

Using Genetic Algorithms for Practical Multi-Objective Production Schedule Optimisation.

Thesis submitted for the candidature of Ph.D.

Katharine Jane Shaw

August 1997

Department of Automatic Control & Systems Engineering

The University of Sheffield,

Mappin Street, Sheffield, S1 3JD

South Yorkshire, United Kingdom.

SUMMARY

Production scheduling is a notoriously difficult problem. Manufacturing environments contain complex, time-critical processes, which create highly constrained scheduling problems. Genetic algorithms (GAs) are optimisation tools based on the principles of evolution. They can tackle problems that are mathematically complex, or even impossible to solve by traditional methods. They allow problem-specific implementation, so that the user can develop a technique that suits the situation, whilst still providing satisfactory schedule optimisation performance.

This work tests GA optimisation on a real-life scheduling application, a chilled ready-meal factory. A schedule optimisation system is required to adapt to changing problem circumstances and to include uncertain or incomplete information. A GA was designed to allow successive improvements to its effectiveness at scheduling. Three objectives were chosen for minimisation. The GA proved capable of finding a solution that attempted to minimise the sum of the three costs. The GA performance was improved after experiments showed the effects of rules and preference modelling upon the optimisation process, allowing 'uncertain' data to be included.

Multi-objective GAs (MOGAs) minimise each cost as a separate objective, rather than as part of a single-objective sum. Combining Pareto-optimality with varying emphasis on the conflicting objectives, a set of possible solutions can be found from one run of MOGA. Each MOGA solution represents a different situation within the factory, thus being well suited to a constantly changing manufacturing problem.

Three MOGA implementations are applied to the problem; a standard weighted sum, two versions of a Pareto-optimal method and a parallel populations method. Techniques are developed to allow suitable comparison of MOGAs. Performance comparisons indicate which method is most effective for meeting the factory's requirements. Graphical and statistical methods indicate that the Pareto-based MOGA is most effective for this problem. The MOGA is demonstrated as being a highly applicable technique for production schedule optimisation.

Acknowledgements

I would like to thank Professor Peter Fleming for his supervision, encouragement and provision of this research opportunity. Many thanks also go to my colleagues in the Department of Automatic Control & Systems Engineering, particularly Carlos Fonseca for the many discussions and explanations of MOGAs, Andy Chipperfield for the great deal of advice on GAs, and research in general, and Sheena MacKenzie for her continual support. All the other members of the Research Group deserve a mention for their help and encouragement.

Many thanks to Pennine Foods ¹, for letting me use their data, in particular Ian Davenport and Richard Middleton for their help with this project and detailed explanations of the chilled ready meal / food industry, and difficulties of practical scheduling.

My thanks go to my parents, Pamela and Bob Shaw, and my brother Geoffrey, for everything they have done to support me. Thanks to Rick Cronan for all his help, - and not least for the extensive use of his PC for writing this thesis - and to all my other friends who have been so supportive throughout.

Cath McBride deserves many thanks for typing revisions and helping with proof-reading and formatting in the final version of this thesis.

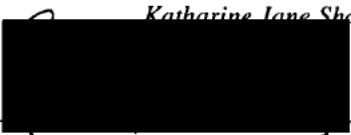
This research was financially supported by the Engineering and Physical Sciences Research Council.

¹ Pennine Foods is a Northern Foods Company

Statement of Originality

Unless otherwise stated in the text, the work described in this thesis was carried out solely by the candidate. None of this work has already been accepted for any other degree, nor is it concurrently submitted in candidature for any other degree.

Candidate: 

Supervisor:  *Katharine Jane Shaw*

Peter Fleming

TABLE OF CONTENTS

SUMMARY	2
ACKNOWLEDGEMENTS	3
STATEMENT OF ORIGINALITY	4
1. INTRODUCTION	5
1.1 INTRODUCTION TO THESIS	5
1.2 OUTLINE OF THESIS	6
1.3 CONTRIBUTIONS	7
1.3.1 MAIN CONTRIBUTIONS	7
1.3.2 ADDITIONAL CONTRIBUTIONS	8

2. INTRODUCTION TO GENETIC ALGORITHMS AND PRODUCTION SCHEDULING

9

2.1 INTRODUCTION	9
2.1.1 INTRODUCTION TO SCHEDULING	9
2.1.2 SCHEDULING PROBLEMS IN THEORY	10
2.1.3 SCHEDULING IN PRACTICE	11
2.1.4 SCHEDULING METHODS	15
2.1.5 LITERATURE ON COMPARISONS OF GAS WITH OTHER METHODS	19
2.1.6 SUMMARY	20
2.2 SURVEY OF PREVIOUS RESEARCH ON EVOLUTIONARY COMPUTATION FOR SCHEDULING	21
2.2.1 INTRODUCTION TO GENETIC ALGORITHMS	21
2.2.2 DEFINITION OF GENETIC ALGORITHMS	22
2.2.3 USE OF GENETIC ALGORITHMS FOR SCHEDULING PROBLEMS	25
2.2.4 IMPLEMENTATION OF GENETIC ALGORITHM FOR PRODUCTION SCHEDULING PROBLEMS	26
2.3 APPLICATIONS OF GENETIC ALGORITHMS TO SCHEDULING PROBLEMS	30
2.3.1 THEORETICAL PROBLEMS	30
2.3.2 REAL LIFE APPLICATIONS	30
2.3.3 CURRENT RESEARCH ISSUES ON GAS FOR SCHEDULING	31
2.3.4 DISTRIBUTED GAS	32
2.4 CONCLUSION OF THIS CHAPTER	34

3. INITIAL APPLICATION OF A GENETIC ALGORITHM TO A REAL LIFE SCHEDULING PROBLEM	36
3.1 INTRODUCTION TO CHAPTER 3	36
3.2 A REAL-LIFE SCHEDULING ENVIRONMENT	36
3.2.1 INTRODUCTION TO CHILLED READY MEALS MANUFACTURING	36
3.2.2 PROBLEM BACKGROUND	37
3.2.3 MODELLING ISSUES	38
3.2.4 RUN TIME ISSUES	39
3.3 INITIAL PROBLEM BASED ON PENNINE FOODS FACTORY	40
3.3.1 AIM OF WORKING ON THIS PROBLEM	40
3.3.2 PROBLEM SPECIFICATION	40
3.3.3 OBJECTIVES	41
3.3.4 PROBLEM SUMMARY FOR INITIAL GA IMPLEMENTATION	41
3.4 APPLYING THE GENETIC ALGORITHM	42
3.4.1 CODE USED	43
3.4.2 ALGORITHM DESIGN	43
3.4.3 OBJECTIVE FUNCTION	44
3.4.4 IMPLEMENTATION OF A SCHEDULE BUILDER	45
3.4.5 OPERATOR CHOICE FOR RECOMBINATION AND MUTATION	45
3.5 RESULTS AND CONCLUSIONS FOR INITIAL SCHEDULING PROBLEM	46
3.5.1 TEST A - OPERATOR PERFORMANCE	46
3.5.2 TEST B - POPULATION SIZE	46
3.5.3 TEST C - NUMBER OF GENERATIONS	47
3.5.4 TEST D - VISUAL EVALUATION OF SCHEDULES	47
3.6 CONCLUSIONS	48

4. INVESTIGATION INTO USING SCHEDULE BUILDERS IN GENETIC ALGORITHMS FOR PRODUCTION SCHEDULING **49**

4.1 INTRODUCTION	49
4.1.1 GA DESCRIPTION	49
4.1.2 REPRESENTATION USED IN THIS WORK	50
4.2 GENETIC ALGORITHMS AND SCHEDULE BUILDERS	50
4.2.1 BACKGROUND TO SCHEDULE BUILDING	50
4.2.2 SCHEDULE BUILDER IMPLEMENTATION	52
4.2.3 GENETIC ALGORITHM USED	53
4.3 METHODS OF LINE ASSIGNMENT	54
4.3.1 HEURISTIC PREFERENCES	54
4.3.2 PRE-EXPRESSED PREFERENCES	55
4.3.3 ESTIMATED FITNESSES	55
4.4 EXPERIMENTS ON THE PERFORMANCE OF SCHEDULE BUILDER	56
4.4.1 EXPERIMENT 1 EXAMINING THE AVAILABLE FITNESS DISTRIBUTION ON MANY INDIVIDUALS FOR EACH SCHEDULE BUILDER	57
4.4.2 EXPERIMENT 2 - LEVEL OF VARIATION IN FITNESS ASSIGNMENT TO A SINGLE INDIVIDUAL	57
4.4.3 EXPERIMENT 3 - PERFORMANCE WITHIN THE GA	57
4.4.4 EXPERIMENT 4 - GENERAL CASE OF SCHEDULE BUILDER WITHIN GA	58
4.4.5 EXPERIMENT 5 - USING REAL-LIFE DATA IN FORM OF BEST SCHEDULE BUILDER	58
4.5 RESULTS	58
4.5.1 EXPERIMENT 1	58
4.5.2 EXPERIMENT 2	60
4.5.3 EXPERIMENT 3	60
4.5.4 EXPERIMENT 4	62
4.5.5 EXPERIMENT 5	63
4.6 CONCLUSION	64

5. GENETIC ALGORITHMS FOR MULTI-OBJECTIVE OPTIMISATION 67

5.1 INTRODUCTION	67
5.1.1 FINDING A TECHNIQUE TO WORK IN THE REAL WORLD	68
5.1.2 MULTIPLE OBJECTIVE PROBLEM SOLVING - OPTIMISATION AND DECISION- MAKING	69
5.1.3 METHODS USED FOR MULTI-OBJECTIVE OPTIMISATION BY GA.	71
5.2 WEIGHTED SUM METHOD	72
5.2.1 BACKGROUND	72
5.2.2 IMPLEMENTATION	73
5.3 PARALLEL POPULATIONS	74
5.3.1 BACKGROUND	74
5.3.2 ADDITIONAL MOTIVATION FOR THE USE OF PARALLEL POPULATIONS	77
5.3.3 IMPLEMENTATION OF <i>PARASOGA</i>	79
5.4 PARETO-BASED METHODS	82
5.4.1 BACKGROUND	82
5.4.2 IMPLEMENTATION	85
5.4.3 ENHANCEMENTS TO MO1	86
5.5 OTHER MOGAS FOR SCHEDULING	87
5.6 SUMMARY OF CHAPTER 5	89

<u>6. COMPARISON OF THREE MULTI-OBJECTIVE OPTIMISATION GA SCHEMES FOR A REAL LIFE SCHEDULING PROBLEM</u>	90
6.1 INTRODUCTION	90
6.2 BACKGROUND TO COMPARISONS OF MOGAS	90
6.2.1 VISUAL COMPARISON OF MINIMA	91
6.2.2 STATISTICAL COMPARISONS	93
6.3 COMPARISON OF THREE MOGA METHODS	94
6.3.1 INTRODUCTION TO EXPERIMENTS	94
6.3.2 QUALITY OF SOLUTIONS IN TERMS OF OPTIMISATION	94
6.3.3 DISTRIBUTION OF SOLUTIONS ACROSS THE TRADE OFF SURFACE	96
6.3.4 ROBUSTNESS AND USE OF STATISTICAL PERFORMANCE ASSESSMENT (SPA)	99
6.3.5 DISCUSSION OF SPA RESULTS	106
6.4 STATISTICAL COMPARISONS OF MOGA METHODS	107
6.4.1 INTRODUCTION	107
6.4.2 USE OF THE SPA FOR STATISTICAL COMPARISONS	107
6.4.3 STATISTICAL TEST	110
6.4.4 RESULTS OF TEST IMPLEMENTATIONS	110
6.4.5 DISCUSSION OF STATISTICAL COMPARISON	115
6.5 DISCUSSION AND CONCLUSIONS	115

7. DISCUSSION AND CONCLUSIONS **118**

7.1 RESTATEMENT OF OBJECTIVES	118
7.2 SUMMARY OF THESIS	118
7.3 DISCUSSION OF RESULTS	120
7.4 FURTHER WORK	122
7.4.1 PROBLEM REPRESENTATION USING SCHEDULE BUILDERS	122
7.4.2 MULTI-OBJECTIVE OPTIMISATION	123
7.4.3 PRACTICAL USE OF GAS FOR SCHEDULING	123
7.5 CONCLUSION	124

A.1. STATISTICAL TEST	A-1
A.2. PERMUTATION TEST	A-1
A.3. CHOOSING A TEST STATISTIC	A-1
A.4. COMPUTING THE TEST STATISTIC FROM THE ORIGINAL PROBLEM	A-2
A.5. EVALUATION OF P-VALUE	A-3
A.6. EXTENSION TO MULTIPLE COMPARISON PROBLEMS	A-3

1. INTRODUCTION

'Observe always that everything is the result of change, and get used to thinking that there is nothing Nature loves so well as to change existing forms and to make new ones like them.'

– Marcus Aurelius, Roman Emperor.

1.1 Introduction to Thesis

In nature, evolution is the process of individuals in a population adapting over time to meet the demands of an environment. Manufacturing provides a particularly demanding environment for a problem-solver, and within this environment, schedules must be found which meet these high demands. It is therefore perhaps appropriate that the principles of evolution are applied to manufacturing problems in the form of genetic algorithms, in which possible solutions to problems adapt according to an evolution-like model to generate optimal answers.

This thesis will explore the use of genetic algorithms (GAs) to find optimal solutions to scheduling problems. Within the broad definition of GAs, many implementations are possible. From the basic principle of taking a population of possible solutions to a problem, repeatedly evaluating them, and combining the better parts of them to create new solutions, there are as many variations possible within the computational model as there are in nature itself. GAs can work with ill-defined data, dynamic problem information, complex or multiple objectives; all of these are common in manufacturing problems. So long as feasible solutions can be modelled - no matter how poor they might be - and the users' requirements from a good solution can be measured, the method can be implemented to solve problems that have proved extremely difficult using other methods.

Real-life manufacturing environments face demands from many directions. The customers need products by a certain time, costs must be kept down to provide a profitable business, staffing requirements must be met, and competitors matched or beaten. If a factory can minimise monetary cost, it will also want to minimise time wasted; if it can minimise the number of unhappy customers, it will also want to minimise the number of unhappy staff. Scheduling production to meet multiple and conflicting objectives is a vital requirement in any factory. Methods that allow the scheduling to be performed optimally offer great potential for the user.

Such capabilities differ from many of those found in traditional methods, which may only have found a single solution, or may have required particular data to be formulated in a certain way. Further difficulties are presented by the need for fast rescheduling in the face of slow scheduling systems; if a scheduling system is to provide a solution, the solution must be able to deal with changed orders, priorities or machine availability.

Therefore, the opportunity to use the flexible and effective genetic algorithm technique is a contemporary research problem that offers great advantages to industry if it is successful.

This thesis demonstrates that GAs present the opportunity to provide both the inspiration and power to find some highly flexible solutions to formerly difficult or even impossible problems, such as those found in a real-life manufacturing problem. It uses such problems not only to demonstrate GA implementation and performance, but also to motivate development of GA methods for solving multi-objective scheduling problems. In addition, it is shown how GAs offer far more than simple solution-finding, but can incorporate preferences, conflicting and changing objectives, real-life knowledge, and reactions to a dynamic environment, to become a powerful decision-making tool.

For any complex real-life problem, there is rarely one set solution that meets all demands. Equally, there is unlikely to be one single method that finds the best solution to any generic scheduling problem. The input and interactivity of the user with the system to update information, provide additional preferences or changed information should be incorporated to allow both satisfactory compromise and flexibility of a system. The context of the problem indicates both the best method and solution for a particular situation. This theme is explored throughout the thesis.

1.2 Outline of Thesis

The thesis is arranged as follows. The remainder of **Chapter 1** introduces the thesis and outlines its content. **Chapter 2** gives the basic background to the main areas covered in this thesis, referring to relevant literature in the field and introducing topics discussed in the subsequent chapters. **Chapter 3** provides a more specific description of the real-life application and its related problems which provide a practical context for the aims of this thesis. This is explored by the experimental work later on, whilst demonstrating an initial implementation of the use of a simple GA for this problem. **Chapter 4** explores in more depth the problem of using real-life data within a computerised optimisation system, and introduces the idea of preferences and the uncertainty in a real-life problem. **Chapter 5** takes the idea of user preferences further by introducing the idea of multi-objective genetic algorithm (MOGA) optimisation in some detail, and provides justification for its use, together with three methods in which we may realise this goal. **Chapter 6** compares the performance of these methods on the problem described in Chapters 3 and 5, and draws conclusions on the experimental evidence provided by these comparisons. It also initiates the development of a statistical method for comparing MOGA performances. **Chapter 7** draws this thesis to a close with a discussion of the main contributions and implications of this work, concluding on potential areas for further work in this field.

1.3 Contributions

1.3.1 Main contributions

The main contributions of this thesis are:

- ***A study of the application of evolutionary computation techniques to real-life scheduling problems.*** Whilst the use of genetic algorithms for solving scheduling problems has been a known technique for over ten years now, since Davis (1985) introduced the method, many methods of applying GAs to scheduling problems have been developed on test functions and job-shop benchmarks. This practice has been changing in recent years as researchers attempt to exploit the advantages of GA techniques for difficult, real-life problems that cannot be solved by traditional methods for schedule optimisation, such as those found in Operations Research. Whilst there is still no generic method for applying GAs to scheduling problems, many techniques from the GA field offer themselves to further development for tackling the specific difficulties of scheduling. This thesis considers these problems from the point of view of a real-life manufacturer. Chapter 3 begins the development of these techniques, and they are modified and extended in subsequent chapters.
- ***The development of objective function implementation that allows real-life scheduling preferences to be used in a schedule optimisation system.*** Genetic algorithms for scheduling are frequently implemented so that the information describing the factory situation is combined with the search power of the GA in order to find accurate and optimal schedules. Chapter 4 discusses and develops a method for allowing this to be done in a realistic way, allowing the inclusion of a degree of uncertainty into the process to reflect the way in which the situation is likely to be seen in real life. (Shaw and Fleming, 1997b).
- ***The application of multi-objective GA methods to a real-life scheduling problem.*** In keeping with the aim of studying practical difficulties faced by schedulers, an optimisation method used for scheduling should be able to find solutions that allow multi-objective optimisation of schedules. Scheduling requirements of manufacturers are very rarely clear-cut, single objective problems, and the development of new optimisation methods needs to reflect this. This thesis discusses the need for multi-objective optimisation GAs in scheduling, and develops three techniques, which might be suitable for providing this. They are developed and compared in Chapters 5 and 6. (Shaw and Fleming, 1996a, 1996b)
- ***The theme of the need for schedule optimisation techniques to allow interactivity, uncertainty and flexibility for the user.*** Whether the system is allowing the manufacturer to incorporate a degree of uncertainty into the model of the factory, (Chapter 4) or providing the opportunity for the manufacturer to change the problem constraints or objectives (Chapter 5), or even choose from a selection of potential schedule solutions which are designed to emphasise different factory requirements (Chapter 6), the manufacturer who is meeting today's customer demands requires a large amount of flexibility to be able to keep up with the rapidly

changing customers' demands. This flexibility must be included in a schedule optimisation system to allow manufacturers' needs to be met realistically. (Shaw and Fleming, 1997a, Shaw and Fleming, 1997c.)

- ***The development and discussion of the need for comprehensive comparisons of MOGA methods when applied to real-life scheduling problems.*** When considering the context of a real-life manufacturing problem, factors beyond simple optimisation performance must be taken into account when a user is choosing which method is best for solving a multi-objective optimisation problem. Chapter 6 explores this idea, and introduces the application of statistical comparison methods for assessing MOGA performances on scheduling problems.

1.3.2 Additional Contributions

The following additional contributions are a result of the work presented in this thesis:

- A toolbox of MATLAB functions developed to provide all the computational results in this thesis. The implementation is designed to allow the user to experiment with a wide range of available GA techniques, and to incorporate a number of different problems based on a real-life manufacturer.
- The creation and administration of Internet resources devoted to genetic algorithms in scheduling. This includes an email list (GAScheduling List), which has over 300 subscribers world-wide, and a World Wide Web area to present various details from the list and from this particular work.
- An experimental trial in conjunction with Pennine Foods[†], a local manufacturer of the GA scheduling system presented in this thesis upon the data which would be used in their daily scheduling.

[†] Pennine Foods is a Northern Foods Company

2. Introduction to Genetic Algorithms and Production Scheduling

'Though Mr. Fulkerson encountered scepticism within Deere, it was no match for the overall eagerness to cure the bottlenecks at the seeding division. A pilot system was developed on a PC adjacent to the loading dock. By last year, a former shop worker named Larry DeClerck was cranking up the software before going home each day, leaving the machine to 'breed' more than 600,000 schedules, each an improvement over the last. Planters now flow smoothly throughout the assembly line, with monthly output up sharply. Overtime has nearly vanished.'

(Petsinger, 1995.)

2.1 Introduction

This research focuses upon real life manufacturing scheduling problems, and the application of genetic algorithms (GAs) to provide schedule optimisation.

Chapter Two introduces the relevant areas of scheduling, GAs, and some of the specific techniques used in the application of GAs to scheduling problems. In the first section, (2.1), scheduling problems are introduced, with a focus on the practical issues faced in industry. This provides the motivation for subsequent work in this thesis. In section (2.2), the field of genetic algorithms is surveyed, and the parts of this rapidly expanding area that are relevant to scheduling are discussed. In section (2.3), the two areas are related, as the application of genetic algorithms to scheduling problems is discussed.

Both scheduling and GAs are extensive subjects for any survey, and this is by no means an exhaustive discussion of either. Davis (1985) first applied the emerging techniques of GA optimisation to a simple job-shop scheduling problem. Since then, research into applying GAs to scheduling problems has grown so much that it is unlikely that a literature search in these areas would cover every contribution to the extensive field in this chapter. The aim, therefore, is to cover the key contributions, current interests and new directions of relevance, to provide context for the research presented in later chapters.

2.1.1 Introduction to Scheduling

Manufacturers make products to meet a demand from customers. Time limits must be set on a date or time when a customer will receive the goods ('*due date*'), and the manufacturer must work so that meeting this order is economically sensible. Good planning is needed to ensure many customers' due dates can be met during a set period of production, whilst making the most of factory resources. The use of scheduling allows these objectives to be met.

Scheduling is the process of timetabling the required products in the factory to be manufactured, usually to meet various constraints and objectives. Whether from a mathematical point of view, in the class of sequencing problems, or the management perspective, of running a factory, scheduling problems are some of the most difficult to solve. Characteristically, the search space is large and complex; there are often many variables involved, and goals conflict, creating difficult objective functions. The process model is complicated and may include informal or undocumented decisions, rules of thumb and incomplete data. Additionally, the production environment is a dynamic and time-critical one, in that orders are continually arriving or changing, requiring a quick and effective rescheduling process. Machine breakdowns and maintenance have to be dealt with, and many needs have to be addressed to keep the process running smoothly.

The effects of a scheduler's work will spread into every area of the business. However, most scheduling problems to which genetic algorithms have been applied so far concentrate mainly on the ordering of jobs for production, and do not take a holistic approach to the full manufacturing environment and its inherent problems. Exceptions to this are discussed in 2.3.2.

The following sections (2.1.2) and (2.1.3) address issues of scheduling in theory, and in practice.

2.1.2 Scheduling Problems in theory

This section introduces some of the basic theoretical definitions found within scheduling work, which provide useful starting points for schedule optimisation work.

Machine Sequencing and Scheduling Theory

Manufacturing systems can be loosely split into two groups. In **discrete event scheduling**, the orders to be manufactured can be placed in discrete units, for example, cars, sandwiches, or bottles. In **continuous process scheduling**, the orders require that a part of a greater volume of the product - for example, chemicals, beer, yoghurt - is divided into a **batch** to meet the order, and then made in a continuous process. The application that motivates the work in later chapters is a discrete event problem, although much of the approach to this problem may be extended to process scheduling.

In the areas of discrete event scheduling, **machine sequencing** is a theoretical starting-point for those interested in solving scheduling problems. French (1982), provides a comprehensive discussion of this area. Early research into applications of GAs to scheduling has been mainly directed at the **job-shop problem** (JSP). A job-shop problem consists of the assignment of n jobs to m machines, to be processed in any order by the machines, with some objective to be minimised by the assignment order. A simpler version of the JSP is a **flow-shop problem** (FSP), in which jobs must be processed by machines in a defined order. This simplifies the scheduling greatly, simply requiring the sequential ordering of the jobs. Complexities may be introduced by adding extra machines or lines in parallel to the first flow line to give increased efficiency. Cartwright and

Tuson (1994) have made detailed examinations of alterations to the flow-shop topology in their study of genetic algorithm schedulers.

Another common problem that has been of interest to the GA scheduling community is associated with continuous process scheduling. **Batching problems** require a solution that splits the total amount of a particular product required for an order into a suitably sized batch for manufacture. The continuous quantities of product found in the process industry make this a more common problem than for discrete event scheduling, where 'batches' are often clearly defined by the discrete nature of the products.

Examples of many benchmark problems for sequencing and scheduling work are available from the Internet site OR-Library, (Beasley, 1990).

Problem complexity

Scheduling problems are frequently described as **NP-complete**. This refers to the time requirements for finding a solution, as related to the size of the problem. NP-completeness can be defined as follows: (French, 1982)

Definition

Time complexity function, $f(v)$ of an algorithm gives the maximum number of operations that would be required to solve a specific problem of size v .

Definition

$f(v)$ is of $O(g(v))$ if $f(v)/g(v) \rightarrow c$, c constant, as $v \rightarrow \infty$

If $f(v)$ is $O(p(v))$ for some polynomial $p(v)$ then the algorithm has polynomial time complexity. Else it has exponential time complexity.

Problems with polynomial time complexity are known as problems in class P; those without, NP. The classification of the problem is not always rigid; a NP problem can be moved into class P if a polynomial time complexity algorithm for its solution is discovered. Garey and Johnson (1979) demonstrate the NP-hardness of scheduling problems. Schedule optimisation therefore requires methods that can deal with such problems; GAs are included in this class.

In practical terms, this means that the exponential time requirements of the search to find a solution would be impractical without a method that could not work with NP-complete problems.

2.1.3 Scheduling in Practice

Whilst JSP, FSP and batching problems are complex theoretical problems to solve, Rodammer and White (1988), point out the inadequacies of using job-shop scheduling methods alone for real-life

scheduling. The JSP is described as being a 'trivial abstraction of the multitude of complex, dynamic and multi-objective scheduling problems, which must be resolved in actual production environments.' Perhaps this typifies the disadvantage of GA / job-shop scheduling research, as applying genetic algorithms to job-shop scheduling alone may only solve an inadequate subproblem. If the orders exist as a predefined group, with no new arrivals or alterations, then the problem is static and solving it requires a search of combinations of jobs assigned to machines. Complications arise with the more realistic introduction of dynamic job arrivals, (Cartwright, 1994). Once found, the scheduling solution to one particular problem may cease to meet new constraints and demands caused by the changing environment. A method must be able to accommodate new orders, breakdowns, changes of orders, or problems with suppliers.

In more dynamic, modern environments, particularly for retail goods, it is common for orders to be co-ordinated via EPOS (electronic point of sale) bar code information to provide instant feedback, on stock or orders, from the shop floor as each product is sold. By contrast, it is still known for large amounts of scheduling still to be done by pencil and paper, a reflection perhaps of the inadequacies of many existing scheduling methods. The introduction of modern technology to improve scheduling performance must be balanced with the reliability and dependence of existing systems, even if these are only rules-of-thumb.

Indeed, as shall be seen in subsequent chapters, real-life scheduling problems are not only concerned with far more complex rules than found within benchmark JSPs, but are also dynamic and multi-objective in nature. Mohanty and Venkatraman (1993) comment further on the practical use of such methods, 'CIMS (computer integrated manufacturing systems) can prove its existence only if there is a value added component associated with it, [...], a CIMS should also help to achieve the operational objectives, and list multiple objectives available'.

A real-life, dynamic, multi-objective problem is described in Chapter 3 and this problem provides an application for the GA methods explored in the following chapters.

Requirements of a practical scheduling system

There often seems to be a gap between the theory and practice of scheduling methods, as theory concentrates on solving over-simplified problems, and practice continues to rely on tried-and-tested methods drawn from experience. Furthermore, the gap between thinking in research and practice can be marked. Individual factories may be innovative, with plants willing to accept new methods of improving production practice. By contrast, some companies are reluctant to see the point in trying to improve on their scheduling methods, particularly where optimisation is concerned.

Bruns (1993) remarked that, 'In spite of a large number of developed methods, only a few practical applications (of theoretical scheduling technique) have entered everyday use in industrial reality.'

Given the logistics involved in introducing a new, unproven scheduling method into a large industrial plant, this is hardly surprising. The ideal of an all-purpose, optimal scheduler is still far from becoming a reality as optimality, speed, accuracy and massive data requirements conflict with each other in its implementation.

Thus implementing a scheduling system effectively in practice may entail many complications. We look at some aspects that should be considered.

Practical Implementation Issues

In understanding a practical problem such as scheduling, it is important to examine current methods in use in factories, the advantages and disadvantages of each, and compare them with the needs of manufacturers. It is true that, given the inherent difficulties with scheduling problems, the reality of the scheduling carried out in real-life factories may not always match up to the theory. Common industrial practice may prefer to rely on a traditional method that works 'well enough' rather than exploring new options.

Modelling

Accuracy of modelling is vital, not only if the optimisation processes are going to work effectively, but if the system is to prove convincing to the users. Some methods will over-simplify a situation or even misrepresent it. Taking this to extremes, an obviously inaccurate model will create a lack of confidence by the users in the solution. This may lead them to compensate for the inaccuracy, working towards their own goals, sometimes without informing the original schedule manager of the changes, and spreading inaccuracy throughout the system. An undocumented change to the schedule can mean that feedback on a schedule's performance will be attributed inaccurately to that schedule, injecting further inconsistencies to the modelled system. Accurate modelling relies heavily on human input, and may need an expert to be available to make decisions in case of a problem, using their added experience. It is questionable whether a fully automated scheduling system should operate without human input.

Consideration of constraints

Scheduling considerations must go far beyond the simple sequencing of jobs according to a single rule. Numerous constraints must be satisfied. Some must be met (*hard constraints*), and others are desirable, but not compulsory (*soft constraints*). Standard constraints include: job and customer priorities, due-dates, cost restrictions, machine and workforce capabilities, operation precedence, materials and supplies availabilities, and stock capabilities.

Other restrictions will be specific to the job; for example, it would be impossible to try to put the tyres on a car before the wheels have been made. Harding (1987), remarks, 'these often form a major deterrent to optimum machine utilisation and become the justification for major plant investment decisions.'

Objectives of scheduling

The purpose of scheduling is not only to produce a work plan, but also to meet other goals. These may be at a **local** level, for example, to minimize the amount of time spent on changeovers between making one product and another. They may also be at a **global** level, for example, to increase profits for that year. Discussion throughout the following chapters of this thesis will show how the choice, modelling and optimisation of these objectives can be an interactive process. This can allow an improvement locally or globally to a business, and also give the manufacturer insights into areas of the business that may have gone unnoticed previously.

Current problems with schedule optimisation methods include the fact that some methods are not designed to optimise at all, but rather to find a feasible solution rather than an optimal one. Indeed, given the complexity of many scheduling problems, there is little guarantee of finding the optimal schedule with any method: at best, a 'good' solution can be found. As we will see in chapter 4, schedules that aim to minimize a measure, which may appear important to the business, may actually be detrimental to the factory's performance.

Measures of performance - costs

For optimisation purposes, many algorithms concentrate on one or more specific measures of performance. Traditionally, these are such quantities as maximum makespan, work-in-progress, lateness, or flow time within a factory. Good definitions of these and other measures are given in French (1982).

Many researchers take the makespan as their measure, when looking for a suitable objective to produce a good schedule. This is the total time between the start of the first operation and the end of the last. Due to the extent of literature available upon the large number of these measures, and proprietary algorithms for minimizing them, this chapter aims not to cover individual cases of algorithms, but to present the main groups of algorithms used in standard management science approaches to the scheduling problem.

➤ **Realising real manufacturing objectives**

There can be an immense difference between the simple problems found in benchmarks (despite their mathematical complexity) and practical situations in a factory. This can be seen in the difference between benchmark objectives, which are usually simple measures, and real life objectives, which often combine many other factors beyond the production floor measures. A more holistic approach may be needed than is offered by the minimisation of a performance measure.

Goldratt (1993), argues that for real progress to be made in improving a factory's performance, it is disadvantageous to concentrate on the isolated problems of minimising one cost within a factory. The whole flow of information, decisions, fixed capacities and constraints must be considered as a

whole to allow the effects and interactions of this complex system to be taken into account. Minimising simple costs alone may not be effective.

User-issues

Naturally in a practical environment, the speed at which a solution can be found is important. Not only is the original run-time an issue, but the time in which rescheduling can be performed is also important.

- ~ Beyond the details of the actual method being used, many human factors must be considered for developing a suitable scheduling system. For the basic accuracy of the system to be reliable, accurate and comprehensive knowledge elicitation from human experts must take place. Staff must subsequently be both willing and able to use the new system, and must have confidence in its working. Fulkerson (1995), remarks: 'Many clever manufacturing technologies have failed to deliver on their promise because they clash with the factory's social structures and organisational dynamics'. The factory must be capable of meeting cost, time, hardware and equipment requirements, and must be willing to remain committed to a system. Staff may need training after the system's implementation. Finally, a method may require that new data is collected or stored for inputting.

2.1.4 Scheduling Methods

The following methods are listed as those being commonly found in literature on GAs for scheduling, either to provide context for a GA method, or for comparison against GAs as scheduling tools.

Objective-specific Heuristics

'Heuristics' is the term used to describe approaches that cannot guarantee that an optimal solution will be reached. These include methods based on rules-of-thumb, common sense, and empirical knowledge gained through experience. These may include some standard mathematical technique as a basis.

Waters (1988) comments, in the favour of heuristics: 'Despite man-centuries spent searching for solution methods, the most effective way of scheduling remains the use of simple rules. These scheduling rules are called heuristic procedures which experience has shown give generally good results.' Theoretical ideas of global optimality may be ignored on the shop floor when a supervisor needs to make a decision on a problem that he knows can be solved well enough using common sense and relevant experience. Until accurate scheduling can be done on the shop floor in real time, with enough flexibility to cover any given possibility, heuristics will continue to play a vital part in the smooth running of production, based on the knowledge and experience of people who work in the environment.

Many of these methods involve very simple selection or priority rules, such as, 'arrange the orders in increasing size of their runtimes'. Using a hierarchy of these rules can give a basis for choosing the next job. The simple rules all model the scheduling situation as a queuing problem by priority given to a job. Examples of these rules are: first come first served, job with longest operation time goes first, rank by delivery date, rank by value of job. These rules are more important for the discipline they provide rather than optimality of schedules. More sophisticated job-shop problem heuristics may make use of the **divide and conquer** technique. The problem is divided into groups of smaller, simpler problems; greedy algorithms are characteristic of such methods.

However, Harding (1987), remarks, 'the scheduling situation is dynamic, and hence one rule would rarely suffice for all situations.' Thus, heuristics are often be effectively combined with other methods into a useful hybrid technique.

It is important to note priority rules do not provide a guaranteed global optimisation of the schedule, unless their objective is identical to the optimisation objective. Their implementation must include all relevant information within the problem to ensure that all effects of the various variables and constraints are taken into account. Dubois et al. (1995), comment that there is no guarantee of the quality of the obtained solution, 'especially if only some of the temporal constraints should be respected'. Grabot and Genete (1994), remark that the performance of a rule on a given criterion often depends on the characteristic of the factory or of the set of manufacturing orders scheduled. Chapter 4 discusses this in more detail.

Constraint based methods

Fox and Sadeh (1990), describe the constraint satisfaction problem (CSP) as follows. 'The problem is defined by a set of variables, each of which has a domain, and a set of constraints. The solution is found by sequential iterations of potential solutions and their testing to see if they satisfy the constraints; the selection of the sequence of solutions is subject to various heuristics'. Tsang (1995), lists the advantages of these methods. Variables need not be numeric and there is no limitation on the type of constraints that may exist, offering CSPs to a wide range of applications. However, Willems and Brandt (1995), remark that CSP methods may suffer from combinatorial explosion.

Branch and bound

Branch and bound is one of the most popular scheduling techniques. It involves exploration of the 'elimination tree' of solutions, in which each order of jobs is considered sequentially, with those providing less than optimal solutions being eliminated during the search.

There are different strategies for searching through the tree. Depth-first searches go down the levels of the tree, whereas branch-from-lower-bound searches move around the tree according to

the lowest bound. The time requirements of the branch and bound method are unpredictable. There is also some uncertainty as to the best search strategy to use within the tree for a certain problem. The search can be speeded up by using heuristics to find a good starting branch. Alternatively, Kohler and Steiglitz (1976), suggest that the acceptance of a sub-optimal solution (say, within 10% of optimum) can give a vast decrease in computational requirements.

For methods to cope with large problems, problem-specific knowledge must be used to form assumptions that can allow the methods to work (Palmer, 1996). It will find optimal solutions, but possibly not in a realistic amount of time. It also requires accurate strategies to guide its search.

Linear Programming

A popular method for solving scheduling problems involves the reformation of problems as mathematical programs, and applying standard operational research algorithms. Linear programming (LP) approaches work well, but are only effective for small problems (French, 1982). LP can only be applied to problems where constraints and objectives are linear. The problem must also be specified by a set of inequalities for the method to proceed; both of these restrictions render the method somewhat inflexible for real-life problems that do not necessarily comply with these restrictions.

Dynamic programming

Dynamic programming is an enumerative method that builds up through optimal subsets of decomposed parts of the main problem into complete solutions. French, (1982), discusses how the method is limited by its computational speed demands, citing an example where, if each operation of a JSP took 1 microsecond, the method would become impractical for use on problems with over around 25 jobs. However, for problems of this size or smaller, it is an effective technique. It has the disadvantage of having to store interim sets of solutions; the final solution is found by working through all the optimal subsets found. To put the method into practice for a large scale problem would require a method of ensuring that the problem is decomposed suitably.

Recent work on emergent behaviour – that is, the effect on a whole system created by the interaction and co-operation of all its parts, may suggest that the practice of decomposing large problems into smaller subsets may lose a certain amount of information about the behaviour of the overall system. Of course, this is true for any method that isolates part of the problem, and again reflects the need to examine problems wholly.

Evolutionary Computation

Genetic algorithms simulate principles of evolution to perform the global optimisation by using populations of potential solutions mimicking the natural process of evolution in parallel towards a certain goal (Holland, 1975). There is no guarantee of an optimal solution being found but empirical evidence suggests that GAs can find comparatively impressive solutions to difficult problems.

Genetic Programming (GP) is also an evolutionary technique, which uses a grammar of possible variables of a problem to allow rules, or in effect, programs to be evolved to meet certain criteria, (Koza, 1992). GP may offer a range of applications within the field of scheduling. For example, Langdon, 1996) uses GP to find suitable rules for scheduling, as well as the schedules themselves.

As Evolutionary Computation is the main technique under discussion in the rest of this thesis, it is described in detail in section 2.2, and subsequent chapters.

Simulated Annealing

Simulated annealing (SA) is a method often used for combinatorial minimisation problems. The search mimics the cooling process (annealing) in metals, in which molecules move around until they settle to a minimal state. Laursen (1996), who discusses the parallel implementation of simulated annealing, describes this as a 'physical optimisation process'. The method is described by Tadei et al. (1995), as a 'generalization of a classic neighbourhood search, such as pairwise interchanges'. The search allows the solution to move from state to state in a 'gradient descent' approach (Rumelhart and McClelland, 1986) according to a predefined topology. As with genetic algorithms, an optimal solution cannot be guaranteed, but a good one is likely to be found. The method is also simple to implement.

