			M3:	I		
121->	M0:	I			M1:	4	2	14	M2:	9	1	1	M3:	I		
122->	M0:	I			M1:	4	2	13	M2:	S		1	M3:	I		
123->	M0:	I			M1:	4	2	12	M2:	9	2	4	M3:	I		
124->	M0:	I			M1:	4	2	11	M2:	9	2	3	M3:	I		
125->	M0:	I			M1:	4	2	10	M2:	9	2	2	M3:	I		
126->	M0:	I			M1:	4	2	9	M2:	9	2	1	M3:	I		
127->	M0:	I			M1:	4	2	8	M2:	S		1	M3:	I		
128->	M0:	I			M1:	4	2	7	M2:	13	0	10	M3:	I		
129->	M0:	I			M1:	4	2	6	M2:	13	0	9	M3:	I		
130->	M0:	I			M1:	4	2	5	M2:	13	0	8	M3:	I		
131->	M0:	I			M1:	4	2	4	M2:	13	0	7	M3:	I		
132->	M0:	I			M1:	4	2	3	M2:	13	0	6	M3:	I		
133->	M0:	I			M1:	4	2	2	M2:	13	0	5	M3:	I		
134->	M0:	I			M1:	4	2	1	M2:	13	0	4	M3:	I		
135->	M0:	S		3	M1:	I			M2:	13	0	3	M3:	I		
136->	M0:	S		2	M1:	I			M2:	13	0	2	M3:	I		
137->	M0:	S		1	M1:	I			M2:	13	0	1	M3:	I		
138->	M0:	4	3	18	M1:	I			M2:	6	0	10	M3:	I		
139->	M0:	4	3	17	M1:	I			M2:	6	0	9	M3:	I		
140->	M0:	4	3	16	M1:	I			M2:	6	0	8	M3:	I		
141->	M0:	4	3	15	M1:	I			M2:	6	0	7	M3:	I		
142->	M0:	4	3	14	M1:	I			M2:	6	0	6	M3:	I		
143->	M0:	4	3	13	M1:	I			M2:	6	0	5	M3:	I		
144->	M0:	4	3	12	M1:	I			M2:	6	0	4	M3:	I		
145->	M0:	4	3	11	M1:	I			M2:	6	0	3	M3:	I		
146->	M0:	4	3	10	M1:	I			M2:	6	0	2	M3:	I		

147->	M0:	4	3	9	M1:	I			M2:	6	0	1	M3:	I		
148->	M0:	4	3	8	M1:	I			M2:	15	0	5	M3:	6	1	6
149->	M0:	4	3	7	M1:	I			M2:	15	0	4	M3:	6	1	5
150->	M0:	4	3	6	M1:	I			M2:	15	0	3	M3:	6	1	4
151->	M0:	4	3	5	M1:	I			M2:	15	0	2	M3:	6	1	3
152->	M0:	4	3	4	M1:	I			M2:	15	0	1	M3:	6	1	2
153->	M0:	4	3	3	M1:	I			M2:	8	0	7	M3:	6	1	1
154->	M0:	4	3	2	M1:	I			M2:	8	0	6	M3:	S		2
155->	M0:	4	3	1	M1:	I			M2:	8	0	5	M3:	S		1
156->	M0:	S		3	M1:	S		3	M2:	8	0	4	M3:	15	1	16
157->	M0:	S		2	M1:	S		2	M2:	8	0	3	M3:	15	1	15
158->	M0:	S		1	M1:	S		1	M2:	8	0	2	M3:	15	1	14
159->	M0:	10	0	9	M1:	4	4	7	M2:	8	0	1	M3:	15	1	13
160->	M0:	10	0	8	M1:	4	4	6	M2:	I			M3:	15	1	12
161->	M0:	10	0	7	M1:	4	4	5	M2:	I			M3:	15	1	11
162->	M0:	10	0	6	M1:	4	4	4	M2:	I			M3:	15	1	10
163->	M0:	10	0	5	M1:	4	4	3	M2:	I			M3:	15	1	9
164->	M0:	10	0	4	M1:	4	4	2	M2:	I			M3:	15	1	8
165->	M0:	10	0	3	M1:	4	4	1	M2:	I			M3:	15	1	7
166->	M0:	10	0	2	M1:	8	1	10	M2:	I			M3:	15	1	6
167->	M0:	10	0	1	M1:	8	1	9	M2:	I			M3:	15	1	5
168->	M0:	S		3	M1:	8	1	8	M2:	I			M3:	15	1	4
169->	M0:	S		2	M1:	8	1	7	M2:	I			M3:	15	1	3
170->	M0:	S		1	M1:	8	1	6	M2:	I			M3:	15	1	2
171->	M0:	10	1	14	M1:	8	1	5	M2:	I			M3:	15	1	1
172->	M0:	10	1	13	M1:	8	1	4	M2:	S		3	M3:	I		
173->	M0:	10	1	12	M1:	8	1	3	M2:	S		2	M3:	I		
174->	M0:	10	1	11	M1:	8	1	2	M2:	S		1	M3:	I		
175->	M0:	10	1	10	M1:	8	1	1	M2:	15	2	3	M3:	I		
176->	M0:	10	1	9	M1:	I			M2:	15	2	2	M3:	I		
177->	M0:	10	1	8	M1:	I			M2:	15	2	1	M3:	I		
178->	M0:	10	1	7	M1:	I			M2:	S		3	M3:	S		2
179->	M0:	10	1	6	M1:	I			M2:	S		2	M3:	S		1
180->	M0:	10	1	5	M1:	I			M2:	S		1	M3:	14	0	9
181->	M0:	10	1	4	M1:	I			M2:	8	2	5	M3:	14	0	8
182->	M0:	10	1	3	M1:	I			M2:	8	2	4	M3:	14	0	7
183->	M0:	10	1	2	M1:	I			M2:	8	2	3	M3:	14	0	6
184->	M0:	10	1	1	M1:	I			M2:	8	2	2	M3:	14	0	5
185->	M0:	I			M1:	10	2	7	M2:	8	2	1	M3:	14	0	4
186->	M0:	8	3	8	M1:	10	2	6	M2:	I			M3:	14	0	3
187->	M0:	8	3	7	M1:	10	2	5	M2:	I			M3:	14	0	2
188->	M0:	8	3	6	M1:	10	2	4	M2:	I			M3:	14	0	1
189->	M0:	8	3	5	M1:	10	2	3	M2:	14	1	7	M3:	I		
190->	M0:	8	3	4	M1:	10	2	2	M2:	14	1	6	M3:	I		
191->	M0:	8	3	3	M1:	10	2	1	M2:	14	1	5	M3:	I		
192->	M0:	8	3	2	M1:	I			M2:	14	1	4	M3:	I		
193->	M0:	8	3	1	M1:	I			M2:	14	1	3	M3:	I		
194->	M0:	I			M1:	I			M2:	14	1	2	M3:	S		5
195->	M0:	I			M1:	I			M2:	14	1	1	M3:	S		4
196->	M0:	14	2	8	M1:	I			M2:	I			M3:	S		3
197->	M0:	14	2	7	M1:	I			M2:	I			M3:	S		2
198->	M0:	14	2	6	M1:	I			M2:	I			M3:	S		1
199->	M0:	14	2	5	M1:	I			M2:	I			M3:	8	4	10
200->	M0:	14	2	4	M1:	I			M2:	I			M3:	8	4	9
201->	M0:	14	2	3	M1:	I			M2:	I			M3:	8	4	8
202->	M0:	14	2	2	M1:	I			M2:	I			M3:	8	4	7
203->	M0:	14	2	1	M1:	I			M2:	I			M3:	8	4	6
204->	M0:	I			M1:	I			M2:	14	3	7	M3:	8	4	5

205->	M0:	I			M1:	I			M2:	14	3	6	M3:	8	4	4
206->	M0:	I			M1:	I			M2:	14	3	5	M3:	8	4	3
207->	M0:	I			M1:	I			M2:	14	3	4	M3:	8	4	2
208->	M0:	I			M1:	I			M2:	14	3	3	M3:	8	4	1
209->	M0:	S		3	M1:	I			M2:	14	3	2	M3:	I		
210->	M0:	S		2	M1:	I			M2:	14	3	1	M3:	I		
211->	M0:	S		1	M1:	I			M2:	0	0	5	M3:	I		
212->	M0:	3	0	7	M1:	I			M2:	0	0	4	M3:	I		
213->	M0:	3	0	6	M1:	I			M2:	0	0	3	M3:	I		
214->	M0:	3	0	5	M1:	I			M2:	0	0	2	M3:	I		
215->	M0:	3	0	4	M1:	I			M2:	0	0	1	M3:	I		
216->	M0:	3	0	3	M1:	0	1	12	M2:	I			M3:	I		
217->	M0:	3	0	2	M1:	0	1	11	M2:	I			M3:	I		
218->	M0:	3	0	1	M1:	0	1	10	M2:	I			M3:	I		
219->	M0:	I			M1:	0	1	9	M2:	3	1	8	M3:	I		
220->	M0:	I			M1:	0	1	8	M2:	3	1	7	M3:	I		
221->	M0:	I			M1:	0	1	7	M2:	3	1	6	M3:	I		
222->	M0:	I			M1:	0	1	6	M2:	3	1	5	M3:	I		
223->	M0:	I			M1:	0	1	5	M2:	3	1	4	M3:	I		
224->	M0:	I			M1:	0	1	4	M2:	3	1	3	M3:	I		
225->	M0:	I			M1:	0	1	3	M2:	3	1	2	M3:	I		
226->	M0:	I			M1:	0	1	2	M2:	3	1	1	M3:	I		
227->	M0:	I			M1:	0	1	1	M2:	I			M3:	I		
228->	M0:	I			M1:	3	2	3	M2:	I			M3:	I		
229->	M0:	I			M1:	3	2	2	M2:	I			M3:	I		
230->	M0:	I			M1:	3	2	1	M2:	I			M3:	I		
231->	M0:	I			M1:	I			M2:	I			M3:	S		5
232->	M0:	I			M1:	I			M2:	I			M3:	S		4
233->	M0:	I			M1:	I			M2:	I			M3:	S		3
234->	M0:	I			M1:	I			M2:	I			M3:	S		2
235->	M0:	I			M1:	I			M2:	I			M3:	S		1
236->	M0:	I			M1:	I			M2:	I			M3:	3	3	1
237->	M0:	I			M1:	I			M2:	I			M3:	I		

M0 Util :41.9492%   M1 Util :46.1864%   M2 Util :58.4746%   M3 Util :46.1864%

Job	0	Start:	211	Finish:	228	Deadline:	100	Penalty:	16384
Job	1	Start:	1	Finish:	64	Deadline:	50	Penalty:	196
Job	2	Start:	16	Finish:	56	Deadline:	90	Penalty:	0
Job	3	Start:	212	Finish:	237	Deadline:	100	Penalty:	18769
Job	4	Start:	97	Finish:	166	Deadline:	0	Penalty:	0
Job	5	Start:	1	Finish:	9	Deadline:	20	Penalty:	0
Job	6	Start:	138	Finish:	154	Deadline:	0	Penalty:	0
Job	7	Start:	65	Finish:	118	Deadline:	0	Penalty:	0
Job	8	Start:	153	Finish:	209	Deadline:	100	Penalty:	11881
Job	9	Start:	82	Finish:	127	Deadline:	0	Penalty:	0
Job	10	Start:	159	Finish:	192	Deadline:	50	Penalty:	20164
Job	11	Start:	1	Finish:	24	Deadline:	100	Penalty:	0
Job	12	Start:	12	Finish:	21	Deadline:	0	Penalty:	0
Job	13	Start:	128	Finish:	138	Deadline:	0	Penalty:	0
Job	14	Start:	180	Finish:	211	Deadline:	0	Penalty:	0
Job	15	Start:	148	Finish:	178	Deadline:	0	Penalty:	0
Job	16	Start:	45	Finish:	97	Deadline:	0	Penalty:	0
Job	17	Start:	1	Finish:	18	Deadline:	120	Penalty:	0
Job	18	Start:	19	Finish:	92	Deadline:	50	Penalty:	1764
Job	19	Start:	37	Finish:	51	Deadline:	80	Penalty:	0

TOTAL TIME: 236   PENALTY: 69158   AVE JOB TIME: 34.2

Schedule Plan for Compromise Solution

1->	M0:	12	0	9	M1:	1	0	12	M2:	17	0	7	M3:	11	0	7
2->	M0:	12	0	8	M1:	1	0	11	M2:	17	0	6	M3:	11	0	6
3->	M0:	12	0	7	M1:	1	0	10	M2:	17	0	5	M3:	11	0	5
4->	M0:	12	0	6	M1:	1	0	9	M2:	17	0	4	M3:	11	0	4
5->	M0:	12	0	5	M1:	1	0	8	M2:	17	0	3	M3:	11	0	3
6->	M0:	12	0	4	M1:	1	0	7	M2:	17	0	2	M3:	11	0	2
7->	M0:	12	0	3	M1:	1	0	6	M2:	17	0	1	M3:	11	0	1
8->	M0:	12	0	2	M1:	1	0	5	M2:	11	1	10	M3:	17	1	10
9->	M0:	12	0	1	M1:	1	0	4	M2:	11	1	9	M3:	17	1	9
10->	M0:	S		3	M1:	1	0	3	M2:	N/A			M3:	17	1	8
11->	M0:	S		2	M1:	1	0	2	M2:	N/A			M3:	17	1	7
12->	M0:	S		1	M1:	1	0	1	M2:	N/A			M3:	17	1	6
13->	M0:	5	0	8	M1:	S		3	M2:	N/A			M3:	17	1	5
14->	M0:	5	0	7	M1:	S		2	M2:	N/A			M3:	17	1	4
15->	M0:	5	0	6	M1:	S		1	M2:	N/A			M3:	17	1	3
16->	M0:	5	0	5	M1:	2	0	6	M2:	11	1	8	M3:	17	1	2
17->	M0:	5	0	4	M1:	2	0	5	M2:	11	1	7	M3:	17	1	1
18->	M0:	5	0	3	M1:	2	0	4	M2:	11	1	6	M3:	S		1
19->	M0:	5	0	2	M1:	2	0	3	M2:	11	1	5	M3:	18	0	8
20->	M0:	5	0	1	M1:	N/A			M2:	11	1	4	M3:	N/A		
21->	M0:	I			M1:	N/A			M2:	11	1	3	M3:	N/A		
22->	M0:	I			M1:	N/A			M2:	11	1	2	M3:	N/A		
23->	M0:	I			M1:	N/A			M2:	11	1	1	M3:	N/A		
24->	M0:	I			M1:	N/A			M2:	1	1	3	M3:	N/A		
25->	M0:	I			M1:	N/A			M2:	N/A			M3:	N/A		
26->	M0:	I			M1:	2	0	2	M2:	N/A			M3:	N/A		
27->	M0:	I			M1:	2	0	1	M2:	N/A			M3:	N/A		
28->	M0:	I			M1:	S		3	M2:	N/A			M3:	N/A		
29->	M0:	I			M1:	S		2	M2:	N/A			M3:	N/A		
30->	M0:	I			M1:	S		1	M2:	N/A			M3:	N/A		
31->	M0:	I			M1:	2	1	6	M2:	N/A			M3:	N/A		
32->	M0:	I			M1:	2	1	5	M2:	N/A			M3:	N/A		
33->	M0:	I			M1:	2	1	4	M2:	N/A			M3:	N/A		
34->	M0:	I			M1:	2	1	3	M2:	N/A			M3:	N/A		
35->	M0:	I			M1:	2	1	2	M2:	N/A			M3:	N/A		
36->	M0:	I			M1:	2	1	1	M2:	N/A			M3:	18	0	7
37->	M0:	I			M1:	I			M2:	N/A			M3:	18	0	6
38->	M0:	I			M1:	I			M2:	1	1	2	M3:	18	0	5
39->	M0:	I			M1:	I			M2:	1	1	1	M3:	18	0	4
40->	M0:	I			M1:	I			M2:	2	2	10	M3:	18	0	3
41->	M0:	I			M1:	I			M2:	2	2	9	M3:	18	0	2
42->	M0:	I			M1:	I			M2:	2	2	8	M3:	18	0	1
43->	M0:	I			M1:	I			M2:	2	2	7	M3:	S		5
44->	M0:	I			M1:	I			M2:	2	2	6	M3:	S		4
45->	M0:	S		3	M1:	I			M2:	2	2	5	M3:	S		3
46->	M0:	S		2	M1:	I			M2:	2	2	4	M3:	S		2
47->	M0:	S		1	M1:	I			M2:	2	2	3	M3:	S		1
48->	M0:	16	0	5	M1:	I			M2:	2	2	2	M3:	1	2	8
49->	M0:	16	0	4	M1:	I			M2:	2	2	1	M3:	1	2	7
50->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	6
51->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	5
52->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	4
53->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	3
54->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	2
55->	M0:	N/A			M1:	I			M2:	N/A			M3:	1	2	1
56->	M0:	N/A			M1:	I			M2:	18	1	10	M3:	S		5

57->	M0:	N/A			M1:	I			M2:	18	1	9	M3:	S		4
58->	M0:	N/A			M1:	I			M2:	18	1	8	M3:	S		3
59->	M0:	N/A			M1:	I			M2:	18	1	7	M3:	S		2
60->	M0:	N/A			M1:	I			M2:	18	1	6	M3:	N/A		
61->	M0:	16	0	3	M1:	I			M2:	18	1	5	M3:	N/A		
62->	M0:	16	0	2	M1:	I			M2:	18	1	4	M3:	N/A		
63->	M0:	16	0	1	M1:	I			M2:	18	1	3	M3:	N/A		
64->	M0:	I			M1:	I			M2:	18	1	2	M3:	N/A		
65->	M0:	I			M1:	I			M2:	18	1	1	M3:	S		1
66->	M0:	I			M1:	18	2	6	M2:	S		3	M3:	1	3	6
67->	M0:	I			M1:	18	2	5	M2:	S		2	M3:	1	3	5
68->	M0:	I			M1:	18	2	4	M2:	S		1	M3:	1	3	4
69->	M0:	I			M1:	18	2	3	M2:	9	0	5	M3:	1	3	3
70->	M0:	I			M1:	18	2	2	M2:	9	0	4	M3:	1	3	2
71->	M0:	I			M1:	18	2	1	M2:	9	0	3	M3:	1	3	1
72->	M0:	18	3	5	M1:	I			M2:	9	0	2	M3:	S		1
73->	M0:	18	3	4	M1:	I			M2:	9	0	1	M3:	16	1	7
74->	M0:	18	3	3	M1:	I			M2:	S		3	M3:	16	1	6
75->	M0:	18	3	2	M1:	I			M2:	S		2	M3:	16	1	5
76->	M0:	18	3	1	M1:	I			M2:	S		1	M3:	16	1	4
77->	M0:	I			M1:	I			M2:	9	1	11	M3:	16	1	3
78->	M0:	I			M1:	I			M2:	9	1	10	M3:	16	1	2
79->	M0:	I			M1:	I			M2:	9	1	9	M3:	16	1	1
80->	M0:	I			M1:	S		3	M2:	9	1	8	M3:	S		2
81->	M0:	I			M1:	S		2	M2:	9	1	7	M3:	S		1
82->	M0:	I			M1:	S		1	M2:	9	1	6	M3:	18	4	13
83->	M0:	I			M1:	16	2	6	M2:	9	1	5	M3:	18	4	12
84->	M0:	I			M1:	16	2	5	M2:	9	1	4	M3:	18	4	11
85->	M0:	I			M1:	16	2	4	M2:	9	1	3	M3:	18	4	10
86->	M0:	I			M1:	16	2	3	M2:	9	1	2	M3:	18	4	9
87->	M0:	I			M1:	16	2	2	M2:	9	1	1	M3:	18	4	8
88->	M0:	I			M1:	16	2	1	M2:	13	0	10	M3:	18	4	7
89->	M0:	I			M1:	S		3	M2:	13	0	9	M3:	18	4	6
90->	M0:	I			M1:	S		2	M2:	13	0	8	M3:	18	4	5
91->	M0:	I			M1:	S		1	M2:	13	0	7	M3:	18	4	4
92->	M0:	I			M1:	9	2	8	M2:	13	0	6	M3:	18	4	3
93->	M0:	I			M1:	9	2	7	M2:	13	0	5	M3:	18	4	2
94->	M0:	I			M1:	9	2	6	M2:	13	0	4	M3:	18	4	1
95->	M0:	I			M1:	9	2	5	M2:	13	0	3	M3:	S		2
96->	M0:	I			M1:	9	2	4	M2:	13	0	2	M3:	S		1
97->	M0:	I			M1:	9	2	3	M2:	13	0	1	M3:	16	3	3
98->	M0:	I			M1:	9	2	2	M2:	8	0	7	M3:	16	3	2
99->	M0:	I			M1:	9	2	1	M2:	8	0	6	M3:	16	3	1
100->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	5
101->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	4
102->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	3
103->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	2
104->	M0:	N/A			M1:	I			M2:	N/A			M3:	4	0	1
105->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	7
106->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	6
107->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	5
108->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	4
109->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	3
110->	M0:	N/A			M1:	I			M2:	N/A			M3:	7	0	2
111->	M0:	4	1	5	M1:	I			M2:	N/A			M3:	7	0	1
112->	M0:	4	1	4	M1:	I			M2:	N/A			M3:	19	0	8
113->	M0:	4	1	3	M1:	I			M2:	N/A			M3:	19	0	7
114->	M0:	4	1	2	M1:	I			M2:	N/A			M3:	19	0	6