Disadvantages of the method include its potential for huge computational demands, and that it also needs to be supplied with a feasible starting point. It is a sequential, rather than parallel method, although Palmer (1996), describes a method of implementing parallel search using SA, by mimicking the style of a GA. Without the parallel implementation, genetic algorithms may be more effective in searches where parallelism can be exploited. Parallel implementation may offer simulated annealing a more effective advantage in the comparison of the two techniques.

An optimistic view of the use of Simulated Annealing for scheduling problems is presented by Palmer (1996), who describes the methods of GAs, SA and Tabu Search (TS) as being similar in that they 'lack a need for guiding heuristics, the lack of a need to evaluate partial solutions, direct operation on cost variations and graceful degradation of performance in the face of increasing problem size.' The capabilities of SA, as being representative of this group of methods, are demonstrated on an integrated production scheduling problem, similar to problems covered by the work of Husbands et al., (e.g. Husbands, 1993). Palmer concludes that its performance is superior to standard dispatching rules on such problems, recommending it in particular for its generality. This compares to dispatch rules that must, by their nature, be narrowly defined by the problem. However, the method is at a disadvantage in this comparison by its lack of execution speed.

Tabu Search

Tabu search (TS) is another heuristic method, which also relies on the search point moving from area to area of the solution space according to pre-defined rules. Usually the point will move to the lower cost region of the solution space, in a greedy iteration search. However, in order to prevent the point settling in a local minimum, a list of 'tabu' (that is, prohibited) moves are kept to define where it may travel next. One example is the move that allows it to go back uphill once it has found a local minimum; another may prevent it returning downhill immediately to the same local minimum, but push it to keep searching elsewhere. Laursen (1996), comments, 'it is not trivial to set these [tabu moves].' The method also requires a feasible starting point for its search. The method is not as well known or popular as EC methods or Simulated Annealing, but appears to be gaining popularity in literature in recent years.

2.1.5 Literature on Comparisons of GAs with Other Methods

Given the number of methods available, much work on the comparison of such methods has been undertaken. The following are mentioned as recent comparisons that include GAs amongst available methods. Genetic algorithms as scheduling techniques themselves are discussed more fully in section (2.2).

Tsang (1995), compares Linear Programming, Branch and Bound, Hill Climbing, Simulated Annealing, Constraint Satisfaction methods, GAs, connectionist methods, and expert systems. He comments on the difference between methods that guarantee an optimal solution, but at the expense of time and those which work fast without guarantee of optimality. Given the NP-completeness of scheduling problems, this amount of time needed may not be trivial.

Tadei et al (1995), compare five search techniques (shifting bottleneck (SB), tabu search (TS), simulated annealing (SA), genetic algorithms (GA) and Lagrangian Relaxation (LR)) on 40 benchmark JSPs from Lawrence (1984). Comparing GA with the LR, they commented that although the GA did better, 'we should bear in mind that on a real shop floor, the information resulting from the cellular algorithm (e.g. the evolution of the objective function and Lagrangian Multipliers) gives the final user a much greater understanding of the problem'. This is also true to a certain extent of GA methods.

McIlhagga et al. (1996), compare simulated annealing with dispatch rules and an implementation of a co-evolutionary distributed GA, applied to the simultaneous optimisation of a number of flexible manufacturing plans. Their GA model improved on the solutions found by the SA. They compared Simulated Annealing, Dispatch Rules and a co-evolutionary GA, and demonstrate that the DGA performs the best out of these methods, commenting on the GA's advantage over SA of having a number of solutions available (in the population) rather than just the one.

Glass and Potts (1996), compare a descent method, threshold accepting methods, SA, TS, GA and a hybrid GA which includes a descent method addition to improve its performance, comparing them on 8 JSP test benchmark problems. The GA itself does not perform well in comparison to the other methods, when tested for minimisation performance, in terms of the minimal values found by each method and robustness, in terms of the frequencies of values found.

There is no clear consensus in these works as to whether any particular method is completely superior. Yet such comparisons may appear redundant to some researchers of the practical problem, when they may simply be concerned whether they can find a solution at all. As Mackle et al., (1995) comment, 'The question of whether [...] scheduling optimisation with genetic algorithms is superior to the other optimisation methods [...] cannot be answered definitively at this stage as no formal comparison has been made. However it is clear that GAs and other evolutionary programming methods can offer a lot of new possibilities for solving these problems, in particular by allowing different constraints and cost factors to be considered more easily than with other methods.'

2.1.6 Summary

In 1988, Rodammer and White drew the conclusion that no single paradigm at that time was able to give a unified method of scheduling, or possibly even represent a scheduling situation efficiently. They suggests a hybrid technique combining some of the best features of several methods.

With such a range of methods available, the user must decide which method is best for the particular application. For example, when looking at time requirements, the choice may be made between a slow method that guarantees an optimal solution, and a fast-running heuristic method. Similarly, there are cost issues; outlay for a system, implementation requirements, and the repercussions of meeting any necessary formalisation of the manufacturing processes to fit the system's requirements. Some methods allow various implementations within their basic design. A user choosing GAs may then include a number of implementations and improvements to allow the GA to perform suitably to requirements. This process is illustrated in subsequent chapters.

This section has aimed to provide the background and detail necessary to put the subsequent work of this thesis into context. The next section will introduce genetic algorithms, with particular reference to scheduling problems.

2.2 Survey of Previous Research on Evolutionary Computation for Scheduling

2.2.1 Introduction to Genetic Algorithms

This section aims to give a view of previous work on genetic algorithms for production scheduling in industry. An overview of scheduling in section 2.1 highlights the difficulties of the subject. Genetic algorithms have been cited as a potential method for producing an effective scheduling method.

This problem is approached by studying the application of genetic algorithm techniques to a real-life manufacturing environment. This section, as with 2.1, is in no way intended to provide an exhaustive survey of the GA field. Several authors provide much detail on general aspects of the theory and practice of applying GAs, (e.g., Goldberg, 1989; Michalewicz, 1992; Mitchell, 1996). Instead, this section concentrates on the relevant literature and background to GAs aimed at solving scheduling problems, and contains details of previous implementations, a summary of other scheduling techniques, and discussion of the requirements of the problem.

Evolutionary Origins

Today, the concept of evolution is well known and widely accepted, since being first generally introduced in Darwin, 1859. The idea of the development of life-forms gradually progressing by means of **survival of the fittest**, over thousands of generations, is a simple concept, and one that can be easily grasped by someone with little biological knowledge. It is this simplicity that makes the initial use of GAs accessible to many users.

Darwin made four major observations in his theory of evolution. Individuals must have:

- **'more than enough offspring'** - all species have the potential to produce many more offspring than they do
- **'the struggle to survive'**. As there are excess offspring, the need for something to counter this surfeit as a 'struggle to survive' means that only the fitter offspring will live long enough to reproduce. If the weaker ones die without reproducing, their characteristics are not passed on to future generations
- **'variation'** - to affect the weaker or stronger individuals, there must be variety in the population. Although Darwin had no knowledge of genetics as we know it today - Gregor Mendel provided the knowledge of that mechanism (Mendel, 1866) - the variety was provided by the huge combinations of genes available in any species
- **'inheritance.'** The mechanism that ensures that offspring receive characteristics from their parents. Though the ideas of genetics were unknown to Darwin, two ideas of inheritance were available at this time. Lamarckian inheritance (Grefenstette, 1991), in which characteristics

learnt during an individual's lifetime could be passed to their offspring, contrasted with Mendel's discovery that the offspring could inherit nothing directly from the parent other than the genes that the parent has at birth (e.g. Mendel, 1866).

The theories of evolution are still being admirably explained today by authors such as Richard Dawkins (e.g., Dawkins, 1986) and Stephen Jay Gould (e.g. Gould, 1980). To begin using genetic algorithms, it is helpful to draw on Darwin's four observations. The concepts are repeated in series: many varying individuals exist in a population, are evaluated for the right to survive, and the fitter individuals allowed to reproduce in order to pass on their characteristics to the next generation.

2.2.2 Definition of Genetic Algorithms

Genetic algorithms are optimisation tools that use a population-based search for the optimum. In doing this, they use principles of natural evolution and genetics to improve the search incrementally over time. They are robust procedures, and avoid many of the pitfalls of traditional search methods and optimisation algorithms. They are noted for their ability to deal with difficult search spaces, and have great potential for problems that are currently judged unsolved or intractable.

The genetic algorithm is based on a **mathematical analogy of natural evolution**. Its search points are **individuals within a population**. The ability of the individuals to 'thrive' within the environment of a particular problem is evaluated and good individuals are selected. This is known as the **fitness** of an individual and is described by a fitness value.

The selected individuals of a population then go through a reproduction process to produce offspring. This may be **unary** or **binary reproduction**; both are discussed in more detail in (2.2.2).

As this process is repeated following the evolution analogy, a 'survival of the fittest' effect allows good characteristics to become predominant in the offspring generations. By storing the best individuals of each generation, it is hoped that an optimal individual will be found. The schema theory of Holland (1975) is fundamental to describing how this process performs optimisation, as the subsequent generations of offspring become increasingly fit within the context of the problem, by exploiting the features of their genetic material necessary for a high likelihood of survival. In practice, there is little doubt that this is an effective optimisation technique. This process is illustrated in Figure 2.1.

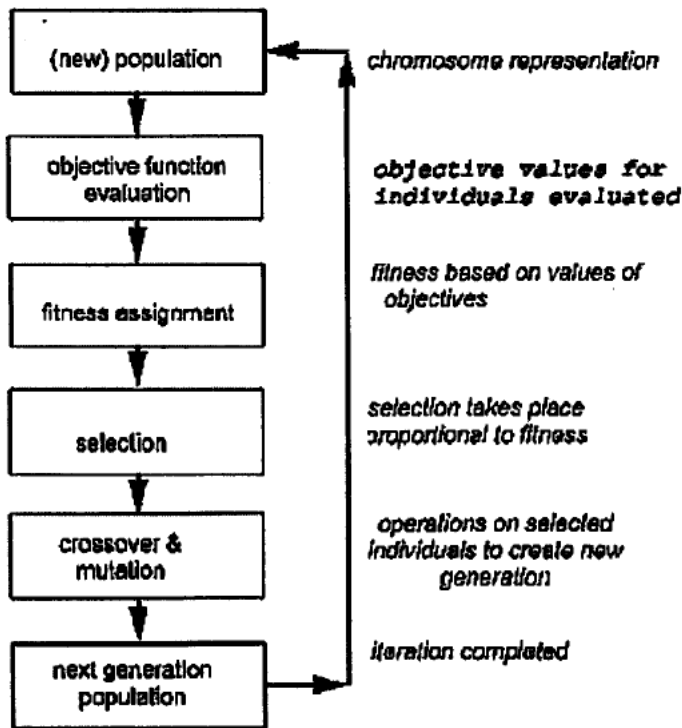


Figure 2.1 - Illustration of the basic GA process

Definitions of GA components

An **individual** is a coded string of information representing a search point. The encoding is chosen by the user, and is defined to suit the problem. The standard approach (e.g. Goldberg, 1989) is to use a string that is a binary representation of the value within the search space. Alternative encodings are commonly used if they are more effective for representing the problem space; for example, Gray codes, (Hollstien, 1971), real-valued strings (Wright, 1991), or, as in this research, sequence strings. The string is a necessary description of the process, and is analogous to a genetic chromosome containing genes, which correspond to the bits in the string. Each bit gives part of the information about that point. Characteristics may be used in the definition of the objective function. Analysis of these characteristics, or **schemata**, can be exploited within genetic algorithms, and a good discussion of this is given in Goldberg (1989). Some problems may use a fixed string length for all individuals; others may use dynamic string lengths. This is also often dictated by the problem.

Each individual's measure of the likelihood that it will survive within the environment representing the objective function is called its **fitness**. The assignment of fitnesses indicates the probability of individuals being selected to reproduce. Assignment of these fitnesses may be done simply by mapping probabilities linearly, but other methods may be used in the context of the problem to bias the selection in some way: roulette wheel, ranking, or stochastic selection, (e.g., Goldberg, 1989).

Once the selection of breeding individuals has been made, they are placed in a **mating pool** in which reproduction, in one of two broad forms, takes place.

Crossover, or **binary reproduction**, of these chromosomes can be performed in many ways. A string representation allows two strings to be paired and an offspring to be produced by rearrangement of some of the each parent's elements. Various crossover operators have been defined, relating to the particular problem being solved.

Parent 1 = 1 1 0 1 0 1 1 0 1 1 0 1 0

Parent 2 = 1 0 1 0 1 1 0 1 1 0 1 0 1

Child 1 = 1 1 0 1 0 1 0 1 1 0 1 0 1 0

Child 2 = 1 0 1 0 1 1 1 0 1 1 1 0 1

Figure 2.2 - Classic Crossover of binary strings

A **mutation or unary reproduction** operator is also used, mainly to prevent a population becoming stagnant, by converging to a local minimum. A genetic algorithm for a simple problem may function adequately using mutation alone. In other case, high levels of mutation cause too much disruption to the optimisation process, doing more harm than good. Good levels of crossover and mutation may be found by experiment. Typically, a small chosen percentage of each generation will undergo mutation. Many users, however, will follow rules-of-thumb for good levels, and adjust them by experimentation.

After reproduction, the offspring individuals replace their parents in the new generation, according to some defined strategy, and the process is repeated many times until the criteria for convergence have been met. In typical scheduling problems, this may be defined by actual run time, although in experimental work it may also be after a set number of generations, or when the GA has appeared to converge on a particular solution.

Advantages of genetic algorithms

Genetic algorithms offer the following advantages to the user that may not be available in the use of other methods.

- The ability to cope with difficult search spaces, with discrete or continuous variables, ill-defined functions, multi-modal problems where many equal optima may exist.
- Adaptable to dynamic environments; they can continue evolving within a changing environment.
- Speed of the search faced with NP-complete problems. Genetic algorithms are often able to search through an NP-space far more effectively than other methods

- They do not need one specific formulation of the problem, so can be implemented to suit many different types of application.
- Their implementation itself can be approached in a very flexible manner, allowing the development of a method suited to the problem being solved
- No need for *a priori* knowledge; they do not need a starting point.
- They perform global, not local, search
- They are simple to understand, being based on easily comprehensible ideas of biology, and so their basis for use as a technique can be easily explained to the user.
- They can approach the optimal solution in a reasonable amount of time, even if they do not find the absolute optimum
- Their parallel nature allows them a greater ability to explore the problem space, and so lessen the risk of becoming stuck in a local optimum.
- Parallel implementation can also improve the speed of the search
- They are capable of multi-objective searching.

Disadvantages of genetic algorithms

Some disadvantages of using genetic algorithms for scheduling problems are:

- They are only as good as the fitness function, so good knowledge capture is vital
- Care must be taken that genetic algorithms for scheduling do not have an over-simplified representation of the problem
- They may be slow for instant decision making if run from an initial state without any *a priori* knowledge.
- Their performance can depend on the careful adjustment of several parameters, and the values assigned to these parameters may effect performance drastically (for a detailed description of this process, see for example, Soares, 1994)
- The effectiveness of GA schedulers may depend on being tailored for them to be problem specific rather than generic to any problem.
- Whilst some scheduling genetic algorithms have found improved results regarding optimality on benchmark problems, they are not yet guaranteed to be able to find an optimal answer in real-life.

2.2.3 Use of genetic algorithms for scheduling problems

As shown by the range of methods in section (2.1.3), the scheduling problem is a great source of research for users of optimisation techniques. Previous work on genetic algorithms suggests that they have the potential to create a tailored, adaptable, accurate and fast scheduler for use in a real-life manufacturing environment.

Genetic algorithms are particularly suited to the problem because of their ability to search large, complicated and unpredictable search spaces. By drawing parallels with their natural basis, further techniques have been used, such as the multi-objective search, which allows the optimisation of

several objectives in the same search. They are efficient for a problem that cannot be tackled in exponential time.

On the subject of genetic algorithms for job-shop scheduling, Heitkoetter (1993) writes: '...genetic algorithms offer much greater simplicity and flexibility, and so, for example, may be the best method for quick high-quality solutions, rather than finding the best possible solution at any cost'. As will be seen later, this is a particularly relevant issue for a real-life problem.

2.2.4 Implementation of Genetic Algorithm for Production Scheduling Problems

The processes for creating a genetic algorithm for application to scheduling problems are now discussed. Whilst the accurate modelling of the process is probably the most critical part of the system, it is not much use without the correct formulation of genetic algorithm. The following components must all be considered in the implementation of a scheduling genetic algorithm.

Chromosome representations

In the application of genetic algorithms to scheduling, various levels of directness of representation have been used. 'Directness' indicates the amount in which the representation attempts to include all relevant information about the problem within the chromosome and operators, rather than by adding information at an additional encoding/decoding stage. The level of direct representation is intrinsically linked with the evaluation stage, as is discussed in 2.2.4.

Indirect Representation

Two main types of indirect representation are commonly used, the binary chromosome and the sequence (or permutation) chromosome. Given the inherent similarities between sequencing and scheduling problems, the binary chromosome is less common due to the difficulty of representing a sequence by a binary string. Various mappings have been explored between the two representations. Some researchers, (Nakano and Yamada (1991), Tamaki and Nishikawa (1992); Ranito and Neto (1993)) have mapped sequences onto binary representations, to allow use of the binary paradigm on a problem. Norman and Bean (1994), create a mapping between the two representations using random number chromosomes, which can be crossed like binary strings, but still mapped to a sequence.

Similarities between the general sequencing problems, such as the Travelling Salesperson Problem (TSP), and scheduling problems motivated many researchers to use sequence-based representations as chromosomes. The TSP is a classic combinatorial problem in which the order of a number of cities visited by a salesperson must be arranged so as to minimise the travelling distance between them. Hoffman and Wolfe (1985) explain problems of optimising such a routine. In many ways, this is a useful approach. The sequences are clear to understand and evaluate, and easily translated

in to a real schedule. The use of edges (Whitley et al.,1989) and other sequence features can be exploited. Grefenstette et al. (1985) presented the application of genetic algorithms to the TSP and demonstrated chromosome representations of the problem and operators that could be applied to these representations. Garfinkel (1985) demonstrated how job sequencing problem could be turned into a TSP, for minimising the total manufacturing time of a sequence of jobs. The comparison between the two problems comes from substituting cities for jobs, and distances for the changeover time between consecutively manufactured jobs.

Sequential chromosomes range from a simple permutation of numbers 1 to n, for n jobs to be scheduled, to more complicated sequence chromosomes, in which each element of the permutation is further divided into a subsequence.

Della Croce et al (1993) use a chromosome based on Faulkenauer and Bouffoix (1991) which is formed using subchromosomes, one representing each machine. Subchromosomes represent preference lists of operations rather than specific sequences and a simulation is used to interpret these preferences. This coding is limited to only describing non-delay schedules, ignoring situations where machines may lie idle, but always produces legal chromosomes.

An example of the more complex chromosome is used by Lee, Sikora and Shaw (1993). In their work on lot-sizing and sequencing they create direct representation chromosomes, this time specifying lot-size, which change in length as they evolve, discarding and combining less useful genes. Bruns (1993), argues for the use of schedules in their entirety as chromosomes, requiring extremely specific operators to manipulate them.

Direct Representation

Norman and Bean (1994), amongst others, have criticised some indirect representation approach as over-simplifying the scheduling problem by treating it as an extended TSP. They ask whether it is this assumption that has led to the surprising lack of 'perfect GA schedulers', against expectations of researchers who believe that the GA should be the ideal tool for solving scheduling problems. Reliance on indirect approaches limits the structure and mechanism required for constructing the GA. Furthermore, the scheduling problem may require a more subtle and complicated approach. Scheduling regularly contains more constraints and complications than the simple sequencing required for a TSP. As Reeves, 1993, remarks: 'GAs have a fairly poor performance for combinatorial problems if implemented in a naive way'.

Therefore, as the name suggests, direct representation attempts to include all relevant information within the chromosomes and operators themselves, avoiding the need for a complex decoding stage. This has the great advantage of ensuring that the search is conducted using individuals that contain all relevant information; in contrast, a GA that uses simple sequence chromosomes can only search to optimise those sequences rather than an entire schedule. The disadvantages of using

direct representation are the additional work needed on representation, choosing a method that will allow all chromosomes to represent valid information, and operators that will preserve this form, or incorporating a repair mechanism if operators are likely to create invalid offspring. A further disadvantage is, of course, that the representation will be highly problem-specific, creating a loss of generality of any method developed.

Evaluating the chromosomes

The representation and evaluation stages are directly connected. All information relating to the problem must be included in one or the other to allow accurate fitness assignment to take place.

Given that problem representation can differ extensively, so can the method of evaluating the chromosomes, depending on the representation used. The level of information contained within the chromosome will indicate the amount of additional supplementary information, included in a decoding stage, which will be required for the chromosome to be evaluated in terms of scheduling objectives.

Direct-encoded chromosomes will only require simple interpretation since they contain most of the information. **Indirect-encoded** chromosomes require extensive additional interpretation, and are evaluated in a decoding process, often termed a **schedule builder**, since it 'builds' a complete schedule for evaluation starting with the information contained within the chromosomes. Syswerda (1991), argues for the use of simple chromosomes as indirect representation, as they are more robust and able to be manipulated more easily manipulated in the case of a large population. Bagchi et al. (1991) disagree, saying that chromosomes and indeed their carefully designed operators should contain as much information as possible.

Bruns (1993), remarks that 'algorithms that do well across a variety of different classes of problem are usually never the best in any particular problem domain.' This highlights the dilemma of the genetic algorithm scheduler attempting to create the perfect scheduling solution. By creating an excellent solution for a specific problem, it may be necessary to sacrifice the generality of the method's application in favour of good performance. Indeed, Davis (1991), concludes that the performance of a genetic algorithm on a particular problem and its robustness are inversely proportional.

→ Evaluation for Indirect Coding

A schedule builder transforms a given chromosome, which can be as simple as a permutation of possible jobs, into a usable schedule. Its job is one of interpretation and application to a real time framework (i.e., setting the schedule within days of the week, for example). It may also be required to schedule maintenance periods, and in some cases, perform local search. Davis (1985) was the first to do this, because he was not using sequence chromosomes and so needed to interpret them somehow. Since then, they have become a general method of augmenting chromosome information into a complete schedule. Syswerda demonstrates three levels of using a schedule

builder, ranging from one that adds fine-tuning to the problem, to one that does a significant amount of search, and is practically a heuristic in itself. The schedule builder has inversely proportional amount of work to do to amount of information contained in chromosomes.

Evaluation for Direct Coding

A more recent development is direct representation: problem-specific chromosomes, which contain all information relevant to the search. Bruns (1993), uses chromosomes that directly represent the whole schedule rather than a simple order of jobs, and reports that these knowledge-augmented chromosomes perform better than indirect representation. There is no need for a schedule builder and all searching is done by the genetic algorithm itself. Genetic algorithms that do not take this approach have the disadvantage that they may not search the entire problem space if it is not adequately represented. They may then have to rely on the schedule builder to supplement the search. Fang, Ross, and Corne (1993), also worked successfully with direct representation chromosomes for scheduling.

The use of schedule builders and indirect coding is discussed in greater detail in Chapter 4, which explores the implementation of such an encoding for a practical scheduling problem.

Operators

The variation in representation generates a need for new operators. Using standard crossover procedures - that is, simply exchanging sections of parent chromosomes - on non-binary chromosomes generally creates illegal chromosomes as offspring. To avoid this, operators must be developed which always create legal offspring. Furthermore, the more complicated sequence chromosomes demand operators sensitive to their configuration, which will not divide subsequences.

Crossover Operators

Many operators have been taken directly from other combinatorial problem genetic algorithms. Popular operators such as Order Crossover (OX), (Oliver, Smith and Holland, 1987) and edge recombination operator (Whitley, Starkweather and D'Ann Fuquay, 1991) are frequently used with success, and examples of the implementations of these are given in Chapter 3. There is an extensive amount found in literature of other operators developed for subsequent scheduling applications, tailored to meet the specific requirements of the problem. Examples include: Partially Matched Crossover, (PMX) (Goldberg and Lingle, 1995) Linear Order Crossover (LOX), Faulkenauer and Bouffoix (1991); Giffler and Thomson Crossover, (Yamada and Nakano (1992)).

Mutation Operators

Mutation operators are unary operators, as unlike crossover operators, they only act on one chromosome, exchanging genetic material within itself. The mutation operator is used to widen the search, by altering existing members of the population into new individuals perhaps with much-

changed characteristics. As with crossover operators, choice of an effective mutation operator can enhance the performance of the GA.

2.3 Applications of Genetic Algorithms to Scheduling Problems

Genetic algorithms for scheduling have been applied to both benchmarks and test problems, and to real-life applications. This research is less concerned with benchmark problems; however, their use in this field should not be overlooked as much research is concentrated upon them.

2.3.1 Theoretical Problems

Standard problems are good indicators of the performance of a new genetic algorithm, as their use as a standard provides useful comparisons with both previous GA implementations, and non-GA methods. One of the most commonly used benchmarks are the *Muth and Thomson* job-shop problems (Muth and Thomson, 1963). They present a 6 x 6 job-shop problem (six commodities to be scheduled in a six machine job-shop), and similarly structured 10 x 10 and 20 x 5 problems are also given. These problems have been used by researchers over the last twenty years, and they provide a wealth of previous results against which a new technique can be tested. For example, Della Croce et al. (1993), are able to use the Muth and Thomson 10 x 10 not only to compare their new design of GA against Nakano and Yamada (1991), but also against solutions using simulated annealing, tabu search, and shifting bottleneck methods. There are also many mathematical algorithms applied to the same problem that can be comparative: (Balas (1969), McMahon and Florian (1975), Barker and McMahon (1985), Carlier (1989))

2.3.2 Real life Applications

In contrast to the use of benchmarks to evaluate a new theory, work on the development of systems for scheduling in real-life situations is characterised by its practical approach and difficulties in performance evaluation. Each problem is unique, and detailed descriptions and modelling of the production process are necessary before the GA optimisation can proceed.

Cleveland and Smith (1989), developed the problem from simple sequencing of jobs in a flow-shop situation to considering job release times, exploring five different crossover operators, all taken from TSP research, within the algorithm. They present and analyse alternative approaches, using a variety of performance measures and crossover operators. It is mainly a comparative survey, but also tests GAs on real-life production application. They note that techniques do not work as well on a real problem as on a strict re-ordering theoretical sequencing problem. This may illustrate that the use of sequential GA approaches, and the use of indirect coding, may not be adequate for a real-life problem. The gap between theory and reality may be illustrated by this fact; that the good performance of a GA in theory may not reflect its performance on a practical problem.

Syswerda (1991), works on a problem of scheduling work in a test laboratory. His system aims to allow manual interaction with a schedule editor within the schedule builder, to compensate for the additional complexities of the real-life problem. A discussion on the level of direct coding is included. This discussion results in the use of a simple chromosome and complex schedule builder, requiring the schedule builder to perform certain amount of local search after the GA has 'taken its best shot'.

Starkweather and Whitley (1993), tested implementation of a production and shipping scheduler for a major brewery with complex logistics. The average inventory level and staff hours were the two measures minimised in this work, as the research took account of the wider manufacturing considerations. The genetic algorithm is also implemented in parallel, giving substantial speed improvements.

Bruns (1993), worked with a real life chemical plant, and used a direct-coded GA. This did significantly better than an indirect-coded algorithm and shows that if a researcher's aim is simply to provide a good scheduler for one particular environment, exploitation of domain knowledge is a vital part of the implementation. From this work, the conclusion was drawn that it is better to evaluate the whole search space by GA alone than by indirect-coded GA plus schedule builder.

Cartwright and Tuson (1994), mention the introduction of genetic algorithm methods into industry. They explore the use of genetic algorithms within a chemical manufacturing flow-shop. It is interesting that in their comments on real-life GA scheduling, they comment on the need to move away from the convenience of TSP approaches to ensure that the GA genuinely tackles the problem presented. This remark holds despite the comparative simplicity of the flow-shop compared to job-shop. They present modifications of flow shop topology, indicating how the problem can quickly move beyond the standard model of FSPs. They also emphasise the dynamic nature of shops, a point overlooked by models that assume static environments. The optimal solution will only be just that so long as no new orders arrive. The work deals mainly with the topology within the flow-shop, including use of a GA to optimise the topology itself.

Further applications of GAs to real-life problems, particularly in the multi-objective sense, are discussed in chapters 4 and 5.

2.3.3 Current Research Issues on GAs for Scheduling

Work on GAs for scheduling continues to diversify with emphasis in the following fields.

- **Representation**
- **Diversity of Application**
- **Industry-motivated problems**
- **Improvements to Methods**

Representation

Representation continues to be a rich field for research, both on individuals within the GA and on the inclusion of domain knowledge. The search for an elusive generic method continues, together with increasingly problem-specific representations, tailored to meet the applications to which scheduling GAs are applied. (e.g., Bierwirth 1995).

Diversity of Applications

The case for the application of GAs in scheduling continues to be strengthened, firstly in the justification of their use, and performance evaluation (e.g., McIlhagga et al., 1996), and secondly in their diversity as an approach for tackling problems related to scheduling (e.g., University examination timetabling - Burke et al., 1995; water distribution – Savic, 1996; railway maintenance – Langdon, 1996; health work - Tuson et al., 1997).

Industry-motivated work

Given the direct links of scheduling problems to industry, and the need to find a good solution to many companies' scheduling problems, increasingly applications-led work seems to be appearing in the GA scheduling field. As earlier work based in theory builds the users' confidence in a new method, real-life applications have ceased to simply provide a more difficult test case for research, but are actually inspiring new developments within the field themselves.

Improvements to Methods

The flexibility of the implementations available for GAs, together with increasingly diverse research within the general field of GAs, means that many enhancements can be included in new implementations of the GA to allow user-defined improvements to the system. These are further discussed below.

2.3.4 Distributed GAs

The concept of geography within a GA serves two major purposes. Firstly, it allows the population to mimic further behavioural aspects of biology. Examples of this include breeding only within a 'local' environment (e.g. Deb and Goldberg, 1989), or defining species separated by geography to allow distinct local characteristics to form, a process called 'allopatric speciation' (e.g., Cohoon et al., 1987). Secondly, it makes an ideal model for the implementation of computer parallelisation of the GA to allow speed-ups of the system. For more details of these implementations, see Chipperfield (1995).

Taking a Wider View of the Problems

Given the influence of industry, it is more likely that additions to the standard scheduling problem will be needed to ensure accuracy of the systems.

The Integrated Planning Approach requires that the method not only optimises the sequence of the jobs (*schedule*), but also the *plans* with which each job is made. Husbands (1993), remarks on this

and the lack of attention paid to this common need, and proposes the method of simulated co-evolution to tackle it. Paredis (1994), also suggests co-evolution as a method of handling constraints. Palmer (1996), investigated this problem using Simulated Annealing.

By using a system that allows the plan and schedule to co-evolve simultaneously, a technique draws on a biological phenomenon that allows two species to co-evolve far more successfully than one evolves alone, and thereby finds superior solutions in terms of optimisation, also solving two problems at once.

Several scheduling methods have drawn inspiration from the field of Artificial Life (ALife) – that is, computer generated populations that exhibit ‘life-like’ behaviours (Langton, 1989). Levy (1994) provides an accessible study of this field.

Colorni et al. (1991), describe the effectiveness of ‘ant’ techniques. These are very simple ‘individuals’ consisting of basic instructions, which interact to create effects of their combined outputs. This allows schedules to be developed from very simple starting points. Fulkerson and Parunak (1995), describe their use of GA / ALife techniques for finding rules for the John Deere & Co. factory, commenting, ‘Alife techniques are one way to develop innovative solutions that transcend ‘managerial savvy’ and enable manufacturing enterprises to be agile’. They have applied ALife techniques in real life to their production sequencing and operation scheduling problems. Fulkerson and Staffend (1995), also describe a system of ‘ants’ for scheduling decisions as a way of meeting agile manufacturing demands in JSP problems.

Mattfeld et al. (1994) demonstrate the incorporation of ‘behavioural’ ideas drawn more from the ALife field with their ‘social-like’ behaviour of individuals within the GA, by giving individuals patterns of behaviour. In this case, being ‘pleased, satisfied, or disappointed’, individuals’ behavioural states influences their interactions, and thus their state within the GA. This method enables them to find good solutions to difficult benchmarks problems.

Reactive Scheduling

Given the dynamic nature of most scheduling problems, work is still needed on scheduling systems that can react to change in the scheduling requirements. Rescheduling is necessary when there is not enough time to be able to start from the beginning again, or when the current schedule is already being run. Fang, Ross and Corne (1993), suggest two methods of tackling rescheduling within a genetic algorithm. Firstly, they suggest augmenting the previous schedule with the new change and iteratively modifying it until it is acceptable, and secondly, creating a smaller scheduling problem, made up from all and only the parts of the previous schedule that are affected by the changes required.

Cobb and Grefenstette (1993) develop the work in Cobb (1990), to experiment with three different measures to improve the responsiveness of the GA to a changing environment. These are: constant mutation within the population, randomly generated individuals added to population ['random immigrants'], and hypermutation. Hypermutation introduces a higher rate of mutation when there is a decrease in the fitness over a time-averaged mean, (Cobb, 1990). They conclude that the random immigrants are a useful method for sudden changes to the problem, and that hypermutation is good for continuous change.

Seeding is also suggested by Cartwright (1994) for a scheduling problem, commenting that, 'many schemata in the solution to the static problem are likely to be useful in deriving the solution to the dynamic problem, provided we have not allowed the constraints to change too dramatically before engaging in a recalculation...a GA which controls a dynamic system is a real possibility.'

Bierwirth et al. (1995) explore the use of GAs for solving dynamic problems, firstly by changing the implementation of the GA but still allowing it global search on the whole dynamic problem, and then, due to this method's poor performance, by decomposing the time-frame into 'windows', allowing the GA to solve the problem for set portions of time; this method appears to work better. They comment that the latter method 'opens the way towards treating real-world problems'. Other researchers mention the need for an on-line or reactive scheduler in discussions of their further work from other research (e.g., Husbands, 1993; Mackle et al., 1995.)

Multi-Objective Genetic Algorithms - improvements in modelling

As mentioned before, a scheduling system, no matter how accurately it optimises a single measure, may actually require that more than one goal is optimised simultaneously. In the last two years, approaches similar to those tested in this thesis, in which GAs which aim to solve a multi-objective production scheduling problem from a real-life application by treating each objective as a separate factor for optimisation have been presented. These may use forms of Pareto-optimisation (e.g., Shaw and Fleming, 1996a, Tamaki et al., 1996) or extensions of adapting weighted sums (e.g. Niemeyer and Shiroma, 1996, and Ishibuchi and Murata, 1996). These are discussed in more detail in chapter 5.

2.4 Conclusion of this chapter

The area of production scheduling by genetic algorithm continues to expand into wider fields and varieties of applications, and yet there is still work to be done before genetic algorithms may be used effectively as a general, practical, scheduling tool. However, the diversity of the applications shown, together with the scope for solving new areas of scheduling and planning, such as reactive, or multi-objective problems, is very promising for developing such a system. Such a development will, in turn, influence theoretical research to work on new problems presented by industry that do not match to the benchmark or job-shop problem standards. The flexibility of GAs shown by their

wide range of representations, may allow difficult problems to be explored which have so far proved too difficult for traditional methods.

It is perhaps apparent that there is no one ideal method for scheduling. Yet genetic algorithms appear to offer particular advantages for the practical scheduler; namely their optimisation capabilities for difficult problems, the flexibility of their implementation, and the possibilities they offer for extending to multi-objective optimisation and reactive scheduling systems.

Chapter 3 will introduce a real-life application to which genetic algorithms have been applied. The initial application of the GA to the data taken from this application will be demonstrated.

3. Initial Application of a Genetic Algorithm to a Real Life Scheduling Problem

(This recipe) requires a special combination of dexterity and brute strength.

Take 1 caper, 1 anchovy, a few drops of oil, 1 garden warbler, 1 ortolan, 1 lark, 1 thrush, 1 quail, a few vine leaves, 1 golden plover, 1 lapwing, 1 partridge, 1 woodcock, 1 teal, 1 pheasant, 1 guinea-fowl, 1 chicken, 1 duck, 1 goose, 1 turkey, 1 bustard.

Make a paste of the anchovy and caper, then stuff the olive with it. Then, proceeding in order down the list, stuff each ingredient into the one below. Put the assembled 'thing' into a big pot with a cloved-studded onion, some pieces of ham, bacon, celery, carrots, coriander seeds, garlic, salt and pepper. Seal down the lid with pastry and cook in a slow oven for ten hours. Serve with chips and frozen peas.'

[Lucan and Gray, 1995]

3.1 Introduction to Chapter 3

In chapter 2, the theories of genetic algorithms and scheduling have been discussed. In chapter 3, a simple genetic algorithm is applied to a scheduling problem in practice. The chapter structure is as follows. Firstly, a real-life application is introduced, upon which subsequent problems are based for applying various GA implementations. The requirements of modelling and GA implementations are discussed and a simple GA is applied to a test problem based on the application. Some simple experiments upon this implementation are conducted. Conclusions and discussion of the development of this work complete the chapter.

3.2 A Real-life Scheduling Environment

Chilled ready meals manufacturing is one of the most demanding applications for scheduling, (Shaw, 1996). Compared to many industries, the timing is on a short scale. Orders must be met in hours rather than weeks. Variables dealing with raw materials such as food ingredients are subject to many hazards. Additional constraints are posed by freshness requirements, stringent hygiene procedures and food safety measures. These all add to the difficulty of producing a feasible, let alone optimal, schedule.

3.2.1 Introduction to chilled ready meals manufacturing

A typical time scale for a production run may be between eight to forty-eight hours from start to finish. Expert knowledge of every area of the production process is vital, and the scheduler will be continually asked to make instant decisions and alterations to the current schedule based on this knowledge. With the consumers' increasingly cosmopolitan knowledge of food matters, supermarkets continually introduce new product lines to match their higher expectations. Many

large supermarkets have decided to open on Sundays, or even for 24 hours a day, (Independent on Sunday, 1997) in recent years. Orders must be met satisfactorily and daily to allow fresh food products to be retailed seven days a week.

The chilled foods industry is booming, (Food Manufacture, 1994), with ready meals showing the biggest increase. A trend has grown over the last decade, towards having good quality, appetising and easy-to-prepare meals. This looks set to continue as the value of the chilled ready meals market as a whole was predicted to increase from £308M in 1992 to £441M by 1997.

3.2.2 Problem Background

Pennine Foods is a manufacturer of chilled ready meals for supermarkets. It provides the application used for the research in this thesis. The information in the sections below gives a broad outline of the process required, although some details have changed during the period of this research, with factory practices being updated, improved, or refined. All problems in subsequent work are therefore based on the factory below but may differ subsequently, as described in the relevant chapters.

Description of the planning process

Provisional orders arrive weekly, and are confirmed daily around 6 a.m., for that day's production. The confirmations are based on the figures from the EPOS system in the supermarket stores. Automatic stock-level adjustment takes place from the bar-code reader at the checkouts, as each product is bought in the stores. The consumer patterns of food purchase are continually subject to changes, making any long-term order forecasting system liable to inaccuracy. There is unpredictability and a need for fast, reactive decision-making within the scheduling process, to ensure that production meets the customers' immediate requirements.

The confirmed orders are divided into batches ready for production. The batches are assigned to the thirteen 'high risk' packing lines to be assembled from prepared ingredients from the cookhouse. The term 'high risk' simply refers to an area of the factory that has strict hygiene rules; 'low risk' areas need not observe such stringent rules, and care is taken to prevent any contact between the two areas. The assumption is made that all ingredients are available; if this is not the case, manual adjustments are made to the schedule to compensate. Some lines are dedicated to one type of packaging, others are multi-purpose and each can require varying numbers of staff. Hygiene issues impose restrictions on the relative positions of certain orders on a line; care must be taken that contamination between products does not occur. 'Messy' products may require extra cleaning of the line after their production.