115->	M0:	4	1	1	M1:	I			M2:	N/A			M3:	19	0	5
116->	M0:	I			M1:	4	2	14	M2:	N/A			M3:	19	0	4
117->	M0:	I			M1:	4	2	13	M2:	N/A			M3:	19	0	3
118->	M0:	I			M1:	4	2	12	M2:	N/A			M3:	19	0	2
119->	M0:	I			M1:	4	2	11	M2:	N/A			M3:	19	0	1
120->	M0:	I			M1:	4	2	10	M2:	N/A			M3:	I		
121->	M0:	I			M1:	4	2	9	M2:	8	0	5	M3:	I		
122->	M0:	I			M1:	4	2	8	M2:	8	0	4	M3:	I		
123->	M0:	I			M1:	4	2	7	M2:	8	0	3	M3:	I		
124->	M0:	I			M1:	4	2	6	M2:	8	0	2	M3:	I		
125->	M0:	I			M1:	4	2	5	M2:	8	0	1	M3:	I		
126->	M0:	I			M1:	4	2	4	M2:	7	1	5	M3:	I		
127->	M0:	I			M1:	4	2	3	M2:	7	1	4	M3:	I		
128->	M0:	I			M1:	4	2	2	M2:	7	1	3	M3:	I		
129->	M0:	I			M1:	4	2	1	M2:	7	1	2	M3:	I		
130->	M0:	S		3	M1:	S		3	M2:	7	1	1	M3:	I		
131->	M0:	S		2	M1:	S		2	M2:	6	0	10	M3:	S		4
132->	M0:	S		1	M1:	S		1	M2:	6	0	9	M3:	S		3
133->	M0:	4	3	18	M1:	8	1	10	M2:	6	0	8	M3:	S		2
134->	M0:	4	3	17	M1:	8	1	9	M2:	6	0	7	M3:	S		1
135->	M0:	4	3	16	M1:	8	1	8	M2:	6	0	6	M3:	7	2	6
136->	M0:	4	3	15	M1:	8	1	7	M2:	6	0	5	M3:	7	2	5
137->	M0:	4	3	14	M1:	8	1	6	M2:	6	0	4	M3:	7	2	4
138->	M0:	4	3	13	M1:	8	1	5	M2:	6	0	3	M3:	7	2	3
139->	M0:	4	3	12	M1:	8	1	4	M2:	6	0	2	M3:	7	2	2
140->	M0:	4	3	11	M1:	8	1	3	M2:	6	0	1	M3:	7	2	1
141->	M0:	4	3	10	M1:	8	1	2	M2:	15	0	5	M3:	6	1	6
142->	M0:	4	3	9	M1:	8	1	1	M2:	15	0	4	M3:	6	1	5
143->	M0:	4	3	8	M1:	7	3	6	M2:	15	0	3	M3:	6	1	4
144->	M0:	4	3	7	M1:	7	3	5	M2:	15	0	2	M3:	6	1	3
145->	M0:	4	3	6	M1:	7	3	4	M2:	15	0	1	M3:	6	1	2
146->	M0:	4	3	5	M1:	7	3	3	M2:	8	2	5	M3:	6	1	1
147->	M0:	4	3	4	M1:	7	3	2	M2:	8	2	4	M3:	S		2
148->	M0:	4	3	3	M1:	7	3	1	M2:	8	2	3	M3:	S		1
149->	M0:	4	3	2	M1:	I			M2:	8	2	2	M3:	15	1	16
150->	M0:	4	3	1	M1:	I			M2:	8	2	1	M3:	15	1	15
151->	M0:	S		3	M1:	4	4	7	M2:	I			M3:	15	1	14
152->	M0:	S		2	M1:	4	4	6	M2:	I			M3:	15	1	13
153->	M0:	S		1	M1:	4	4	5	M2:	I			M3:	15	1	12
154->	M0:	10	0	9	M1:	4	4	4	M2:	I			M3:	15	1	11
155->	M0:	10	0	8	M1:	4	4	3	M2:	I			M3:	15	1	10
156->	M0:	10	0	7	M1:	4	4	2	M2:	I			M3:	15	1	9
157->	M0:	10	0	6	M1:	4	4	1	M2:	I			M3:	15	1	8
158->	M0:	10	0	5	M1:	I			M2:	I			M3:	15	1	7
159->	M0:	10	0	4	M1:	I			M2:	I			M3:	15	1	6
160->	M0:	10	0	3	M1:	I			M2:	I			M3:	15	1	5
161->	M0:	10	0	2	M1:	I			M2:	I			M3:	15	1	4
162->	M0:	10	0	1	M1:	I			M2:	I			M3:	15	1	3
163->	M0:	S		3	M1:	I			M2:	I			M3:	15	1	2
164->	M0:	S		2	M1:	I			M2:	I			M3:	15	1	1
165->	M0:	S		1	M1:	I			M2:	S		3	M3:	8	3	13
166->	M0:	10	1	14	M1:	I			M2:	S		2	M3:	8	3	12
167->	M0:	10	1	13	M1:	I			M2:	S		1	M3:	8	3	11
168->	M0:	10	1	12	M1:	I			M2:	15	2	3	M3:	8	3	10
169->	M0:	10	1	11	M1:	I			M2:	15	2	2	M3:	8	3	9
170->	M0:	10	1	10	M1:	I			M2:	15	2	1	M3:	8	3	8
171->	M0:	10	1	9	M1:	I			M2:	14	0	11	M3:	8	3	7
172->	M0:	10	1	8	M1:	I			M2:	14	0	10	M3:	8	3	6

173->	M0:	10	1	7	M1:	I			M2:	14	0	9	M3:	8	3	5
174->	M0:	10	1	6	M1:	I			M2:	14	0	8	M3:	8	3	4
175->	M0:	10	1	5	M1:	I			M2:	14	0	7	M3:	8	3	3
176->	M0:	10	1	4	M1:	I			M2:	14	0	6	M3:	8	3	2
177->	M0:	10	1	3	M1:	I			M2:	14	0	5	M3:	8	3	1
178->	M0:	10	1	2	M1:	I			M2:	14	0	4	M3:	I		
179->	M0:	10	1	1	M1:	I			M2:	14	0	3	M3:	I		
180->	M0:	S		3	M1:	10	2	7	M2:	14	0	2	M3:	I		
181->	M0:	S		2	M1:	10	2	6	M2:	14	0	1	M3:	I		
182->	M0:	S		1	M1:	10	2	5	M2:	S		3	M3:	I		
183->	M0:	8	4	5	M1:	10	2	4	M2:	S		2	M3:	I		
184->	M0:	8	4	4	M1:	10	2	3	M2:	S		1	M3:	I		
185->	M0:	8	4	3	M1:	10	2	2	M2:	14	1	7	M3:	I		
186->	M0:	8	4	2	M1:	10	2	1	M2:	14	1	6	M3:	I		
187->	M0:	8	4	1	M1:	I			M2:	14	1	5	M3:	I		
188->	M0:	3	0	7	M1:	I			M2:	14	1	4	M3:	I		
189->	M0:	3	0	6	M1:	I			M2:	14	1	3	M3:	I		
190->	M0:	3	0	5	M1:	I			M2:	14	1	2	M3:	I		
191->	M0:	3	0	4	M1:	I			M2:	14	1	1	M3:	I		
192->	M0:	3	0	3	M1:	I			M2:	0	0	5	M3:	14	2	13
193->	M0:	3	0	2	M1:	I			M2:	0	0	4	M3:	14	2	12
194->	M0:	3	0	1	M1:	I			M2:	0	0	3	M3:	14	2	11
195->	M0:	I			M1:	I			M2:	0	0	2	M3:	14	2	10
196->	M0:	I			M1:	I			M2:	0	0	1	M3:	14	2	9
197->	M0:	I			M1:	0	1	12	M2:	3	1	8	M3:	14	2	8
198->	M0:	I			M1:	0	1	11	M2:	3	1	7	M3:	14	2	7
199->	M0:	I			M1:	0	1	10	M2:	3	1	6	M3:	14	2	6
200->	M0:	I			M1:	0	1	9	M2:	3	1	5	M3:	14	2	5
201->	M0:	I			M1:	0	1	8	M2:	3	1	4	M3:	14	2	4
202->	M0:	I			M1:	0	1	7	M2:	3	1	3	M3:	14	2	3
203->	M0:	I			M1:	0	1	6	M2:	3	1	2	M3:	14	2	2
204->	M0:	I			M1:	0	1	5	M2:	3	1	1	M3:	14	2	1
205->	M0:	I			M1:	0	1	4	M2:	14	3	7	M3:	I		
206->	M0:	I			M1:	0	1	3	M2:	14	3	6	M3:	I		
207->	M0:	I			M1:	0	1	2	M2:	14	3	5	M3:	I		
208->	M0:	I			M1:	0	1	1	M2:	14	3	4	M3:	I		
209->	M0:	I			M1:	3	2	3	M2:	14	3	3	M3:	I		
210->	M0:	I			M1:	3	2	2	M2:	14	3	2	M3:	I		
211->	M0:	I			M1:	3	2	1	M2:	14	3	1	M3:	I		
212->	M0:	I			M1:	I			M2:	I			M3:	S		2
213->	M0:	I			M1:	I			M2:	I			M3:	S		1
214->	M0:	I			M1:	I			M2:	I			M3:	3	3	1
215->	M0:	I			M1:	I			M2:	I			M3:	I		

M0 Util :39.7196%   M1 Util :48.1308%   M2 Util :64.9533%   M3 Util :64.0187%

Job	0	Start:	192	Finish:	209	Deadline:	100	Penalty:	11881
Job	1	Start:	1	Finish:	72	Deadline:	50	Penalty:	484
Job	2	Start:	16	Finish:	56	Deadline:	90	Penalty:	0
Job	3	Start:	188	Finish:	215	Deadline:	100	Penalty:	13225
Job	4	Start:	100	Finish:	158	Deadline:	0	Penalty:	0
Job	5	Start:	13	Finish:	21	Deadline:	20	Penalty:	1
Job	6	Start:	131	Finish:	147	Deadline:	0	Penalty:	0
Job	7	Start:	105	Finish:	149	Deadline:	0	Penalty:	0
Job	8	Start:	98	Finish:	188	Deadline:	100	Penalty:	7744
Job	9	Start:	69	Finish:	100	Deadline:	0	Penalty:	0
Job	10	Start:	154	Finish:	187	Deadline:	50	Penalty:	18769
Job	11	Start:	1	Finish:	24	Deadline:	100	Penalty:	0

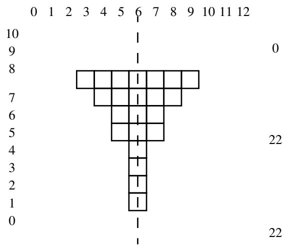
```
Job   12   Start:      1   Finish:    10   Deadline:     0   Penalty:      0
Job   13   Start:     88   Finish:    98   Deadline:     0   Penalty:      0
Job   14   Start:    171   Finish:   212   Deadline:     0   Penalty:      0
Job   15   Start:    141   Finish:   171   Deadline:     0   Penalty:      0
Job   16   Start:     48   Finish:   100   Deadline:     0   Penalty:      0
Job   17   Start:      1   Finish:    18   Deadline:   120   Penalty:      0
Job   18   Start:     19   Finish:    95   Deadline:    50   Penalty:   2025
Job   19   Start:    112   Finish:   120   Deadline:    80   Penalty:   1600

TOTAL TIME: 214    PENALTY: 55729    AVE JOB TIME: 35.05
```