The cookhouse will take the order figures and the supervisors down there will plan the cooking according to their own scheduling methods. The difference in planning techniques can lead to a

certain lack of integration between the cookhouse and packing lines, and a product that is scheduled to be packed first may not be always ready, demanding rescheduling of the lines.

Using these restrictions, the scheduler can build up a provisional schedule assigning jobs to lines. This schedule can require drastic changes because of accidents, altered orders, or lack of prepared ingredients. Breakdowns and stoppages can occur regularly. Further constraints include staffing considerations, for example, the minimisation of unnecessary overtime, or attempts to ensure that all staff work roughly similar hours. The schedule must also include breaks for staff meals and cleaning. Once orders are completed, there are seven deliveries, which leave at regular periods throughout the afternoon. The current main objective of the factory is to meet these delivery times. However, it is an important research issue that a developing schedule optimisation system designed for this environment should be capable of meeting more than one objective to allow additional goals to be met if possible.

→ ***Summary of the Scheduling Process***

The main steps in the scheduling process can be summarised as follows:

- Orders confirmed in early morning, the provisional ones having been received weekly
- Different amounts of products in each order for varying delivery times throughout the afternoon must be met
- Cookhouse plans separately but supplies prepared ingredients to packing lines
- Jobs defined by batches of orders to be packed for each product, and each job is assigned to one line, according to many constraints
- Rescheduling takes place throughout production run

Identification of problem to work on

One major purpose of collaborating with Pennine Foods was to work with real-life data, to provide both suitable complexities and realistic situations with which a GA scheduling system could work. The main application for GA scheduling within the factory is the line assignment, which is a progression from the standard job shop or assignment problem, with many constraints. There is much GA literature on this type of problem (as seen in Chapter 2). However, new aspects of this problem develop from the use of real-life data from the factory, an examination of the formation of multiple objectives for optimisation and the demand for flexible, reactive scheduling. Due to the number of disruptions and general instability of the environment, this would suggest the use of some type of reactive scheduling system.

3.2.3 Modelling Issues

To model the production environment at Pennine Foods requires a good deal of constraint handling and modelling of complex processes, many of which are interconnected or exert influences on each other. Accurate representation of these is required to provide a suitable objective function. In any GA scheduling system, the GA needs information to assess the scale of fitness, based on how

"good" a chromosome representing a schedule might be. The chromosome needs to be converted into an accurate schedule. Some form of performance measure, which is often dependent on the current factory set up, is used to evaluate the schedule provided.

This evaluation requires an accurate representation of all the necessary information of the manufacturing system. It was decided to limit the area of the factory explored in this work by concentrating on the packaging lines of the factory ('low-risk'). All the orders for the day have to be assembled and packed on one of thirteen lines. This is a highly constrained allocation problem. Particular products can only go on certain lines, limited crews are available for each line, certain products cannot be packed adjacently and the whole process is reliant on all the necessary ingredients being ready at the right time.

Modelling this environment required additional work from Pennine Foods themselves. It would involve large amounts of collecting data, formalising some of their current systems, and examining current decision-making processes. This is a well-documented requirement in literature on the application of a scheduling system. One employee was assigned to examine the manufacturer's information systems used in the factory, to collect data and implement computerisation of current planning and production information.

Assumptions made for factory modelling

The work that follows makes several assumptions, mainly that both raw ingredients and the prepared ingredients from the cookhouse are all ready on time. Ideally, the cookhouse should be integrated and could also be scheduled by genetic algorithm. At present, the cookhouse scheduling is done simply by rule-of-thumb and is independent of the packaging planning.

In addition, the following assumptions were included:

- All crews available, and with the correct number of staff for each required product
- Machinery took a constant amount of time to clean, maintain, or reconfigure
- All equipment was working correctly - no breakdowns.
- Orders were not subject to change once received initially (i.e., static problem).
- Breakfast and lunch breaks must be taken of 35 and 45 minutes on each line, rather than by each crew, as would actually happen.

Further extensions could be included in the schedule builder to add information that would allow these assumptions to be removed, using the correct information instead. This use of uncertain information in place of assumptions is discussed in chapter 4.

3.2.4 Run time issues

The factory made just one requirement regarding runtime of the system on the test problem; it had to run in less than an hour. However, given the limited time available in which to find a solution, it was necessary to consider whether the robustness of the method could be guaranteed. Given that

the GA is a stochastic optimisation method and liable to some variation in its individual performances per run, it is common practice to draw on results averaged over several runs, or perhaps to take the best from a run. For time-dependent applications, such as this one where a schedule is needed as soon as possible, it is not particularly desirable to have to re-evaluate the solution several times in order to be sure of its quality.

This question is discussed in more detail in Chapter 6, when we look at methods of measuring the robustness of solutions from a GA run. For this initial work in chapter 3, the common practice of taking the solutions from the combined data of a set of repeated runs was used.

3.3 Initial Problem based on Pennine Foods Factory

3.3.1 Aim of working on this problem

To learn more about the initial techniques and problems of the application of genetic algorithms to a scheduling problem, initial work was undertaken to apply a GA to a simple scheduling problem, adapted from parameters taken from real-life information in the Pennine Foods situation.

The aims of this first application were:

- to develop practical implementation of theory drawn from the literature in chapter 2
- to provide a simple demonstration of the performance of a basic genetic algorithm on a scheduling problem
- to work with real-life parameters and constraints as a test problem
- to set up a suitable basis of a GA scheduler, on which to expand in possible future implementations
- to provide some portable MATLAB-based GA scheduling tools, in conjunction with the departmental GA toolbox (Chipperfield, 1995)

3.3.2 Problem Specification

The factory manufactures thirty six products. Orders are received each day for a certain amount of each product. These orders are transformed into batches (3.2.2) to define the size and duration of the thirty six jobs. Each job is to be assigned to one of the thirteen lines, according to certain constraints, in order to minimise the three objective costs.

Strong Constraints

- Certain lines are configured for particular types of products
- Adjacency constraints: some products may not be made before or after certain other products
- Time constraints: some products may not be made before a certain time, or must be made to meet a certain time.

Weak Constraints

- The number of products that cannot be scheduled given the current constraints should be minimised
- Total makespan, i.e., the total length of the production run, should be minimised
- Total changeover / maintenance time between jobs should be minimised
- The number of products which are late for their allocated delivery should be minimised
- Variation in end of shift times should be minimised

3.3.3 Objectives

The three costs that were being minimised for this problem were:

- **Cost 1**- the number of unplaced jobs, this being a simple integer total of the number of those jobs
- **Cost 2** - the lateness of any order regarding its delivery time, this being a weighed sum of each late order, weighted by the importance of the depot and its overall lateness,
- **Cost 3** - the variation in the end of the run of each of the 13 production lines, this being the total deviation in ends of each line run

3.3.4 Problem Summary for Initial GA Implemenation

Given:

36 jobs, J_i , $i=1,2,\dots,36$

13 lines, L_k , $k=1,2,\dots,13$

7 depots, D_m $m=1,2,\dots,7$, of importance I_m , $m=1,2,\dots,7$, with deliveries leaving at T_m

Then $J_i = \sum_{m=1}^7 j_{i,m}$ each $j_{i,m}$ being the proportion of J_i which goes to depot D_m

RJ_i = run time needed for J_i

$Rj_{i,m}$ = run time needed for $j_{i,m}$

$Bj_{i,m}$ = start time of $j_{i,m}$

RL_k = total run time of L_k (including breaks or other stops)

Allocate one L_k to each J_i , such that no hard constraints are broken, and to minimise:

$$\text{cost 1} = \sum_{i=1}^{36} J_i A_i$$

where $A_i =$ 1 if no L_k can be found for J_i which satisfies the hard constraints
 0 if any L_k can be found for J_i which satisfies the hard constraints

$$\text{cost 2} = \sum_{i=1}^{36} \sum_{l=1}^7 I_l P_{j_i,m}$$

$$\text{where } P_{j_i,m} = \begin{cases} (B_{j_i,m} + R_{j_i,m}) - T_m & \text{if } (B_{j_i,m} + R_{j_i,m}) - T_m > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{cost 3} = \sigma(RL_k)$$

The methods of combining the three costs into one or more objective functions were implemented in various ways, and are discussed in more detail during the relevant experiments.

3.4 Applying the Genetic Algorithm

It was decided to use as simple a GA as possible, whilst exploiting features recommended by other researchers to improve performance within this, choosing operators, objective function, and chromosome structure. A schematic of the GA implementation is provided in Figure 3.1

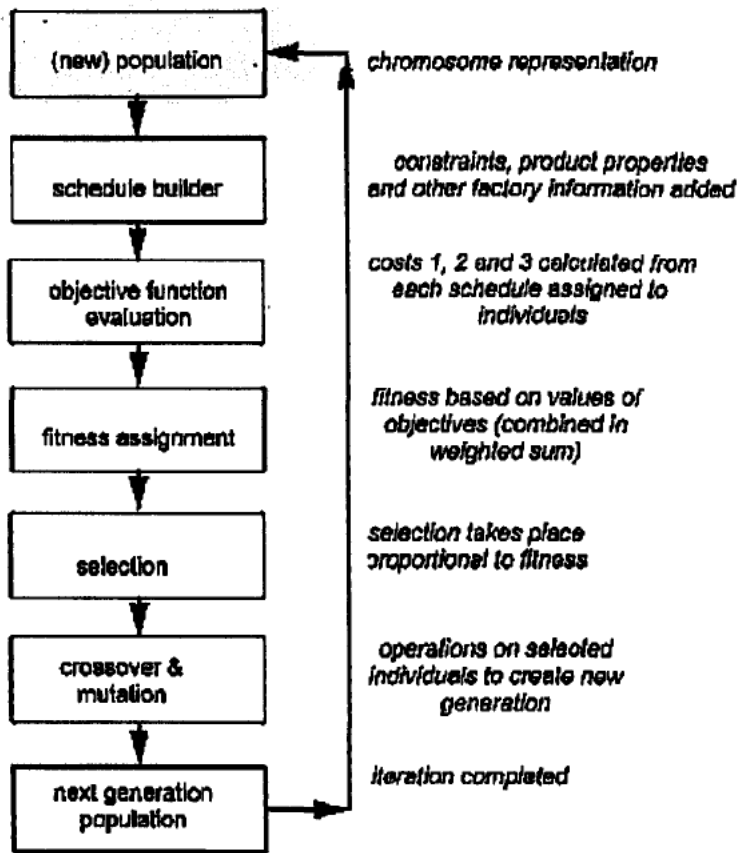


Figure 3.1 - Schematic of Genetic Algorithm used for scheduling

3.4.1 Code Used

The genetic algorithm was implemented in MATLAB^{®1} and draws on the MATLAB GA toolbox, (Chipperfield, 1995), but uses new implementations of chromosomes and operators. A set of menu-based options allows simple running of the GA for a user not familiar with MATLAB, (e.g., see Figure 3.2). All GAs were run on an IPX, 16MB workstation.

The user may choose new settings to use with the GA, or simply use pre-set values for size of populations, number of generations, types and levels of crossover and mutation operators. The products required must be input manually. The genetic algorithm runs as configured, and displays options for the user: examining the performance of the GA, or seeing the final optimal schedule found from the search.

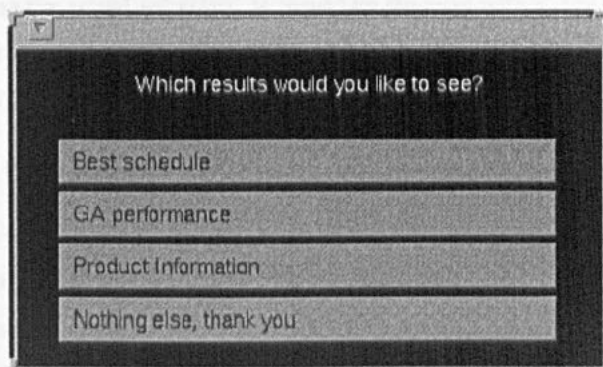


Figure 3.2 Example of menu of output options

3.4.2 Algorithm Design

The configuration is based on one of the two main GA-scheduling techniques (e.g., Davis, 1985; Syswerda, 1991) of using simple combinatorial chromosomes with a schedule builder to construct valid schedules and assess fitnesses. Effectiveness of this representation, compared to direct representation is undecided, as has already discussed in 2.2.3.

Each chromosome takes the form of a permutation, each element of which represents one of the jobs to be scheduled. The order in which the jobs appears dictates the priority placed on the allocation of the jobs, a task performed by the schedule builder (3.4.4).

The advantages of this representation are its robustness, and its ability to cope with a problem given little domain knowledge. This reflects one of the well-known advantages of the general GA: that it is able to optimise in a domain where it has a less well-defined model of the environment than may be required by other techniques. This particular problem provided a novel application for GA scheduling techniques, and further had some incomplete information and a variety of constraint types. Therefore, the indirect method allowed a broader approach to applying the GA to

¹ MATLAB is a registered trademark of The MathWorks, Inc.

the problem, without a requirement to find carefully tailored chromosome and operators as required by direct representation.

Finally, the indirect representation is more flexible, which is of great advantage for expanding on this initial work. Writing a problem-specific GA for one problem would limit further adaptations to the problem or GA configuration, and allows little scope to incorporate any additions to the problem once it had been completed.

The disadvantage of this representation might be a loss of performance, compared with that of direct representation. This is due to the fact that indirect representations mean the GA is searching the permutation space (i.e., the space represented by the permutation chromosomes), rather than the space representing the actual solutions, as would be the case with a direct representation. The degree of this effect depends on the accuracy of the mapping between chromosome and fitness assignment. This is discussed further in Chapter 4.

3.4.3 Objective function

An accurate objective function is one of the hardest parts of constructing an effective genetic algorithm. At this stage, a simple linear combination of the main costs was used. These costs were weighted to normalise the costs between their ranges of possible values. The implications of this approach are discussed in much detail in Chapter 5, when it is compared with alternative approaches designed to allow multiple-objective optimisation.

Thus, the objective function was provided as follows in (3.1).

$$\begin{aligned}
 & \text{minimise } Os, \\
 & \text{where } Os = \sum_{i=1}^3 w_i \text{cost } i, \\
 & w_1 = \frac{1}{\max(\text{cost } 1)} \\
 & w_2 = \frac{1}{\max(\text{cost } 2)} \\
 & w_3 = \frac{1}{\max(\text{cost } 3)}
 \end{aligned}$$

with cost 1, cost 2, and cost 3 being defined in 3.4.4

(3-1)

3.4.4 Implementation of a Schedule Builder

Each chromosome supplies the schedule builder with the order of the jobs, as represented by the components of the chromosome, and develops this order into a workable schedule, by assigning a job to a line at a certain time. The schedule builder currently assesses possible lists of jobs by completing an assignment that satisfies the hard constraints, and then evaluating the costs of the objectives, which go forward as the main cost to be minimised by the GA. This algorithm imposes a strict observation of the hard constraints, in that, if they cannot all be met, the job will not be placed at all. A summary of this process is provided in Table 3.1. Alternative methods, which use a penalty function to assess the weight of the effect of a constraint being broken, may also be possible.

Example Individual	Translation stage	Evaluation Stage
[2 7 24 9 13... 17]	<ul style="list-style-type: none"> • Assign job 2 to an available line which satisfies all hard constraints. • Assign job 7 to an available line which satisfies all hard constraints. • Repeat successively with jobs 24, 9, 13, continuing along the individual, to job 17. • Store any jobs which cannot be placed at all. 	The three costs are calculated from the resulting schedule.

Table 3.1 - Summary of Stages of Schedule Builder

As the representation allows an individual to be any permutation of the integers 1,2,...,36, this would provide a search space of 36! possible individuals within the population. However, given the mapping between individuals and schedules, where certain constraints will mean that two different sections of an individual may provide the same job allocation, and equally, where one job within an individual may be mapped to more than one available line, it is less easy to estimate the potential number of schedules available within the search.

3.4.5 Operator Choice for Recombination and Mutation

There is a certain amount of flexibility for the user of the GA, given a choice of two crossover and four mutation operators. These were provided to allow a choice of operators for a particular given problem, following the suggestion of Whitley et al., (1989), that there are different operators suited to various permutation problems. The following operators were implemented: **Crossover Operators:** Order Crossover (OX), (Davis, 1985; Oliver, Smith and Holland, 1987), Edge Recombination (ERX) (Whitley, Starkweather and D'Ann Fuquay, 1989); **Mutation Operators;** Inversion, Swap, Splice and Position-based Mutation.

It is possible to add further operators to this implementation if needed; those provided below were included to allow some choice of crossover and mutation.

3.5 Results and Conclusions for Initial Scheduling Problem

Some initial experiments were conducted to indicate the implementation of the GA which would allow the best performance, and to demonstrate this performance on the problem. The tests themselves are standard practise for the implementation of a practical genetic algorithm, and details of these tests are summarised below.

3.5.1 Test A - Operator Performance

Eight combinations of operators were tested with two sizes of search. The best crossover operator, by minimum average value found over the runs, was the order crossover and the best mutation operator was the swap operator. Both these operators preserve absolute positions rather than relative positions and this property is likely to be important to reaching the optimisation goal for this problem.

This conclusion is not surprising, since constraints dealing with relative positions, e.g. adjacency constraints, are enforced whereas constraints about time, including the total run time and the deviation in shift ends, are heavily weighted in the cost function. It would be worth investigating the difference in operator performance given a changed emphasis on other constraints. It may be necessary in a practical scheduling system for the user to have a number of operators at hand in a toolbox environment. A scheduler configured to use one type of operate may perform badly once the costs are re-configured.

3.5.2 Test B - Population Size

The performance of the GA when supplied with three population sizes (20, 50 and 100 individuals) was tested. In addition, a set of 40000 evaluations of random individuals was run in an attempt to find the minimum of the problem by random search. The latter search was curtailed after two days.

The results of the effect of population were somewhat inconclusive, with an unsurprising general trend towards a larger number of evaluations finding a better result. However, the small runs performed adequately for the purposes of these tests, and it may be that runs as small as 400 evaluations, taking less than 10 minutes, are quite effective for this particular problem. Compared with a random search that took approximately two days to find the same value as a GA managed in six minutes, it seems that tailoring the parameters to give a minimal running time may not be such an important issue at this stage.

3.5.3 Test C - Number of Generations

In the above runs, the speed of effective search was measured by storing the generation in which the minimum found on that run was first discovered, this being the point at which the search might have effectively stopped if the optimum had been known. This was calculated as a percentage of the total number of generations needed.

From an optimistic point of view, results from test C again indicate that some of the time spent on searching may be unnecessary. However, if the search is not converging prematurely, it may be that it is stagnating because the global optimum is difficult to find. Theory suggests manipulation of operators, changes in mutation levels or use of hybrid techniques, say neighbourhood searching technique combined with the normal use of genetic algorithms.

3.5.4 Test D - Visual Evaluation of Schedules

One of the best schedules found was compared with one generated by Pennine Foods, and also presented to the factory team for visual assessment. It is no minor result that such a schedule had been generated in less than an hour, which than the current methods. Although there will be far more constraints on the original schedule, this allows the factory a verifiable result in a familiar form, by simple visual assessment of the schedule based on their knowledge of the environment. Although this is not a rigorous test, the satisfaction of the factory and realistic detail within the scheduling system are an important issue within this work. The schedule is shown in Figure 3.3.

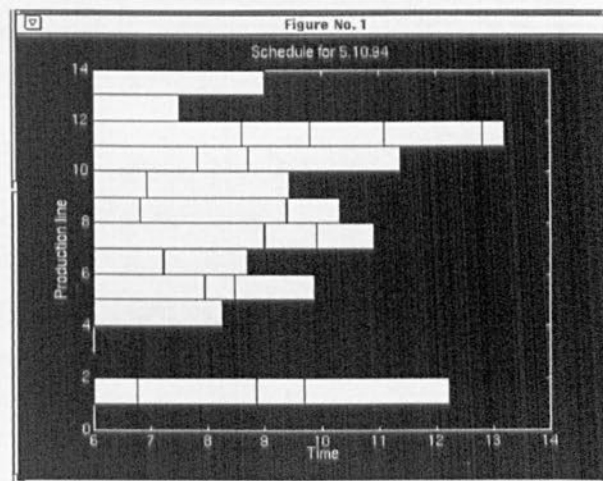


Figure 3.3 - Test E - visual result: schedule found by genetic algorithm scheduler for test case

It is difficult to compare the schedules found exactly as the schedule found by the GA is based on an approximation of the real problem, although the same set of orders has been scheduled.

- The schedule was visually assessed by the factory planning manager as being a satisfactory solution to the problem described above.

- Notice that lines 5 - 14, (excluding 12), are relatively balanced in their finishing times, i.e., there is no one line with all the orders assigned to it. This is due to the heavy weighting of the cost to minimise deviation in finishing times.
- Lines 1 and 3 are very specialised, hence the lack of orders scheduled on these.
- Lines 1 and 4 are configured to suit far fewer products than lines 7 - 13, again this is shown in the schedule found.

3.6 Conclusions

This chapter has demonstrated the basic, initial application of a genetic algorithm to a problem based on a real-life manufacturing environment, and some tests upon the parameters have been illustrated.

Not least is the simple demonstration that it is possible to produce a satisfactory, albeit not necessarily optimal, schedule for this difficult problem with a very simple implementation of GA. The global optimum to the problems is not known; all means of minima must be relative. This is particularly relevant for demonstrating how techniques such as GAs can be applied in industry. The previous software tested by the factory could not find an adequate schedule after 24 hours of running.

Of course, many assumptions and simplifications have been made in this implementation, but the GA has been applied with indirect implementation, which allows further constraints or data to be included in the schedule builder stage. We will look at this more closely in chapters 4 and 5, to demonstrate how greater detail can be included in the GA schedule optimisation process.

The manufacturing environment has been described in some detail, and some of the issues presented by this problem will provide context for the work as it develops in subsequent chapters. So far, it has borne little relevance to this simple implementation, and the major needs for multi-objective optimisation, and flexible reactions to changing situations and priorities within the scheduling process has yet to be tackled. Chapter 5 will start to explore this work in detail.

4. Investigation into Using Schedule Builders in Genetic Algorithms for Production Scheduling

4.1 Introduction

Representation is a vital issue for users of genetic algorithms for practical problems. It is essential to the performance of the GA that the accurate information is incorporated, if the GA is to optimize measures based on the modelled application. This chapter looks at one particular GA representation in use for optimizing production scheduling problems, and issues associated with implementing this representation, in terms of representing rules and preferences used within a real-life manufacturing environment.

The work is presented as follows. The genetic algorithm implementation that this work develops is introduced in section 4.1.1. The issues of problem representation are discussed in section 4.2. Sections 4.3 and 4.4 discuss the experiments based on this implementation. The conclusions are presented in section 4.5, together with suggestions for further work.

4.1.1 GA Description

The problem in this chapter is based on the chilled ready meal problem described previously, in Chapter 3, in that forty-five products have to be assigned to thirteen production lines, given certain constraints, and aiming to optimize three separate costs.

Opinions conflict as to the most effective way of including knowledge of the domain within the genetic algorithm. It is important to represent the situation of the factory accurately by the inclusion of all necessary information for describing it. At some stage, the fitness function, and knowledge of the environment optimized by the GA has to be included. There are varying degrees of direct representation for doing this. Direct chromosomes may have complex configurations, as they are designed to contain as much problem information as possible. Such methods commonly require specially designed operators to ensure that crossover or mutation create legal offspring. However, they can be simply interpreted, as they already represent much of the information directly. Bagchi et al., (1991), recommend this representation.

Indirect representation has already been introduced in chapter 2. A simple permutation chromosome uses a schedule builder to add the further information to the sequence of orders that the permutation represents, to allow the individual to be evaluated in terms of scheduling measures. This chapter intends to discuss the particular use of problem information within the schedule builder.

Bruns, (1993), remarks that 'algorithms that do well across a variety of different classes of problem are usually never the best in any particular problem domain.' This highlights the dilemma of the genetic algorithm scheduler - and indeed for GA users in many other fields, attempting to create a schedule optimisation technique. By creating an excellent solution for a specific problem, there may be a loss of generality for applying the solution to other scheduling environments.

4.1.2 Representation used in this work

There may be no definite consensus on the best form of representation in GAs for scheduling. Indirect representation was chosen for this particular research, the choice being influenced much by the real-life nature of the scheduling problem. A major advantage in the use of the schedule builder was the allowance of extremely flexible inclusion of the environmental data. Given the level of change within the factory, it was vital that this GA representation did not restrict inclusion of changed data in any way. Problem changes can be implemented by a simple alteration to the code contained within the schedule builder, without any need to alter the actual representation or operators used. In addition, the indirect representation seemed less restrictive to the experimental work used in this project on other aspects of GA research, such as multi-objective optimisation and development of a GA MATLAB toolbox (Chipperfield, Fleming and Pohlheim, 1994).

The main disadvantage of indirect representation is that it may inhibit accuracy of the optimisation process by representing only part of the problem within the population. In this case, the permutations represent only the priorities on the order of the products to be scheduled. However, this disadvantage must be put in context by the fact that the solutions found by the system could only be measured relatively rather than absolutely, given the real-life problem had no known solutions with which to compare ours. At this stage, complete accuracy of optimisation is less of an issue than the ability to find fast, flexible and reasonably accurate solutions to an extremely challenging problem.

Michalewicz, (1996), discusses the inclusion of problem-specific knowledge, and in particular, problem-dictated hybridism within the GA implementation, quoting Davis, (1989), stating his belief that real-world knowledge should be incorporated into the GA by operator development or a decoding stage, such as schedule builders, to allow GAs to become useful tools for real-life applications.

4.2 Genetic Algorithms and Schedule Builders

4.2.1 Background to Schedule Building

In this section, the use of schedule builders is discussed in more detail. Their purpose is to transform a simple chromosome into a schedule that can be evaluated according to the objectives required in optimisation. Given the permutation, the schedule builder takes it as the priority order

in which it must assign the products to the available machinery, according to the rules and constraints that it contains. The transformation of permutation chromosome into a schedule ready for evaluation provides an essential part of fitness assignment and thus performance of the GA.

Davis, (1985), was the first to suggest a schedule building technique; the representation used required a transformation stage from chromosome to schedule. Whitley, et al., (1989), detail a 'greedy scheduler' that transforms initial job sequence into a completed schedule to be evaluated. The problem is designed to ensure that each mapping from chromosome to schedule is unique by including rules to resolve conflicts ('if there are multiple jobs that could be set up in place of X, [...], first, the machine decides that job has the shortest set-up time; second if these are the same, the one with the shorter execution time is chosen.')

Syswerda and Palmucci, (1991), describe their implementation of a schedule builder for a constrained resource scheduling problem, 'using knowledge about the constraints of the problem, the scheduling builder uses this ordering to constrain a legal schedule without worrying about the much harder problem of producing a good schedule'. However, they suggest that the schedule builder could perform a local search for the best position, given the schedule already structured. They add that, 'as the schedule builder is made smarter, more and more chromosomes will result in the same schedule and thus the same evaluation. This hides information from the GA, since a chromosome and a somewhat worse chromosome can both result in the same schedule.' Their schedule builder uses a preference vector, constructed by computing fitnesses according to each hour of the week for every unscheduled task, to construct the schedule according to the constraints given.

Syswerda, (1991), works on a problem of scheduling work in a test laboratory. This discusses the sudden leap between the simplicity of many theoretical problems and the immediate difficulties presented in real-life. The system aims to allow manual interaction with a schedule editor within the schedule builder. Discussion is included over the level of complication of chromosome versus a simple chromosome and hard-working schedule builder, requiring the schedule builder to perform certain amount of local search after the GA has 'taken its best shot'.

Uckan, et al., (1993), provide a comprehensive discussion comparing the use of schedule builders against the use of a direct chromosome representation - that is, one using the schedule itself as a chromosome. They remark that a schedule builder extends the GA's search capabilities into areas not covered by the GA itself (i.e., by drawing on the extra information contained within the schedule builder to evaluate the fitness of chromosomes), but, for their implementation, that results are only optimal locally. The optimal schedule designed by the schedule builder has been made using optimal process plans whereas 'a globally optimal result might require the use of suboptimal process plans and resource allocations for some job orders,' an area that has subsequently been

explored by Husbands et al's 'Co-evolutionary Distributed GA' (e.g. McIlhagga, Husbands and Ives, 1996). It is a useful point to note that when splitting GA optimisation into distinct stages (first do this, THEN this), the effects created by the separate parts must still contribute to the global optimisation goal.

Dorndorf and Pesch (1995), describe the use of the *Giffler and Thompson algorithm* (Giffler and Thompson, 1969) for assigning operations to machines. This solves the conflict of two (or more) operations competing for the same machine randomly - the authors suggest the use of one of twelve priority rules to resolve these conflicts.

Recent work on schedule builders includes a method using genetic programming to evolve the best scheduling heuristics to use within the schedule builder (Langdon, 1996). Also highly relevant to this area is the 'Evolving Heuristic Choice' (EHC) method that produced good results, although these may be at the expense of computing time. The genetic algorithm (Fang, Ross and Corne, 1994) includes heuristic rules to be used in the schedule builder as part of the chromosome encoding, allowing the rule choice to evolve alongside the schedule. This produces superior results on a set of benchmark problems. Attention is drawn to the effectiveness of including a degree of hybridization within the GA process, by such a technique.

4.2.2 Schedule Builder Implementation

Given the permutation, the schedule builder takes it as the priority order in that it must assign the products to the available machinery, according to the rules and constraints that it contains. The transformation of permutation chromosome into a schedule ready for evaluation provides an essential part of fitness assignment and thus performance of the GA.

Figure 4.1 shows the pseudo-code for this process.

```
for I=1 to number of elements (products) in chromosome
    find set of lines L1 that product may be assigned to (hard
        constraint)
    find set of lines L2 that are available to be used at the given
        time - some lines do not start running until later in the day, or
        may already be finished for the day (hard constraint)
    find set of lines L3 that are free to have product placed on
        them given adjacency restrictions (some products may not be
        placed immediately after some other products)
    find intersection of L1, L2 and L3
    if intersection is empty,
        there are no available lines at present and product is not
        able to be placed
    end if
    if intersection is not empty and only contains one line,
```



```

        this is chosen as the line to be assigned to the product
    end if
    * if intersection contains more than one line,
        one is chosen from the set as the line to be assigned to
        the product
    end if
    check through unplaced products to see if any can now be placed
    add product to schedule on assigned line for required amount of
    time
    add in any changeover and maintenance time that may be needed
    in between products
    if a tea break or lunch break is needed once that product is
    finished
        add tea or lunch break
    end if
end for
all orders are now placed on completed schedule

```

Figure 4.1 - pseudo code for schedule builder

Putting this into practice with the input of a permutation chromosome includes the schedule builder as an essential part of fitness evaluation; this process is summarized in Table 4.1.

Permutation given	Schedule Builder	Evaluation
[9 4 7 9 3 8...]	Takes product 9 first, looks for the line on that it can be placed. Having done so it takes product 4...repeats until end.	Schedule is evaluated according to objective measures and fitness is assigned to permutation.

Table 4.1 - Working of the schedule builder

4.2.3 Genetic Algorithm Used

The genetic algorithm that was initially implemented to solve this problem was designed to be as simple as possible, for reasons discussed below.

The GA used a generational population of individuals that were permutations of the integer values (1,2,...,45), each element within the permutation representing one product. The population size was set at fifty individuals, over forty generations. Fitness evaluation took place after the individuals had been coded into complete schedules by the schedule builder, which is discussed in more detail below. The objective values were calculated as a weighted sum of the three costs provided, with the later intention of using a multi-objective optimisation GA. The Order Crossover, and Swap

Mutation Operators were used, these being operators that had been designed to create legal offspring from permutation strings, and that had been shown experimentally to work well for this particular problem.

4.3 Methods of Line Assignment

As the GA individuals are permutations, it is the role of the schedule builder to take each element of the permutation in turn to assign the order represented by that element to a suitable and available line. The method of selecting between more than one available line for a particular job must be addressed. As may be the case when working with real-life problem data, no further explicit information was readily available as to how to make this choice. Line assignment in the factory takes place by the scheduling team's experience, by rule-of-thumb, or according to an unquantifiable combination of factors at the time of scheduling. Further consideration was needed upon the method of line assignment given a choice of available lines, both from the theoretical point of view - that line would provide with the best schedule for a global optimal solution, and the practical - whether the choice was sensible and within the scope of the factory's own experience.

The simplest method of such decision-making, given the lack of any alternative information was simply to pick a line at random from those available. However, there were complications for the case in which several lines might be available simultaneously. This situation would then have the effect of creating many mappings of fitness to individual. The fitness assignment would be unreliable, as any given individual might have a whole range of fitnesses available depending on the random selections of lines made. Clearly, a method that allowed a more informed selection from the available lines was required, and alternative methods of informing this choice are explored below.

4.3.1 Heuristic Preferences

One simple method of choosing a line from a given set is by the use of heuristic rules to make the final decision. Any such rule would provide a one-to-one mapping of fitness to individual. Common examples of such rules are: 'select the line that finishes soonest' or 'select the line requiring the minimum changeover'. Whilst this is useful in modelling accurately the events taking place in the factory, it is perhaps necessary to question its effectiveness from the optimisation point of view. Does using such a heuristic actually limit the search of possible schedules too much? Although a heuristic is used in the factory, it might not necessarily be the most efficient method to help find an optimal schedule. It may be convenient to use, as the data is already available, or it may be a common rule-of-thumb in other factories. A heuristic may also limit the use of the GA by fixing the elements too rigidly and not allowing sufficient exploration of the available schedule space. One example given was that by using a 'choose the line that finishes earliest' heuristic, the rule is implicitly encouraging the optimisation of early finishing times, which might not necessarily always be desirable, especially with the emphasis on trade-offs between conflicting objectives

(Shaw and Fleming, 1996) requiring that all objectives be treated equally, without an implicit bias towards one particular objective. If this method were to be used, it would be necessary to choose between the heuristic that best represents the action taken in the factory in real life, or choosing between a large selection of available heuristics to see that allows us most effective use of the GA as an optimisation tool. The actual search for a best rule to use in this situation has been explored previously. (Fang, Ross and Corne, 1994; Dorndorf and Pesch, 1995; Langdon, 1996) This work aims to examine the effectiveness of using rules against an alternative method.

For this particular example, examination of the problem, data and factory practices provided one practical heuristic that could be included in the problem at this stage, that of choosing a line with the minimum changeover period. Although there are many other heuristics well-known in the literature on scheduling, data collection and factory preference elicitation beyond the scope of this work would have been necessary to include such rules; although this would certainly be an area for further work.

4.3.2 Pre-expressed Preferences

A less rigid decision could be made by the use of pre-expressed preferences of which line to use for any given product. By creating a matrix that explicitly expressed a value of preference for each assignment of a product to a line, the choice of lines could be made by simply from the line available with the highest expressed preference. These preferences could be fixed throughout the run of the problem, or again, could be adaptively altered according to the situation on the shop floor (for example, if a machine is starting to fail during a shift, preferences might be expressed to allow that line to 'go easy' when possible). As a trial, these preferences are expressed on a scale from 1 to 10, or perhaps in simpler terms of 'good', 'neutral' and 'bad' (mapped 1-3).

Further investigation into the use of these preferences may take place to see how exactly they need to be expressed. If each product/line preference value is expressed as a unique value, there will only ever be one possible outcome when a choice between lines has to be made. This allows us to provide a one-to-one mapping for the chromosome-schedule interpretation. However, this could not only be fixing the problem too rigidly, limiting the problem to the extent that the potential for search possible schedules is over-constrained. Use of integer preferences, where some lines may have the same preference for a given product, would reintroduce a certain random element in their one-to-several mapping of fitness to chromosome, where there may have to be a further choice between two lines with the same preference. However, the extra level of constraints would provide a narrower choice than in the original schedule builder. Experimentation should show how far this choice could be relaxed. ✓

4.3.3 Estimated Fitnesses

It was suggested that one way of stabilising the assignment of fitnesses to individuals by the schedule builder, in the case of there being stochastic elements in the assignment, would be to

generate the distributions shown above for each individual, and then use either the mean, (Cronan, 1996), median, or best value found from the distribution as the actual fitness, rather than relying on one random sample from this distribution, as had been the case previously.

Whilst this method would create stability, it would be extremely intensive computationally - each time an individual needed evaluating, repeated runs of the schedule builder would be required to create the sampled distribution of fitnesses with a degree of accuracy to allow the GA to use a measure of this as the individual's fitness.

Experimentation on this current problem shows how many samples are needed in this distribution to provide some accuracy. It must be remembered that in a combinatorial problem, there may be very high number of possible schedules available from just one individual if random choice at the assignment stage is used.

It may be possible, with some analysis of the actual number of possible lines presented the assignment stage, that it can be predicted how large the number of possible schedules available might be for any given individual, and so estimate the sampled distribution needed to accurately predict the mean value for these to allow more accurate assignment of fitness.

4.4 Experiments on the performance of schedule builder

Given the possible rules that might be used for the allocation of jobs between multiple available lines, experiments were conducted to explore their performance within the GA optimisation process. The performance of the following schedule building techniques were compared: (Table 4.2).

	Technique	Notation	Implementation
SB1	'Random Line' Schedule Builder	<i>pref_0</i>	Given a choice of lines when assigning a job, choose a line at random
SB2	Preferences - unique	<i>pref_13</i>	Assign unique preferences (1-13) for each job to line assignment
SB3	Preferences - 1 - 10	<i>pref_10</i>	Assign a preference on the scale of 1 - 10 for each job-to-line assignment
SB4	Preferences 1 - 3	<i>pref_3</i>	Assign a preference on the scale of 1 - 3 for each job-to-line assignment
SB5	Problem Specific Heuristic	<i>maintenance</i>	Given a choice of lines when assigning a job, choose the line with the minimum maintenance time
SB6	Estimated fitnesses	n/a	Generate an empirical distribution of all the possible fitnesses from a random assignment and use the sampled mean as the fitness.

Table 4.2 - Schedule Building Techniques under comparison

SB1 is described above in section 4.3, SB2 - SB4 are variations of the method discussed in section 4.3.2. The problem specific heuristic, SB5, discussed in section 4.3.1, was chosen as being the best example of such a rule for this problem. The schedule builders were initially implemented with estimations of the factory's own preferences, based on the data originally given. In the following experiments, these implementations were used for experiments 1 - 4; in experiment 5, the factory supplied the data directly for the schedule builder.

In initial experiments, it was apparent that the use of SB6 discussed in 4.3.3 was infeasible given the computing power available. Even using a relatively low number of samples, 99, the technique required an excessive number of evaluations of the schedule builder, making one run of the GA take over a week. However, this method may be worthy of future investigation given greater computing power.

Tests on the capabilities of the schedule builders were planned as follows:

4.4.1 Experiment 1 Examining the available fitness distribution on many individuals for each schedule builder

Firstly, the general fitness landscape created by each schedule builder upon the general population was explored by allowing 1000 random individuals to be generated and evaluated by each of the schedule builders. This would give an empirical measure of the distribution of fitnesses for the population as a whole. As seen in Table 4.2, some schedule builders were likely to assign a relatively wide range of fitnesses, whereas others would assign a unique value.

4.4.2 Experiment 2 - Level of variation in fitness assignment to a single individual

It has already been suggested that it is undesirable to have a high level of instability in the individual-to-fitness mapping. Each schedule builder was run 500 times on one specific individual, to provide sampled distributions of the fitnesses created by each method for an individual chromosome. This allows a profile of the stability of each method to be shown, by indicating the distribution of possible fitnesses that could be assigned to each individual by the schedule builder.