Appendix E- Containership Information

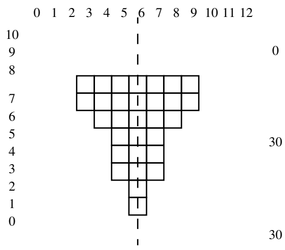
E.1 Containership Bay Plan

Drawn Looking Forward

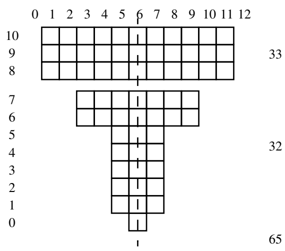
Bay 0



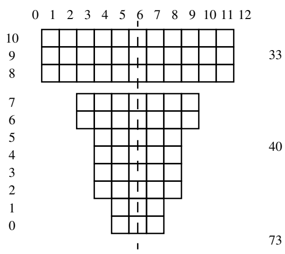
Bay 1



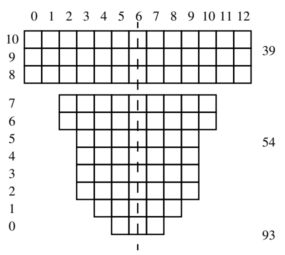
Bay 2



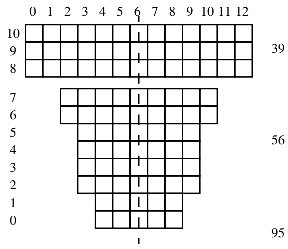
Bay 3



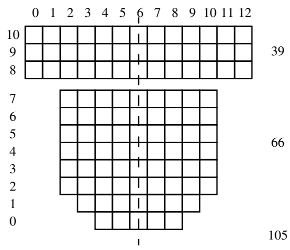
Bay 4



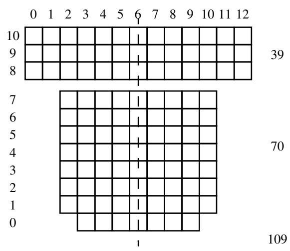
Bay 5



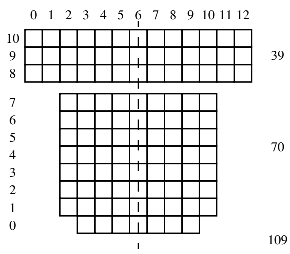
Bay 6



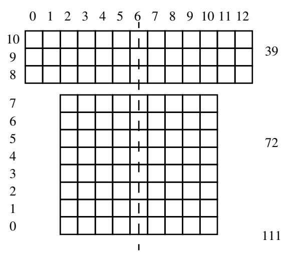
Bay 7



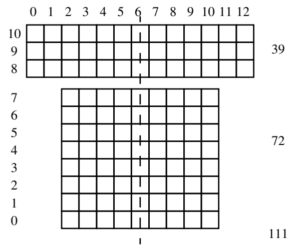
Bay 8



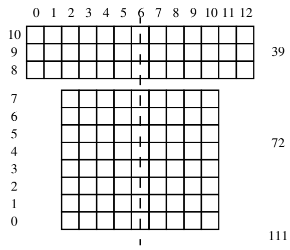
Bay 9



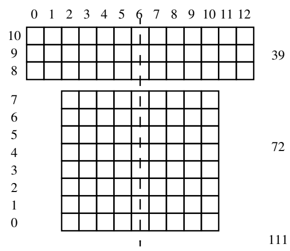
Bay 10



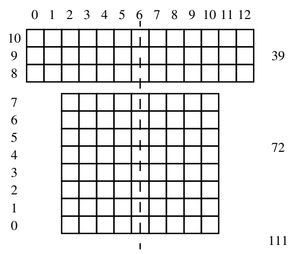
Bay 11



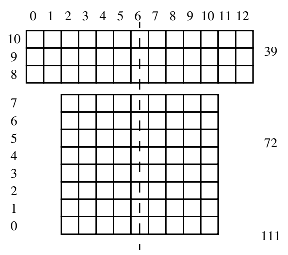
Bay 12



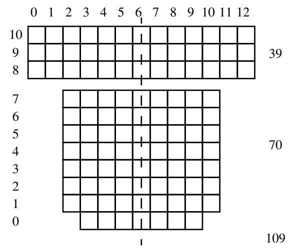
Bay 13



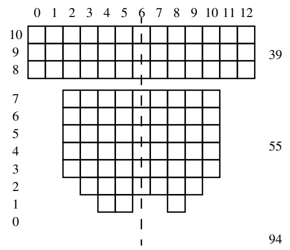
Bay 14



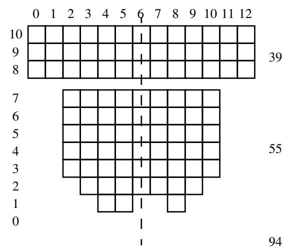
Bay 15



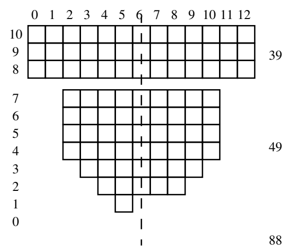
Bay 16



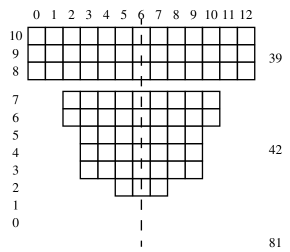
Bay 17



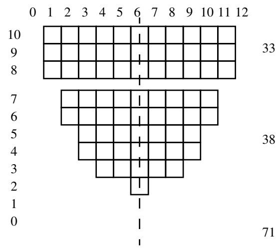
Bay 18



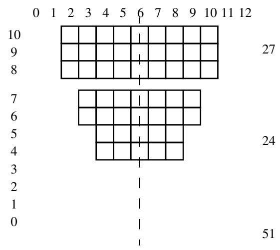
Bay 19



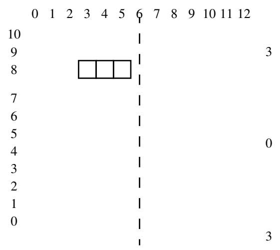
Bay 20



Bay 21

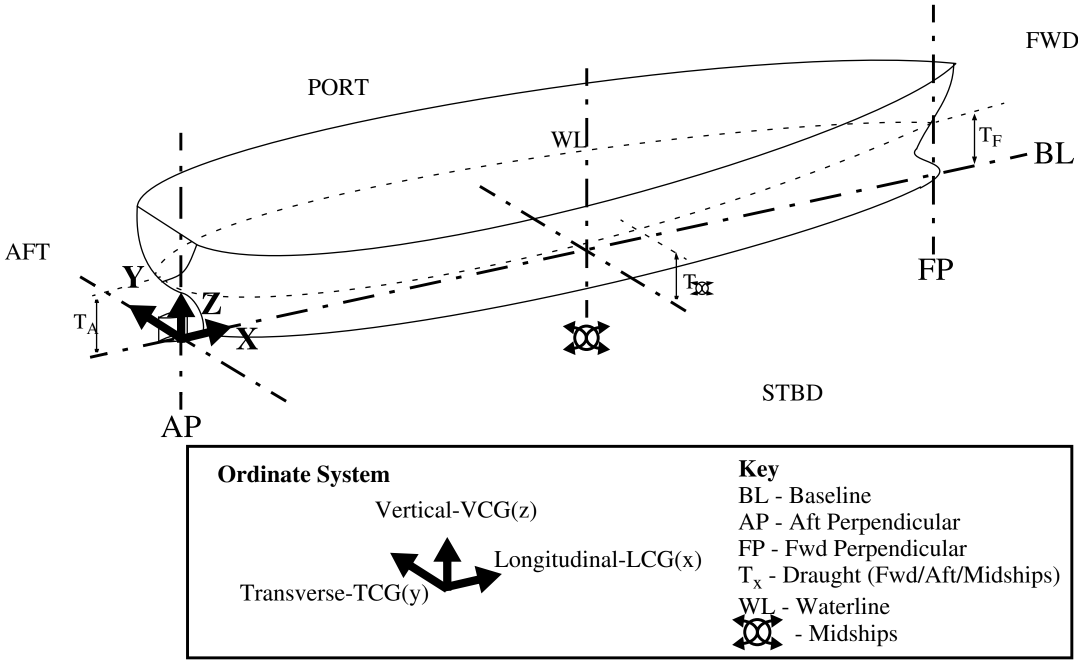


Bay 22



Deck = 753
In Holds = 1205
TOT = 1958

E.2 Calculation of Containership Trim and Stability



HEEL - Minimise the Magnitude of Transverse Deadweight Moment of the Containers.

$$DwtMt_{TCont} = \sum_{i=1}^{n} \Delta_i y_i$$

n = total number of containers.

Δ_i= mass of container i.

y_i = TCG for container i.

STABILITY - Ensure Limiting Deadweight Moment is not exceeded.

Performed for three cases, 100%, 50% and 10% of variable load (Fuel/Supplies).

1.Calculate Total Vertical Deadweight Moment (initially with ballast=0).

$$TotalDwtMt_V = DwtMt_{VCont} + Mt_{VVar} + Mt_{VBal}$$

$$DwtMt_{VCont} = \sum_{i=1}^{n} \Delta_i z_i$$ = Vertical Deadweight Moment of containers.

Mt_{VVar} = the Vertical moment of the variable load (for the three conditions).

Mt_{VBal} = the Vertical moment of the ballast.

2.Calculate total displacement.

$$\text{Total}\Delta = \text{Dwt} + \text{Ls}$$

$$\text{Dwt} = \Delta_{\text{Containers}} + \Delta_{\text{Var}} + \Delta_{\text{Bal}}$$

Ls = Mass of Lightship.

3.Compare $\text{Total}\Delta$ and $\text{TotalDwtMt}_{\text{V}}$ to Limiting DwtMt_{V} curve (Revised T&S book) to see if stability constraints are met, i.e. Limiting DwtMt_{V} is not exceeded.

TRIM - Calculate ship trim ensuring 0 < t < -1 (stern trim).

$$t = \frac{\text{Total}\Delta(\text{LCG} - \text{LCB})}{100 \times \text{MTC}} = t_a + t_f$$

$$\text{LCG} = \frac{\text{DwtMt}_{\text{LCont}} + \text{Mt}_{\text{LVar}} + \text{Mt}_{\text{LBal}}}{\text{Total}\Delta}$$

LCB = Longitudinal Centre of Buoyancy (Revised T & S Book).

MTC = Moment to change Trim one Centimetre (Revised T & S Book).

$$\text{DwtMt}_{\text{LCont}} = \sum_{i=1}^{n} \Delta_i x_i$$

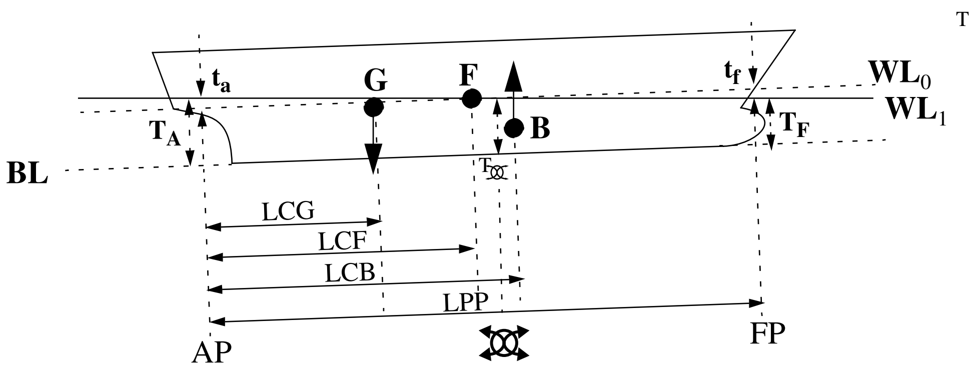
Mt_{LVar} = the Longitudinal moment of the variable load (for the three conditions).

Mt_{LBal} = the Longitudinal moment of the ballast.