4.4.3 Experiment 3 - Performance within the GA

The actual performance of each schedule builder within the GA was put to the test by comparing the performance of ten runs of a standard, weighted sum scheduling GA (Shaw and Fleming, 1996) for each schedule builder. The GA used 50 individuals, for 40 generations, using order crossover (Davis, 1985), splice mutation (Fox and McMahon, 1990), that were found in previous tests to work well on this problem.

4.4.4 Experiment 4 - General case of schedule builder within GA

To test for a general extension of experiment 3, an exploration of the stability of this effect was tested by using a variety of different preference values in the matrix. For those schedule builders that could be given alternative values within their preference matrices [*pref_3*, *pref_10* and *pref_13*], ten random examples of each type of preference matrix were generated and used in the genetic algorithm to explore further the effectiveness of the matrices used in Experiment 3 - to show whether a more general conclusion could be drawn for a generic approach using this technique. For example, given a different problem on which to apply this method, is there any one schedule builder that seems to work better with different preferences?

4.4.5 Experiment 5 - Using real-life data in form of best schedule builder

In order to see if these results held in practice, data was then elicited from the factory in order to create a schedule builder according to the method found to be the best in the previous experiments. Runs of this schedule builder were tested against the randomly generated schedule builder performances from 4.4.4.

4.5 Results

The results of the above experiments are presented below.

4.5.1 Experiment 1

Shown below in Figure 4.2, Figure 4.3 and 4.4 are histograms showing the distribution of fitnesses found by the random generation and evaluation of individuals using each of the schedule builders.

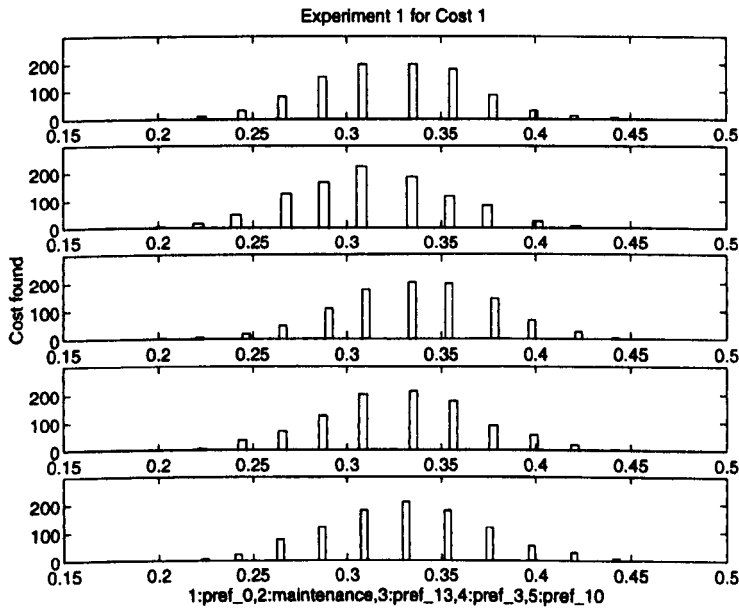


Figure 4.2 - Distribution of values found for cost 1 by evaluation of 1000 random individuals

[Key: subplot 1 - *pref_0*; subplot 2 - *maintenance*; subplot 3 - *pref_13*; subplot 4 - *pref_3*; subplot 5 - *pref_10*]

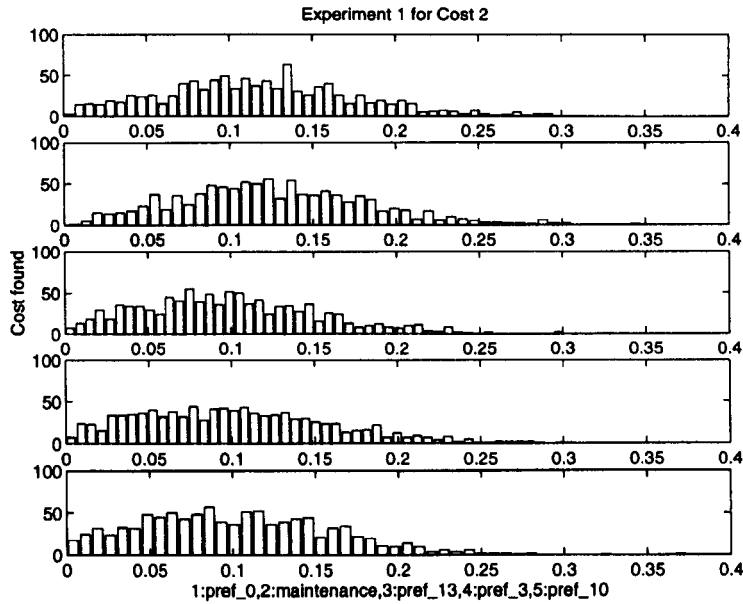


Figure 4.3 - Distribution of values found for cost 2 by evaluation of 1000 random individuals

[Key: subplot 1 - pref_0; subplot 2 - maintenance; subplot 3 - pref_13; subplot 4 - pref_3; subplot 5 - pref_10]

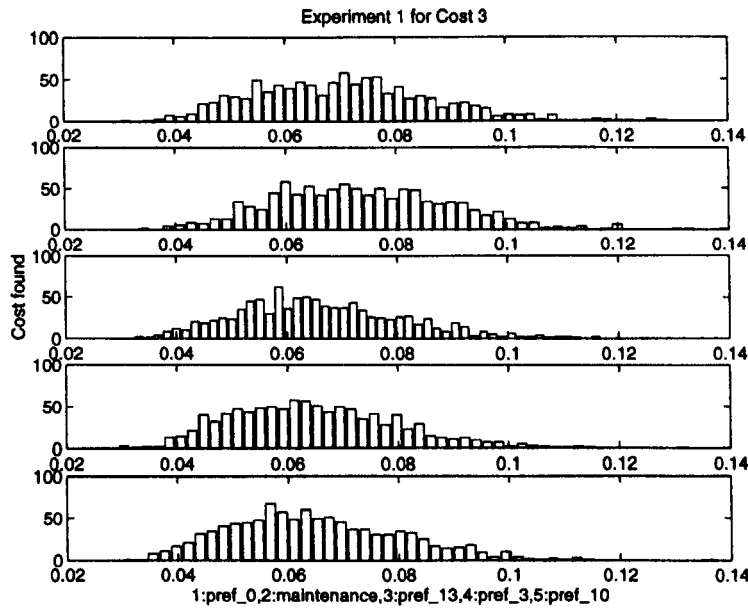


Figure 4.4 - Distribution of values found for cost 3 by evaluation of 1000 random individuals

[Key: subplot 1 - pref_0; subplot 2 - maintenance; subplot 3 - pref_13; subplot 4 - pref_3; subplot 5 - pref_10]

It can be seen that there is little noticeable difference between the range of values for the different costs found by each of the methods, demonstrating that the change in the actual method of schedule building does not produce any overall change in the potential range of fitnesses that may be available to the genetic algorithm. This would suggest that we could reasonably experiment with

changing the different implementations of schedule builder without constraining the potential of the performance of the GA.

4.5.2 Experiment 2

Experiment 2 examined the actual variation in fitness assignment to each individual for the different schedule builder methods. The means and standard deviations for the fitnesses produced by 500 repeated runs of each schedule builder on one single individual are given in Table 4.3 and Table 4.4. Clearly, the original method (*pref_0*) provides a wide range of fitnesses for the same individual, with a difference of over 0.3 between its best and worst. *pref_3* gives a similar amount of variation, with somewhat less extreme values at the top of the range of fitnesses. *Pref_13* and *Maintenance* provide the same value throughout - a one-to-one mapping of fitness to chromosome. *Pref_10* also shows variation across the range of fitnesses.

	Cost 1	Cost 2	Cost 3
pref_0	0.3428	0.1600	0.0749
pref_13	0.3111	0.2013	0.0865
pref_3	0.3556	0.1574	0.0741
pref_10	0.3440	0.1559	0.0733
maintenance	0.3674	0.1354	0.0697

Table 4.3 - Means of distributions in fitnesses assigned to one individual by each method

	Cost 1	Cost 2	Cost 3
pref_0	0.0280	0.0592	0.0138
pref_13	0.0000	0.0000	0.0000
pref_3	0.0000	0.0000	0.0000
pref_10	0.0273	0.0607	0.0132
maintenance	0.0188	0.0438	0.0073

Table 4.4 - Standard deviations of values of distributions in fitnesses assigned to one individual by each method

4.5.3 Experiment 3

Experiment 3 applied the schedule builders' capabilities to the problem by running each within a genetic algorithm.

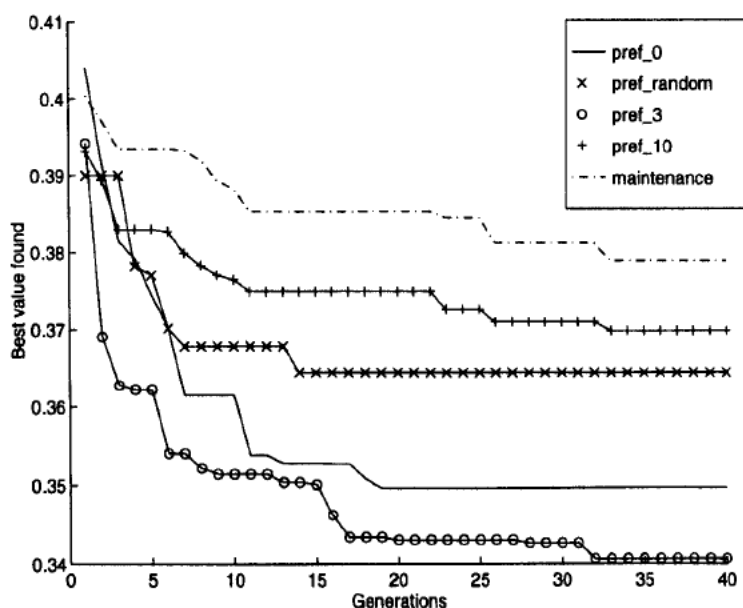


Figure 4.5 - Plot of mean performance over ten runs of each schedule builder within a genetic algorithm

The mean performances of each of the ten runs are shown averaged for each schedule builder in Figure 4.5. From this, it can be clearly seen that *pref_3* performs best, followed by *pref_0* (the original 'no additional information' technique), and so the other three methods actually impair the performance of the GA compared to its original state.

It is interesting that the priority rule method ('*maintenance*') is actually the worst of all the methods shown. This rule-of-thumb is perhaps the most realistic representation of a method that might be used to make a decision in the factory, but as can be seen from this, it is certainly not the most effective way of doing so. An approach that may seem common sense in the factory - to select the line with the shortest maintenance time in order to get the production of the next order running as soon as possible - damages the overall optimisation of the schedule.

The *pref_13* and *pref_10* techniques do not fare very much better, suggesting that again, the rigid prescription of preferences (whether unique, in the case of *pref_13*, or with only a slight amount of flexibility, as in *pref_10*) may be too rigid in constraining the problem to finding sub-standard solutions - without any flexibility, the schedule builder may not be capable of getting near an optimal solution.

The original GA, *pref_0*, actually performs better than these other methods, indicating the degree to which care should be taken in choice of preference rules within the schedule builder - that the GA can perform so well given the amount of variation contained within it is encouraging for supporters of GAs in itself, but that it outperforms methods intended to improve it by limiting the variation gives us a cautionary example.

Finally, the best performance is given by *pref_3*. This method limits the variation within the schedule builder somewhat - but by preserving an element of this variation, actually allows the GA to find far better solutions. As the preference matrices that would best enhance the GA's ability to find an optimal solution are not known, allowing a little variation within the schedule builder is better than rigidly prescribing preferences that, as can be seen, do little to help the optimisation performance.

4.5.4 Experiment 4

In order to see whether these results would extend to the general case, ten random examples were generated of each of the preference matrices where this was possible (*pref_3*, *pref_10*, and *pref_13*), and ran genetic algorithms with each of these matrices inside the schedule builder.

The performances of the individual runs (Figure 4.6) and the mean performance for each method (Figure 4.7) are shown below.

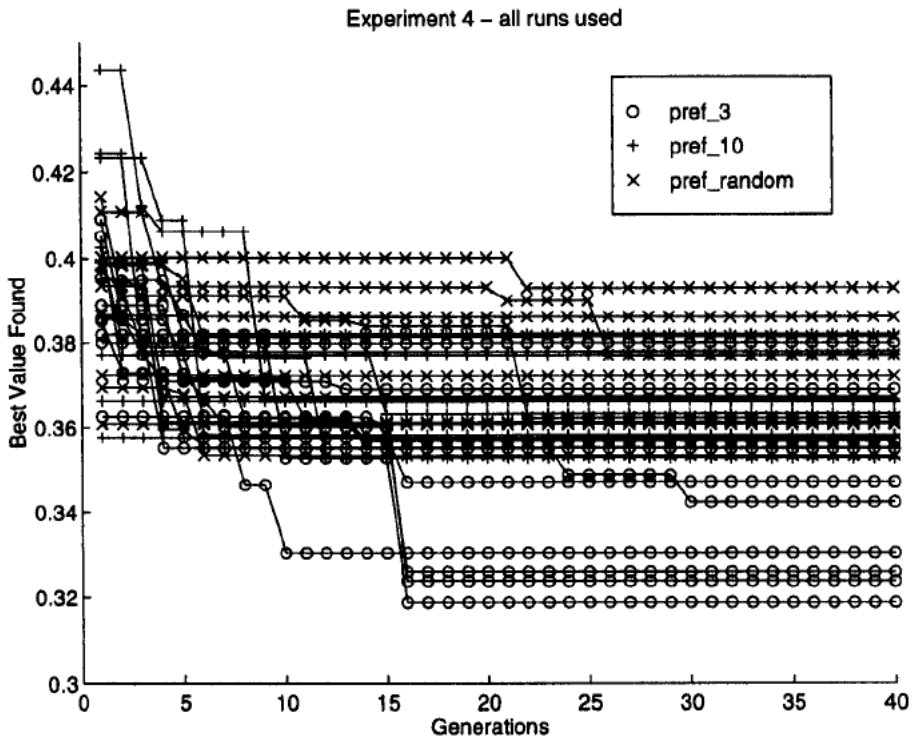


Figure 4.6 Plot of performance of each of ten randomly generated schedule builders for each method

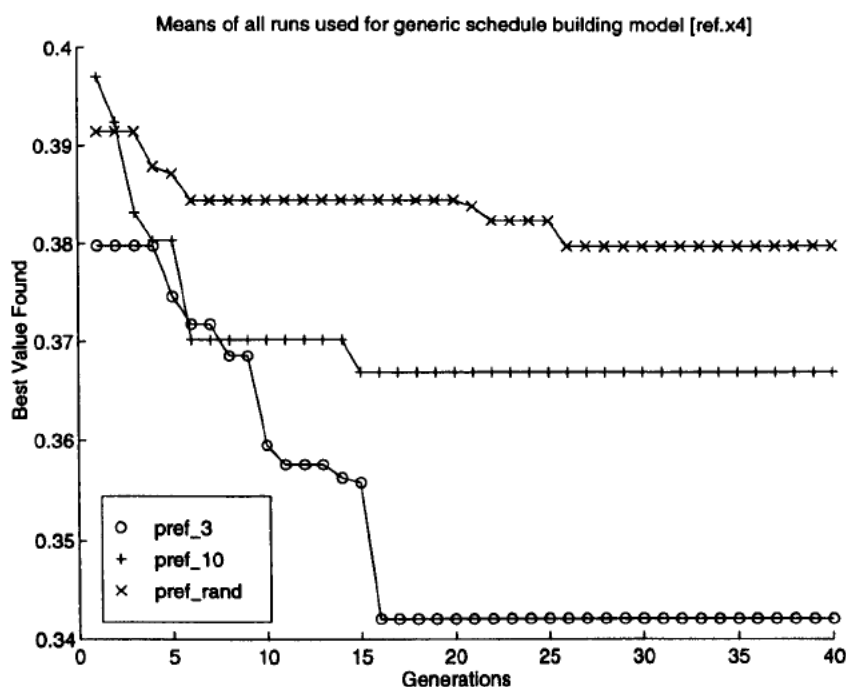


Figure 4.7 Plot of means of performance for each method using ten randomly generated schedule builders

It can be clearly seen that again, *pref_3* is the overall best performer, indicating that the results from Experiment 3 may be extended to a general case. *Pref_13* and *pref_10* do not perform well in comparison - indeed the overall result reverses that shown in Experiment 3, as *pref_10* performs slightly better than *pref_13*, perhaps confirming that even a small amount of flexibility within the schedule builders preferences can be beneficial.

4.5.5 Experiment 5

Experiment 5 tested the results of Experiments 3 and 4 in practice by taking the best-performing technique of schedule building (*pref_3*) and using them to add data from the factory into the GA. By asking the production team involved in creating schedules on the shop floor to supply the 'good / neutral / bad' preferences required to supplement a *pref_3* schedule builder ['*pref_fac*'], it was possible to include their data in this form within the genetic algorithm.

Ten runs of the GA using this data allowed us to compare the performance of this schedule builder to the 'randomly generated' builders used in the previous experiments. The plot of the mean performances of each method is shown in Figure 4.8.

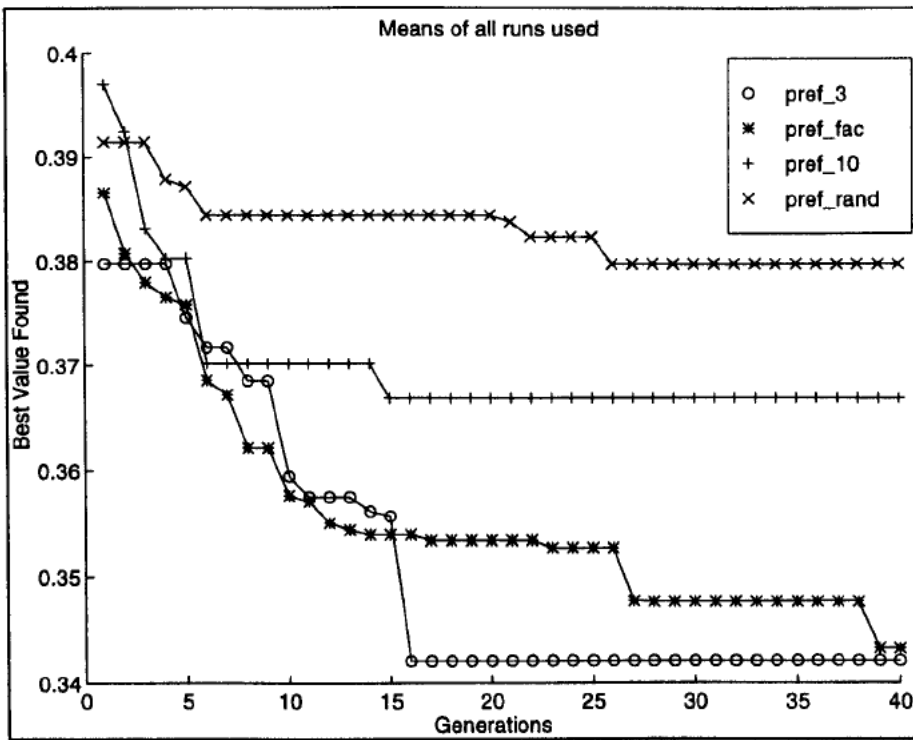


Figure 4.8 - Performance of the factory-set *pref_3* ('*pref_fac*') schedule builder against the experimental versions.

The performance of *pref_fac* matches the set of *pref_3* runs quite closely - this should be expected given that *pref_fac* is one particular example of this implementation. It outperforms the examples of the other methods (*pref_10* and *pref_0*, the method that was being used originally), and almost manages to reach the same level of minimised solutions as the 'generic' *pref_3*.

4.6 Conclusion

The use of an interpretation stage within a GA, such as a schedule builder, for a large problem based on real-life information depends on both the accuracy and degree of consistence in assigning fitnesses to individuals. This work explores issues involved in including such information for a real-life manufacturing problem.

Whether the schedule builder representation should be used at all is still an undecided issue. In this work, such representation has offered many advantages, such as flexibility for fast-changing problem data, portability and ease of use for experimental work. Some of the problems of rule selection within the GA might have been avoided by using a more direct representation of the problem, and this would offer scope for future work. However, it has been an aim of this work to develop schedule builders that allow flexible incorporation of such data, and to explore new types of preference representation, without losing any accuracy of the schedule optimisation.

Estimated fitnesses were suggested as supplying an accurate measure of an individual's fitness without the need to provide any additional information to the problem. For use in the practical

problem, where a run of the GA with only one evaluation of each individual almost fills the maximum time limit required for a schedule program (set at one hour by the factory), it is clear that this method is infeasible. However, it is worth including here for future reference, when a more powerful computer may make it possible. It should also be included as being worthy of investigation for problems where there can be no further information whatsoever available as to which line to choose at the assignment stage - in the absence of any further information, it may not make any sense to include a heuristic or preferences that can accurately aid the search. It may also be of use in a situation where the time of the GA run is not of consequence.

Within the framework of schedule building, heuristics may be the solution most accurately reflecting behaviour in factory decision making, but the results bring into question the actual effectiveness of certain heuristics when working with optimisation. Accurate reflection of factory decision making may actually impair genuine optimisation rather than aid it if there is no influence in the choice of the heuristic towards the actual goal of the optimisation. Choosing a rule because, 'it's always done this way,' may maintain accuracy between an optimisation system and the factory, but will not necessarily be the ideal for optimisation of costs. Williams (1996) makes this point clearly; in industry, occasionally finding cases where, 'a fair number of rules that had become law, were merely because of original programming bugs, or [were] developed in the 60s and copy of the source was now lost!' A rule chosen in the factory may not be at all beneficial to a global optimisation goal. Methods such as Langdon, (1996), and Fang et al. (1994), may allow the selection of a heuristic that is beneficial to the optimisation process.

There is much scope for modelling and inclusion of alternative heuristics from the factory, that may provide better performances within the GA, and this might be an area for future work. However, it is worth noting that it is the case when working with real-life problems that the obvious choice of heuristic or data available to model it is not always available, as was the case in this application.

Preferences allow us to explore the degree of constraints that might be included in the schedule builder before the search either becomes too narrowly constrained or too randomly implemented. It allows information from the factory to be included and allows less rigid control than a heuristic might. However, the use of this method would depend on the ability to capture useful knowledge and have them expressed as a numerical preference at this stage. The generality of the results shown in Experiment 4 may indicate a guideline for implementing this method in practice - if extracting preference information from a factory to include in a schedule builder, it may be better to choose a form in which preferences more broadly defined (*pref_3*) than one that assigns rigid values for preferences (*pref_13*) - given that this makes sense in the context of your problem, and does not conflict with any fixed requirements in the factory.

It should be noted that the preferences at the moment as used in 4.4.5 were only elicited from one person - the operations manager - and there is scope for further work on this in defining not only a method of combining preferences from more than one person, with varying points of view and priorities, but also in defining what the actual preferences given reflect in terms of practical situations. The manager might define preferences that reflect very different considerations from those of the shop-floor worker. This idea of trade-offs of information is explored in another context in the use of GAs for multi-objective optimisation in Chapters 5 and 6.

The demonstration of this method in practice, when using the *pref_3* formulation to include real factory data in the scheduler, a far better performance can be achieved than by the previous methods. This allows progress with a more accurate, better performing GA for scheduling. It should be noted that the data provided by the factory was provided without a view to aiding the optimisation performance of the GA, so much as for improving the accuracy of the model. As seen with the disappointing performance of the maintenance time heuristic, these two goals are not always compatible - however, in this case both accuracy and performance have been improved.

It is suggested that the idea of use of a certain amount of flexibility within the schedule builder, as demonstrated by the performance of *pref_3*, seems to be an interesting area for further exploration of improving the performance of any genetic algorithm that includes a degree of uncertainty in its evaluation of the population individuals' fitnesses, as is the case here.

There may also be scope to investigate, when using this technique, to use a combination of factory-defined preferences and GA-helpful (that is, aiding the performance of the GA) preferences in the case when factory-defined preference knowledge is 'gappy' in a hybrid schedule builder.

For both of the other methods (heuristic preferences and pre-expressed preferences), it should be noted that although both might constrain the search far more than is desirable for pure global optimisation, it may be that, given the practical time constraints and factory requirements it is better to use these modifications to give a reliable search that is less optimal in its treatment of the problem space than one that is unconstrained in its exploration of the problem space but is unstable and unpredictable (method SB1) or prohibitively expensive in computation (method of estimated fitnesses, SB6).

More work on these modifications may suggest particular applications in which they may be useful, and may indicate how further work can allow such problems to be effectively searched by GAs in future.

5. Genetic Algorithms for Multi-Objective Optimisation

'One of the driving forces in the applications of GAs is that manufacturing is becoming highly oriented towards customer production' (Raymond, 1995).

5.1 Introduction

So far in this research, the genetic algorithm has been applied to single objective optimisation problems. This chapter deals with the use of genetic algorithms for performing multi-objective optimisation, and includes some detail on the use of multi-objective optimisation GAs (MOGAs) for solving scheduling problems. It includes discussion of the reasons for exploring multi-objective rather than single objective optimisation, with particular reference to this application, and explores methods found in literature to conduct such a task, comparing the available implementations. In addition, three methods are suggested for the experimental work that will be conducted in Chapter 6, for performing multi-objective optimisation with a GA on test problems based on the application discussed in Chapter 3. Two initial comparisons of these methods were presented in Shaw and Fleming (1996a), and Shaw and Fleming(1996b). These results are now discussed in more detail.

In this and subsequent chapters, the main theme of this thesis will be discussed, based on the proposition that it is possible to implement a robust genetic algorithm that allows the optimisation of more than one objective from a scheduling problem within the same GA run. Such a GA should have improved performance in terms of optimisation and search capabilities, and within a time scale satisfactory to the application used.

The implementation of a GA method that aims to provide multi-objective optimisation for a practical problem such as scheduling should aim to satisfy the following criteria:

- that the optimisation capability is satisfactory. The method needs to be able to find values for the objectives that can match or improve on those found by a single-objective GA (SOGA). It also needs to be sure of fully covering the available search space to find the best solutions available for multi-objective optimisation.
- that the method can meet feasible time requirements - does the time the method takes to run fit in with the requirements of the factory?
- that it is reasonably 'user-friendly' - being feasible and reasonably simple to implement, not demanding an extensive amount of work from the user for, say, setting parameters, and lending itself to a generic development of a method for similar problems

- that it provides robust and reliable results, which can be used with confidence in the factory.

5.1.1 Finding a technique to work in the real world

A modern manufacturing environment is defined by a huge number of variables involved in the determination of the problem, leading to multiple objectives that the factory would like to optimise. Methods for scheduling with multiple objectives have often used one single function, which aims to combine the costs in some meaningful way. This means predefining priorities and preferences, and adjusting weights within the function to reflect accurately the decision-making process.

However good a technique in theory, if it is to be adopted by the industry that initiated its development, it must be able to meet their needs. Furthermore, it must convince staff at all levels that it is capable of working in a way that is acceptable to them.

Manufacturers are driven by the need to make products that conform to consumers' requirements. The effects of consumers having increasingly high requirements is illustrated by Esmail and Saggi (1996) who highlight how the culture of choice fundamentally influences the new thinking in manufacturing. 'It has been suggested by numerous sources that the era of mass production is ending, and that a period of 'mass customization' is burgeoning.' Wogan (1993), stresses the problems of this effect with particular reference to the short-life product industry. Shaw (1996), stresses that, 'short shelf life of only a few days is regarded as a **virtue** by the consumer.' Fulkerson and Staffend (1995), discuss the need for agile manufacturing, with the need to tailor products to each customers' requirements, rather than produce one standard product. In short, the traditional ideas of a service personal to each customer's requirements are being combined with the economics of mass production, to provide the most competitive businesses. Production must be flexible enough to cope with much greater demands and preferences.

In the manufacturing process, these compromises and trade-offs are being made continually. It is becoming increasingly unrealistic to concentrate upon a scheduling tool that optimises only one objective and provides only one solution. It may be possible to minimize makespan, but only by penalizing other measures, and with no flexibility to meet other criteria, beyond adjusting the implementation and starting again.

Dubois, et al. (1995), point out that 'optimising a single criterion leads to a very limited view of the real problem'. Grabot and Genete (1994), agree: 'JSS [job shop scheduling]...uses various kinds of dispatching rules...each...aims at satisfying a single criterion although workshop management is a multi-criteria problem.' They also conclude that, 'the problem in scheduling is not to find a hypothetical optimal solution, but to be able to modify decision criteria when objectives change.'

An additional consideration in meeting the customers' exact needs is that representing all relevant data may be an almost impossible task. Methods of modelling improve all the time (for example,

fuzzy representation may improve the true-life reflection of data). However, it may still be extremely difficult to provide an accurate and automated reflection of the decision-making process of a human expert for a scheduling problem. Ideally, a human expert should work in conjunction with a tool that provides data on the optimal choices. The human would provide an understanding not only of the practical aspects of the shop-floor, but also the subtler political and psychological factors needed, and would be able to provide 'fine-tuning' based on these effects.

This chapter presents the background to the use of GAs as multi-objective optimisation tools. It describes the implementations of three particular GA models, which may be able to allow the user some form of accurate representation and optimisation of more than one goal in a scheduling problem.

5.1.2 Multiple Objective Problem Solving - Optimisation and Decision-Making

Hwang and Masud (1979), provide an extensive survey of multiple objective decision making, and Fonseca (1995), puts many of the methods that they discuss into the context of genetic algorithm optimisation.

Definitions within the multi-objective optimisation problem are provided as follows:

Definition - Multi-Objective Optimisation Problem

A **multi-objective** (maximization) **optimisation problem** can be described for n objective functions f , given m constraining functions, g , as:

$$\begin{aligned} & \max (f_1(\underline{x}), f_2(\underline{x}), \dots, f_k(\underline{x})) \\ & \text{where } \underline{x} = (x_1, x_2, \dots, x_n) \\ & \text{subject to } g_j(\underline{x}) \leq \underline{0}, j = 1, 2, \dots, m \end{aligned}$$

Clearly, it follows that this definition can be equally applied to minimization problems.

Then an **optimal solution**, \underline{x}^* , to a multi-objective optimisation is one that results in the maximum value of each of the objective functions simultaneously. That is, \underline{x}^* is an optimal solution to the problem iff $\underline{x}^* \in X$ and $\underline{f}(\underline{x}^*) \geq \underline{f}(\underline{x})$ for all $\underline{x} \in X$, (Hwang and Masud, 1979).

This means that a solution must be found in which all values of the objectives are optimal for each objective function f . This is not necessarily a trivial task, as we shall see.

The three major methods that have been used as methods for allowing GAs to solve multi-objective optimisation problems, particularly within the context of scheduling problems, are:

- Weighted sum methods
- Population based methods
- Pareto-optimal methods.

In the following sections, the earlier methods used for multi-objective optimisation are compared with current GA multi-objective optimisation methods, and discussed in greater detail. Implementation details of the three methods are also provided.

An important difference between methods used for single objective optimisation and multi-objective optimisation is that when multiple objectives are involved, there are almost certainly multiple solutions available for the answer. Indeed, the problem is further complicated, as Hwang and Masud (1979) comment: 'since it is the nature of MODPs (multiple objective decision problems) problems to have conflicting objectives, usually there is no optimal solution.'

The reason for this is that many problems involve a trade-off between the objectives that means that changing the emphasis of one objective means that a different subset of the available solutions will be explored. The three methods in 5.2, 5.3 and 5.4 are inspired by the different approaches to allowing these trade-offs to be exploited. The **weighted sum** method uses different values of weights applied to each objective in order to provide the particular emphasis. The **Pareto-based methods** use the idea of non-dominated solutions to express possible solutions, which satisfy both the trade-offs available and potential different emphasis that might be applied. This idea is illustrated in Figure 5.1, in which three non-dominated solutions points represent various possible solutions. The **parallel populations** method attempts to tackle the trade-off more implicitly, by treating each objective individually within a different population, and using some form of interaction between each population to try to ensure that all compromises between objectives are covered.

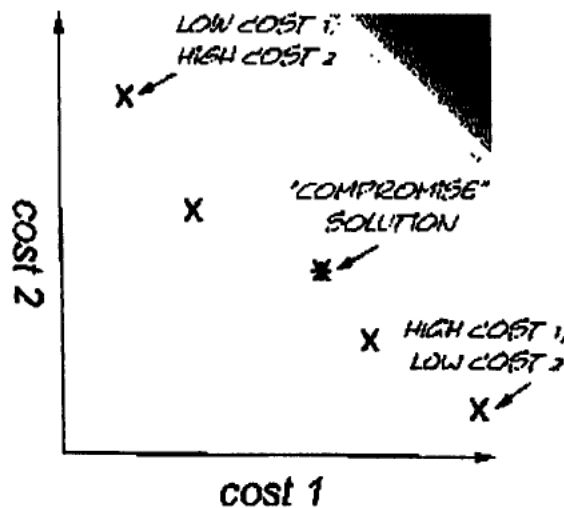


Figure 5.1 - Trade-off between two conflicting objectives

In addition, scheduling provides highly constrained problems, as we have already discussed. Fonseca and Fleming (1994), emphasize that additional choices can be made as to which constraints to enforce and which to relax. As we have seen in chapter 4, the use of preferences in schedule builders within GAs for scheduling demonstrates how the use of preferences rather than hard-defined constraints can affect the optimisation performance and exploration of potential solutions. Therefore, the potential of a method that allows the choice of different emphasis on constraints is also worth considering.

Categories of Decision Making Processes

Given that there will be more than one solution available, the introduction of the 'decision maker' (DM) is of particular importance, as well as highly relevant to the ideas of flexible scheduling systems. Hwang and Masud (1979) provide a taxonomy of the way in which methods allow the user to express preferences when presented with a selection of alternative solutions:

- no articulation of preferences is needed from the DM
- *a priori* articulation of preference - expressed before the search is run.
- *interactive* articulation of preferences - the preferences are expressed and can be altered as the search is running
- *a posteriori* articulation of preference - expressed after the search is run, the DM chooses from a set of possible solutions provided at the end of the run

A priori methods clearly allow a degree of certainty by fixing the targetted outcome in advance of the optimisation run. However, doing this is likely to have some shortcomings, which will be discussed in 5.2, when the weighted sum method, which is generally used as an *a priori* method, is introduced. *Interactive* methods allow the user both to react to changing situations in the application problem, and to interact with the optimisation process by updating objectives or goals as the optimisation is conducted. *A posteriori* methods have the advantage of allowing no possible solution to be eliminated prematurely in the optimisation process by preserving all potential outcomes. However, this may also be a disadvantage if there are a large number of solutions as the user may be presented with an excessive number from which to make a choice. Further problem information may be needed to provide context of the solution, or additional DM methods may be needed to enable the user to make the final choice. Ultimately, the preferred method of DM is likely to be influenced by the problem requirements.

5.1.3 Methods used for multi-objective optimisation by GA.

In the following sections, the implementations of the three techniques are reviewed in order to explore multi-objective optimisation in the context of a scheduling genetic algorithm. The generic implementation is discussed, followed by the specific implementation used in the experimental comparisons of the three methods. The first method is a commonly used method of combining all objectives into an aggregate, **weighted sum** function. The second is based on the use of a

migration model, in which sub-populations are assigned to work on one objective each. The third is a global genetic algorithm that uses the ideas of Pareto optimality to select sets of possible solutions that satisfy criteria for optimising several costs simultaneously.

5.2 *Weighted Sum Method*

5.2.1 Background

The weighted sum has been used commonly for addressing the problem of how to deal with more than one objective in an optimisation problem by combining the objectives into a single objective function. This method is well documented in literature, and examples of its implementation can be found for a huge range of applications in which an optimisation of more than one objective is required.

Weights are used to try to balance the difference between the objectives, and to allow the search to deal with each objective according to the user's preferences.

Thus for minimizing objective functions $f_i, i=1, \dots, n$, we would create a new single objective, O_s , to minimize as: (5.1):

$$\begin{aligned} &\text{minimize } O_s, \\ &\text{where } O_s = \sum_{i=1}^n w_i f_i, \end{aligned}$$

where f_i are the objective functions, and weights w_i are chosen by the user. (5.1)

The approach is simple and allows the user to proceed with the optimisation with little adjustment to the running of the GA. However, the main problem lies in the choice of weights. Care must be taken in their choice, and a certain amount of knowledge of the objective functions may be needed to ensure that the search space is being searched adequately. Without such care, one particular objective could assert more influence on the sum than others. Richardson et al. (1989), comment, 'typically the GA user finds some ad-hoc function of the multiple attributes to yield a scalar fitness function. Often-seen tools for combining multiple attributes are [...] weights for linear combinations of attribute values...the final GA solution is usually very sensitive to small changes in the [...] weighting factors.'

This method reflects the concerns found with other methods in which there is a need to choose 'weights or similar'. Commenting on this problem with regard to other methods, Fleming and Pashkevich (1985), comment that, 'it is desirable that the declared goals of the designer specify precisely what is required rather than be artificial devices – it is necessary, therefore, to construct a sound repertoire of well-posed design specifications'.

With particular reference to the implementation of a weighted sum GA to provide multi-objective optimisation, Grabot and Genete (1994), discuss a method that uses weights within a fuzzy logic system. This modifies the respective influence of an aggregation of elementary priority rules in order to try to meet a multi-criteria problem. They point out that the importance of a rule or objective is often neglected in research works, but is often relevant to the users.

In many implementations, the user provides only the weights *a priori* and so if the problem should change, the emphasis of the weights may no longer be accurate. For example, if the emphasis on the influence of a particular objective changes within the problem, then the weights must be redefined with equal care to reflect this. Morad and Zalzal (1995), suggest an approach in which the weights are altered to allow the exploration of other possible solutions for a manufacturing cell problem. ✓

Another disadvantage of using the weighted sum for exploring the available trade-offs between objectives inherent in a multi-objective optimisation problem is discussed by Fonseca (1995). The **search front** is the boundary attained by the successive points of the search. By the nature of its linearity, if the trade-off surface is concave, the WS search front cannot reach inside the concave surface.

The other major advantage of these methods is that the use of weights does not actually impair optimisation performance (Srinivas and Deb, 1994). Therefore, although the user can be sure of finding a good solution in the Pareto-optimal sense, it will be the one Pareto-optimal solution with the pre-defined priorities given to the weights. Murata (1997), suggests avoiding any problems caused by such linearity by implementing a form of weighted sum in which the weights are assigned randomly for every generation.

5.2.2 Implementation

The following simple implementation was used for the experimental work upon MOGAs for scheduling (Figure 5.2).

Pseudo code for weighted sum

- generate initial population**
- **start iteration**
 - **evaluate population to find values for f_1, f_2, f_3 for each individual**
 - **determine value of $O_i = \sum w_l f_l, l=1,2,3$, for each individual**
 - **assign fitness to each individual based on O_i**
 - **population selected for crossover according to fitness**
 - **crossover takes place to produce new population**
 - **mutation takes place**
 - **end iteration**
- **solution is selected as individual with best value of O_i**

Figure 5.2 - Pseudo code for weighted sum

The weights were set simply to normalize the objectives available, the maximum values of each being known or calculated from factory data, ((5.2)

$$\begin{aligned} \text{weight1} &= \frac{1}{(\max(\text{cost 1}))} \\ \text{weight2} &= \frac{1}{(\max(\text{cost 2}))} \\ \text{weight3} &= \frac{1}{(\max(\text{cost 3}))} \end{aligned}$$

(5.2)

5.3 Parallel Populations

5.3.1 Background

Chipperfield (1995) reviews the use of migration genetic algorithms as parallel tools for optimisation. In global genetic algorithms, all individuals belong to one population, and there is no restriction on which individuals may mate with others. Migration models use the division of the population into *sub-populations*, or *species*, and confine the reproduction to create new individuals within the species overall - a *polytypic* model (literally 'many types'; breeding is distributed amongst semi-isolate groups; (Chipperfield, 1995). Although most of the papers discussed are using PGAs with the motivation of trying to improve performance or robustness, Schaffer (1985), and Collins and Jefferson (1991), exploit this particular GA model for the purposes of multi-objective optimisation.