PROPELLER IMMERSION - Draught Aft TA> 7.8 meters.

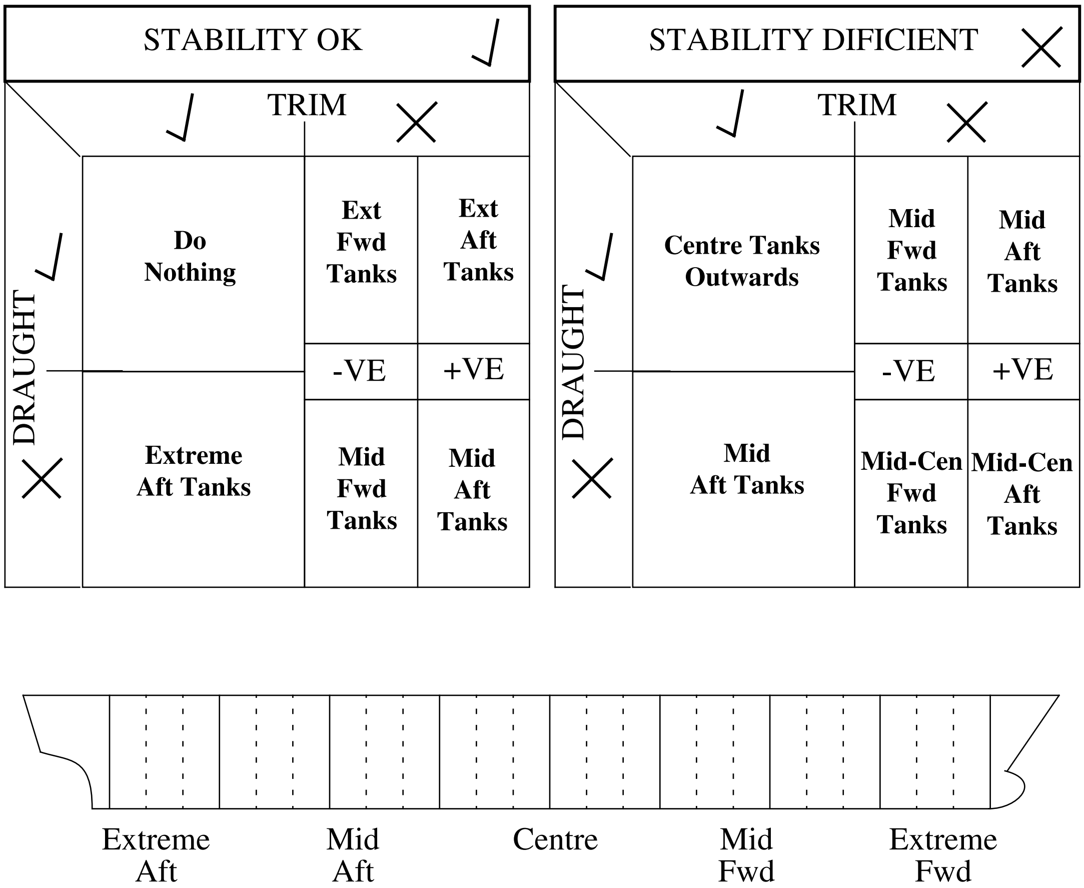
$$t_a = \frac{t(\text{LCF})}{\text{LCB}}$$

$$\therefore T_A = T_{\otimes} + t_a ,$$



BALLASTING SCHEDULE

If any of the stability, trim or propeller immersion criteria are not met, ballasting must be used and the calculations for 2,3 and 4 are repeated until all criteria are met. Ballasting is applied as follows:



E.3 MCGA Containership Layout

Notation:

818 C 7

Container Number — Destination — Mass

Bay 0

0X 0	204B 4	345B 4	0X 0	0X 0	689B 5	0X 0
	6B 8	614B 9	7B 9	4B 9	58B 7	
		12B 9	683B10	11B10		
		8B11	5B13	693B14		
			628B15			
			9B16			
			10B18			
			13B18			

Bay 1

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	147B 6	163B 8	123B 5	499B 7	249B 4	22B 5
	557B16	573B 9	17B11	29B 9	30B 9	
		20B11	31B13	27B11		
		35B13	16B16	24B16		
		23B16	14B18	15B17		
			34B19			
			18B19			

Bay 2

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
606B 7	70B 6	1391B 6	0X 0	588B 5	0X 0	0X 0	0X 0	561B 4	0X 0	350B 4
208B 9	210B 7	64B 7	67B 7	632B 8	74B 8	46B 8	68B 9	56B 9	54B 9	668B10
		625B10	1101B10	49B11	52B11	66B12	636B13	40B13		
		600B14	584B14	69B14	623B15	62B15	73B15	691B16		
				71B16	41B17	51B17				
				688B18	36B18	50B18				
				42B19	45B19	53B19				
				48B19	633B20	570B20				
				57B20	37B20	55B20				
					39B20					

Bay 3

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	699B 5	590B 6	0X 0	0X 0	320B 6	0X 0	266B 8	0X 0	0X 0
593B 8	675B 8	353B 6	91B 6	617B 8	686B 5	84B 8	313B 4	113B 9	292B 5	65B 5
		87B 8	88B 8	114B11	86B 8	116B11	106B 8	125B 9		
		1204B14	92B10	1265B12	109B11	120B12	127B10	559B20		
			117B11	118B14	89B13	90B13	124B11			
			107B14	564B15	77B14	98B14	103B13			
			93B15	108B15	97B17	119B16	82B15			
			95B17	94B18	85B19	76B19	79B17			
				104B19	115B19	105B19				
				122B20	83B20	110B20				

Bay 4

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	434B 7	0X 0	203B 7	0X 0	660B 6	0X 0	255B 4	0X 0	0X 0	0X 0
637B 7	380B 6	368B 5	181B 7	479B 5	1011B13	241B 7	667B 7	333B 5	154B 7	132B 7	1284B 4	0X 0
		131B 7	128B 8	133B11	138B13	179B11	184B 8	171B 8	156B 8	589B 9		
		176B 9	164B10	555B12	183B14	157B13	170B11	134B11	151B10	137B 9		
			182B11	178B12	662B18	162B14	152B12	198B12	197B11			
			160B13	161B13	1357B19	173B18	174B14	169B13	159B13			
			150B14	165B15	139B19	149B18	155B17	187B16	136B14			
			145B18	129B18	143B19	140B19	166B19	141B18	142B18			
				185B19	571B20	196B19	193B19	148B19				
					177B20	130B20	168B20					

Bay 5

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	692B 4	0X 0	0X 0	0X 0	0X 0	0X 0	563B 4	0X 0	647B 6	0X 0
273B 6	282B 5	322B 8	288B 4	209B 9	218B 9	227B10	195B 8	189B 4	225B 6	219B 4	275B 8	690B17
		239B10	407B11	206B11	228B11	229B11	212B10	186B 6	235B 9	579B 9		
		605B20	634B13	237B13	585B14	232B13	245B11	250B 9	213B11	201B11		
			259B15	205B14	224B14	257B15	233B14	207B14	230B13			
			202B16	236B15	243B16	231B16	264B16	211B15	252B15			
			656B17	256B17	270B18	269B18	215B17	268B16	244B16			
			254B18	214B19	238B19	246B19	226B19	216B18	265B18			
				247B19	251B19	274B20	276B20	248B19				
				267B20	222B20	220B20	221B20	272B20				

Bay 6

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	386B 4
344B 4	121B 4	412B 6	305B 6	684B 8	1247B 8	349B 8	293B 7	341B 7	639B 7	311B 7	324B 7	329B 7
		339B11	355B11	695B10	295B10	342B10	340B10	325B 9	316B 9	359B 9		
		291B12	338B12	351B12	331B12	309B12	310B11	283B11	286B11	336B11		
		642B15	568B14	565B14	312B14	279B14	301B14	677B13	335B13	357B12		
		1361E16	289B15	303B15	294B15	287B15	326B15	334B15	343B15	332B15		
		567B16	347B16	318B16	306B16	361B16	300B16	284B16	278B16	299B16		
		562B19	352B18	315B18	297B18	307B18	290B17	328B17	358B17	285B17		
			672B19	302B19	317B19	321B19	319B19	314B19	330B19			
				604B20	356B20	337B20	281B20	308B20				

Bay 7

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
401B 6	0X 0	0X 0	0X 0	387B 6	2B 6	367B 6	376B 4	389B 6	399B 6	0X 0	75B 7	612B 7
		413B10	363B11	404B11	398B10	425B10	1027B 9	424B 9	397B 8	418B 7		
		680B12	379B12	144B11	406B11	414B11	438B11	442B11	432B11	423B11		
		364B13	408B13	419B13	439B13	443B13	437B13	390B13	371B12	392B12		
		678B15	566B15	374B15	382B15	394B15	1375B14	698B14	436B14	383B14		
		650B16	416B16	420B15	395B16	411B16	430B16	391B16	435B15	384B15		
		643B17	370B16	362B16	421B17	431B17	429B17	402B17	369B16	373B16		
		433B17	422B18	409B18	375B19	393B19	385B19	388B18	427B18	440B17		
			396B19	428B20	400B20	365B20	372B20	403B20	410B19			

Bay 8

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	348B 6	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	153B 4	0X 0
33B10	1360B 8	0X 0	472B 6	468B 8	72B 7	1077B 4	240B 5	38B 4	461B 6	101B 4	43B 5	490B 6
		0X 0	476B 7	649B 9	629B 9	483B 7	478B 7	514B 6	508B 8	462B 6		
		644B 9	450B 8	594B10	509B 9	506B 9	507B 9	444B 8	503B 9	449B 8		
		486B 9	477B 9	465B10	481B10	516B10	467B 9	485B 9	487B 9	497B 9		
		452B 9	489B 9	505B12	496B12	455B11	492B10	470B10	459B 9	466B 9		
		501B11	448B10	687B14	510B14	446B13	493B12	502B12	511B12	504B11		
		456B13	517B12	581B15	491B14	463B15	454B15	498B14	480B13	460B13		
		494B16	513B14	482B17	473B18	515B19	464B18	488B18	445B17	469B15		
			500B16	471B19	453B20	451B20	484B20	512B20	458B19			

```
Bay 9
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0   556B12     0X 0   377B 5   192B 5   200B 4   544B 4     0X 0     0X 0   474B 4    21B 6   682B13   640B10
                     194B 4   366B 5   126B 6   674B 4   447B 5   415B 4   405B 4   578B 4   260B 8
                     100B 6   609B 6  1024B 7   158B 6   652B 6   280B 5   495B 4   524B 4   223B 8
                     426B 6    60B 6   102B 8   175B 7  1376B 7   261B 6   199B 7   519B 4   531B10
                     525B 8    80B 6   671B12   529B11   527B 8   381B 7   648B11   457B 5   616B11
                     518B12   262B 8   610B16   701B15   576B14   146B 7   635B13   112B 5   574B12
                     580B18   624B15   613B16  1236B17   558B17   611B11   522B17   258B 6   679B14
                     681B18   602B15   526B17   572B18   603B18   608B15  1064B18   521B14  1214B15
                     659B20   530B16   586B18   645B19   523B19   520B18   554B20   528B15   598B20

Bay 10
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0   676B19     0X 0     0X 0     0X 0     0X 0    25B 4   417B 4   234B 4   441B 4   378B 4   626B 4   577B 4
                      47B 4  1275B 4   298B 4   631B 4   553B 4    96B 4   167B 4   253B 5   354B 5
                      78B 5   190B 5   180B 5   304B 5    61B 5    44B 6   546B 6   135B 6   263B 6
                     346B 7    19B 7   360B 7   545B 7   271B 7   575B 7   666B 7   670B 7   191B 8
                     599B 8   663B 8   537B 8   533B 8   277B 8   323B 8   327B 8   654B 9   685B 9
                     543B 9   534B 9    32B 9   653B 9   627B10   657B10   560B10   651B10   536B10
                    1238B11   535B11   532B11   541B11   619B12   540B12  1004B12    59B13   591B13
                     547B13   669B16  1390B16   622B17  1089B17   550B16   646B15   551B13  1277B13
                     694B19   700B20   587B19   655B18   542B20   538B19   549B17   539B17   548B14

Bay 11
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0  1069C 5     0X 0     0X 0     0X 0     0X 0
   601C 4   475C 4   217C 4  1262C 6   703C 6  1093C 7  1078C 5     0X 0   764C 6   704C 5     0X 0  1041C18   709C 4
                     729C 5   734C 7   737C 9   705C10   723C 7   706C 9   751C 6   722C 6   735C 4
                     721C 5     3C 8   760C11   750C11   748C10   719C11   746C 8   761C 8   755C 5
                     715C 7   716C11  1211C13  1034C12   749C12  1007C13   743C11   714C 8   708C 8
                     765C10   766C13  1017C14   702C14   730C13   717C13   707C13   757C11   767C10
                     725C13   718C14   726C14  1070C15   732C14  1202C15   741C15   739C13   754C13
                     768C14   724C16   752C16   713C15   733C15   762C16   756C15   731C14   728C14
                     710C15  1273C18  1044C19   769C16   712C17   630C19   759C16   747C16   711C15
                     738C17   740C18   758C20   745C20   720C20   742C20   753C20   736C18   763C17

Bay 12
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0   833C 4   789C 4   791C 4   798C 4   803C 4   821C 4   832C 4   829C 4   830C 5
                    1105C 6   607C 7  1383C 8   818C 7   822C 7   777C 7   787C 6   773C 6   788C 5
                    1038C 9   839C 8   820C 8   812C 8   809C 8   808C 8   802C 8   810C 8   823C 8
                     824C10   800C10   819C10   792C 9   781C 9   796C 9   838C 9   828C 9   806C 9
                    1016C13   790C13   804C12   813C12   784C12   696C11   785C11   772C11   811C11
                     596C15   697C15   817C15   775C15   552C14   836C14   783C13   780C13   826C13
                     831C17   770C16   779C16   793C16   776C16   837C16   771C15   795C15   835C15
                     569C18   794C18   778C17   797C18   827C18   774C17   814C17   825C17   834C17
                    1025C19   799C19   786C20   815C20   816C20   807C20   805C20   782C20   801C19

Bay 13
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0     0X 0
     0X 0   615C15  1229C 4   849C 4   848C 4  1102C 5     0X 0   296B 6   932C 4   916C 4   843C 4  1057C 7  1006C19
                     859C 4   878C 4  1092C 5   905C 8   658C 7   857C 7   877C 5   897C 4   891C 4
                     869C 7   872C 7   888C 8   976C 9   873C 8   882C 8   887C 6   883C 5   866C 5
                     902C 8   896C10   889C10   863C10   867C11   841C10   856C 8   890C 7   860C 7
                     881C11  1055C12  1294C12   898C12   884C12   874C12   893C10   880C 9   854C 9
                     845C12   855C12  1073C14  1084C13   875C12   861C13   885C12   903C12   840C12
                     997C16   894C14  1063C15   895C14   846C16   847C13   871C13   899C13   901C13
                     842C16   864C17   868C17   892C18   844C18   870C18   886C17   858C17   851C17
                     876C19   862C19   850C20   904C20   879C20   865C20   853C20   852C20   900C19
```

Bay 14

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	1043C 4	0X 0	1097C 9	0X 0	0X 0	0X 0	1028C 5	912B 5	1062C 9	26B 9
		925C 5	1079C 6	188C 4	970C11	0X 0	0X 0	917C 6	1047C 6	28C 7		
		592C 8	1099C 7	1061C 9	935C15	0X 0	1095C 5	909C 7	941C 7	910C 9		
		919C 8	937C 7	1052C11	1042C16	0X 0	952C 7	950C 8	954C 9	1087C 9		
		914C 9	959C 8	942C14	1030C17	946C 8	934C 7	915C 9	992C10	921C10		
		923C12	929C 9	1046C17	665C19	944C12	906C12	938C 9	922C10	930C10		
		618C17	939C11	1056C18	1100C19	927C16	920C15	1067C13	940C14	928C10		
		1012C17	926C13	641C19	908C19	1075C18	1023C17	943C16	945C17	931C12		
		597C20	936C18	933C19	1386E20	907C20	913C20	924C19	911C18	918C17		

Bay 15

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
		1326C C	1083C 4	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0		
		621C C	1022C 5	1029C 5	951C 5	727C 5	963C 5	1082C 4	948C 4	242B 4		
		1019C 9	1037C 6	1050C 6	583C 7	63C 9	1013C 9	949C 9	1380C 7	958C 9		
		1080C11	1032C11	1072C11	1058C12	1091C10	955C 9	1065C10	967C10	969C10		
		1081C14	1039C13	1074C13	962C13	957C14	956C14	960C14	1098C15	1053C16		
		1026C16	1256C16	964C16	966C16	1096C17	1094C17	1040C17	1066C18	1269D18		
		1005C19	620C19	1045C19	1020C19	1003C19	1048C19	968C19	947C19	1259D20		
			1088C20	1068C20	1015C20	1076C20	965C20	953C20	961C20			

Bay 16

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	1253D 4	111B 4	1222B 5
1287D10	673D18	1239D 8	1237D 8	1279D 8	1153D 8	1134D 8	1135D 8	1162D 8	1286D 7	1116D 7	1292D17	638D14
		1119D 8	1249D 9	1115D 9	1250D11	1010C11	1168D11	582B10	1136D 9	1117D 7		
		1104D11	1252D12	1122D12	1125D12	1128D12	1132D11	1139D11	1226D10	1118D 7		
		1254D13	1298D13	1244D13	1170D12	1160D12	1147D12	1140D12	1124D10	1138D 9		
		1156D13	1142D13	1111D13	1133D15	1155D15	1150D14	1171D14	1110D13	1149D13		
		1158D15	1106D16	1154D18	1129D17	1146D17	1127D17	1137D16	1169D16	1164D15		
			1161D18	1114D18	1131D19	1167D19	1141D19	1126D18	1148D17			
				1159D19	1152D20			1151D20				

Bay 17

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	99B 4
1144D 4	1203D 4	1166D 4	1179D 4	1113D 4	1210D 5	1121D 5	1293D 5	1189D 5	1109D 5	1163D 5	1255D 6	1120D 6
		1143D 6	1205D 7	1246D 9	744D 7	1280D 7	1263D 7	1123D 6	1108D 6	1130D 6		
		1173D 7	1276D 9	1270D 9	1289D 9	1251D 8	1285D11	1186D 7	1112D 6	1157D 6		
		1297D10	1223D10	1218D10	1215D12	1266D13	1399D14	1235D11	1212D 8	1281D 8		
		1185D10	1172D13	1290D12	1181D12	1267D17	1272D17	1033D11	1182D10	1176D10		
		1288D14	1184D14	1219D15	1174D15	1190D19	1216D18	1274D16	1183D15	1187D15		
			1188D17	1180D19	1282D20	1206D20	1283D18	1209D18	1177D16			
				1220D20	1175D20			1178D20				

Bay 18

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
1268D 4	1165D 4	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
		1291D 8	1240D 8	1199D 6	1197D 5	1195D 4	1201D 5	1257D 7	1241D 7	1264D 7		
		1301D 8	595D 8	1208D 9	1232D11	1145D 8	1107D 9	1230D 9	1258D 9	1103D12		
		1295D12	1242D13	1196D13	1193D13	1217D14	1278D14	1260D14	1231D14	1261D14		
		1198D14	1191D14	1367D15	1233D15	1213D16	1200D16	1248D17	1299D17	1224D18		
			1271D18	1243D18	1300D18	1192D18	1234D19	1245D19	1225D19			
				1194D19	1393D20	1207D20	1296D20	1221D20				
					661D20							

Bay 19

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	1008E 5	1401E 4	0X 0	0X 0	0X 0	0X 0	1337E 6	0X 0	1355E 6	0X 0	1318E11	0X 0
1372E20	1378E16	1364E 7	1389E 7	1370E 4	1329E 4	1316E 6	1309E 7	0X 0	1359E 8	0X 0	1387E20	1021E12
		1371E10	1368E 8	1366E 5	1308E 4	1381E 8	1352E 9	1388E 5	1336E 9	1311E 5		
		1002E14	1323E 9	1319E 8	1313E 5	1365E11	1305E11	1328E 7	1330E11	1369E12		
			1392E12	1331E11	1307E 7	1303E12	1356E17	1332E10	1314E14			
			1353E18	1317E13	1302E17	1306E17	1322E17	1325E17	1321E18			
			1333E18	1312E18	1327E19	1377E20	1304E19	1335E19	1315E20			
					1334E20	1324E20	1320E20					

Bay 20

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	0X 0	0X 0	0X 0	1362E 4	81B 4	1398E 5	1338E 5	1310E 5	664E 5	1227E 5
	994E 6	172E 7	1339E 7	995E 7	1340E 7	1349E 6	989E 6	979E 5	1000E 5	
	1382E11	999E11	1049E11	1385E10	1343E10	1054E 9	1354E 8	1348E 8	993E 8	
		1363E14	1358E14	1394E13	1341E13	1397E12	1373E12	1346E11		
		1379E17	1400E17	1345E17	1395E16	1396E16	1059E16	1344E15		
			1384E20	1374E18	1351E18	1347E17	1350E17			
					1342E20					

Bay 21

0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
0X 0	1031C 5	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0	0X 0
1051C17	1071C10	1018C 4	0X 0	1036C12	1009C 5	0X 0	980C10	1228C17
	1035C13	1090C 7	0X 0	1085C13	1060C 8	977C10	996C13	
	987C13	973C13	974C15	986C16	971C17	981C14	990C13	
		984C17	998C17	982C17	1014C19	983C16		
		988C19	972C19	1001C20	975C19	978C19		

Bay 22

991C 4	1086C 4	985C 5

References

ANDERSON 91 - E.J.Anderson and M.C.Ferris, “Genetic Algorithms for Combinatorial Optimization: The Assembly Line Balancing Problem”, Research Paper 11/91, Cambridge University Engineering Department, UK, 1991.

ANTONISSE 89 - J.Antonisse, “A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp86-91,1989.

AUSTIN 96 - S.A.Austin, A.N.Baldwin, T.Hassan and A.J.Newton, “Techniques for the Management of Information Flow in Building Design”, Information Processing in Civil and Structural Engineering Design, B.Kumar, I.A.Macleod and A.Retik (Eds.), Civil-Comp Ltd. Edinburgh Scotland, pp119-123, 1996.