Schaffer (1985), provided the VEGA (Vector Evaluated GA) as one of the first suggested methods for solving multi-objective optimisation problems by GA. Although it was actually Schaffer who introduced the idea of exploiting the Pareto front in order to find multi-objective optimising solutions in this paper, the VEGA worked on different principles. This method influences many approaches to multi-objective optimisation for GAs in its use of *population-based methods*. The VEGA method worked by splitting the GA population into subpopulations, each of which were evaluated according to a different objective chosen from the vector of objectives available. (See Figure 5.3).

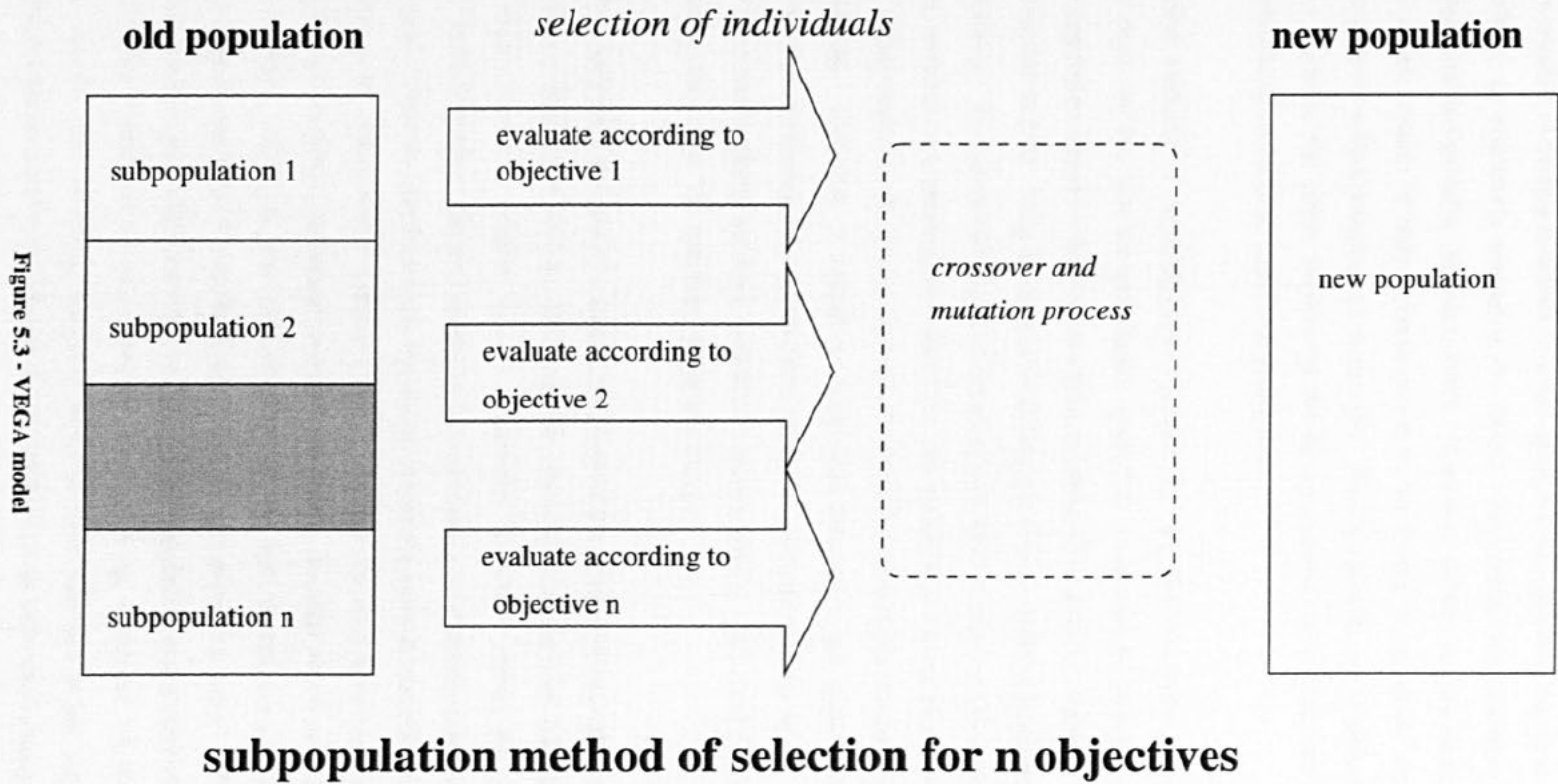


Figure 5.3 - VEGA model

The method is capable of finding solutions to a two-objective test problem, but it is criticized by Richardson (1989), as effectively averaging the fitness components and therefore not actually treating the objectives individually, (Fonseca, 1995). Horn et al. (1994), remark on the flaw of this method; '[it] seems capable of only extreme points on the Pareto front, where one attribute is maximal, since it never selects according to *trade-offs*.' They suggest the use of niches (see section 5.4.3) to try to moderate this effect, introducing the NPGA (niched Pareto GA), and apply it to a real-life test problem of optimising water-well placements.

Population-based methods of multi-objective optimisation are less common generally in GA literature, and there are few that are specifically applied to scheduling problems. Cohoon et al (1987), test a population-based method on an Optimal Linear Arrangement Problem, which is NP complete and related to TSP. They describe the advantages of incorporating *allopatric speciation* by using this method. This allows the rapid evolution of new species that have been geographically separated. By working on separating the objectives, the scaled-down populations are allowed to evolve better, uninterrupted by the confusing amount of genetic information potentially available in a larger population. However, it should be noted that this might not necessarily be to the advantage of the true multi-objective optimisation process. Although Cohoon et al., mention one of the uses of their methodology as being useful for a system where, 'supposing fitness is a multi-objective function', they add, 'we will not explore this further'.

→ Kursawe (1991), addresses the use of a similar idea within the context of evolution strategies, by suggesting a similar idea to the VEGA. However, the objective function used for selection of the individual is either chosen at random, or at a probability defined to reflect the user's / DM's preference for each objective. He also mentions the importance of ensuring that there is enough **recessive** material within the chromosomes. Recessive material is genetic material that may not be directly relevant to the fitness as it is currently being evaluated, but may have a useful bearing on the problem given an absence of **dominant** material, the genetic material which does directly effect the objective function. Kursawe also recommends that 'the best results were achieved with a probability of around one third for exchanging dominant and recessive genes'. The notion of dominance and recessive genes here can be translated to the population based methods effectively. Genes that do not contribute to the objective that is being evaluated of that particular subpopulation can be seen as being recessive, whereas those relevant genes can be seen as dominant. This would therefore indicate that the probability of an individual changing objective (or migrating to another subpopulation) should be one-third. This is explored experimentally in 5.3.3.

Husbands and Mill (1991) use a variation on the theme of speciation for multi-objective optimisation. Rather than having subpopulations, two 'species' are used, in order to exploit the principles of co-evolution, which allow the effects of each species on the other to influence the

evolution of both. This idea of the 'Red Queen theory' where 'an evolutionary change to one species is experienced by co-existing species as a change in their environment' allows Husbands and Mill to take an innovative approach to optimising for two different subproblems, rather than just two objectives of the same problem.

5.3.2 Additional motivation for the use of parallel populations

In this particular work, there is an additional motivation for wishing to work with parallel populations. It is initiated by the work with GAs on the TSP, which has shown that there are a number of possible crossover operators that can be used to create valid offspring from two parent chromosomes. Whitley, Starkweather and D'Ann Fuquay (1989) argue that different operators will exploit different features of the problems. They recommend the edge recombination operator for problems where the relative order of jobs within the permutation is important. There are similar considerations when choosing a suitable mutation operator.

In Chapter 3, this choice was illustrated for a single objective problem, in which three objectives were combined into one and the best operators were chosen based on this single objective.

A GA which was configured to minimize each of the three given objectives singly, performed best given different sets of operators to allow overall exploration of the multi-objective optimisation problem. The details of experiments to justify this are presented below in section 5.3.2. Whilst Bagchi et al. (1991), use a probability-based selection of operators when requiring more than one for a particular problem, the results here motivated our experiments using a VEGA-type implementation of GA, allowing each objective / population its own set of operators. These allow the optimisation of each cost to be treated mostly as a distinct process.

Problem Description

The details of the application providing the problems for this work have already been presented in Chapter 3. In summary, there are three costs / objectives to be minimised:

- Cost 1 - the placement of jobs within a schedule at all
- Cost 2 - the lateness of any order regarding its delivery time
- Cost 3 - the variation in the end of the run of each of the 13 production lines.

Results of Runs

Nine sets of runs were undertaken on each of the costs singly, by using the simple GA described in Chapter 3. Each cost was isolated within the weighted sum to allow it to be treated as the sole objective. The comparison of the results is made very simply by averaging the best value found on each run for each combination of operators, shown in Table 5.1, Table 5.2 and Table 5.3. The two crossover operators and four mutation operators introduced in Chapter 3 were considered in pairs, for each possible combination of crossover and mutation operator. These comparisons could be

performed in greater detail, and statistical tests could be carried out to measure the significance, if any, of the difference in performance for each set of operators. In each table, the best value for each operator is highlighted.

	ERX	OX	Average
InvMut	0.0000	0.0000	<u>0.0000</u>
PosMut	0.0111	0.0056	0.0083
Splice	0.0056	0.0056	0.0056
Swap	0.0111	0.0000	0.0056
Average	0.0070	<u>0.0028</u>	

Recommended Operators for Cost 1: Order Crossover & Inversion Mutation

(Key: *InvMut* - inversion mutation; *PosMut* - position mutation; *splice* - splice mutation; *swap* - swap mutation; *ERX* - edge recombination crossover; *OX* - order crossover)

Table 5.1- Average best value found over run for cost 1

	ERX	OX	Average
InvMut	1.3333	1.8333	1.5833
PosMut	1.0833	1.3333	1.2083
Splice	1.5000	1.7500	1.6250
Swap	1.4166	1.3333	1.3750
Average	1.3333	1.5625	

Operators Recommended for Cost 2: Edge Recombination Crossover & Position Mutation

Table 5.2 - Average best value found over run for cost 2

	ERX	OX	Average
InvMut	0.5083	0.4894	<u>0.4989</u>
PosMut	0.5557	0.5347	0.5452
Splice	0.5335	0.5506	0.5420
Swap	0.5610	0.5406	0.5508
Average	0.5396	<u>0.5288</u>	

Recommended Operators for Cost 3: Order Crossover & Inversion Mutation for best result

Table 5.3 - Average best value found over run for cost 3

The following sets of operators were shown (simply) to provide the best results for the optimisation of each cost individually:

- **Cost 1 (non-placement)** - Order Crossover & Inversion Mutation
- **Cost 2 (lateness)** - Edge Recombination Crossover & Position Mutation
- **Cost 3 (shift ends)** - Order Crossover & Inversion Mutation

It has already been shown by Whitley et al. (1989), that the Edge Recombination Operator performs well on problems where the optimal schedule depends on relative positions of jobs. An example of this is cost 2, where a highly constrained, time-demanding job ahead of another in the order can affect the lateness of the rest of the jobs. The order crossover performs better upon problems that depend on absolute positions of jobs / genes within the chromosome, such as the non-placement cost. Putting a job that is difficult to place later in the chromosome means that the job will have less available possibilities when it comes to being assigned a line within the schedule builder. However, it may be easily placed if given first choice of all the lines at the beginning of the chromosome. The shift end problem depends on much the same factors. This performance factor is also reflected in the choice of mutation operators. Indeed the same combination of operators are chosen for costs 1 and 3, whilst the position mutation, which largely preserves relative positions (and 'edges' or pairs of neighbouring genes) is chosen by cost 2.

Conclusions of Experiments

These results are somewhat informal, but in conjunction with the conclusions of Whitley et al. (1989), they provide some indication that there may be a need for different sets of operators to work on the individual objectives if such an implementation is possible.

It is indicated by this study that whilst costs 1 and 3 appear to have similar requirements, cost 2 seems to rely on different performance factors for the optimisation process to be most successful. It is suggested that some attention be paid to the operator requirements of individual costs in the use of multi-objective optimisation for scheduling problems, a consideration that may not occur so frequently for problems that use binary coding and standard crossover operators.

Whether it is more important in optimisation to improve the performance of the GA by using well-chosen operators or by implementing the multi-objective optimisation techniques remains to be seen. Experimentation should show how important it is for the coverage of multi-objective optimisation that each cost is allowed to benefit from its own preferred operators.

5.3.3 Implementation of *parasoga*

As a result of the above conclusion, the parallel populations implementation that is used subsequently ('*parasoga*' or 'parallel populations with single objectives GA') in this work will be configured to as to allow different sets of operators to work on each subpopulation. The model also differs from the VEGA-like implementation in that it uses the epoch and migration cycles to allow each subpopulation a chance to evolve using these operators before the subpopulations are

merged. The amount of migration between subpopulations was set at one-third, as described in section 5.3.3.

Pseudo code for the migration model is used here with three sub-populations to optimise three objectives (Figure 5.4)

```
generate initial sub-population i ('subpop_i'), i=1 to 3
start iteration - loop for generations to run between each migration
    • iteration for epoch of each of three subpopulations;
    • for i=1 to 3
        • evaluate subpop_i to find values for fi for each individual
        • assign fitness to each individual in subpop i based on fi
        • population selected for crossover according to fitness
        • crossover using takes place to produce new sub-population i
        • mutation takes place
    • end iteration
    • % migration takes place
    • for i=1 to 3
        • new subpop_i is formed from proportion of each subpop selected at random
    • end
    • end iteration - loop for generations to run between each migration
    • solutions are selected from all three generations that best meet the criteria for multi-objective optimisation
```

Figure 5.4 - Pseudo code for parasoga

Comparison of Migration Levels for Parasoga

A suitable level for the migration in the *parasoga* scheme must be found. Initial tests using a scheme where a third of the population remained and the other two thirds exchanged showed to perform poorly. This was further examined by running a batch of tests that set the migration level at five levels - 10%, 30%, 50%, 70% and 90% to see whether this would improve performance in the model.

The genetic algorithms were allowed to run over thirty generations, with twenty individuals per sub-population. Each level is run ten times, to allow some averaging of results. As we can see from the figures of comparative performance for cost 2 and cost 3, (Figure 5.5 and Figure 5.6), little improvement took place beyond the 20th generation, and in the case of cost 2, beyond the fifteenth generation. The performance for cost 1 is not illustrated because, as already seen in Table 5.1, this proved simple to minimise in these experiments and no informative variation was shown between levels.

Comparison of the best values found at each level at (a) the tenth generation (b) the fifteenth generation (c) the final generation, all proved that the difference between migration level

performance at these stages was statistically insignificant. However, it is clear from both graphs (Figure 5.5 and Figure 5.6), that the 50% level seems to perform best for both costs.

This means that each sub-population would retain a larger number of its own individuals with only 25% migrating to the other population. Higher levels might allow too much genetic diversity in the sub-populations after migration, hindering the search rather than improving the performance.

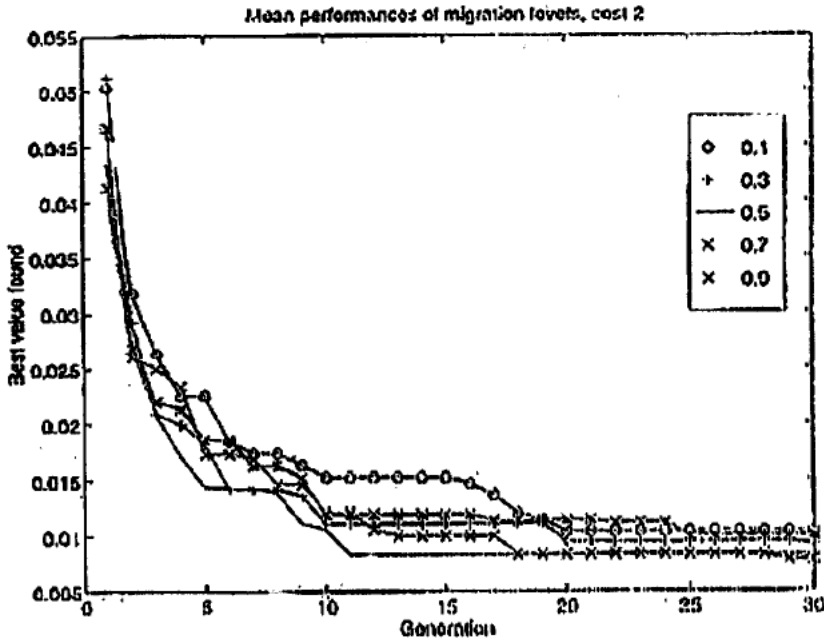


Figure 5.5 - Comparative mean performances of migration levels for cost 2

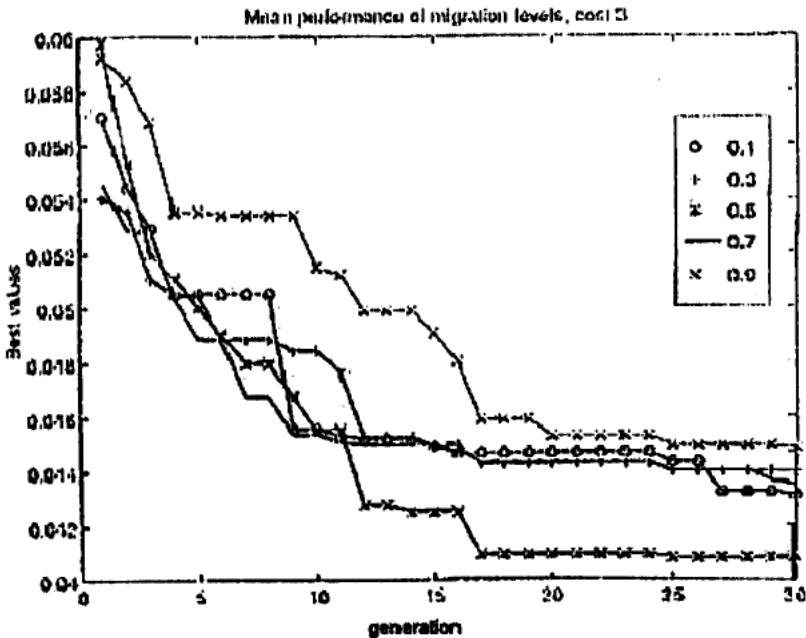


Figure 5.6 - Comparative mean performances of migration levels for cost 3

This concludes the experiments for implementing the PP method.

5.4 Pareto-based methods

5.4.1 Background

Although Schaffer (1985), suggested the idea of exploiting Pareto-optimal ideas to solve a multi-objective optimisation problem using GAs, he did not do so as such, concentrating on the VEGA instead.

Pareto-based methods differ from population-based methods in that they allow selection of individuals according to some consideration of the effects of every objective simultaneously, rather than picking a single objective on which to base the selection, as with the parallel population methods. However, when finding solutions according to several criteria, there is likely to be a selection of possible solutions available, according to the different emphasis put on the available trade-offs between objectives, by the DM. As seen in Figure 5.1, a DM could choose between emphasis on cost 1 or cost 2, resulting in different solutions.

Fourman (1985), introduced a forerunner to the idea of selection of individuals by looking at the values of each objective separately. A form of tournament selection is used, in which individuals are selected by comparison of each objective's values in the order of a pre-assigned priority (either user-defined in advance, or randomly assigned for each comparison). Fonseca (1995), comments that this method corresponds to the averaging of fitness across the components.

The use of Pareto-optimality on a vector of all the evaluated objectives has been demonstrated in various implementations, e.g., Goldberg (1989), Fonseca and Fleming (1993), Horn, Nafpliotis, and Goldberg (1994), Srivinas and Deb (1995). Pareto-optimality applied to a set of individuals, which represent potential solutions to a multi-objective optimisation problem, can be defined as follows:

Definition of Pareto-Optimality

For the n-objective maximization problem, let the solutions of the problem be measured by an n-dimensional vector in which the *i*th element is the value of the *i*th objective's fitness for that individual.

Then:

i) For any two vectors, \underline{x} and \underline{y} , $\underline{x} < \underline{y}$ iff $x_i < y_i$ for all *i* for which the two vectors are defined, and we say that \underline{x} is dominated by \underline{y} . Similarly, $\underline{x} > \underline{y}$ iff $x_i \geq y_i$ for all *i*.

ii) For any two vectors, \underline{x} and \underline{y} , if neither $x_i < y_i$ nor $x_i \geq y_i$ for all i for which the two vectors are defined, then \underline{x} and \underline{y} are non-dominated.

Example

For a problem with 4 objectives, let vectors \underline{x} , \underline{y} , and \underline{z} be such that $\underline{x} = [4 \ 7 \ 3 \ 5]$, $\underline{y} = [2 \ 6 \ 1 \ 4]$, and $\underline{z} = [5 \ 7 \ 2 \ 8]$. Then \underline{x} dominates \underline{y} , as $x_i > y_i$ for all x_i in \underline{x} ($i=1, \dots, 4$). However, neither $x_i > z_i$ nor $z_i > x_i$ for all $i, i=1, \dots, 4$, and therefore \underline{x} and \underline{z} are non-dominated, and can therefore be regarded as equivalent solutions to the problem.

In the above example, both \underline{x} and \underline{z} could be presented as potential solutions to the problem out of that set of individuals, and information from the DM would be needed to specify the priority of the objectives in order to make the final choice between \underline{x} and \underline{z} .

Goldberg (1989), suggests a method of Pareto-ranking in which a rank of 1 is assigned to the non-dominated individuals in a population. These are then effectively ignored, and a rank of 2 is assigned to the individuals that now dominate the rest of the population - a 'layering' of the levels of domination. Selection can then be based upon this ranking. This is shown in Figure 5.7

Srinivas and Deb (1994), implement Goldberg's suggestions with the addition of sharing, as discussed in 5.4.3, in the *Non-dominated Sorting GA*. The sharing is implemented on the phenotypic values rather than the objective values, and tested on three two-objective test problems, including two previously used by Schaffer (1985). The method is demonstrated to provide a wider selection of solutions along the Pareto-optimal front than the VEGA, as might be expected given the documented lack of ability of the VEGA at finding an even spread of such solutions.



Figure 5.7 - Goldberg's Pareto Ranking Method

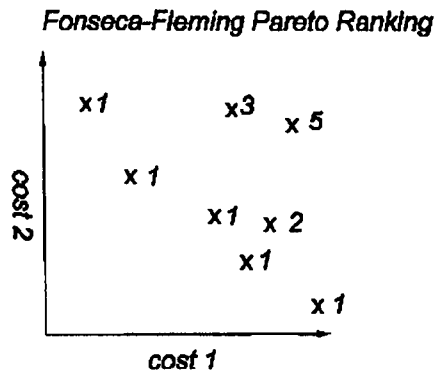


Figure 5.8 - Fonseca and Fleming Pareto Ranking Method

Fonseca and Fleming (1993), use the Pareto-based ranking assignment differently. This assignment takes place by ranking the individuals according to how many other individuals dominate them in the population. Figure 5.8 illustrates this ranking.

Horn et al. (1994), offer an alternative method, in their Niche Pareto GA (NPGA), which uses Pareto-optimality in tournament selection, the dominant individual in a pair winning the tournament. In the case of a draw, the equivalence class sharing is implemented, in which the individual with the fewest neighbours in its niche or neighbourhood wins.

The work of Surry, Radcliffe and Boyd (1995) on the COMOGA method (Constrained Optimisation by Multi-Objective Genetic Algorithms) develops a detailed discussion and methodology of dealing with real-life, highly constrained problems - in this case, gas networks - in addition to performing multi-objective optimisation. They use a Pareto ranking scheme, which is measured by the constraint violations of the solutions. There is also overlap with the VEGA, as they purposefully exploit the VEGA property of finding extreme solutions, by allowing a varying proportion of the population to be selected according to VEGA methods (by a single objective). This method is compared with a penalty-based GA for the same problem and displays a similar performance. However, it has the advantage of requiring many fewer parameter settings, and is thus more easily applicable. The problems of setting values to apply the penalties to constraint violations are similar to for choosing weights for the WS method.

Osyczka and Tamura (1996), use a method in which they preserve all Pareto-optimal solutions from the previous generation of the population to create the subsequent generation, adding other individuals at random. The results on test problems appear to be compared favourably with the VEGA, and Srinivas and Deb method in terms of the number of Pareto-optimal solutions found overall. This method has the advantage of being extremely simple. However, the obvious disadvantage must be that a lot of information about the search space and the population's

performance within it must be lost by disregarding any fitness-based information other than the simple Pareto-optimality.

Murata (1997) demonstrates a GA that uses a similar form to Osyczka, in which a form of elitism operates to preserve the Pareto-optimal solutions from one generation to the next. However, the objective function is then created as a weighted sum, in which the weights are assigned at random each generation, avoiding the need for a priori definition of weights. This is intended to allow a full search of the space. One disadvantage of this method must be that no matter which weights are chosen, the search front will always be linear. However, the problems usually associated with this are countered by a demonstration that this method can cope overall with certain concave search fronts. The method is applied to a flowshop problem, and this is further discussed in 5.5.

It is apparent that there is no clear agreement on the best way to incorporate Pareto-optimality into a GA. Methods that use Pareto-based ranking (e.g., Goldberg, Fonseca and Fleming, Srinivas and Deb) allow information from the whole of the population to be incorporated into the search capabilities of the GA. Methods that use Pareto-based selection or preservation as an elitism strategy, will lose a certain amount of information that may be provided by the dominated individuals in the population. Compared with Pareto-ranking, Pareto-preservation uses only the information provided by individuals with rank 1. It may be that there are individuals in the rank 2 that are very close to the Pareto optimal front, but not actually in it, which may have useful genetic information for future generations. Their effects do not contribute directly to the search. However, the use of Pareto preservation is usually combined with additional exploitation of the information contained in the remaining population, for example, from a weighted sum approach (e.g. Murata, 1997).

5.4.2 Implementation

Two forms of the MOGA, MO1 and MO2, were implemented for the experiments that follow in Chapter 6. The first was a simple MOGA in which the search used only the Pareto-based ranking. The second included fitness sharing and mating restrictions, as discussed in 5.4.3.

Pseudo-code for the simple Pareto-based MOGA (MO1)

generate initial population

• **start iteration**

- **evaluate population to find values for f_1, f_2, f_3 for each individual**
- **evaluate Pareto-rank of individual based on the dominance of each by other members of the population**
- **assign fitness to each individual based on Pareto-ranking**
- **population selected for crossover according to fitness**
- **crossover takes place to produce new population**
- **mutation takes place [optionally: according to mating restriction]**
- **non-dominated solutions are stored for all generations**

- end iteration
- solutions are selected as all individuals which remain non-dominated throughout the run

Figure 5.9 - Pseudo code for MO1

5.4.3 Enhancements to MO1

Genetic drift is the phenomenon observed both naturally and in artificial evolution where, although there are several optima available, the population settles only on one of them. It is, of course, important to ensure that this does not happen when using methods that aim to find several Pareto-optimal solutions simultaneously. *Niching methods* encourage the population to avoid genetic drift. These methods include crowding (Mahfoud, 1992), sharing (Fonseca and Fleming, 1994; Horn et al., 1994). It is desirable that this problem should be avoided within the implementation of the MOGA, to avoid losing all available options for finding schedules.

Mating restriction and fitness sharing are niching methods suggested by Fonseca and Fleming (1995) as suitable for including with the Pareto-based MOGA to avoid this problem. To promote a broad diversity within the population, it is desirable that we have solutions at the extremes of the trade-off surface. However, two parents of good fitnesses from opposite extremes of the trade-off may mate to create an offspring that is 'central' to both, and consequently of poor fitness, known as a '*lethal*'. By ensuring that individuals only mate with other individuals in their neighbourhood, which is defined as an area in which individuals of similar fitnesses are positioned, attempts can be made to prevent the deterioration of the performance due to creation of lethals.

The MO2 method is an extension of MO1 presented in 5.4.2, including sharing and mating restriction. Figure 5.10 provides the pseudo-code for this implementation.

Pseudo-code for MO2

- ```
generate initial population
```
- start iteration
    - evaluate population to find values for  $f_1, f_2, f_3$  for each individual
    - evaluate Pareto-rank of individual based on the dominance of each by other members of the population
    - assign fitness to each individual based on Pareto-ranking and fitness sharing
    - population selected for crossover according to fitness
    - crossover takes place to produce new population
    - mutation takes place within the mating restriction
    - non-dominated solutions are stored for all generations
    - end iteration
  - solutions are selected as all individuals which remain non-dominated throughout the run

Figure 5.10 - Pseudo-code for MO2

## 5.5 Other MOGAs for Scheduling

Chapters 2 and 3 have described some of the difficulties of the scheduling problem, particularly in the context of a real-life manufacturing environment. As already discussed in 5.1.1 and 5.1.2, manufacturers must be capable of tailoring their production to meet increasingly exacting demands, taking more than one objective into account in the optimisation process. Given shorter lead times than in the past, they must also be capable of reacting to change by the provision of a selection of possible schedules to match varying situations, rather than only having one set schedule that cannot meet a changed problem. Change may be external, as a customer's order changes unexpectedly, or internal, as a machine fails after it has been scheduled for use. For these two reasons, a multi-objective optimisation technique that offers a variety of solutions would be highly beneficial to manufactures.

- In the last two years, approaches similar to that presented in this thesis have been suggested, in which GAs aim to solve a multi-objective production scheduling problem. Indeed, the ranges of methods used here demonstrate that the variety in methods discussed is still evident.

Viennet et al. (1995), use a hybrid of VEGA and Pareto ideas, by using subpopulations to optimise each objective, and then find the Pareto-optimal set of solutions from the combined population. They demonstrate this on a manufacturing problem. However, the results are only briefly explained, with more emphasis on the effectiveness of using the Pareto-optimal solutions to allow the DM to have a choice in the trade-off of solutions than on the relative accuracy or performance advantages of this method.

- A combination of VEGA and Pareto-based strategies are also used in the work of Tamaki et al. (1992), and (1996). The method of *Pareto Reservation Strategy* is introduced in detail in Tamaki et al. (1992), and applied to a three-objective steel rolling process. It effectively combines a Pareto-based method with the parallel populations method. In this implementation, the selection stage uses a strategy of dividing the population into subpopulations and selecting by a single-criteria from each, as used in the VEGA (hereafter referred to as '*VEGA selection*'). This is combined with an elitism strategy of preserving any non-dominated individuals from the whole population into the next generation. In Tamaki et al. (1996), this method is reversed. The new population is created from the non-dominated individuals of the previous generation, and then either supplemented with VEGA selected individuals from the rest of the population, or limited by VEGA-selection if there are too many non-dominated individuals for the required population size.
- This method is compared against others when Tamaki, et al. (1996), provide a useful discussion of the different methods available for multi-objective optimisation in their review of MOGAs. They compare four methods on the test problem of finding the vertices of a quadrant of a sphere, these being:

- i. 'parallel selection' - that is, the subpopulation-based selection method as seen in the VEGA
- ii. Pareto rank-based selection (Fonseca and Fleming (1993),
- iii. Pareto tournament selection and sharing,
- iv. their own hybrid method of *Pareto reservation strategy*.

Method (iv) is shown to work best. However, the lack of sharing techniques in the implementation of (ii) may explain their comments that the Pareto rank-based selection only finds a few solutions.

Both Viennet and Tamaki's methods demonstrate the combination of the VEGA ideas with Pareto optimality. The *parasoga* method presented in 5.3.3 combines the use of parallel populations and Pareto-optimality, but only in the limited sense that the final set of solutions are selected from the combined population by using Pareto-optimality criteria.

Niemeyer and Shiroma (1996) provide an excellent and useful discussion of the application of MOGAs to a difficult real-life scheduling problem, and indeed their application bears many similarities to the problem discussed in this thesis. They model several factories around the country, which require schedules for around 20 products daily. They comment that, 'daily schedules need to be generated overnight; as a result the company is willing to forsake the goal of true optimality in favour of generating the best possible schedules in the amount of time available.' This agrees with the point made by Shaw and Fleming (1997), that given certain real-life applications of GAs for scheduling, the user has to interact with the further trade-off of solution quality against GA execution time. Niemeyer and Shiroma use two objectives, time and cost (i.e., expense), commenting on the trade-off typically inherent between these two measures. However, the implementation used is an interesting development on the standard weighted sum - rather than using a weighted sum of the objective values, they assign fitness according to a weighted sum of the ranks of the individual objectives:

$$\text{cost} = w_1(TM) + w_2(CM)$$

where  $w_1, w_2$  are weights,

$TM$  = value based on ranked fitness of individual's time measure,

$CM$  = value based on ranked fitness of individual's cost measure.

(5.3)

They finally include a Tabu list method within the GA, which is designed to improve diversity in the small population. This is necessitated by the need for a fast implementation by preventing new individuals from being created which are identical to those already in the population.

Extensive experiments are then conducted on five different methods to demonstrate the effectiveness of this:-

- i. Monte-Carlo evaluation
- ii. Simple Tabu search
- iii. Mutation only search
- iv. GA implemented with fitness as in (5.3 )
- v. As (iv) but with a Tabu list restriction

Comparing 30 trials of each, they demonstrate clearly that (v) works well. It would be interesting to see how this method compares to a Pareto-based approach.

Murata (1997) applies a form of MOGA as already described in 5.4 to two and three-objective flowshop problems. For the two-objective problem, five runs of the MOGA method are compared with a VEGA implementation and a constant-weighted GA (CWGA) - the weighted sum described above.

It is clear that there is increasing interest in the use of GA methods for multi-objective optimisation problems, and equally apparent that there is yet to be agreement on the best GA implementation to perform this task. Out of the two main methods of performing multi-objective optimisation using GAs, researchers are starting to combine features of both techniques to exploit both their advantages.

It is difficult to comment on the effectiveness of any implementation without a more rigorous comparison of the methods available. At this point in time, where several fundamentally different implementations of GA for solving multi-objective optimisation problems are being offered, it may make an interesting subject for a more rigorous comparison, with all available methods being compared over an extensive set of test problems. Chapter 6 will discuss an initial suggestion for providing such a comparison.

## ***5.6 Summary of Chapter 5***

This chapter has provided an overview of the use of GAs for solving multi-objective optimisation problems. The methods available can roughly be divided into three available categories. Firstly, there is the 'traditional' weighted sum approach, which does not really treat each objective as an individual optimisation goal, but remains popular as a method for performing multi-objective optimisation in the face of insufficient motivation for using other methods. Secondly there are those methods based on parallel populations, which share implementation issues with other parallel genetic algorithms, and also with the GA community more interested in artificial life techniques, as they allow the incorporation of 'biological-like' behaviour in their interactions, migrations, speciation and subpopulations. It is clear that there may be much of interest in the implementation

of such techniques for multi-objective optimisation, and the recent move towards combining these methods with Pareto-optimality may signify a renewal of interest in such methods. Thirdly, there are the methods based on Pareto-optimality, treating the population as a whole entity but examining the objectives individually. These methods are perhaps the ones that best allow true optimisation of all objectives simultaneously. The implementation of the best method of applying Pareto-optimality is not agreed, although the use of Pareto-based ranking seems to be gaining some popularity in new research. Other users of Pareto-optimality use it as a method of selecting an elite subgroup of the population, rather than implementing any form of ranking with it.

In Chapter 6, the three methods of Weighted Sum, Parallel Populations and Pareto-based ranking are compared on the scheduling problem from Chapter 3.

# **6. Comparison of Three Multi-Objective Optimisation GA Schemes for A Real Life Scheduling Problem**

## ***6.1 Introduction***

In Chapter 3, a simple, single-objective genetic algorithm was applied to a scheduling problem based on real-life data taken from Pennine Foods. Chapter 5 discussed the implementation of genetic algorithms as optimisation tools in the form of multi-objective genetic algorithms (MOGAs), and described three possible such configurations. These themes are combined in this chapter, as experiments are discussed which apply these MOGAs to further problems based on the Pennine Foods data.

It has already been shown in chapter 3 that GA methods are capable of finding schedule solutions to the manufacturing problem. This chapter is more concerned with the comparison of the MOGA methods for providing effective optimisation of a multi-objective scheduling problem.

The difficulties of comparing the solutions of MOGA implementations are explored. This is an area that has hardly been covered in the literature, but which must necessarily be explored if any conclusions are to be drawn from such comparisons. Various methods of comparing performance of MOGAs are discussed, together with the conclusions that might be drawn from these comparisons. The conclusion of this chapter discusses the effectiveness of the MOGA implementations for scheduling problems, both concerning this particular problem, and in the general case. Some suggestions are made towards deciding which method is best for a particular problem.

## ***6.2 Background to comparisons of MOGAs***

The comparison of simple, single-objective GAs is commonly performed in terms of minimisation performance. This simply means that the method that finds the minimal solution is indicated as being the best; this may be measured in terms of time taken or simply the method that finds the minimal value, regardless of time taken. Yet as already seen in Chapter 5, the definition of a minimal solution when working with multiple objectives is less clearly defined. If the objectives conflict, as in this problem, there will be a set of non-dominated solutions to the problem rather than a single overall solution. Therefore, it is necessary to compare sets of solutions rather than one single value. Both Schaffer (1985) and Fonseca and Fleming (1996) comment on the difficulty of comparing MOGAs. To compare the performance of MOGAs designed to optimise a real-life,

multi-objective scheduling problem, further aspects of performance should be taken into account. These should include:

- distribution of solutions across the trade-off surface,
- quality of solutions in terms of optimisation, in terms of values found, time taken or use of the optimisation performance measures.
- robustness - the methods' reliability in providing answers to a certain standard
- interpretation of results in terms of the scheduling problem and the manufacturer's requirements.

These aspects are explored in more detail in the following sections. Comparison methods can also be divided into 'visual' and 'non-visual'; non-visual methods are usually statistical in nature. These are discussed below.

### 6.2.1 Visual comparison of minima

In the literature, GA optimisation performances are generally compared only in terms of finding the minimum. The standard method of doing so has been to plot the performance of, say, the best or mean fitnesses found over time for the run of the GA. For two MOGAs, it could be possible to plot the individual costs' fitness performance separately, but given the interaction between the costs, this does not provide the ideal illustration of the minimising behaviour of the algorithms.

If the fitness performance over time is difficult to assess, the values found by way of solution, or sets of solutions can at least be compared. Many of the problems used in MOGA literature to date have been for test problems with just two objectives. For these, the solutions for one cost against another can be plotted, as shown in Figure 6.1.