BELEGUNDU 94 - A.D.Belegundu, R.R.Salagame and E.W.Constans, “Multi-Objective Optimization of Laminated Ceramic Composites Using Genetic Algorithms” Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimisation, Panama City, FL, pp10-15-1022, AIAA Inc., 1994.

BRIDGES 87 - C.L.Bridges and D.E.Goldberg, “An Analysis of Reproduction and Crossover in a Binary Coded Genetic Algorithm”, Proceedings of The Second International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp9-13, 1987.

BRUNS 93 - R.Bruns, “Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling”, Proceedings of the Fifth International Conference on Genetic Algorithms, pp352-359, Morgan Kaufmann Publishers, 1993.

BULL 97 - L.Bull, “Model-based Evolutionary Computing: A Neural Network And Genetic Algorithm Architechture”, Proceedings of the IEEE International Conference on Evolutionary Computation, pp611-616, 1997.

CHANG 95 - C.S.Chang, W.Wang, A.C.Liew, F.S.Wen and D.Srinivasan, “Genetic Algorithm Based Bicriterion Optimisation for Traction in DC Railway System”, IEEE International Conference on Evolutionary Computation (ICEC ‘95), Volume 1, pp11-16, Piscataway, NJ, 1995.

CHENG 96 - R.Cheng, M.Gen and Y.Tsujimura, "A Tutorial Survey of Job-Shop Scheduling using Genetic Algorithms - I.Representation", Computers and Industrial Engineering, Vol.30, No.4, pp983-997, 1996.

CHO 96 - K.K.Cho, K.R.Ryu and H.R.Choi, “A Scheduling System for Block Assembly Operations in Shipbuilding using Genetic Algorithms”, Applied Operations Research: Advances in Industrial Engineering Applications and Practice I, pp476-481, 1996.

CHU 89 - C-H.H.Chu, “A Genetic Algorithm Approach to the Configuration of Stack Filters”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp219-224,1989.

CIENIAWSKI 95 - S.E.Cieniawski, J.W.Eheart and S.Ranjithan, “Using Genetic Algorithms to Solve a Multiobjective Groundwater Monitoring Problem”, Water Resources Research, Vol.31, No.2, pp399-409, 1995.

CLEVELAND 91 - G.A.Cleveland and S.F.Smith, “Using Genetic Algorithms to Schedule Flow Shop Releases”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp160-169,1989.

CRAMER 85 - N.L.Cramer, “A Representation for the Adaptive Generation of Simple Sequential Programs”, Proceedings of The First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, pp183-187,1985.

DAVIS 85 - L.Davis, “Job Shop Scheduling with Genetic Algorithms”, Proceedings of The First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, pp136-140,1985.

DASGUPTA 92 - D.Dasgupta and D.McGregor, “Structured Genetic Algorithms: The Model and First Results”, Department of Computer Science, University of Strathclyde, UK, 1992.

DEB 89 - K.Deb and D.E.Goldberg, “An Investigation of Niche and Species Formation in Genetic Function Optimization”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp42-50,1989

DE JONG 75 - K.A.DeJong, “Analysis of the Behaviour of a Class of Genetic Adaptive Systems”, Doctoral Dissertation AAC7609381, University of Michigan, 1975.

DE JONG 89 - K.A.DeJong and W.M.Spears, “Using Genetic Algorithms to Solve NP-Complete Problems”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp124-132,1989.

DILLINGHAM 86 - J.T.Dillingham and A.N.Parakis, “Application of Artificial Intelligence in the Marine Industry”, Fleet Management Technology Conference, Boston, Mass, 1986.

EASTON 93 - F.F.Easton and N.Mansour, "A Distributed Genetic Algorithm for Employee Staffing and Scheduling Problems", Proceedings of the Fifth International Conference on Genetic Algorithms, pp360-367, Morgan Kaufmann Publishers, 1993.

FANG 93 - H-L.Fang, P.Ross and D.Corne, "A Promising Genetic Algorithm Approach To Job-Shop Scheduling, Re-Scheduling and Open-Shop Scheduling Problems", Proceedings of the Fifth International Conference on Genetic Algorithms, pp375-382, Morgan Kaufmann Publishers, 1993.

FILIPIC 95 - B.Filipic, “A Genetic Algorithm Applied to Resource Managament in Production Applications”, Evolutionary Algorithm for Management Applications, J.Biethan and V.Nissen (Eds.), pp101-111, Springer, Berlin, 1995.

FISHER 96 - K.A.Fisher, “The Application of Genetic Algorithms to Optimising the Design of an Engine Block For Low Noise”, Proceedings of IEE/IEEE International Conference: GALESIA 95, Publication No:414, pp18-22, 1995.

FONSECA 93 - C.M.Fonseca and P.J.Fleming, “Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization”, Proceedings of The Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-299-2, pp416-423, 1993.

FOURMAN 85 - M.P.Fourman, “Compaction of Symbolic Layout using Genetic Algorithms”, Proceedings of The First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp141-153, 1985.

GEBALA 91 - D.A.Gebala and S.D.Eppinger, “Methods for Analysing Design Procedures”, Design Theory and Methodology, DE-Vol.21, pp227-233, ASME 1991.

GEN 94 - M.Gen, Y.Tsujimura and E Kubota, "Solving Job Shop Scheduling Problems by Genetic Algorithm", IEE International Conference on Systems, Man and Cybernetics, Vol.2, pp1577-1582, 1994.

GERO 95 - J.S.Gero and S.J.Louis, “Improving Pareto Optimal Designs Using Genetic Algorithms”, Microcomputers in Civil Engineering, Vol.10, pp239-247, 1995.

GOLDBERG 85 - D.E. Goldberg and R.Lingle Jr., “Alleles, Loci and the Travelling Salesman Problem”, Proceedings of The First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp154-159, 1985.

GOLDBERG 87 - D.E.Goldberg and J.Richardson, “Genetic Algorithms with Sharing for Multimodal Function Optimization”, Proceedings of The Second International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp41-49, 1987.

GOLDBERG 89 - D.E.Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley, ISBN 0-201-15767-5, 1989.

GOLDBERG 89b - D.E.Goldberg, “Sizing Populations for Serial and Parallel Genetic Algorithms”, Proceeding of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, California, pp70-79, 1989.

GOLDBERG 91 - D.E.Goldberg, "Real Coded Genetic Algorithms, Virtual Alphabets, and Blocking", Complex Systems, vol 5, p141-167, 1991.

GOODMAN 86 - E.D.Goodman, R.C.Averill, W.F.Punch III, Y.Ding and B.Malott, “Design of Special-Purpose Composite Material Plates via Genetic Algorithms”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp3-9, 1996.

GREFFENSTETTE 85 - J.J.Greffenstette (ed.) “Proceedings of The First International Conference on Genetic Algorithms and Their Applications”, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

GREFFENSTETTE 85b - J.J.Greffenstette, R.Gopal, B.Rosmaita, D.VanGucht, “Genetic Algorithms for the Travelling Salesman Problem”, Proceedings of The First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp160-168, 1985.

GREFFENSTETTE 89 - J.J.Greffenstette and J.E.Baker, “How Genetic Algorithms Work: A Critical Look at Implicit Parallelism”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp20-27,1989.

GRIERSON 96 - D.E.Grierson, “Conceptual Design Using Emergent Computing Techniques”, Emergent Computing Methods in Design: NATO ASI Series F: Computer Systems and Sciences, Vol.149, Springer-Verlag, pp150-161, 1996.

HAJELA 92 - P.Hajela and C.Y.Lin, “Genetic Search Strategies in Multicriterion Optimal Design”, Structural Optimization 4, p99-107, 1992.

HAMADA 95 - K.Hamada, T.Baba, K.Sato and M.Yufu, "Hybridizing a Genetic Algorithm with Rule-Based Reasoning For Production Planning", IEEE Expert, pp60-67, October 1995.

HOLLAND 62 - J.H.Holland, “Outline for a Logical Theory of Adaptive Systems”, Joint Association of Computing Machinery, Vol.9, pp297-314, 1962.

HOLLAND 75 - J.H.Holland, “Adaption in Natural and Artificial Systems”, 1st Edition, University of Michigan Press, Ann Arbor, 1975. 2nd Edition, MIT Press, Cambridge, MA, 1992.

HOPFIELD 84 - J.J.Hopfield, “Neurons with Graded Response have Collective Computational Properties Like Those of Two State Neurons”, Proceedings of the National Academy of Sciences, Vol.81, pp3088-3092.

HORN 93 - J.Horn and N.Nafpliotis, “Multi-Objective Optimization using the Niched Pareto Genetic Algorithm”, IlliGAL Report No, 93005, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, IL, USA, 1993.

HULIN 91 - M.Hulin, “Analysis of Schema Distributions”, Proceedings of the Forth International Conference on Genetic Algorithms, pp204-209, Morgan Kaufmann Publishers, 1991.

HUSBANDS 96 - P.Husbands, I.Harvey, D.Cliff, A.Thompson and N.Jakobi, “The Artificial Evolution of Robot Control Systems”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp41-49, 1996.

HUANG 81 - C.L.Huang and K.Yoon (Eds.), “Multiple Attribute Decision Making Methods and Applications: a State of the Art Survey”, Springer Verlag, New York, 1981.

JACQUET-LAGREZE 82 - E.Jacquet-Lagreze and J.Siskos, “Assessing a Set of Additive Utility Functions for Multiple Criteria Decision Making, The UTA Method”, European Journal of Operational Research, Vol.10, pp151-164, 1982.

JOG 89 - P.Jog, J.Y.Suh and D.VanGucht, “The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Travelling Salesman Problem”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp110-114,1989.

KEANE 96 - A.J.Keane and S.M.Brown, “The Design of a Satellite Boom with Enhanced Vibration Performance Using Genetic Algorithm Techniques”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp107-113, 1996.

KEHAT 73 - E.Kehat and M.Shacham, “Chemical Process Simulation Programs, Part 2: Partitioning and Tearing of System Flowsheets”, Process Technology International, Vol.19, No.3, pp115-118, 1971.

KIM 96 - Y.K.Kim, Y.J.Kim and Y.Kim, “Genetic Algorithms for Assembly Line Balancing with Various Objectives”, Computers and Industrial Engineering, Vol.30, No.3, pp397-409, 1996.

KRAUSE 95 - M.Krause and V.Nissen, “On Using Penalty Functions and Multicriteria Optimisation Techniques in Facilities Layout”, from Evolutionary Algorithms in Management Applications, J.Biethan and V.Nissen(Eds.), Springer-Verlag, Berlin, ISBN 3-540-60382-4, 1995

KURSAWE 91 - F.Kursawe, “A Variant of Evolution Strategies for Vector Optimization”, Parallel Problem Solving For Nature, Volume 496 of Lecture Notes in Computer Science, pp193-197, Springer Verlag Berlin, 1991.

KUSIAK 91 - A.Kusiak and J.Wang, “Concurrent Engineering: Simplification of the Design Process”, Computer Applications in Production and Engineering: Integration Aspects, G.Doumeingts, J.Browne and M.Tomljanovich(Eds.), Elsevier Science Publishers (Holland), 1991.

LAWLER 85 - E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Schmoys, “The Travelling Salesman Problem”, John Wiley and Sons Ltd, ISBN 0 471 90413 9, 1985.

LEDET 70 - W.P.Ledet and D.M.Himmelblau, “Decomposition Procedures for the Solving of Large Scale Systems”, Advances in Chemical Engineering, Vol.8, pp 185-254, 1070.