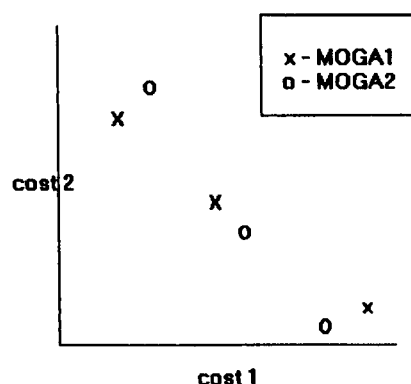


Figure 6.1- Example of plot of best solutions found for two costs for two objectives

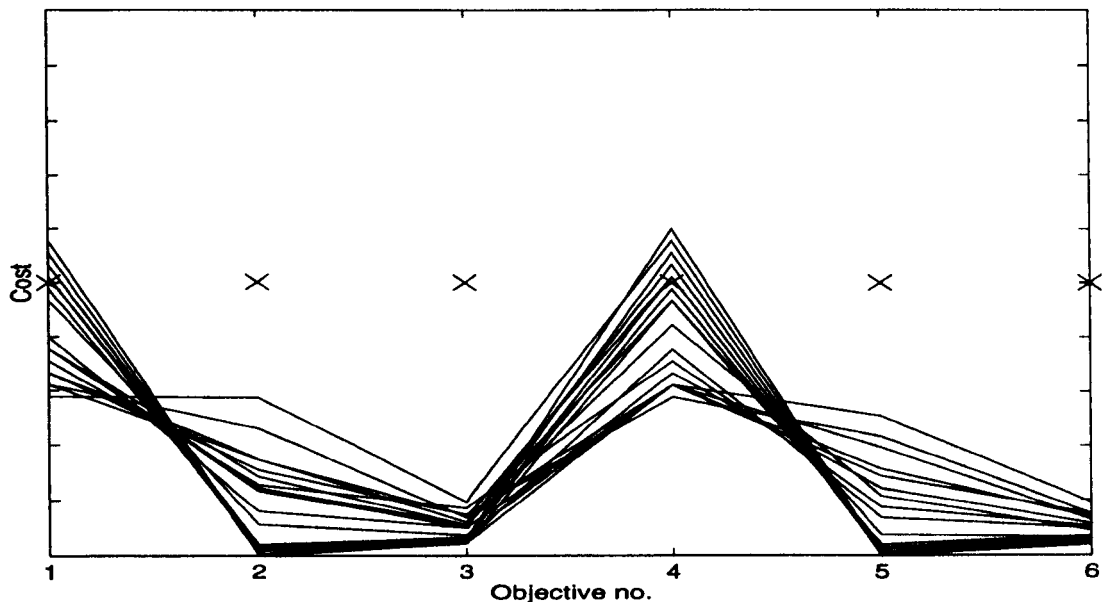
Examples of this can be seen in much of the literature on MOGAs (e.g., Schaffer, 1985; Viennet et al., (1995), Murata, 1997). This is simple to use to show the methods' relative abilities at



minimising costs 1 and 2. Here it can be seen that MOGA1 finds the better value overall for cost 1, and MOGA2 finds the best for cost 2. However, many problems are likely to have more than two objectives and the problems of visualising such plots when the number of objectives is greater than 2 is evident. It quickly becomes unfeasible to use this form of visualisation to compare such problems.

Murata, (1997), avoids this problem by comparing the performance of GAs on a three-objective problem by pairwise plots of combinations of the three objectives; cost 1 against cost 2, cost 2 against cost 3, and cost 1 against cost 3. This compromise is a reasonably satisfying method of allowing the information to be visualised. However, by only showing two dimensions of the trade-offs that might be visible in a three-dimensional plot, the effects of the third objective on solution quality cannot be seen.

A different form of plot for comparing solutions of more than one objective may be needed to allow us to draw any meaningful conclusions. Figure 6.2, uses a trade-off plot to demonstrate the different solutions for a six-objective MOGA (Schroder, 1997). The different objectives are numerated along the x-axis, whilst the values of each are plotted against the y-axis. This allows alternative visualisation of information from that shown in Figure 6.1 for more than two objectives.



**Figure 6.2 - Example of plots for the comparison of non-dominated solutions for a six objective problem.**

Whilst this and other similar graphs allow a certain amount of comparison to be made, the need for methods that can visually demonstrate relative performances when a large amount of data or runs are available is still needed.

## 6.2.2 Statistical Comparisons

### *Niemeyer and Shiroma Measure*

Niemeyer and Shiroma (1996), define an *overall performance indicator* for their two-objective problem (the two objectives being 'Time' and 'Cost') comparing five methods, as follows:

$$\text{Total Fitness of Method} = 1 - ((\text{Time Fitness} \times \text{Weight}_{\text{Time}}) + (\text{Costs Fitness} \times \text{Weight}_{\text{Costs}}))$$

where

$$\text{Time Fitness for this Algorithm} = \frac{\text{Best Time for this Algorithm} - \text{Best Time for all Algorithms}}{\text{Worst time for all Algorithms} - \text{Best Time for all Algorithms}}$$

$$\text{Cost Fitness for this Algorithm} = \frac{\text{Best Cost for this Algorithm} - \text{Best Cost for all Algorithms}}{\text{Worst Cost for all Algorithms} - \text{Best Cost for all Algorithms}}$$

*Weights are chosen by the user to provide the emphasis within the weighted sum.*

The closer the value to 1.00, the better the performance of the algorithm at optimising all costs simultaneously; the best algorithm will receive a score of 1.00 exactly. This compares the performance in terms of finding the relative minimum of each cost.

The comparison of the performances is relative between the methods rather than measured against the ability to find the globally optimal answer to the problem - if one exists. However, the particular interest both of this research and of Niemeyer and Shiroma is to find MOGA methods, which can work on practical scheduling problems, in which the solution or solutions are simply not known. In this case, relative performance is certainly acceptable.

### *Statistical performance assessment*

The *statistical performance assessment* (Fonseca and Fleming, 1996) is used to provide a demonstration of the 'typical performance' of a genetic algorithm. The method provides a quantitative comparison of MOGA performance, based on the *statistical attainment surfaces* of each method, given a set of experimental runs. The surfaces define, 'the tightest goal vectors which, given the data, are known to be attainable'. They show both optimisation power and distribution of non-dominated points across the trade-off surface.

The idea of attainment can be applied to show successive levels of optimisation capability. The user might calculate, for example, the 'best attainment surface'; that is, the points which have only been attained in one of all the runs. The 'median attainment surface' is defined by those points that were attained in 50% of the runs. The 'worst attainment surface' is formed by those points attained by all runs. Having found these surfaces, it is possible to compare the relative performances of

multi-objective genetic algorithm schemes and to provide statistical demonstrations of their best, median and worst-case performances. This is particularly appropriate here, as one of the major problems with transferring genetic algorithm techniques into industry is being able to demonstrate performance robustness compared with other optimisation systems. By using a quantitative performance assessment such as this, a measured variety of the 'typical performance' can be demonstrated with confidence, (Shaw and Fleming, 1997a).

The application of this technique as a visual comparison method is shown in the experiments below. In section 6.4.3, new work explores the development of this method to allow statistical comparison tests of MOGA methods to be conducted.

## ***6.3 Comparison of three MOGA methods***

### **6.3.1 Introduction to Experiments**

Some of these methods are now put into practice, in the comparison of three implementations of the MOGA discussed in Chapter 5. These are the Pareto-MOGA, (again, without and with sharing and mating restrictions, providing methods MO1 and MO2 respectively), the weighted sum (WS) and the parallel population (PP) method. The problem explored used extended information from the previous problems shown in Chapter 3, as an additional nine products and their respective orders, earlier and additional delivery times, and varying changeover times were included. New data from the factory, together with changes to the representation of preferences used in the schedule builder, as described in Chapter 4, added to the complexity of the problem.

Each MOGA method was run 30 times. A reasonably high number of repeat runs has been suggested in order to ensure the statistical performance assessment is accurate. In other respects, each GA was implemented as detailed in Chapter 5 to allow multi-objective optimisation.

The main difference in the treatment from the single-objective GA implementations in terms of data was that non-dominated solutions were stored in each generation for all runs, to allow the implementation of the MOGA techniques. The storage of these solutions was in the vector form:

$$\text{solution} = [\text{cost 1}, \text{cost2}, \text{cost3}, \text{generation in which solution was found}]$$

### **6.3.2 Quality of solutions in terms of optimisation**

This section examines techniques that compare how well the methods have succeeded in actually finding an optimum solution, or sets of optimum solutions in the Pareto sense. These are answers that might be of interest to an experimental user, wanting to find a minimal solution to a problem. This assumes such a user is given time to perform many runs in order to counter the stochastic nature of the GA, in which an optimal solution may not be guaranteed from any single run. The

term 'optimal' as used throughout the section refers to relative rather than absolute optimality, as the actual optimal solution to the real-life problem is simply not known.

Minimum and median values of the costs found in all non-dominated solutions found by each method is given in Table 6.1

| Method     | Cost 1         |               | Cost 2         |               | Cost 3         |               |
|------------|----------------|---------------|----------------|---------------|----------------|---------------|
|            | <i>Minimum</i> | <i>Mean</i>   | <i>Minimum</i> | <i>Mean</i>   | <i>Minimum</i> | <i>Mean</i>   |
| <b>MO1</b> | <u>0.2000</u>  | 0.4340        | 0              | 0.0897        | <u>0.0151</u>  | 0.0504        |
| <b>WS</b>  | 0.2444         | 0.4307        | 0              | <u>0.0670</u> | 0.0185         | <u>0.0452</u> |
| <b>PP</b>  | 0.2889         | 0.4448        | 0              | 0.0958        | 0.0159         | 0.0538        |
| <b>MO2</b> | 0.2222         | <u>0.4303</u> | 0              | 0.0927        | 0.0162         | 0.0512        |

**Table 6.1 - Minimum and means of minimum values found for each cost by method. (The underlined value indicates the minimum in each category.)**

MO1 finds the minimum of cost 1, and MO2 has the lowest mean performance. All methods find the minimum value of cost 2 (zero), but the WS finds a slightly lower mean. For cost 3, MO1 finds the minimal value of the cost, but the mean values found by the WS are less than the other methods. PP is particularly poor at finding any low cost solutions - the other three methods all perform better throughout.

This table appears to suggest that the MO1 method is best for finding minimal solutions, whilst the WS can generally find low solutions, as indicated by the lower mean values, yet this conclusion is somewhat misleading as shall be seen in 6.3.4, when a more detailed analysis indicates otherwise. However, it is difficult to draw any firm conclusions from such results to allow us to conclude which is the best method for use in an industrial scheduling problem.

The **Niemeyer-Shiroma measure** (equation 6.1) is designed to measure multi-objective optimisation performance in terms of the methods' abilities to minimise all costs simultaneously. This was calculated on the data and found as shown in Table 6.2

| Method     | Value of Measure |
|------------|------------------|
| <b>MO1</b> | 1.0000           |
| <b>WS</b>  | 0.8615           |
| <b>PP</b>  | 0.7821           |
| <b>MO2</b> | 0.9419           |

**Table 6.2 - Values for Niemeyer-Shiroma measure**

MO1 receives the best score of 1, and MO2 performs better than WS for this measure. As is perhaps already becoming apparent, the PP is performing poorly compared with the other methods,

given its low scores. For this measure, the Pareto-MOGAs score better than the WS. This is perhaps as expected, given the design of the measure to compare the performance of the algorithm at optimising all costs simultaneously, a property for which the MOGAs were designed.

### 6.3.3 Distribution of solutions across the trade off surface

Although the minimum solutions can be examined, one of the themes of this work has been that, given the changing priorities and requirements of a factory, there is unlikely to be one ultimate solution to the scheduling problem. Indeed, if one is found, given the dynamic environment in a factory, it may not be valid for very long. A motivation for implementing the MOGA methods was to allow a thorough exploration of the available solutions across the trade-off surface created by the interaction between the conflicting costs. Therefore, the ability of the MOGA methods to cover the available trade-offs should be considered.

#### 3-D plots

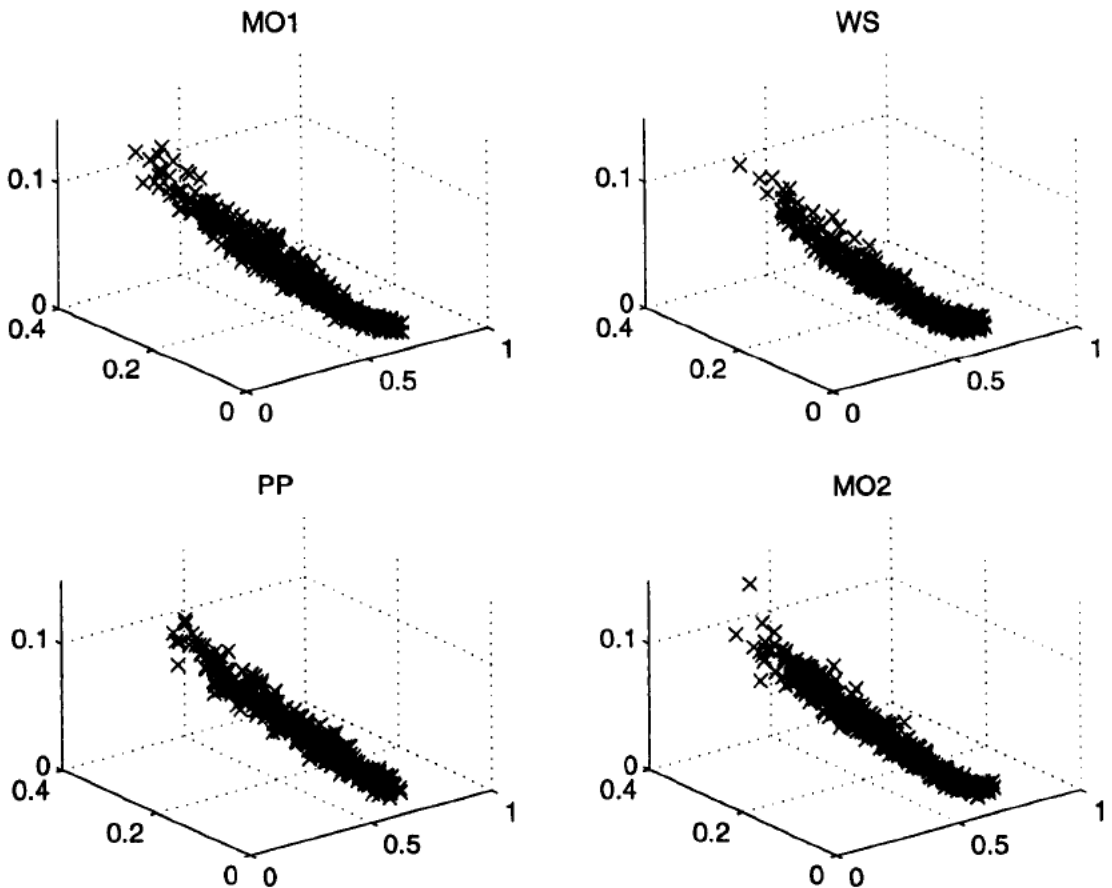


Figure 6.3 - trade-off plots in 3D for all methods

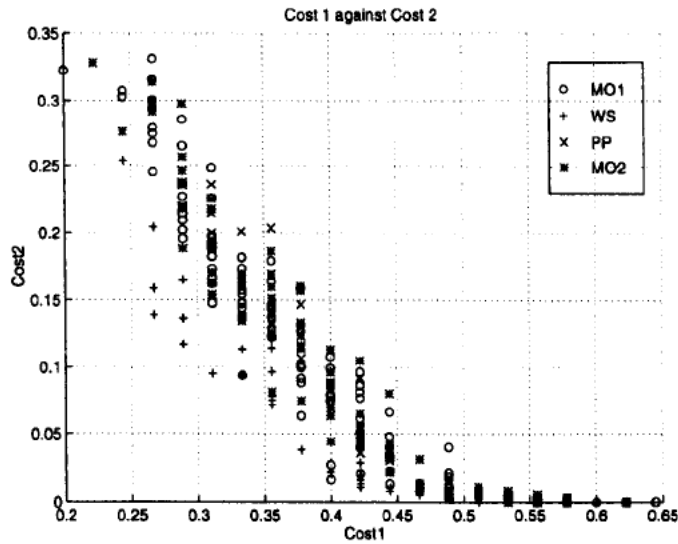
Figure 6.3 shows the plots of the non-dominated solutions found for each method plotted across the trade-off surface of costs, as discussed in 6.3.2. All methods seem equally capable of finding many solutions at the lower end of the trade-off surface. Despite the initial impression of little difference between these plots, closer examination shows that the MOGAs appear to cover a slightly wider

area of the trade-off surface. The MO1 in particular finds many more solutions at the upper end of the surface. A further point of interest is the shape of the areas covered by these solutions. The PP solutions appear to form a linear block whereas the others appear to curve to fit the inherent trade-off. This lessened ability to fit the curve of the trade-off surface could be due to the 'averaging' effect of the individuals' fitnesses across the subpopulations, as already remarked in Chapter 5 by several sources, combined with the somewhat inferior search capabilities of this method. The WS also effectively averages solutions but seems more capable of reaching extremes of the search surface than the PP.

Finally, it is interesting that again, in terms of the spread across the trade-off surface, the MO1 is performing somewhat better than the MO2. This is somewhat surprising given that the MO2 has enhancements to try to ensure it covers the trade-off surface more effectively than the MO1.

**Two-dimensional plots**

These effects can perhaps be seen more clearly in the 'flattened' 2-D plots below - Figure 6.4, Figure 6.5, and Figure 6.6. This allows comparison of performance in only two of the objectives simultaneously, and loses any information about the performance of a solution in the third dimension.



**Figure 6.4 - Two-dimensional trade-off plot for cost 1 against cost 2**

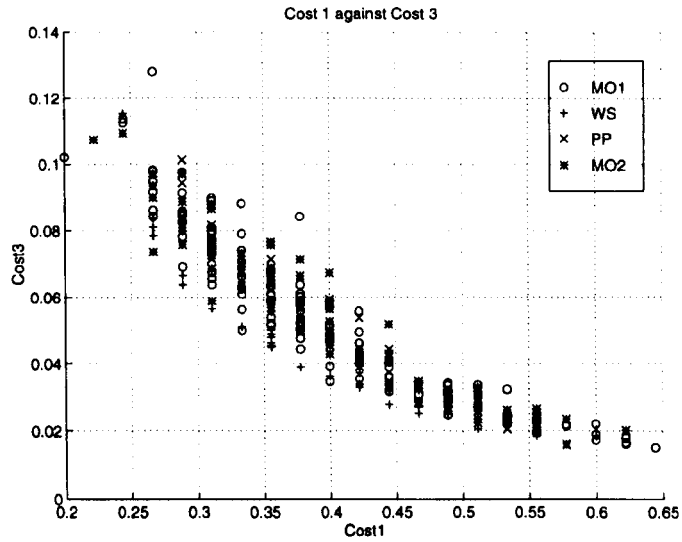


Figure 6.5 - Two-dimensional trade-off plot for cost 1 against cost 3

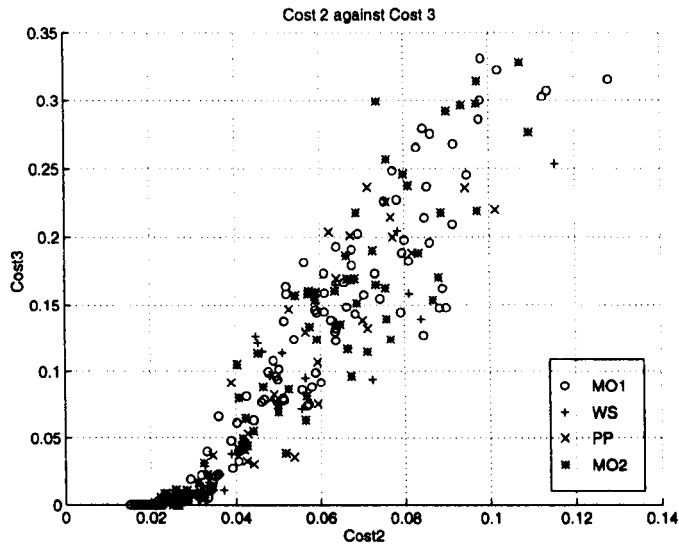


Figure 6.6 - Two-dimensional trade-off plot for cost 2 against cost 3

It can be seen that the Pareto-based methods are reaching the extreme solutions better than the WS and again that the MO1 is doing so slightly better than the MO2. The modifications to MO2 may not make any difference to its ability to find extreme solutions; the mating- restriction and fitness sharing distribute solutions between known extreme individuals across the trade-off surface, rather than pushing points to further extremes. In addition, the poor performance of the PP can be seen in all plots.

WS finds good 'central' solutions in Figure 6.4, although it does not have such an advantage in Figure 6.5 or Figure 6.6. It is to be expected that the WS is capable of finding such solutions, these being close to the single solution for which it is formulated to search. As already explained in

previous chapters, this implementation provided a very simple form of weighting, by simply normalising each of the costs. Improved choices of weights may allow the user to target particular solutions more effectively by directing the search in a different way.

The loss of relevant information regarding the 3D surface is illustrated in Figure 6.6. Without information on the values of cost 1 of the solutions illustrated, the solution closest to the origin of the axes would be chosen as the overall minimum solution for the problem. However, it is the case that all solutions in this plot must be preserved as equally non-dominated for the three-objective problem, and in fact, this point is no more an overall solution to the problem than any other non-dominated point, when its value for Cost 1 is included.

This concludes the analysis of these results using standard techniques. However, it is vital to examine these results in the context of the problem, and this requirement may heavily change the emphasis of what is meant by the 'best method'. The inclusion of robustness and the users' requirements for a practical system are shown in the next section.

#### **6.3.4 Robustness and Use of Statistical Performance Assessment (SPA)**

These relative performances can be examined in more detail by using the *statistical performance assessment* ('SPA') to provide a quantitative method of demonstrating 'typical performance' of genetic algorithms (Fonseca and Fleming, 1996). This method allows us to compare the relative performances of the schemes and to provide statistical demonstrations of their best, median and 'worst-case' performances, by plotting the non-dominated points which form *empirical attainment functions*, (EAFs). This is particularly appropriate here given the context of needing a fast, reliable and reasonably accurate solution to the problem, rather than a perfect but unreliable or inflexible one. One of the major problems with transferring genetic algorithm techniques into industry is being able to demonstrate some guarantee of performance robustness compared with other optimisation systems. This quantitative performance assessment provides useful information for anyone wanting to know just how well the method might work as a one-off run, typically, (the 'worst case' performance) or under repeated experimental trials (the 'best case' performance).

- Again, the shortfall in this method must be noted. As a visual method, it works very well for a two-dimensional / two-objective problem but may lose information on the three-objective problem. Figure 6.7- Figure 6.15 therefore plot the statistical performance assessment in two dimensions for this problem, whilst Figure 6.16- Figure 6.18 show the same data for all three objectives.



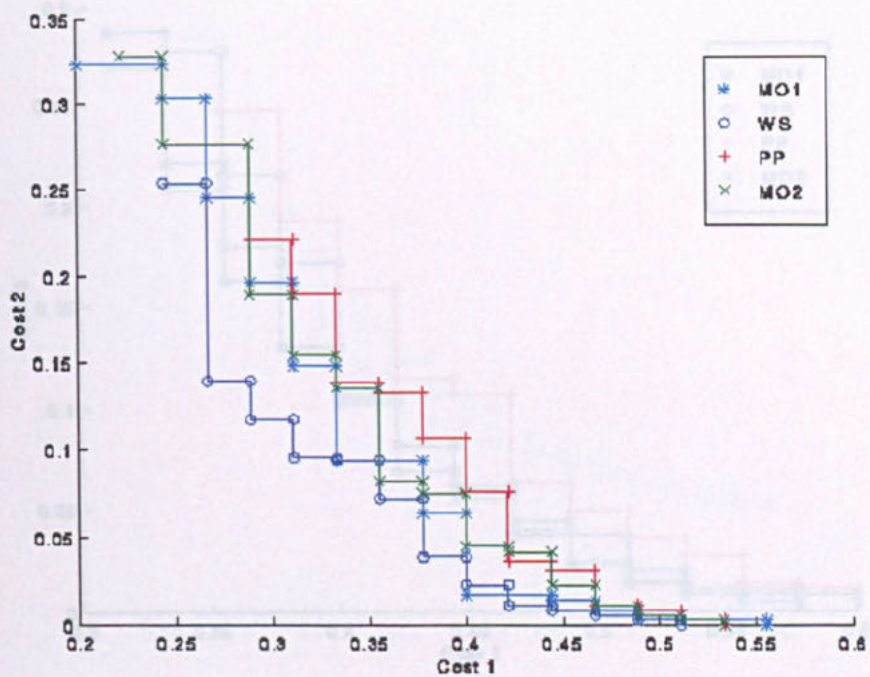


Figure 6.7 - SPA for Cost 1 against Cost 2 - best

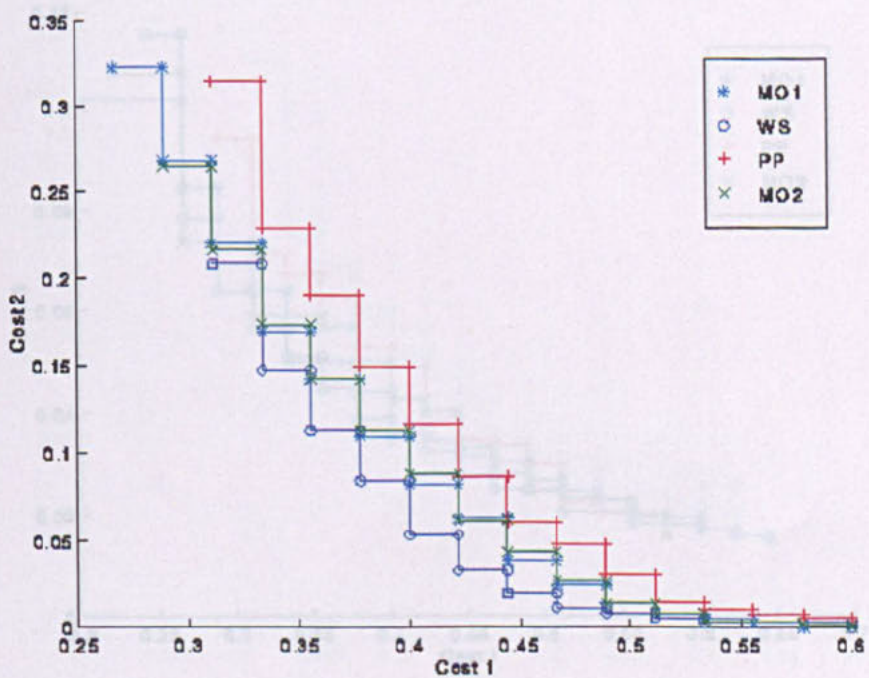


Figure 6.8 - SPA for Cost 1 against Cost 2 - median

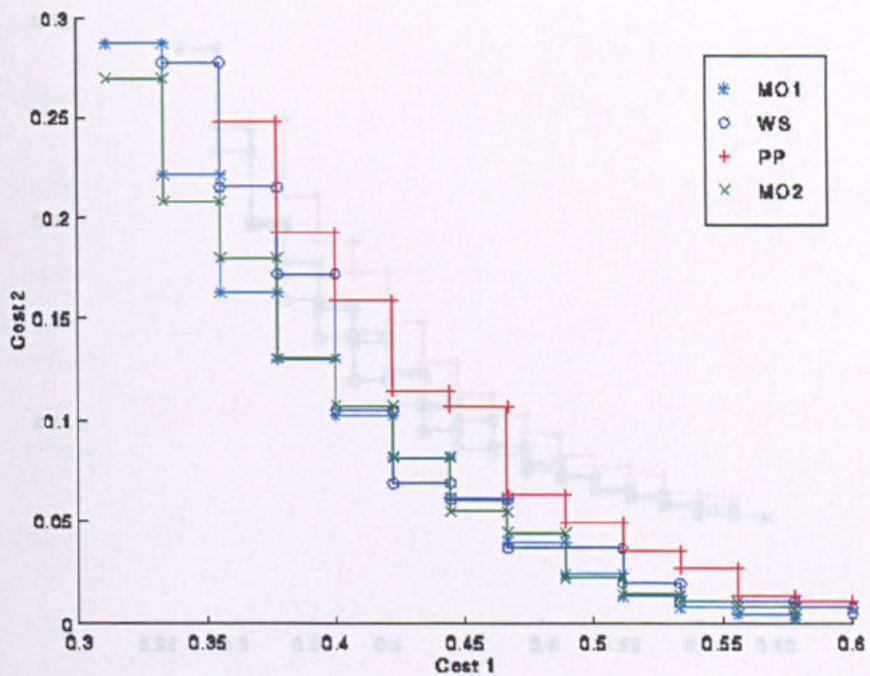


Figure 6.9 - SPA for Cost 1 against Cost 2 - worst

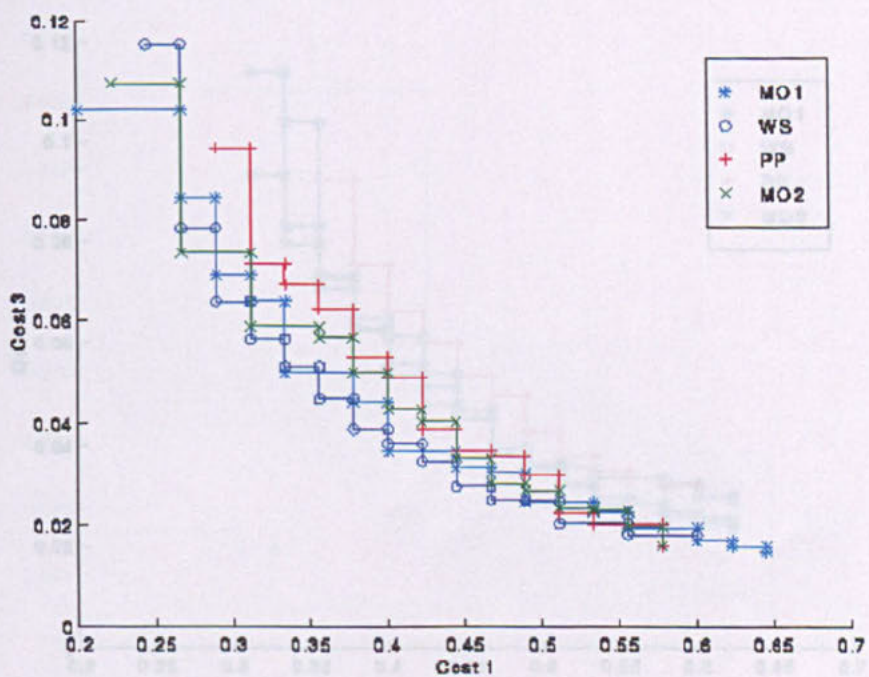


Figure 6.10 - SPA for Costs 1 against Cost 3 - best



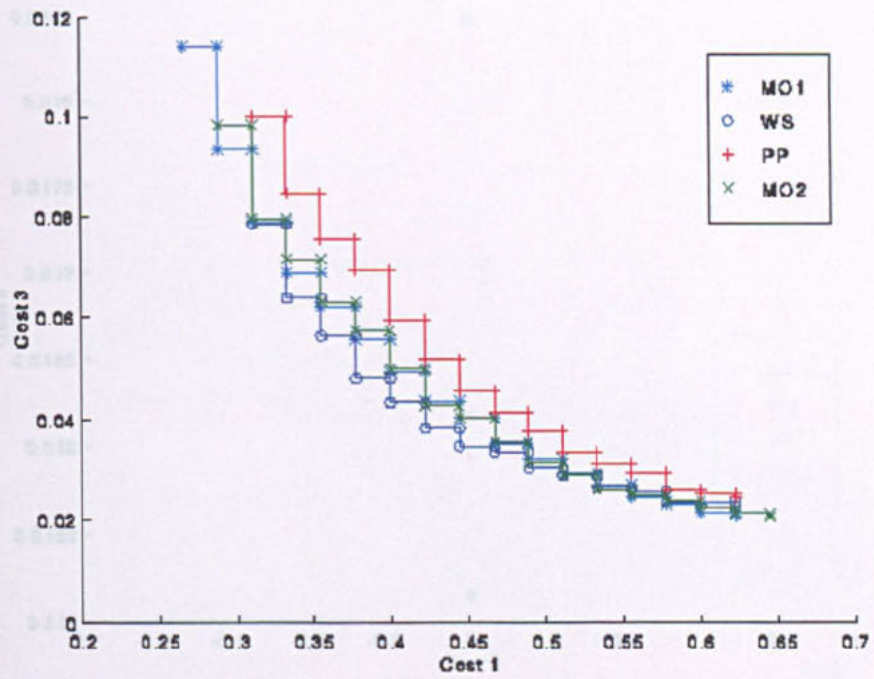


Figure 6.11 - SPA for Cost 1 against Cost 3- median

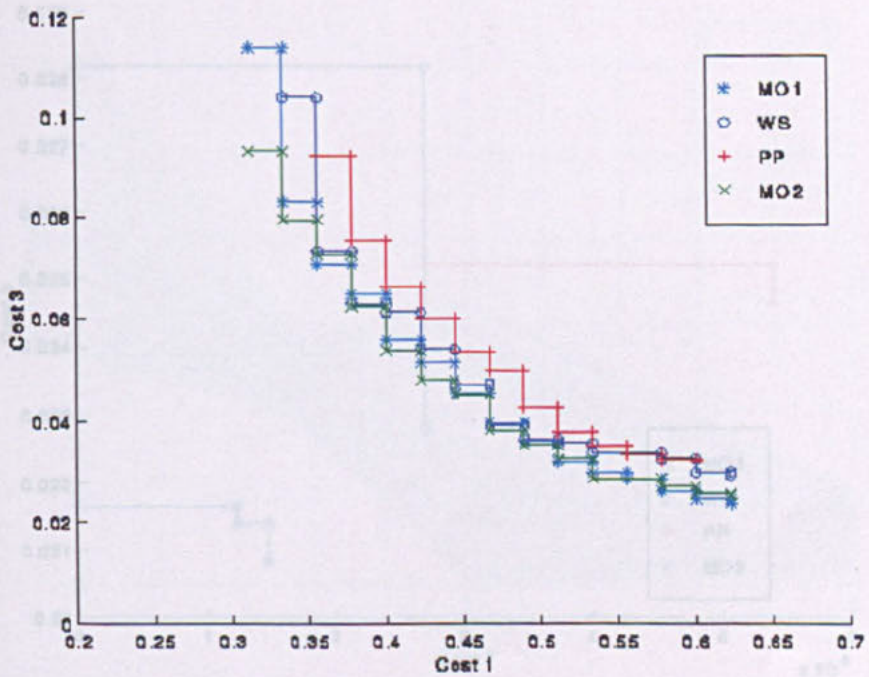


Figure 6.12 - SPA for Cost 1 against Cost 3 - worst

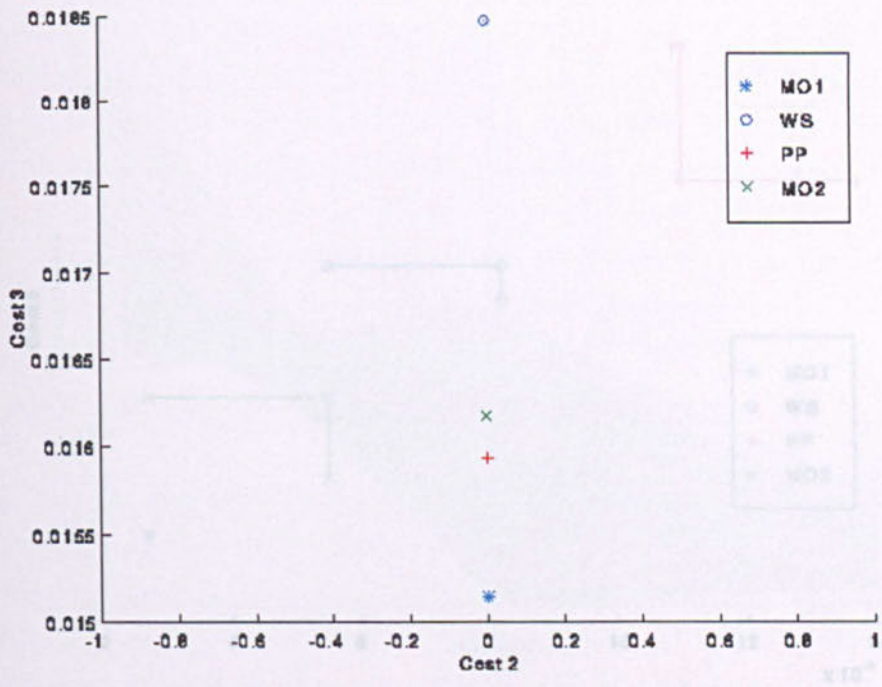


Figure 6.13 - SPA for Cost 2 against Cost 3 - best

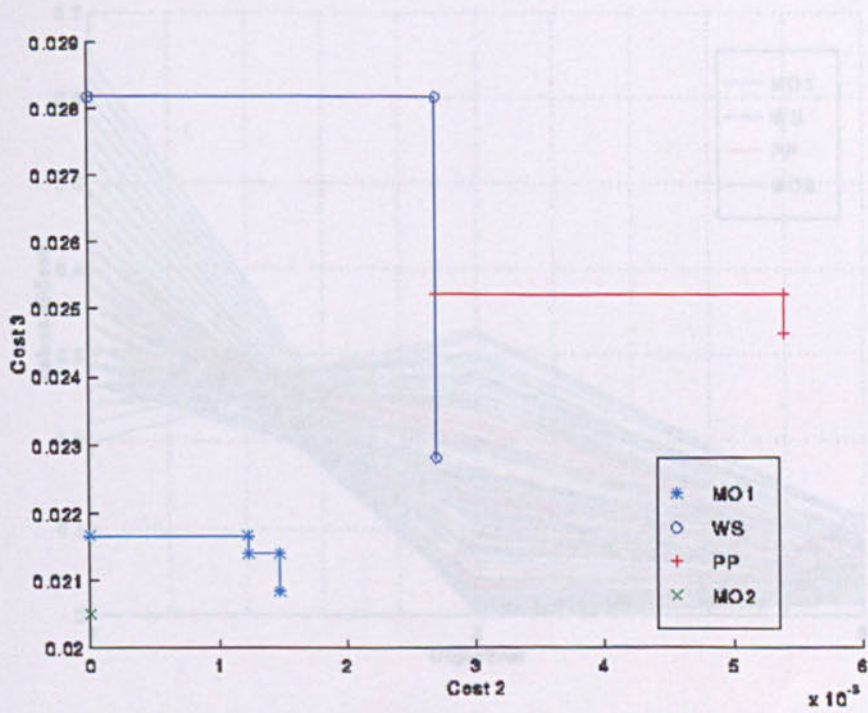


Figure 6.14 - SPA for Cost 2 against Cost 3 - median



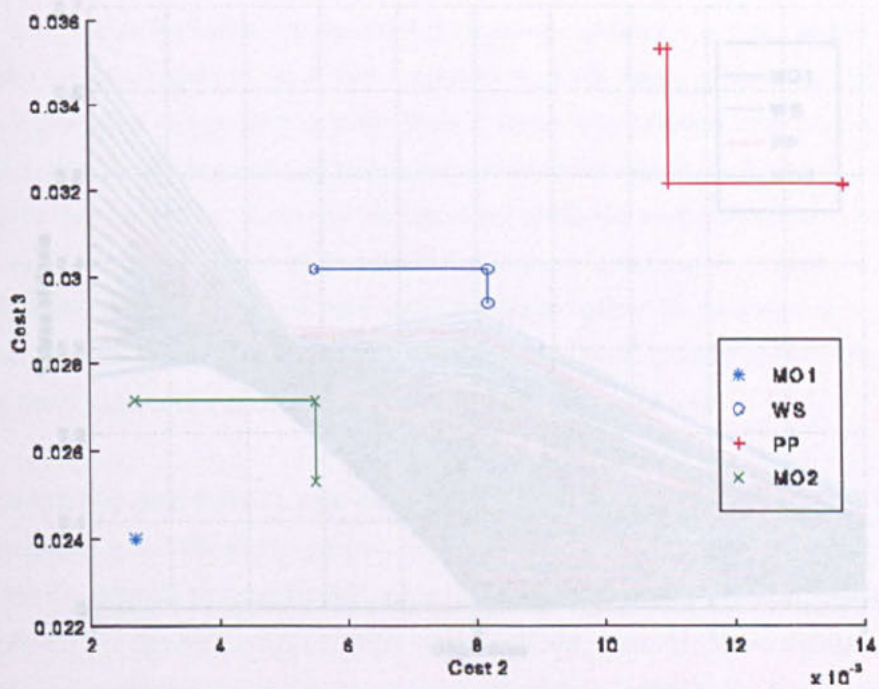


Figure 6.15 - SPA for Cost 2 against Cost 3 - worst

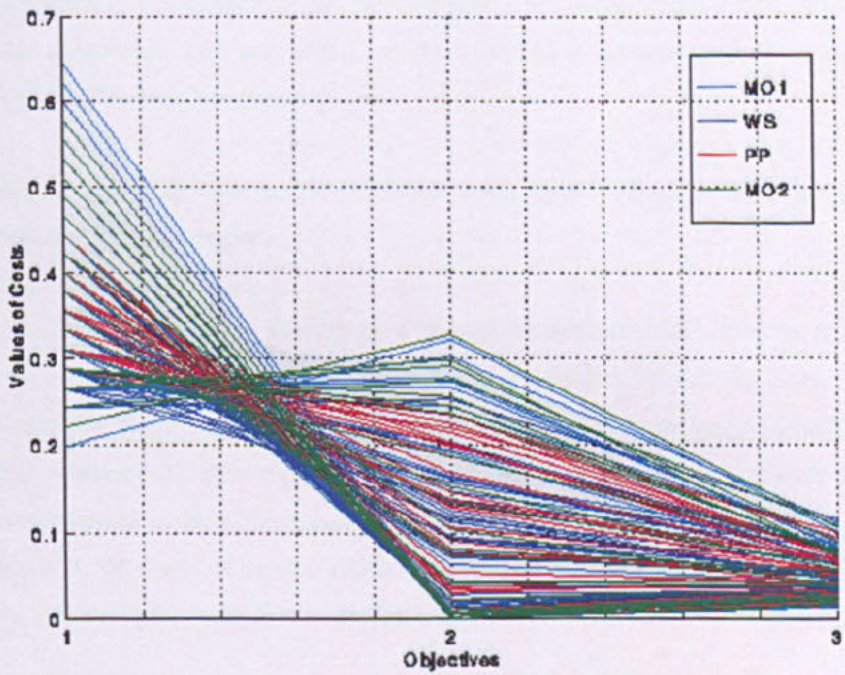


Figure 6.16 -- Best Attainment Surface with 3 objectives



### 6.3.5 Discussion of SPA results

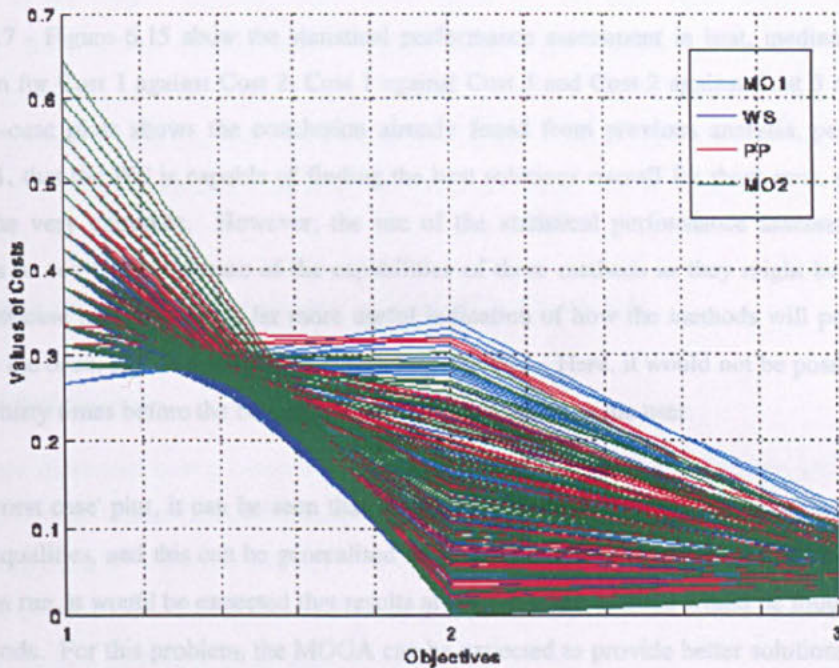


Figure 6.17 - Median Attainment Surface with 3 objectives

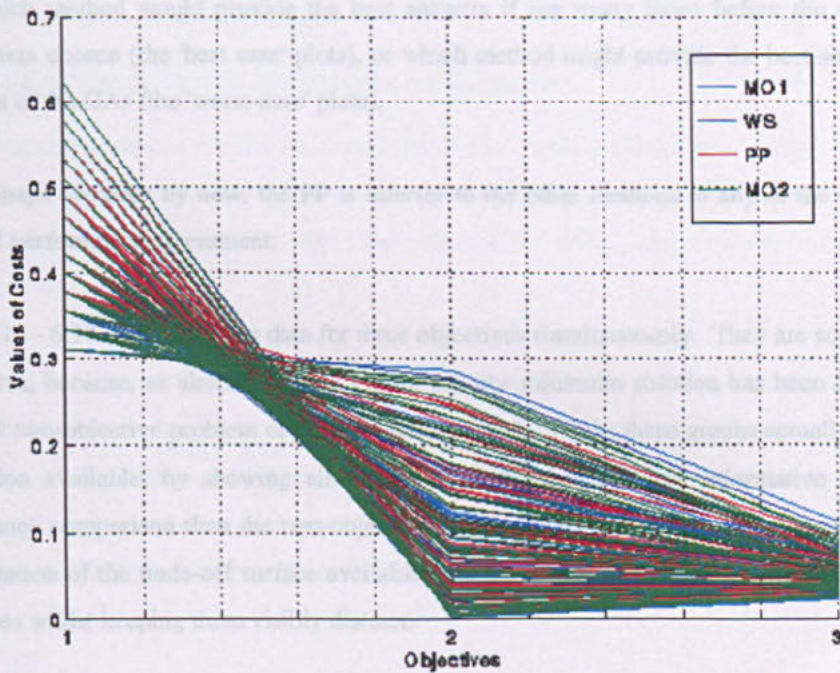


Figure 6.18 - Worst Attainment Surface with three objectives

### 6.3.5 Discussion of SPA results

Figure 6.7 - Figure 6.15 show the statistical performance assessment in best, median and worst-case form for Cost 1 against Cost 2, Cost 1 against Cost 3 and Cost 2 against Cost 3 respectively. The best-case plots shows the conclusion already found from previous analysis, particularly in Table 6.1, that the WS is capable of finding the best solutions overall for these runs, apart from a few at the very extremes. However, the use of the statistical performance assessment method allows us to see a fuller picture of the capabilities of these methods as they might be in practice. The worst-case plot provides a far more useful indication of how the methods will perform when used just the once, say, in a shop-floor scheduling situation. Here, it would not be possible to run a method thirty times before the best answer is finally presented to the user.

In the 'worst case' plot, it can be seen that the MOGA methods actually overtake the WS in their solution qualities, and this can be generalised to indicate the relative robustness of the method. In any given run, it would be expected that results at least as good as these would be found by each of the methods. For this problem, the MOGA can be expected to provide better solutions in terms of minimisation in *any* given run; given a set of repeated runs, and allowing the best solutions over all to be pooled the WS would be expected to find a better solution. The use of the statistical performance assessment, therefore, is to provide a comparison that also supplies an indication of the robustness of the result. This allows an informed choice to be made, whether the user wants to know which method would provide the best answers if run many times before the overall best solution was chosen (the 'best case' plots), or which method might provide the best solutions in a single run of the GAs (the 'worst-case' plots).

As is perhaps apparent by now, the PP is inferior to the other methods in any of the cases of the statistical performance assessment.

Figure 6.16 - 6.18 show the same data for three objectives simultaneously. They are somewhat less informative, because, as already seen in Figure 6.6, the minimum solution has been found in the 'flattened' two-objective problem of Cost 1 against Cost 3. Whilst these graphs actually include all information available, by showing all three objectives, they are less informative in terms of performance comparison than the two-objective graphs. This is due partly to their less intuitive representation of the trade-off surface available, and partly due to the practicalities of printing so many lines whilst keeping them visibly distinct.

However, even given this limited representation, several indications of performance can be seen. The lines represented can be interpreted as follows. Those on the outsides of the shaded areas at each of the values 1, 2 and 3 for the 'costs' axis are solutions containing extreme values of one or more of the costs. Those lying in the centre of the shaded area are less extreme solutions. As must

already be evident by now, the parallel populations method, shown in red on these diagrams, is confined to the centre, providing only central solutions, and even then, fewer than the other methods. Both MOGA methods show good coverage of the extreme values in all three graphs, with the green and pale blue lines. One interesting property of the WS that can be seen clearly is its high quality of solutions that cover trade-offs between costs 1 and 2 in the best and median surfaces. However, as already seen in the two-objective plots, this advantage is lost in the worst-case plot.

## **6.4 Statistical Comparisons of MOGA Methods**

### **6.4.1 Introduction**

It has been illustrated how a comparison of MOGA methods may be undertaken effectively from graphical means. Yet so far, conclusions have only been drawn by basic comparisons of figures, or visual inspection of trade-off surfaces. A quantifiable method of comparing performances by statistical tests is highly desirable to enable the user to know with a degree of certainty whether one particular MOGA method is better than the other.

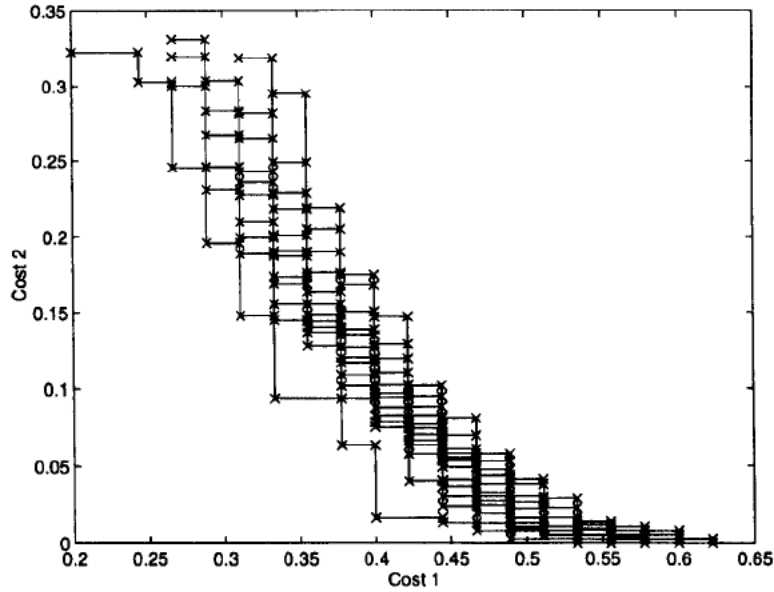
This section aims to introduce some experimental work to develop such a test. One long-term aim of doing so should be to provide a generic statistical test for comparing the performance of MOGAs for any given problem. The method is outlined in section 6.4.2, and full details of the calculations are provided in Appendix 1.

### **6.4.2 Use of the SPA for statistical comparisons**

The methods presented in the statistical performance assessment (SPA) have already been seen to be effective tools for allowing a clear visual comparison of MOGA methods across the trade-off surfaces. However, the SPA also offers the potential for being developed into a full statistical comparison method.

The tests are based on the data provided by the SPA method in forming the attainment surfaces, or *empirical attainment functions*, (EAFs) as seen in 6.2.2. This data represents the successive capabilities of the methods at finding points on the non-dominated front. Thus, attainment surface 1 provides the best points found by such a method - i.e. those found in only one run of all the trials; attainment surface 30 provides the points found by every run of the method.





**Figure 6.19 - Example of EAFs**

Initially it seems obvious that a test might be developed comparing areas or volume underneath the attainment lines. Such a test is currently being explored as a method of providing a simple comparison test. However, a test based on volumes would be limited by assumptions of the scale and cardinality of the objectives. These assumptions can be avoided by instead comparing the relative positioning of the non-dominated points found by two or more methods, allowing inferences to be made as to the similarities between their overall optimisation performance.

In the simple case of comparison of methods for multi-objective optimisation, two GAs are compared, labelled MOGA1 and MOGA2. There are further difficulties in comparing more than two methods simultaneously; these are discussed later in 6.4.4. It would be expected that similarly performing MOGAs would provide similar EAFs. For example, the points forming the first attainment surface should be roughly similar in position, as should those in the second, third and subsequent surfaces. Figure 6.20 illustrates such a case.

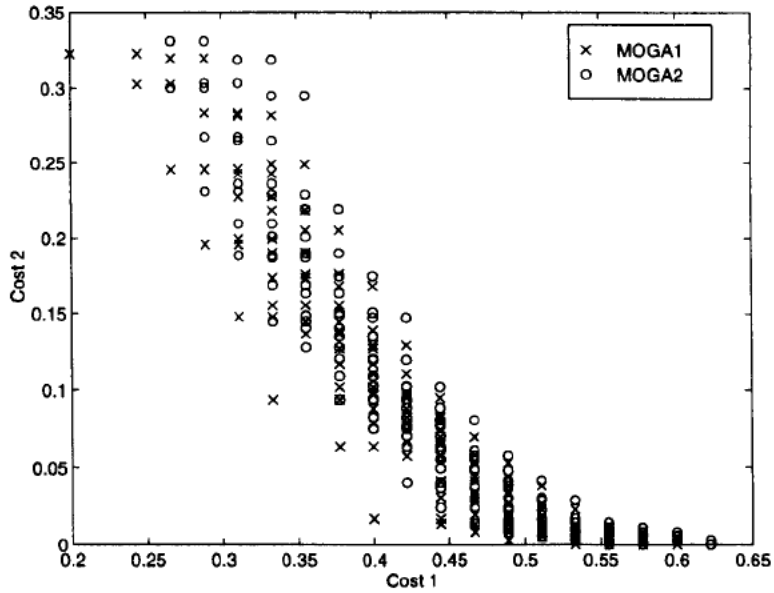


Figure 6.20 - Example of MOGAs with similar EAFs

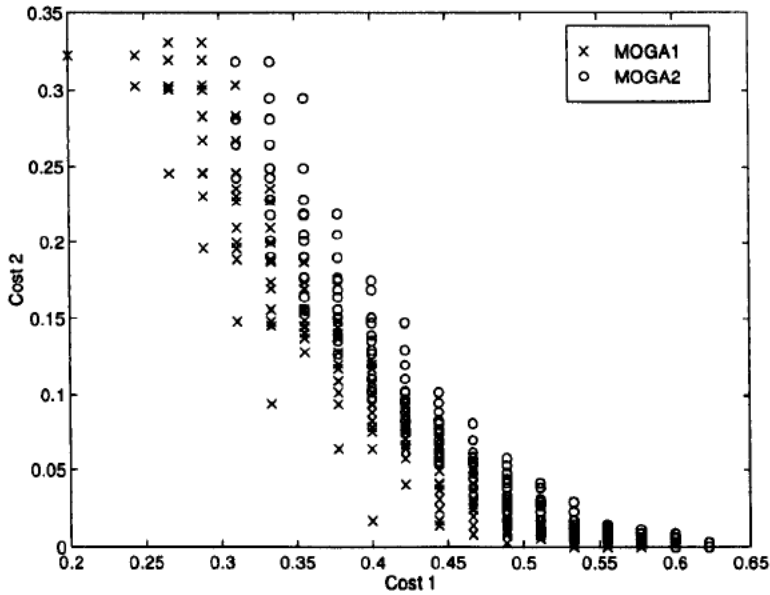


Figure 6.21 - Example of MOGAs with dissimilar EAFs

Practically, this difference can be measured by sampling the relative orders in which the points of the EAFs are arranged. A random bisection of the EAFs in Figure 6.20 might provide a sampled arrangement of methods as (MOGA1, MOGA2, MOGA1, MOGA2, MOGA1), suggesting that these two GA methods are reasonably similar. Figure 6.21 shows a different arrangement. A random sampled bisection of these lines would provide the order (MOGA1, MOGA1, MOGA1, MOGA2, MOGA2); suggesting that there is some differences between these two methods.

Thus, an appropriate test statistic would be the maximum difference found between the sampled positions of the two methods upon such arbitrary bisections. If this difference were small, the EAFs of the two GAs would be expected to have little difference between them.

An suitable test for this situation is a permutation test (Good, 1994). The details for the development of this test, and the calculations required to apply it to this problem are given in Appendix 1.

### 6.4.3 Statistical Test

The following hypotheses are chosen for testing the differences between the MOGAs:

***H<sub>0</sub>: There is no difference between the performances of MOGA1 and MOGA2***

***H<sub>1</sub>: There is a difference between the performances of the methods***

This particular hypothesis is two-sided in that it will only be possible to show that there is a difference between the methods. A one-sided test would show which of the methods had a better performance. Clearly, this would be one of the immediate requirements in future work. For the moment, having established that there is a difference between methods, it is still necessary to analyse it informally, based on the graphical results.

To implement a test of this hypothesis, a permutation test is used. Permutation tests have the advantage of making very few assumptions about the underlying distribution. The essential assumption is that the data in the samples can be exchanged between categories if  $H_0$  were true. (Good, 1994). This assumption is valid for these MOGA comparisons, as the non-dominated results being examined could be equally produced by any of the methods under  $H_0$ .

### 6.4.4 Results of test implementations

Initial attempts at implementing these tests on data provided by the GAs for multi-objective optimisation, which were compared above, have provided the following results. These results are by no means exhaustive in their description of the comparisons made. Indeed, they may provide more questions than they answer. It is hoped that this implementation simply provides an initial example and motivation for further work in this area.

#### ***First set of tests - pairwise comparisons for all methods***

In these tests, the hypotheses:

***H<sub>0</sub> - MOGA<sub>m</sub> and MOGA<sub>n</sub> show no difference in performance***

***H<sub>1</sub> - MOGA<sub>m</sub> and MOGA<sub>n</sub> show difference(s) in performance***

***(m, n = 1,2,3,4)***

are tested for all possible pairs of the four MOGA methods; MOGA1 being the MO1, MOGA2 being the WS, MOGA3 being the PP, and MOGA4 being MO2. The p-values provided by such tests, and as calculated in Appendix A ,are provided in Table 6.3 below.

| <b>Comparison</b>  | <b>p-value</b> | <b>Conclusion</b> |
|--------------------|----------------|-------------------|
| <b>MO1 vs. WS</b>  | 0.000          | Reject $H_0$      |
| <b>MO1 vs. PP</b>  | 0.000          | Reject $H_0$      |
| <b>MO1 vs. MO2</b> | 0.200          | Accept $H_0$      |
| <b>WS vs. PP</b>   | 0.000          | Reject $H_0$      |
| <b>WS vs. MO2</b>  | 0.000          | Reject $H_0$      |
| <b>PP vs. MO2</b>  | 0.000          | Reject $H_0$      |

**Table 6.3 - p-values found for pairwise significance tests**

The initial impression of these results is that the test appears to be reasonably sensitive to be capable of finding almost all pairs of methods to be performing differently, with the exception of the two MOGA methods. As discussed above, there seemed to be little graphical evidence that there was much difference between the performance of these two versions of the Pareto-MOGAs, and the statistical test now provides further evidence that this is true for this problem. The conclusion that the other pairs of methods are all different in their performance is another satisfactory conclusion to be able to draw from this work. However, the conclusions are now deficient due to the fact that only a two-sided test has been implemented; clearly it would be extremely useful to know which method is better than the other. As seen in the visual comparisons, whilst PP was not the better method of any pair, it would be harder to draw a definite conclusion over the relative performances of MO1/MO2 and the WS.

It should be noted that these tests have been performed upon the combined data for all attainment levels; significant differences between methods at various attainment levels would indicate further flexibility for the user, as described previously, and therefore the tests are implemented on the data relating to each attainment level separately.

### ***Second set of tests - pairwise comparisons at each attainment level***

In these tests, the hypotheses:

***$H_0$  -  $GA_i$  and  $GA_j$  have same attainment surface performance for attainment level  $L$***

***$H_1$  -  $GA_i$  and  $GA_j$  do not have the same attainment surface performance for attainment level  $L$***

are tested for all possible pairs of the four MOGA methods, as in the previous experiments. It is difficult to set values for significant rejection of the hypotheses when more than one test is involved in the conclusions. This work instead follows Efron and Tibshirani, 1993, in presenting the p-values found as evidence for or against the hypotheses, without any acceptance or rejection test being applied. The adjustment of p-values for multiple tests is discussed by Lehmann, (1986)

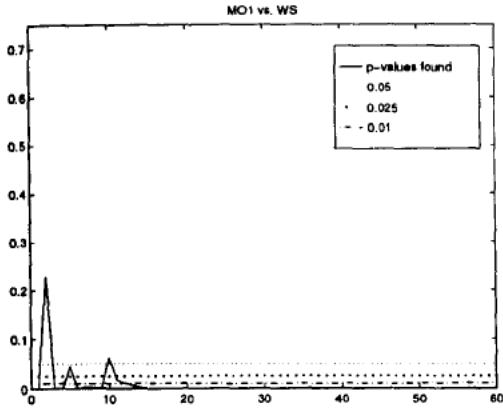
and Hochberg, (1987), and would be needed if more exacting conclusions were needed to be drawn.

Multiple tests must be even stronger to allow a conclusion to be still valid. Conover, 1971, explains: 'two or more samples may be further analysed using (such a) test until the differences between populations have been satisfactorily detected. However, the level of significance in all but the first test is distorted and almost completely devoid of meaning, except possibly to aid one in ordering the differences from smallest to largest.' Simply, it is not useful to perform multiple tests in order to isolate one particular result without some additional adjustments to the significance of the results.

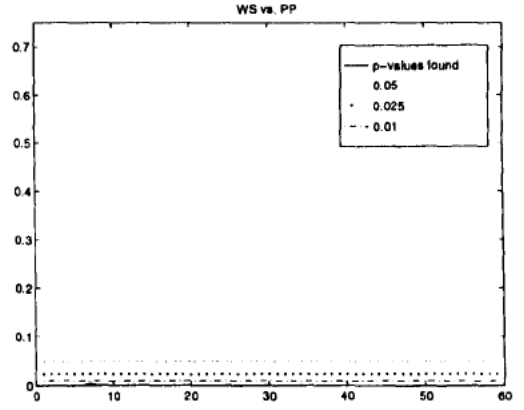
The p-values provided by such tests for the MOGAs at each comparison level are illustrated in Figure 6.22 - Figure 6.27. The full values for each test are provided in the appendix A, in Table A-1. The height of the peaks reflects the probability at each attainment level that the two methods compared have similar performances; therefore the lower the peaks shown on these graphs, the greater the difference between the methods.

Again, it is important to note that each of these conclusions must be considered individually, and that they cannot be combined to provide any overall conclusions of the performances of the methods.

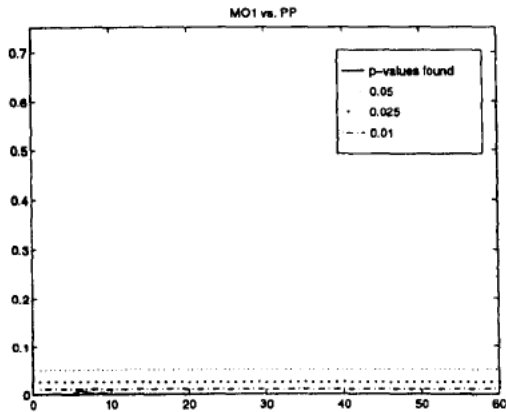
The results of these tests generally correspond to the previous test. However, it is interesting to note that whilst the tests generally show there to be significant differences between the pairs of the methods apart from the two versions of the Pareto-optimal MOGA: MO1 and MO2, this difference arises when comparing these methods with the WS method on a few of the higher attainment levels. It would require further work to show whether these few results indicate a significant trend towards such a difference, as that there is no significant difference between the MOGA methods and WS on lower (i.e., first to tenth) attainment levels. As discussed above, given a large number of statistical tests from which to draw one conclusion, there is a far higher probability of drawing the wrong conclusion, and a suitable adjustment would have to be made to take this into account.



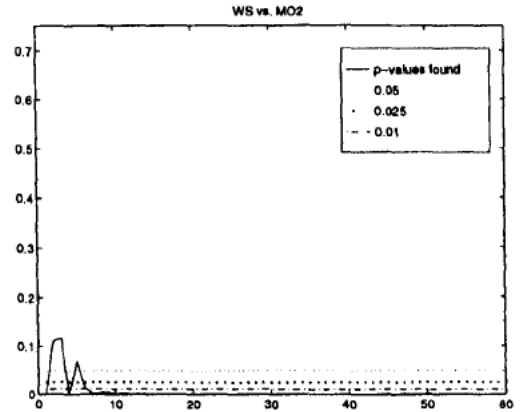
**Figure 6.22 - P-levels of tests on each attainment level for MO1 vs. WS**



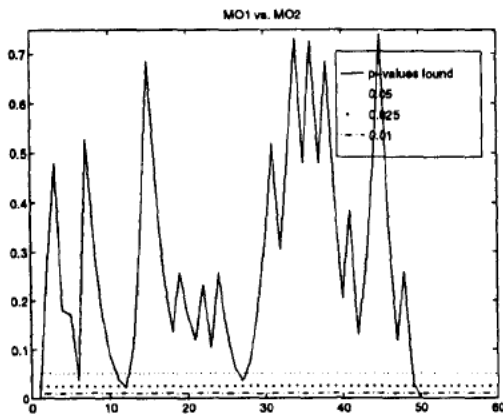
**Figure 6.25 - P-levels of tests on each attainment level for WS vs. PP**



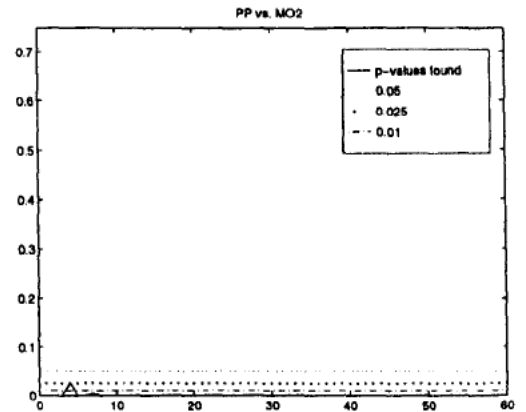
**Figure 6.23 - P-levels of tests on each attainment level for MO1 vs. PP**



**Figure 6.26 - P-levels of tests on each attainment level for WS vs. MO2**



**Figure 6.24 - P-levels of tests on each attainment level for MO1 vs. MO2**



**Figure 6.27 - P-levels of tests on each attainment level for PP vs. MO2**

### ***Third set of tests - 4-sample test***

Finally, an attempt is made to compare all four methods simultaneously, rather than comparing each pair of methods separately as shown above.

In this test, the hypothesis

***H<sub>0</sub>: There is no difference between the performances of MOGA1, MOGA2, MOGA3 and MOGA4***

***H<sub>1</sub>: There is a difference***

is tested, by all four methods being compared simultaneously.

Having conducted this test, using the TS indicated in Equation A4, this p-value of this test was found to be p-value = 0.000, indicating that it is practically impossible that all methods can be said to be performing the same on this problem.

This test in itself is not particularly difficult to apply, and indeed its conclusion, that there is some difference between the methods overall, is not particularly useful for providing any more detailed conclusions from this work. But having implemented this test, it is then possible to perform a set of 'step-down' tests to identify further details concerning which methods are causing a difference. Having established that this first test finds in favour of H<sub>1</sub>, subsequent tests are conducted on the sets of methods using a series of eliminations of each method until differences are identified. This is discussed in Lehmann, 1986 and Hochberg, 1987. Unfortunately, such tests involve lengthy computations and time constraints have prevented these from being implemented on this example. This would also be an area for further work.

The resulting tests would allow an answer to the question that is still particularly outstanding from this work:

***H<sub>0</sub>: There is no difference between the performances of MOGA1, MOGA2, MOGA3 and MOGA4***

***H<sub>1</sub>: There is a difference, ordering the methods from best to worst as follows: {MOGA1, MOGA2, MOGA3, MOGA4}***

with the details of such differences supplemented by the pair-wise tests conducted in above sections.

### **6.4.5 Discussion of statistical comparison**

This section has presented the initial experimental work for developing a generic, statistical test, which can provide conclusions concerning the comparative performances of MOGAs. This test can be extended to compare the performance of any stochastic multi-objective optimisation method which provides sets of non-dominated points over a number of runs. Thus, it should be possible to compare not only various implementations of MOGA against each other, but also MOGAs against entirely non-GA based methods.

It is apparent that there is a great deal of further work that might be conducted in this area. Immediate requirements of such tests for future implementations would be the use of a one-sided test, to allow conclusions to be drawn as to whether one method is better than another, and use of the step-down tests for multiple comparisons, also to provide such a conclusion. Further work is also needed on the adjustment of the significance of such results in the multiple cases to ensure the accuracy of such a conclusion.

However, some groundwork has been provided for pursuing such aims, and the implementations of MOGA provided in Chapter 5 have been compared statistically to a certain extent. It has been demonstrated that we can conclude that there is a significant difference in the performance of the methods. However, further work is needed before any conclusions concerning the details of such a difference may be drawn. For this reason, these tests have only limited utility at this stage. The remaining work on this method should provide an extremely useful tool for comparing and improving future MOGA implementations.

## **6.5 Discussion and Conclusions**

Clearly the question that we would best like to answer at this stage is; 'is any method better overall for optimising scheduling problems?' The above work has presented extensive graphical representations of the relative performances of such methods, and a more exact quantifying method has been introduced to attempt to find an answer to this question.

Perhaps it is easier to indicate the worst method rather than the best. The PP method generally provides the least good performance throughout, both in terms of inferior solutions, and in its coverage of the available non-dominated points. This has been previously indicated in the literature discussed in chapter 5, and might, perhaps have been expected.

It is important to define exactly what is needed from a method in the context of the problem it is required to solve, in order to say it is 'the best'. Undoubtedly, methods must provide good optimisation capabilities, yet for real-life problems such as those represented here, finding the globally optimal solution is not the only consideration. Reliability and compatibility with the users' changing problem specifications are equally important. Methods must also be developed for



measuring the performance of GAs for multi-objective optimisation, under varying circumstances. It is harder to choose the best method from the Pareto-MOGAs or WS without suitable context. As with the choice of best answer from a set of non-dominated solutions, context has to be added to the available selection criteria to define the 'best' solutions. Knowledge of the application at hand is vital to allow a decision to be made.

The original intention of this research was to develop a MOGA suitable for schedule optimisation in a real-life manufacturing environment. The key requirements for this problem were that the MOGA method should be fast, reasonably accurate, reliable and should be able to cater for the various changes in circumstance demanded in the factory.

When compared visually, the WS and MO methods (MO1 and MO2) offer reasonably similar solutions in terms of optimisation performance (section 6.3.2). MO1 provides the best coverage of the trade off surface (section 6.3.3) and therefore the best selection of solutions suitable for offering to the user in the sense of multi-objective optimisation. There is little practical difference in terms of speed; MO1 and MO2 take slightly longer than the WS yet both run within the allocated time of one hour in these tests. However, indications of speed can be as misleading as measures of optimisation performance, if we do not consider the robustness of the methods. A method may run extremely fast but is of little use if the solution is not satisfactory. Similarly a method may run be capable of finding extremely good solutions, but if it cannot be guaranteed to provide such a result when needed, such ability is of less use.

For this reason, the results of the SPA are significant in indicating the reliability of methods. The provision by MO1 and MO2 of the best 'worst case' solutions demonstrates that they are more robust than the WS for guaranteeing a reasonable solution under the conditions in which the factory might use them, run just once rather than repeated many times. This reliability may surpass any advantage the WS might have in terms of speed or absolute optimisation ability for the context of this problem. The factory cannot afford to repeat a run of the scheduler if the first solution is unsatisfactory.

The additional information provided by the statistical comparison is both helpful and limiting for the user. It has indicated that, indeed, the MOGA methods are significantly different in their performances in terms of optimisation. The only exceptions to this are the two Pareto MOGAs, MO1 and MO2; the differences in their performances are indistinguishable. Thus, it can be concluded that the use of sharing and mating restriction of MO2 adds no improvement for this example.

Yet, the result that the performances of the MOGAs are 'different' is inconsequential without additional information provided by the implementation of the one-sided test, to tell which one is

the better method. The SPA graphs do not indicate that one method is better than another outright, but differ depending on the attainment levels illustrated. With no clear visual information to aid such a conclusion, the statistical test does not add greatly to the decision-making process between methods at present. Further work could enable this test to offer a greater contribution to the MOGAs comparison; indeed this area would allow substantial comparisons of MOGA performances to be made in future.

One limitation of this comparison has been that the problem only has three objectives. It may be that given a problem with a higher number of objectives, the difference in the methods' performances would be more marked.

In summary, the MOGA scheduling method that the factory demanded had the following requirements:

- reasonable accuracy in the solutions found
- ability to provide a wide range of alternative solutions to the multi-objective scheduling problem
- reasonable speed and accuracy
- reliability / robust provision of solutions under the time constraints for finding a schedule.

Given all these factors, it can be concluded that for the optimisation application presented here, the Pareto-optimal method (MO1) is the best for meeting the users' requirements in this case. There is no significant difference between Pareto-based methods, MO1 and MO2 and therefore, the simpler implementation, without the additional mating restriction or fitness sharing, is chosen. As discussed previously, the method is not chosen solely in terms of absolute minimisation performance, but in the context of the problem presented.

Chapter 7 provides the conclusions to this work.

## **7. Discussion and Conclusions**

### ***7.1 Restatement of Aims***

This thesis has aimed to demonstrate the use of GA techniques for solving real-life production scheduling problems. The requirements of a chilled ready-meal factory have provided influences upon the work, by motivating the need for a reactive, interactive scheduling system. The factory has also allowed the demonstration of multi-objective GA techniques to a problem that is well known to be extremely demanding.

The real-life problem differs from benchmark problems in several ways. It is dynamic, rather than static, demanding that scheduling methods are designed to allow the user to change the problem information easily, during a run, and that the method can adapt to keep working with any new information. It is often uncertainly defined, whereas in a benchmark, an assumption may be made that something is so, or a fixed value provided for any necessary variable. It has been seen, particularly in Chapter 4, that it is often the case in real-life that necessary problem data simply is not known, or defined, as well as might be liked for the application of rigid techniques that demand a particular set of information before they can be used. The problem therefore also requires a method that can deal with uncertainty, 'fuzzy' information or a problem defined in 'non-standard' terms. Finally, as is so often the case in real life, there is not one simple answer to the scheduling problem that can satisfy all demands and objectives equally. Compromise in the form of trade-offs between the conflicting objectives must be made by the user for a solution to be found. The GA has been shown to be a tool that can satisfy all these demands.

### ***7.2 Summary of Thesis***

The general requirements of scheduling problems and other methods of dealing with them were presented in chapter 2, along with an introduction to genetic algorithms. Several of the many scheduling methods available were discussed. GA methods were demonstrated to be contenders in the search for a suitable scheduling method for a real-life problem. Whilst their overall ability at finding optimal solutions was perhaps less marked than some other methods, they could offer solutions to problems that could not be solved in other ways. They also allowed specific implementations to meet the users' requirements indicated their advantages. Problem-specific features must indicate the choice of method best suited to finding solutions within a particular environment.

Chapter 2 also demonstrated that within the field of GAs itself, there are many different implementations and variations upon the basic theme. As with the choice of scheduling methods

themselves, the choice of best implementation of a GA for a particular problem must be indicated by the problem requirements.

Particular problem requirements specifically influence the development of GA methods for the remainder of the thesis. A difficult real-life problem, based on data drawn from a local chilled ready meal factory was introduced in Chapter 3, provided an application that included novel situations for the GA scheduling application. The general implementation of genetic algorithm methods has been demonstrated, starting from the standard application of a single-objective genetic algorithm to this problem.

One result that should not be overlooked is the fact that the GAs can find schedules for the problem faster and more effectively than the current factory methods. This result in itself was satisfying for the initial experiments, and the implementation itself was designed to allow further developments upon the basic method. Yet, examining the additional needs of the factory provided motivation for further developments. The problem was more complex than that originally presented for solution in Chapter 3. Rather than a static, single-objective problem, it was dynamic, requiring multiple-objectives, and the incorporation of changing priorities. Subsequent chapters explored methods for meeting these requirements.

Using the chosen representation of the simple individual combined with a schedule building system to add problem information to the search for the solution, difficulties were found within the translation stage of individual to completed schedule. Chapter 4 introduced the ideas of using real-life preferences rather than fixed, heuristic rules regarding the constraints of the problem. This demonstrated that a certain amount of freedom of definition combined with the search capabilities of the GA allowed improved performance.

The schedule builder allowed easy alteration of data for a changing problem, a vital requirement from the system. It also allowed the GA to incorporate uncertain information and user preferences within that schedule builder, and to deal with the assignment of fitnesses in which the fitness to chromosome mapping was not necessarily unique. Such situations may be found in many types of real-life problem, when data may not be as clearly defined as in a theoretical problem. It was shown that the use of such data actually allowed an improved GA performance on the alternative method of using a problem-specific heuristic. However, further comparisons using other heuristics were not conducted and general conclusions about the advantages offered by this method are therefore not available. The inclusion of additional rules may actually create detrimental optimisation performance. The preference-based method was put into practice with factory data and shown to be very acceptable to the users.

Chapter 5 introduced the same ideas of allowing the user to interact with the optimisation system and to provide preferences for the objectives, by implementing multi-objective optimisation. Chapter 5 also reviewed previous work in the field of multi-objective GAs and introduced three common methods for implementing multi-objective search within a GA. The models of parallel populations, Pareto-based methods and weighted sums were discussed, and an implementation of each was used for the experimental work in Chapter 6. A survey of the current field of applying MOGAs to scheduling completed this chapter, together with some comments on these implementations in relation to this work.

Chapter 6 demonstrated the MOGA methods being applied to the scheduling problem introduced in Chapter 3. This provided a detailed comparison of their relative performances. It also discussed the implementations and demonstrated a method of statistical comparison that may allow significant conclusions to be drawn concerning the potential performances of these methods. It is hoped that such comparisons will be useful for inspiring the confidence of industry in using relatively novel techniques such as genetic algorithms. To develop MOGAs for scheduling problems, the user must be able to measure improvement by having effective comparisons for their performance. The statistical method used offered much further work.

Finally, the comparison concentrated on answering the question as to whether it was possible to choose a 'best method' of those presented. The idea of context to indicate the 'best' method was discussed. Out of the three methods, the parallel populations method was discarded as performing poorly in all comparisons. The original experimental indication, that by using a model that allowed each objective to be searched using operators best fitted to it, a good performance would result, did not match the practical results. The WS worked well as an optimisation method, but the comparisons indicated that it might be more suited to experimental problems, for which the user has the opportunity to repeat the GA runs many times and draw solutions from the pooled results of the all runs. For the application presented in Chapter 3, the simple Pareto-optimal MOGA (MO1) was felt to be the best method for meeting the manufacturers' requirements from a schedule optimisation system.

### ***7.3 Discussion of Results***

As a simple attempt to apply GA techniques to the Pennine Foods problem, this work has been successful in all its aims. Chapter 3 demonstrated a simple implementation capable of improving on previous methods for the test problem provided, and later chapters provided further developments in terms of modelling, interactivity and multi-objective optimisation. These points have all moved towards addressing issues in the development of new, realistic scheduling practice. The implementation of three methods of performing multi-objective schedule optimisation using GAs have been demonstrated. The implementation of the parallel populations method in particular

is novel, with the provision of separate operator choices for each subpopulation. These have been applied to the test data of a real-life problem, and have been shown to be simple and effective tools for performing schedule optimisation. The schedule builder work in Chapter 4 has addressed issues of representation and use of problem data within schedule optimisation systems, and offers an alternative to the approach of using heuristic rules to construct schedules. The conclusions drawn in the work may be extended to many other applications.