LIANG 94 - S.J.Liang and J.M.Lewis, “Job Shop Scheduling Using Multiple Criteria”, Proceedings of the Joint Hungarian-British Mechatronic Conference, pp77-82, 1994.

LIEPINS 87 - G.E.Liepins and M.R.Hilliard, “Greedy Genetics”, Proceedings of The Second International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp90-99, 1987.

LIN 72 - S.Lin and B.W.Kernighan, “An Effective Heuristic Algorithm for the Travelling Salesman Problem”, Operations Research, pp498-516, 1972.

LIN 92 - C-Y. Lin and P.Hajela, "Genetic Algorithms in Optimization Problems with Discrete and Integer Design Variables", Engineering Optimization, vol 19, p309-327, 1992.

McCLELLAND 88 - J.McClelland and D.Rumelhart , “Parallel Distributed Processing: A Handbook of Models Programs and Exercises”, MIT Press, Cambridge, MA, USA, 1988.

MINSKY 69 - M.L.Minsky and S.A.Papert, “Perceptrons”, MIT Press, Cambridge, MA, USA, 1969.

MUHLENBEIN 88 - H.Muhlenbein, M.Georges-Schleuter and O.Kramer, “Evolution Algorithms in Combinatorial Optimization”, Parallel Computing 7, pp65-85, 1988.

MUTH 63 - J.F.Muth and G.L.Thompson, “Industrial Scheduling”, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1963.

MURATA 94 - T.Murata and H.Ishibuchi, “Performance of Genetic Algorithms for Flowshop Scheduling Problems”, Proceedings of the IEEE Conference on Evolutionary Computation, pp289-294, 1995.

MURATA 96 - T.Murata, H.Ishibuchi and H.Tanaka, “Multi-Objective Genetic Algorithm and it’s Application to Flowshop Scheduling”, Computers in Industrial Engineering, Vol.30, No.4, pp957-968, 1996.

NAKANO 91 - R.Nakano and T.Yamada, "Conventional Genetic Algorithm for Job Shop Problems", Proceedings of the Fourth International Conference on Genetic Algorithms, pp474-479, Morgan Kaufmann Publishers, 1991.

NISSEN 95 - V.Nissen and J.Biethan , “An Introduction to Evolutionary Algorithms”, from Evolutionary Algorithms in Management Applications, J.Biethan and V.Nissen (eds.), Springer Verlag, ISBN 3-540-60328, 1995.

OLIVER 97 - I.M.Oliver, D.J.Smith and J.R.C.Holland, “A Study of Permutation Crossovers on the Travelling Salesman Problem”, Proceedings of The Second International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp225-230, 1987.

PARETO 1896 - V.Pareto, “Cours d’Economie Politique”, 1. Lausanne, Switzerland: F.Rouge, 1896.

PARSAEI 93 - H.R.Parsaei and W.G.Sullivan, “Concurrent Engineering - Contemporary Issues and Modern Design Tools”, Chapman and Hall, ISBN 0-412-46510-8, 1993.

POLONI 96 - C.Poloni, M.Fearon and D.Ng, “Parallelisation of Genetic Algorithm for Aerodynamic Design Optimisation”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp59-64, 1996.

RATCLIFFE 87 - A.T.Ratcliffe and P.Sen “Computer Aided Stowage and Lashings Strategies for Containerships”, Cargo Systems: Safe Ship-Safe Cargo Conference Proceedings, pp113-117, London, 1987.

REEVES 95 - C.R.Reeves, "A Genetic Algorithm for Flowshop Scheduling", Computers in Operations Research, Vol.22, No.1, pp5-13, 1995.

RICHARDSON 89 - J.T.Richardson, M.R.Palmer, G.Liepins and M.Hilliard, “Some Guidelines for Genetic Algorithms with Penalty Functions”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp191-197,1989.

RICHE 95 - R.G.Riche, C.Knopf-Lenoir and R.T.Haftka, “A Segregated Genetic Algorithm for Constrained Structural Optimization”, Proceedings of The Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-370-0, pp558-565, 1989.

RITZEL 94 - B.J.Ritzel, J.W.Eheart and S.Ranjithan, “Using Genetic Algorithms to Solve a Multiple Objective Groundwater Pollution Containment Problem”, Water Resource Research, Vol.30, No.5, pp1589-1603, 1994.

ROSENBLATT 58 - F.Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Retrieval in the Brain”, Computational Neuroscience Symposium, Indiana University-Perue University at Indianapolis, pp.73-78.

ROY 96 - R.Roy, I.C.Parmee and G.Purchase, “Integrating the Genetic Algorithm with the Preliminary Design of Gas Turbine Blade Cooling Systems”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp228-235, 1996.

RUMMELHART 86 - D.Rumelhart and J.McClelland, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", MIT Press, Cambridge, MA, USA, Vols. 1 and 2, 1986.

RUMMELHART 86b - D.Rumelhart G.E.Hinton and R.J.Williams, "Learning Representations by Back-Propagation Errors", Nature, Vol.232, No.9, pp533-566, 1986.

SAGINAW 89 - D.J.Saginaw and A.N.Parakis, "A Decision Support System for Containership Stowage Planning", Marine Technology, Vol.26, No.1, pp47-61, 1989.

SCHAFFER 85 - J.D.Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms", Proceedings of The First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp93-100, 1985.

SEN 88 - P.Sen and T.Busfield "Computer Assisted Container Stowage Planning", International Conference on Maritime Communications and Control, Institute of Marine Engineers, London, Paper 9, 1988.

SEN 94 - P.Sen and J.B.Yang, "Design Decision Making Based upon Multiple Attribute Evaluation and Minimal Preference Information", Mathematical and Computer Modelling, Vol.20, pp107-124, 1994.

SEN 95 - P.Sen and J.B.Yang, "Multiple Criteria Decision Making in Design Selection and Synthesis", Journal of Engineering Design, Vol.6, No.3, pp207-230, 1995.

SHA 85 - O.P.Sha, "Computer Aided on Board Container Management" from P.Banda and C.Kuo(eds.), Computer Applications in the Automation of Shipyard Operation and Ship Design V, pp177-187, Elsevier Science Publishers B.V. (Holland), 1985.

SHIELDS 84 - J.J.Sheilds, "Containership Stowage: A Computer Aided Preplanning System", Marine Technology, Vol.21, No.4, 370-383, 1984

SOUTH 93 - M.C.South, G.B.Wetherill & M.T.Tham, “Hitch-Hikers Guide to Genetic Algorithms”, Journal of Applied Statistics, Vol.20, No.1, pp153- 174, 1993.

SRIDHAR 96 - J.Sridhar and C.Rajebdran, "Scheduling in Flowshop and Cellular Manufacturing Systems with Multiple Objectives - A Genetic Algorithmic Approach", Planning Production and Control, Vol.7, No.4, pp374-382, 1996.

SRINIVAS 94 - M.Srinivas and Lalit M.Patnaik “Genetic Algorithms : A Survey”, IEEE Computer, June, 1994.

SRINIVAS 95 - N.Srinivas and K.Deb, “Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms”, Evolutonary Computation, Vol.2, No.3, pp221-248, 1995.

STARKWEATHER 91 - T.Starkweather, S.McDaniel, K.Mathias, D.Whitley and C.Whitley, “A Comparison of Genetic Sequencing Operators”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-308-9, pp69-76, 1991.

STEWARD 81 - D.V.Steward, “Systems Analysis and Management - Structure Strategy and Design”, Petrocelli Books inc., USA, ISBN 0-89433-106-X, 1981.

STEWARD 81b - D.V.Steward, “The Design Structure System: A Method for Managing the Design of Complex Systems”, IEEE Transactions on Engineering Management, Vol.EM-28, No.3, August 1981.

STEWART 92 - T.J.Stewart, “A Critical Survey on the Status of Multiple Criteria Decision Making and Practice”, International Journal of Management Science, Vol.20, No5/6 pp569-586, 1992.

SUBRAMANI 96 - D.Subramani and P.Sen, “Ship Subdivision Using Genetic Algorithm”, Proceedings of Adaptive Computing in Engineering Design and Control, University of Plymouth, ISBN 0-905227-61-1, pp152-158, 1996.

SUH 87 - J.Y.Suh and D.VanGucht, “Incorporating Heuristic Information Into Genetic Search”, Proceedings of The Second International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ, pp100-107, 1987.

SYSWERDA 91 - G.Syswerda and J.Palmucci, "The Application of Genetic Algorithms to Resource Scheduling", Proceedings of the Forth International Conference on Genetic Algorithms, pp502-508, Morgan Kaufmann Publishers, 1991.

TANOMARU 95 - J.Tanomaru, "Staff Scheduling by a Genetic Algorithm with Heuristic Operators", IEEE Conference on Evolutionary Computing, Vol.1, pp456-461, 1995.

THOMAS 95 - G.M.Thomas, R.Gerth, T.Velasco and L.C.Rabelo, “Using Real-Coded Genetic Algorithms for Weibull Parameter Estimation”, Computers and Industrial Engineering, Vol.29, pp377-381, 1995

TODD 97 - D.S.Todd and P.Sen, “A Multiple Criteria Genetic Algorithm for Containership Loading”, Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-487-1, pp674-681, 1997.

TODD 97b - D.S.Todd and P.Sen, “Multiple Criteria Scheduling Using Genetic Algorithms in a Shipyard Environment”, International Conference on Computer Applications in Shipbuilding (ICCAS 97), Yokohama, Japan, October 13-17, 1997.

TSOI 95 - E.Tsoi, K.P.Wong and C.C.Fung, “Hybrid GA/SA Algorithms for Evaluating Trade-off Between Economic Cost and Environmental Impact in Generation Despatch”, IEEE International Conference on Evolutionary Computation, Vol.1, pp132-137, Piscataway, NJ, 1995.

TSUJIMURA 97 - Y.Tsujimura, R.Cheng and M.Gen, "Improved Genetic Algorithms for Job-Shop Scheduling Problems", Engineering Design and Automation, Vol.3, No.2, pp133-144, 1997.

UNGER 93 - R.Unger and J.Moult, “Genetic Algorithm for 3D Protein Folding Simulations”, Proceedings of The Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-299-2, pp581-588, 1993.

WEBSTER 69 - W.C.Webster and P.VanDyke “Automated Procedures for the Allocation of Containers Shipboard”, Hydronautics Inc., Laurel, Md, 1969

WEBSTER 70 - W.C.Webster and P.VanDyke, “Containership Loading: A Containership Allocation Model”, Computer-Aided Ship Design Engineering Summer Conference, University of Michigan, Ann Arbor, 1970.

WERBOS 74 - P.J.Werbos, “Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences”, PhD Thesis, Havard University, Cambridge, MA, USA, 1974.

WHITLEY 89 - D.Whitley, T.Starkweather and D.Fuquay, “Scheduling Problems and the Travelling Salesman: The Genetic Edge Recombination Operator”, Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kaufmann, ISBN 1-55860-066-3, pp133-140, 1989.

WRIGHT 92 - A.H.Wright, "Genetic Algorithms for Real Parameter Optimization", Proceedings of a Workshop on the Foundations of Genetic Algorithms, p205-218, 1991.

ZHANG 95 - Y.Zhang, “Lecture Notes on Back Propagation Neural Networks”, Department of Marine Technology, University of Newcastle, UK.