The comparison of MOGAs has illustrated various methods that can be used to measure performance of multi-objective optimisers. Many of these extend to use with a general multi-objective optimisation tool, and need not be confined simply to use with genetic algorithms. It has initiated scope for future work on the formal comparison of MOGAs via statistical tests. The ability to measure the progress of MOGA performance can allow future developments and implementations to be explored, and can provide for a comprehensive comparison of MOGA methods on many other types of application. This can be performed by comparing one form of MOGA with another, and by comparing MOGAs with non-GA methods.

An overall theme of the thesis has been that there is rarely 'one right answer' to meet the complexities of a real-life manufacturing problem. This fact has encouraged the development of techniques that not only address this fact, but also offer additional advantages to the user in terms of accuracy, flexibility and relevance to the problem.

The general result of developing a schedule optimisation system that works towards meeting real-life users' needs has been addressed. The MOGA system does not confine the user to a static problem representation or definition, and allows more freedom to work with compromises between objectives to the user's advantage.

This work's main limitation is the use of one set of particular test problems, which can still not adequately represent the full detail found in a factory. However, the methods have been developed to allow additional detail to be added, and it is hoped that these will allow a more comprehensive scheduling system to be implemented in future. The limited example provided by using one particular application rather than a set of various problems may also limit the extent to which general conclusions may be drawn concerning other problems. However, many other applications have similar requirements; time constraints, changing or uncertain problem information, or the need for multi-objective optimisation, and the comments made on these aspects would hold true under similar circumstances.

Over all, this work has provided a comprehensive demonstration of MOGA methods to one difficult real-life scheduling problem and has indicated how these methods can provide advantages to the user. The concepts from which the MOGA is developed may be extended to other problems

with similar criteria, and the work offers illustrations of how the performance MOGA methods may be tested.

## ***7.4 Further Work***

Suggestions have been made for future work throughout the relevant chapters. These can be summarised as follows.

### **7.4.1 Problem Representation using Schedule Builders**

The comparison of schedule building techniques in chapter 4 was somewhat limited by the use of just one heuristic rule against three of the preference-based methods. Other heuristics might be included for comparison to allow more extensive conclusions to be drawn, and such work would require necessary data and preference elicitation to be defined.

The development of the preference-based methods might include exploration of the preference-definitions themselves, including the case where preferences might be defined by multiple users. Further work may also be conducted on how these modifications can improve the use of GAs for scheduling. Finally, the use of schedule builders themselves might be examined against alternative, more direct representations for this problem.

Throughout the thesis, the idea of decision making is explored in various contexts. Whether in long-term management strategy, or as an immediate reaction to a problem in the factory, the incorporation of realistic decision patterns within optimisation techniques should be considered. The method should have accuracy from a theoretical point of view, and credibility from a practical point of view.

## **7.4.2 Multi-Objective Optimisation**

Multi-objective optimisation gives the user interaction with the optimisation process for changing problems. This provides immediate relevance as the solution is being developed. Current methods in practical scheduling may meet changing circumstances by restarting a scheduling method with new information. Alternatively a current solution may be changed to meet the new circumstances without consideration of the new solution's optimality. The MOGA allows a new approach to meeting changing circumstances, without loss of optimality or reaction time.

More work is still to be done if effective comparison of MOGA performance can be made. Comparisons allow the general development and improvement of MOGA techniques. As seen in Chapter 5, there are various implementations of GA for solving multi-objective optimisation problems. One useful study would be to provide a comprehensive comparison of all such methods.

The idea that there is no absolute best method for performing MOGA optimisation on this problem may indicate further areas for categorisation of MOGA performance. In this example, the WS method is more effective for experimental work, for which many runs of a GA can be performed to find results, whilst the Pareto MOGA lends itself to practical use, for a situation where the method is run just the once. Whether the advantageous properties of these methods can be combined to create a MOGA that performs well under any circumstances is also an area for research.

The statistics test is also presented in its initial stages. Immediate work on this would include the ability to perform one-sided tests, adjustments to p-values for multiple tests, step-downs for multiple comparisons, and additional work on the interpretation of the results.

## **7.4.3 Practical Use of GAs for scheduling**

Previous work on GAs for scheduling has focused on finding a good method of providing a solution to a set problem. The work in this thesis has presented the notion of a user working with a scheduling system to develop a set of solutions for a problem situation, so that the schedule finally chosen may bear immediate relevance to a continually changing environment.

The implementation of GAs allows the user a great deal of choice, in implementation, modelling and design. By exploiting these features further, it is possible to include objective weighting or priority assignment, and indication of the best solution to meet particular circumstances.

The evolution-like basis of GAs makes them well suited to working with changing problem environments even in their simplest form (Cobb, 1990). Allowing the user interaction within the optimisation process increases this power and offers a truly flexible and immediately relevant system for production scheduling.



## ***7.5 Conclusion***

This work originally aimed to demonstrate that GAs could be used to find satisfactory schedules for a difficult real life manufacturing environment. Starting from the provision of schedules that performed single-objective optimisation on a static problem, the work has developed the technique to allow multi-objective optimisation to be extended to dynamic problem information.

The flexibility of the MOGA technique makes it an effective tool for meeting manufacturer's needs in the field of optimisation. From a theoretical point of view, it demonstrates the variety of implementations that are possible, which still provide optimisation, both for single-objective and multiple-objective problems.

Given the potential for further development of scheduling optimisation methods, interactive manufacturing systems, multi-objective optimisation methods and decision-making tools, this work has barely begun to address the possibilities in this field. The flexible manufacturing decision-making systems that might be provided by the use of MOGAs may allow industry a great competitive advantage in future; their development offers many interesting research issues for the GA theorist. Already GA optimisation methods are increasing in their use in industry, (e.g., Petsinger, 1996); it will be absorbing to see what further developments MOGAs can bring to production scheduling.

# A. Appendix.

## A.1. Statistical test

The following null hypothesis is chosen to compare performances of two MOGAs, in this case labelled 'MOGA1' and 'MOGA2'.

*H<sub>0</sub>: There is no difference between the performances of MOGA1 and MOGA2*

*H<sub>1</sub>: There is a difference between the performances of the methods*

## A.2. Permutation test

To implement a test of this hypothesis, a permutation test is used. Permutation tests have the advantages of making very few assumptions about the underlying distribution. The essential assumption is that the data in the samples can be exchanged between categories under  $H_0$ . (Good, 1994).

This is valid for these MOGA comparisons, as the non-dominated results being examined could be equally produced by any of the methods under  $H_0$ . The permutation test has the advantage of allowing a wide range of test statistics to be used. One of its key advantages is that the user is allowed to define their own, best suited to the application.

As the test is non-parametric, there is also no need to assume normality of the underlying distributions; this is an advantage in our case as normality cannot be assumed for the underlying distribution of this test statistic. The distribution of the test statistic is unknown and will even depend on the data, because GAs are stochastic processes. Permutation tests are exact even in these conditions. All these assumptions make this method ideal for testing such hypotheses as those provided in this work.

## A.3. Choosing a test statistic

In theory, as seen in Chapter 6, the test statistic provided by the distribution is a measure of the ordinal distances between all possible arrangements of the EAF for each method. In practice, permutations of the data points that form the EAFs are used. The test is therefore implemented as follows.

Matrix  $A_{ij}$  is calculated

where

$a_{ij}$  is the number of points:

reaching attainment level  $i$  for run  $j$  of method

MOGA1, for  $i=1,2,\dots,30$

(reaching attainment level  $(i-30)$  for run  $j$  of method

MOGA2, for  $i=31,32,\dots,60$

From this, the test statistic, TS, can be calculated:

$$TS = \max_j \left| \sum_{i=1}^{30} f(a_{ij}) - \sum_{i=31}^{60} f(a_{ij}) \right|$$

where

$$f(a_{ij}) = \begin{cases} 1 & \text{if } a_{ij} > 0 \\ 0 & \text{if } a_{ij} \leq 0 \end{cases} \text{ for all } a_{ij} \text{ in } A_{ij}$$

(A-1)

to give the maximum difference between the attainments of each method.

## A.4. Computing the Test Statistic from the original problem

In this example, there are 30 runs from each algorithm, in which each run is independent. It would be possible, although impractical, to calculate the test statistic for such an arrangement, by working out every permutation, measuring the maximum distance, and thereby forming a distribution of the distances against which to measure our hypothesis.

Therefore, a Monte-Carlo estimate of the underlying distribution of the test statistic under  $H_0$ ,  $TS'$ , is generated. Many random permutations of  $[1,\dots,60]$  are generated and used to permute the rows of the matrix  $A_{ij}$  into matrix  $\hat{A}_{ij}$ , as this gives the case where there is no difference between the solutions found by MOGA1 and MOGA2. This is used to generate the distribution by replacing  $A_{ij}$  with  $\hat{A}_{ij}$  in the test statistic.

$$TS' = \max_j \left| \sum_{i=1}^{30} f(\hat{a}_{ij}) - \sum_{i=31}^{60} f(\hat{a}_{ij}) \right|$$

$$f(\hat{a}_{ij}) = \begin{cases} 1 & \text{if } \hat{a}_{ij} > 0 \\ 0 & \text{if } \hat{a}_{ij} \leq 0 \end{cases}$$

for all  $\hat{a}_{ij}$  in  $\hat{A}_{ij}$ , the matrix created by permuting the rows of  $A_{ij}$

(A-2)

By repeating this, say, 1000 times, it is possible to generate an empirical distribution for the test statistic under  $H_0$ .

## A.5.Evaluation of p-value

The p-value is the probability that the TS would take the observed value (or one less likely) under the null hypothesis. The lower the p-value, the less likely it is that the GA methods being compared have similar performances.

The p-value of this test is given by:

$$p\text{-value} = \frac{(\text{number of values in the distribution of } TS' < TS)}{1000}$$

The distribution of  $TS'$  is given in (A-2)

(A-3)

## A.6.Extension to multiple comparison problems

An extension to this test is required for the experiments presented above, as multiple tests are required, given that there are more than two GAs to compare. The TS is simple to extend for this case, as follows.

In this test, the hypothesis:

***H<sub>0</sub>: There is no difference between the performances of MOGA1, MOGA2, MOGA3 and MOGA4***

***H<sub>1</sub>: There is a difference***

is tested, by all four methods being compared simultaneously.



$$TS' = \max (\Delta \hat{A}_k), \quad k = 1, 2, \dots, 6$$

where

$$\Delta A_1 = \left| \sum_{i=1}^{30} f(\hat{a}_{ij}) - \sum_{i=31}^{60} f(\hat{a}_{ij}) \right|$$

$$\Delta A_2 = \left| \sum_{i=1}^{30} f(\hat{a}_{ij}) - \sum_{i=61}^{90} f(\hat{a}_{ij}) \right|$$

$$\Delta A_3 = \left| \sum_{i=1}^{30} f(\hat{a}_{ij}) - \sum_{i=91}^{120} f(\hat{a}_{ij}) \right|$$

$$\Delta A_4 = \left| \sum_{i=31}^{60} f(\hat{a}_{ij}) - \sum_{i=61}^{90} f(\hat{a}_{ij}) \right|$$

$$\Delta A_5 = \left| \sum_{i=31}^{60} f(\hat{a}_{ij}) - \sum_{i=91}^{120} f(\hat{a}_{ij}) \right|$$

$$\Delta A_6 = \left| \sum_{i=61}^{90} f(\hat{a}_{ij}) - \sum_{i=91}^{120} f(\hat{a}_{ij}) \right|$$

and

$$f(\hat{a}_{ij}) = \begin{cases} 1 & \text{if } \hat{a}_{ij} > 0 \\ 0 & \text{if } \hat{a}_{ij} \leq 0 \end{cases}$$

for all  $\hat{a}_{ij}$  in  $\hat{A}_{ij}$ , the matrix created by permuting the rows of  $A_{ij}$

(A-5)

The p-value for this test is given in (A-3) as before.

| Test             | MO1 vs. WS | MO1 vs. PP | MO1 vs. MO2 | WS vs. PP | WS vs. MO2 | PP vs. MO2 |
|------------------|------------|------------|-------------|-----------|------------|------------|
| Attainment Level | p-value    | p-value    | p-value     | p-value   | p-value    | p-value    |
| 1                | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 2                | 0.229      | 0.000      | 0.261       | 0.000     | 0.110      | 0.000      |
| 3                | 0.000      | 0.000      | 0.477       | 0.000     | 0.117      | 0.000      |
| 4                | 0.000      | 0.000      | 0.181       | 0.000     | 0.000      | 0.026      |
| 5                | 0.046      | 0.000      | 0.171       | 0.000     | 0.069      | 0.000      |
| 6                | 0.000      | 0.008      | 0.036       | 0.000     | 0.013      | 0.000      |
| 7                | 0.004      | 0.004      | 0.527       | 0.004     | 0.001      | 0.004      |
| 8                | 0.002      | 0.000      | 0.316       | 0.000     | 0.002      | 0.000      |
| 9                | 0.001      | 0.004      | 0.169       | 0.001     | 0.004      | 0.000      |
| 10               | 0.063      | 0.000      | 0.089       | 0.000     | 0.000      | 0.000      |
| 11               | 0.017      | 0.001      | 0.040       | 0.000     | 0.002      | 0.000      |
| 12               | 0.010      | 0.000      | 0.021       | 0.000     | 0.000      | 0.000      |
| 13               | 0.006      | 0.000      | 0.105       | 0.000     | 0.000      | 0.000      |
| 14               | 0.002      | 0.000      | 0.339       | 0.000     | 0.000      | 0.000      |
| 15               | 0.000      | 0.000      | 0.685       | 0.000     | 0.000      | 0.000      |
| 16               | 0.000      | 0.000      | 0.436       | 0.000     | 0.000      | 0.000      |
| 17               | 0.000      | 0.000      | 0.250       | 0.000     | 0.000      | 0.000      |
| 18               | 0.000      | 0.000      | 0.136       | 0.000     | 0.000      | 0.000      |
| 19               | 0.000      | 0.000      | 0.257       | 0.000     | 0.000      | 0.000      |
| 20               | 0.000      | 0.000      | 0.171       | 0.000     | 0.000      | 0.000      |
| 21               | 0.000      | 0.000      | 0.119       | 0.000     | 0.000      | 0.000      |
| 22               | 0.000      | 0.000      | 0.232       | 0.000     | 0.000      | 0.000      |
| 23               | 0.000      | 0.000      | 0.104       | 0.000     | 0.000      | 0.000      |
| 24               | 0.000      | 0.000      | 0.257       | 0.000     | 0.000      | 0.000      |
| 25               | 0.000      | 0.000      | 0.144       | 0.000     | 0.000      | 0.000      |
| 26               | 0.000      | 0.000      | 0.063       | 0.000     | 0.000      | 0.000      |
| 27               | 0.001      | 0.000      | 0.035       | 0.000     | 0.000      | 0.000      |
| 28               | 0.000      | 0.000      | 0.079       | 0.000     | 0.000      | 0.000      |
| 29               | 0.000      | 0.000      | 0.168       | 0.000     | 0.000      | 0.000      |
| 30               | 0.000      | 0.000      | 0.319       | 0.000     | 0.000      | 0.000      |
| 31               | 0.000      | 0.000      | 0.520       | 0.000     | 0.000      | 0.000      |
| 32               | 0.000      | 0.000      | 0.304       | 0.000     | 0.000      | 0.000      |
| 33               | 0.000      | 0.000      | 0.515       | 0.000     | 0.000      | 0.000      |
| 34               | 0.000      | 0.000      | 0.731       | 0.000     | 0.000      | 0.000      |
| 35               | 0.000      | 0.000      | 0.480       | 0.000     | 0.000      | 0.000      |
| 36               | 0.001      | 0.000      | 0.727       | 0.000     | 0.000      | 0.000      |
| 37               | 0.000      | 0.000      | 0.479       | 0.000     | 0.000      | 0.000      |
| 38               | 0.000      | 0.000      | 0.686       | 0.000     | 0.000      | 0.000      |
| 39               | 0.000      | 0.000      | 0.411       | 0.000     | 0.000      | 0.000      |
| 40               | 0.000      | 0.000      | 0.205       | 0.000     | 0.000      | 0.000      |
| 41               | 0.000      | 0.000      | 0.383       | 0.000     | 0.000      | 0.000      |
| 42               | 0.000      | 0.000      | 0.130       | 0.000     | 0.000      | 0.000      |
| 43               | 0.001      | 0.000      | 0.266       | 0.000     | 0.000      | 0.000      |
| 44               | 0.000      | 0.000      | 0.475       | 0.000     | 0.000      | 0.000      |
| 45               | 0.000      | 0.000      | 0.740       | 0.000     | 0.001      | 0.000      |
| 46               | 0.000      | 0.000      | 0.366       | 0.000     | 0.000      | 0.000      |
| 47               | 0.000      | 0.000      | 0.118       | 0.000     | 0.002      | 0.000      |
| 48               | 0.000      | 0.000      | 0.259       | 0.000     | 0.000      | 0.000      |
| 49               | 0.000      | 0.000      | 0.033       | 0.000     | 0.000      | 0.000      |
| 50               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 51               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 52               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 53               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 54               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 55               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 56               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 57               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 58               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 59               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |
| 60               | 0.000      | 0.000      | 0.000       | 0.000     | 0.000      | 0.000      |

Table A-1 - P-values of tests at each attainment level

## References

- Bagchi, S., Uckan, S., Miyabe, Y., Kawamura, K., 1991. Exploring Problem-Specific Recombination Operators for Job-Shop Scheduling, *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers
- Balas, E., 1969. Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithms, *Operations Research*, 17, 941-957.
- Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail, *Journal of the Operational Research Society*, 4111, 1990, pp.1069-1072. All the files in OR-Library are available via anonymous ftp to <ftp://mscmga.ms.ic.ac.uk>. The numeric equivalent of this ftp address is 155.198.66.4
- Bierwirth, C., 1995. A Generalized Permutation Approach to Job-shop Scheduling with Genetic Algorithms, *OR-Spektrum*, Vol. 17 No. 2/3, pp. 87 - 92.
- Bierwirth, C., Kopfer, H., Mattfeld, D. C., and Rixen, I., 1995. Genetic Algorithm based Scheduling in a dynamic manufacturing environment, *Proc. IEE Conference on Evolutionary Computation*, pp. 439 - 443, 1995.
- Bruns, R., 1993. Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling, *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. Stephanie Forrest. Morgan Kaufmann Publishers
- Burke, E., Elliman, R., and Weare, R., 1995. Specialised Recombinative Operators for timetabling Problems, *Evolutionary Computing, Lecture Notes in Computer Science 993*, ed. Fogarty, 1995.
- Carlier, 1989. The one machine sequencing problem, *European Journal of OR*, 11, 42-27.
- Cartwright, H. M., 1994. Getting the Timing Right – The Use of Genetic Algorithms in Scheduling, *Applications of Modern Heuristics*, ed. V. J. Raymond-Smith, 1994.
- Cartwright, H.M., and Tuson, A., 1994. Genetic Algorithms and Flow shop scheduling: towards the development of a real-time process control system, *Proceedings of the AISB Workshop on Evolutionary Computing*, Leeds University, 1994, available by anonymous ftp from <ftp://muriel.pcl.ox.ac.uk>
- Chipperfield, A. J. Fleming, P. J., and Pohlheim, H., 1994. A genetic algorithm toolbox for MATLAB, *Proc. Int. Conf. on Systems Engineering*, Coventry, UK, pp. 200 - 207, 1994.
- Chipperfield, A. J., 1995. Parallel processing in computer aided control system design. *Ph.D. Thesis*, University of Sheffield, Department of Automatic Control and Systems Engineering, 1995.
- Cleveland, G. A., and Smith, S. F., 1989. Using genetic algorithms to schedule flow-shop releases, *Proceedings of the Third International Conference on Genetic Algorithms*.



- Cobb, H. G., 1990. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having continuous Time-Dependent Nonstationary Environments. *Available from <http://www.aic.nrl.navy.mil/papers/1990/ml.html>*
- Cobb, H. G., and Grefenstette, J. J., 1993. Genetic Algorithms for tracking changing Environments, *Proceedings of Fifth International Conference on Genetic Algorithms and their Applications*, pp. 523 - 529.
- Cohon, J. P., Hegde, S. U., Martin, W. N., and Richards, D., 1987. Punctuated Equilibrium: a parallel genetic algorithm, *Proceedings of Second International Conference on Genetic Algorithms and their Applications*, pp. 148 – 155.
- Collins R. J., and Jefferson, D. R., 1991. A Multi-Population Genetic Algorithm for solving the K-Partition Problem on Hyper-Cubes. *Proceedings of Fourth International Conference on Genetic Algorithms and their Applications*, 1991.
- Coloni, A., Dorigo, M., Maniezzo, V., 1991. Distributed Optimization by Ant Colonies, *Proc. First European Conference on Artificial Life*, pp. 134 - 142, MIT Press.
- Conover, W. J., 1971. *Practical nonparametric statistics*, New York;Chichester:Wiley, 1971.
- Cronan, R. H. J., 1996. *Email Correspondence*.
- Darwin, C., 1859. *The Origin of the Species*, Dent.
- Davis, L., 1985. Job Shop Scheduling with Genetic Algorithms, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, ed. J. J. Grefenstette. Lawrence Erlbaum Publishers
- Davis, L., 1989. Adaptive Operator Probabilities in Genetic Algorithms, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, 1989, pp. 60 - 69.
- Davis, L., editor, 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.
- Dawkins, R., 1986. *The Blind Watchmaker*, Longman.
- Dawkins, R., 1996. *Climbing Mount Improbable*, Viking.
- Deb, K. and Goldberg, D. E., 1989. An investigation into niche and species formation in genetic function optimization, *Proceedings of Third International Conference on Genetic Algorithms and their Applications*.
- Della Croce, F., Tadei, R., Volta, G., 1993. A Genetic Algorithm for the Job-shop Problem. *Report*, Politecnico di Torino Italy
- Dorndorf and Pesch, 1995. *Evolution Based Learning in a Job Shop Scheduling Environment*, Computers and Operations Research, Vol. 22, Issue 1, pp. 25 - 40.
- Dubois, D., Fargier, H., and Prade, H., 1995. Fuzzy Constraints in Job Shop Scheduling, *Journal of Integrated Manufacturing*, 6, pp. 215 – 234, 1995.
- Efron, B. and Tibshirani, R. J., 1993. *An introduction to the bootstrap*, Chapman and Hall, 1993.

- Esmail and Saggi, 1996. *A changing paradigm: agile manufacturing*, *Manufacturing Engineer*, vol. 75, no. 6, pp. 285 - 288. December 1996.
- Falkenauer, E.; Bouffouix, S., 1991. A genetic algorithm for job-shop, *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Cat. No.91CH2969-4, p. 824-9 vol. 1
- Fang, H. L., Ross, P. and Corne, D., 1993. A Promising Genetic Algorithm Approach to Job-shop Scheduling, Rescheduling and Open-Shop Scheduling Problems. *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. Stephanie Forrest. Morgan Kaufmann Publishers
- Fang, H. Ross, P. and Corne, D., 1994. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problem, *ECAI 11th European Conference on AI*, Wiley.
- Fleming, P.J., and Pashkevich, A. P., 1985. Computer Aided Control Systems Design using a Multiobjective Approach, *Proc. Control '85*, pp. 174 - 179.
- Fonseca, C. M., Fleming, P. J., 1993. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. S. Forrest. Morgan Kaufmann Publishers.
- Fonseca, C. M. and Fleming, P. J., 1995. *An overview of Evolutionary Algorithms for Multiobjective optimisation*, *Evolutionary Computation*, 3, 1. pp. 1 - 16, 1995.
- Fonseca, C. M., 1995. Multi-Objective Genetic Algorithms with Application to Control Engineering Problems. *Ph.D. Thesis*, Department of Automatic Control & Systems Engineering, University of Sheffield.
- Fonseca, C.M. and Fleming, P. J., 1996. *On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers*, *Parallel Problem Solving From Nature IV*, pp. 584 - 594, Berlin.
- Fonseca, C. M., 1997. *Personal Correspondence*.
- Food Manufacture, 1994. Editorial, 'Chilled Food Market Booming', *Food Manufacture*, August 1994, vol. 69, no. 8, p. 9.
- Fourman, M. P., 1985. Compaction of Symbolic Layout using GAs, pp. 141 - 153. *Proceedings of International Conference on Genetic Algorithms*.
- Fox and McMahon, 1990. *Genetic Operators of Sequencing Problems*, *Foundations of Genetic Algorithms*, ed. G. J. E. Rawlings, 284 - 301.
- Fox, M., S, and Sadeh, N., 1990. Why is Scheduling Difficult? A CSP Perspective, *ECAI90 - Proceedings of the Ninth European Conference on AI*, Pitman, 1990.
- French, S., 1982. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood.
- Fulkerson, B., and Parunak, V., 1995. The Living Factory: Applications of Artificial Life to Manufacturing, *2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ., 1995.

- Fulkerson, B., and Staffend, G., 1995. Multi-Agent Simulation and Decentralized Control, *Proc. INFORMS 1995*.
- Garey, M. R., and Johnson, D. S., 1979. *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman.
- Garfinkel, R. S., 1985. Motivation and Modelling, Chapter 2, in *Lawler et al.*
- Giffler B., and Thompson G. L., 1969. Algorithms for solving production scheduling problems, *Operations Research* 8: 487 -503,.
- Goldberg, D. A., 1989. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley
- Goldberg, D. A., and R. Lingle, Jr., 1985. Alleles, loci and the travelling salesman problem. *Proceedings of International Conference on Genetic Algorithms*, pp. 154-159
- Goldberg, D. E., 1989. *GAs in Search, Optimization and Machine Learning*, Addison-Wesley.
- Goldratt, E. M, 1993. *The Goal; A process of ongoing improvement*, Gower.
- Good, P., 1994. *Permutation Tests, A Practical Guide to Resampling Methods for Testing Hypotheses*, Springer Series in Statistics, Springer-Verlag.
- Gould, S. J., 1980. *The Panda's Thumb*, Norton.
- Grabot, B., and Genete, L., 1994. Dispatching Rules in Scheduling: A Fuzzy Approach, *International Journal of Production Research*, vol. 32, no. 4, pp. 903-9, 1994.
- Grefenstette, J. J. 1992. Genetic Algorithms for changing environments, *Parallel Problem Solving from Nature* 2, pp. 137 - 144.
- Grefenstette, J. J., 1991. Lamarckian Learning in Multi-Agent Environments, *Proceedings of Fourth International Conference on Genetic Algorithms and their Applications*, pp. 303-310, 1991.
- Grefenstette, J., Gopal, R., Rosmaitta, B., Van Gucht, D., 1985. Genetic Algorithms for the travelling salesman problem. *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, ed. J. J. Grefenstette. Lawrence Erlbaum Publishers
- Harding, H. A., 1987. *Production Management, The M & E Handbook Series*, Fourth Edition.
- Heitkoetter, Joerg, ed. 1993 - 1997 The Hitch-Hikers Guide to Evolutionary Computation: A list of Frequently Asked Questions FAQ, Usenet: comp.ai.genetic. Available via anonymous ftp from rtfm.mit.edu in pub/usenet/news.answers/ai-faq/genetic/
- Hilliard, M.E., Liepins, G.E., Palmer, M., Morrow, M., Richardson, J., 1987. A classifier-based system for discovering scheduling heuristics. *Proceedings of the Second International Conference on Genetic Algorithm*, Lawrence Erlbaum Associates
- Hoffman, A. J., Wolfe, P., 1985. History, Chapter 1, in *Lawler et al.*
- Holland, J., 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press

- Hollstien, R. B., 1971. Artificial genetic adaptation in computer control systems, *Doctoral dissertation*, University of Michigan.
- Horn, J., Nafpliotis, N, and Goldberg, D. E., 1994. A Niche Pareto GA for Multi-Objective Optimisation. *Proceedings of the First IEEE Conference on Evolutionary Computation*, 1994.
- Husbands, P., Mill, F., 1991. Simulated co-evolution as the mechanism for emergent planning and scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers
- Husbands, P., 1993. An ecosystems model for integrated production planning, *International Journal of Computer Integrated Manufacturing*, 1993, vol. 6, no. 1-2, pp. 74-86.
- Hwang C. L., and Masud, A. S. M, 1979. Multiple Objective Decision Making - Methods and Applications, *Lecture Notes in Economics and Mathematical Systems*, No. 164, Springer Verlag, 1979.
- Independent on Sunday, *News item on 24-hour supermarket opening*, March 1997.
- Ishibuchi, H., and Murata, T., 1996. Multi-Objective Genetic Local Search Algorithm, *International Conference on Evolutionary Computation '96*.
- Kohler and Steiglitz, 1976 in *French*, 1982
- Koza, J. R., 1992. *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press.
- Kursawe, F., 1991. A Variant of Evolution Strategy for Vector Optimization, *Parallel Problem Solving from Nature1*, pp. 193 - 196.
- Langdon, W., 1996. Scheduling Maintenance of Electrical Power Transmission Networks Using Genetic Programming, *Proc. GP-96 Conference*, John Koza ed., Stanford Bookstore.
- Langton, C. G., 1989. *Artificial Life: the Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Addison Wesley.
- Laursen, P. S., 1996. Parallel Hearst Search – Introduction and a New Approach, *Lecture Notes in Computer Science*, 1054, Springer-Verlag, 1996.
- Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B. editors 1985. *The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization*, John Wiley & Sons.
- Lee, I., Sikora, R., Shaw, M. J., 1993. Joint Lot Sizing and Sequencing with Genetic Algorithms for Scheduling - Evolving the Chromosome Structure, *Proceedings of the Fifth International Conference on Genetic Algorithms*, ed. S. Forrest. Morgan Kaufmann Publishers.
- Lehmann, E. L., 1986. *Testing Statistical Hypotheses*. Second Edition. Wiley Series in Probability and Statistics, Wiley.
- Levy, S., 1994. *Artificial Life*, Penguin.
- Lucan and Gray, 1995. *The Decadent Cookbook*. Dedalus.

- Mackle, G., Savic, D. A., and Walters, G. A., 1995. Applications of Genetic Algorithms to Pump scheduling for Water Supply, *Proceedings of GALESIA '95*, pp. 400 - 405.
- Mahfoud, S. W., 1993. Simple Analytical Models of Genetic Algorithms for Multimodal Function Optimisation, *IlliGAL Report, no 93001*, February 1993.
- Mattfeld, D. C., Kopfer, H., and Bierwirth, C., 1994. Control of Parallel Population Dynamics by Social-Like Behaviour of GA-Individuals, *Parallel Problem Solving from Nature 3*, p. 15 - 24.
- McIlhagga, M., Husbands, P., and Ives, R., 1996. A Comparison of Optimization Techniques for Integrated Manufacturing Planning and Scheduling, *Parallel Problem Solving from Nature IV*, Berlin, 1996.
- McMahon, G. and Florian, M., 1975. On Scheduling with ready times and due dates to minimise maximum lateness, *Operations Research*, 23, 475-482.
- Mendel, G., 1866. *Versuche über Pflanzenhyriden*, Trans. Brünn Natural History Society, Brno, Czech Republic.
- Michalewicz, Z., 1992. *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.
- Michalewicz, Z., 1996. *Evolutionary Computation; Practical Issues*, International Conference on Evolutionary Computation, Int. Conf. On Evolutionary Computation '96. pp. 30 - 39.
- Mitchell, M., 1996. *An Introduction to Genetic Algorithms*, MIT Press.
- Mohanty, R. P. and Venkatraman, S., 1993. Justification study for computer integrated manufacturing, *IJCIM*, 6, 6, pp. 366-374.
- Morad, N., and Zalzala, A., 1995. The Formation of Manufacturing Cells Using Genetic Algorithms, *Research Report 575*. Department of Automatic Control & Systems Engineering, The University of Sheffield, May 1995.
- Murata, T., 1997. *Genetic Algorithms for Multi-Objective Optimisation*, Ph.D. Thesis, Osaka Prefecture University, submitted for examination in February 1997.
- Muth, J. F., and Thompson, G. L., 1963. *Industrial Scheduling*, Prentice Hall.
- Nakano, R., Yamada, T., 1991. Conventional Genetic Algorithm for Job-shop Problems, *Proceedings of the Fourth International Conference on Genetic Algorithms*, ed. R. K. Belew and L. B. Booker. Morgan Kaufmann Publishers
- Niemeyer and Shiroma, 1996. Production Scheduling with GA and Simulation, *Parallel Problem Solving from Nature 4*, Berlin, pp. 930 - 936, September 1996.
- Norman, B., A., and Bean, J., C., 1994. Random Keys Genetic Algorithm for Job-shop Scheduling, *Technical Report*, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109
- Oliver, I. M., Smith, D. J., and Holland, J. R. C., 1987. A Study of Permutation Crossover Operators on the Travelling Salesman Problem, *Proceedings of the Second International*

- Conference on Genetic Algorithms and their Applications*, ed. J. J. Grefenstette. Lawrence Erlbaum Publishers
- Osyczka, A. and Tamura, H., 1996. Pareto Set Distribution Method for Multi-Criteria Optimisation Using GA. *Proceedings of Mendel 96, the 2nd International Mendel Conference on Genetic Algorithms*, TU Brno, Czech Republic, June 1996.
- Palmer, G., J., 1996. A Simulated Annealing Approach to Integrated Production Scheduling, *Journal of Intelligent Manufacturing*, pp. 163 - 176, Vol. 7, June 1996.
- Paredis, J., 1994. Co-evolutionary Constraint Satisfaction, *Parallel Problem Solving from Nature 3*. pp. 47 - 55
- Petsinger Jr., Thomas, 1995. *At Deere, they know a mad scientist may be a firm's biggest asset*, Wall Street Journal, Page B1, July 14, 1995.
- Ranito, J. & Neto, J., 1993. Using normal crossover and mutation on order - based problems: a new representation for permutations, *First draft of report*, May 3rd, 1993.
- Raymond, M. K., 1995. Gin Rummy Tactics on the Factory Floor, *American Machinist*, December 1995.
- Reeves, C. R., 1993. Hybrid Genetic Algorithms for Bin-packing and Related Problems, submitted to appear in a special issue of *Annals of Operations Research*.
- Richardson, J. T., and Palmer, M. R., 1989. Some Guidelines for Genetic Algorithms with Penalty Functions, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*.
- Rodammer and White, 1988. A Recent Survey of Production Scheduling, *IEEE Trans. Systems Man and Cybernetics*, November 1988, pp. 841 - 851.
- Rumelhart, D. E., and McClelland, J. L., 1986. *Parallel Distributed Processing*, Vol. 1, MIT Press, Cambridge, MA, pp. 287 - 288, 322 - 324.
- Savic, D. and Walters, G., 1995. An Evolution Program for Optimal Pressure Regulation in Water Distribution Networks, *Engineering Optimization*, vol. 24, no. 3, pp. 197 - 219.
- Schaffer, J. D., 1985. Multiple Objective Optimisation with Vector Evaluated Genetic Algorithm, *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 93 - 100.
- Schroder, P., 1997. Example illustration of seven-objective trade-off graph. Department of Automatic Control & Systems Engineering, University of Sheffield.
- Shaw, K. J., and Fleming, P. J., 1996a. Initial Study of Multi-Objective Genetic Algorithms for Scheduling the Production of Chilled Ready Meals. *Proceedings of Mendel 96, the 2nd International Mendel Conference on Genetic Algorithms*, Brno, Czech Republic, June 26th - 28th, 1996.

- Shaw, K. J., and Fleming, P. J., 1996b. An Initial Study of Practical Multi-Objective Production Scheduling, Using Genetic Algorithms. *Proceedings of International Conference on Control '96*, University of Exeter, September 2nd - 5th, 1996.
- Shaw, K. J., and Fleming, P. J., 1997a. An Overview of Multi-Objective Genetic Algorithms for Production Scheduling, - *Proceedings of the DTI/ACTT Regional Seminar in Finite Capacity Scheduling in the Process Industries*, Manchester, UK, 23rd January 1997.
- Shaw, K. J., and Fleming, P. J., 1997b. Use of Rules and Preferences for Schedule Builders in Genetic Algorithms Production Scheduling, - To be Published in the *Proceedings of the AISB '97 Workshop on Evolutionary Computation*, Springer-Verlag.
- Shaw, K. J., and Fleming, P. J., 1997c. Including Real-Life Problem Preferences in Genetic Algorithms to Improve Optimisation of Production Schedules, *to appear in the Proceedings of the GALESIA '97*, Glasgow, 2nd - 4th September, 1997.
- Shaw, K. J., and Fleming, P. J., 1997d. Which Line is it Anyway? Use of Rules and Preferences for Schedule Builders in Genetic Algorithms Production Scheduling. *Proceedings of the AISB '97 Workshop on Evolutionary Computation*, April 7th-8th, 1997.
- Shaw, R. 1996. "Extending the Shelf Life of Chilled Ready Meals", *Developments in Meat Packaging*, ed. Taylor A. A., ECCEAMST, Utrecht, The Netherlands.
- Soares, C., 1994. *Evolutionary Computation for the Job Shop Scheduling Problem*, Minho University, Braga, Portugal.
- Srivinas, N. and Deb, K., 1994. Multiobjective Optimisation Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, Vol. 2, No. 3, pp. 221 - 249.
- Starkweather, T., McDaniel, S., Mathias, K., Whitley, D., Whitley, C., 1991. A Comparison of Genetic Sequencing Operators. *Proceedings of the Fourth International Conference on Genetic Algorithms*, ed. R. K. Belew and L. B. Booker. Morgan Kaufmann Publishers
- Starkweather, T., Whitley, D., 1993. A Genetic Algorithm for Scheduling and Resource Consumption. *Operations Research in Production Planning and Control, Lecture Notes in Economics and Mathematical Systems*, pp. 567 - 583, Springer - Verlag 1993.
- Stewart, Liaw, and White, 1994. A Bibliography of Heuristic Search Through 1992. *IEEE Transactions of Systems, Man and Cybernetics*, vol. 24, no. 2, February 1994.
- Surry, Radcliffe and Boyd, 1995. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks: the COMOGA Method , *Evolutionary Computation: AISB95*, 1995.
- Syswerda and Palmucci, 1991. The Application of Genetic Algorithms to Resource Scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 502 - 508.
- Syswerda, G., 1991. Schedule Optimisation using Genetic Algorithms, *Handbook of Genetic Algorithms*, ed. Davis, L., Van Nostrand Reinhold.
- Tadei, R., Della Croce, F., Menga, G., 1995. Advanced Search Techniques for the job shop problem: a comparison., *RAIRO Recherche Operationelle*, vol. 29, Iss. 2, pp. 179 - 194.

- Tamaki, H., Kita, H., and Kobayashi, S., 1996. Multi-Objective Optimization by Genetic Algorithms; A Review. *International Conference on Evolutionary Computation '96*. pp. 517 - 522.
- Tamaki, H., Nishikawa, Y., 1992. A paralleled GA based on a neighbourhood model and its application to Job-shop scheduling, *Parallel Problem Solving from Nature*, 2 1992, Ch. 60, pp. 573-582.
- Tsang, E. P. K., 1995. Scheduling techniques - a comparative study, *BT Technology Journal*, vol. 13 no. 1, January 1995.
- Tuson, A., Ross, P. Wheeler, R, 1997. Emergency Resource Redistribution in the Developing World: Towards a Practical Evolutionary/ Meta-Heuristic Scheduling System, to appear in *GALESIA '97*.
- Uckan, S., Bagchi, S, Kawamura, K., 1993 Managing Genetic Search in Job-shop Scheduling, *IEEE Expert*, vol. 8, no. 5, October 1993
- Viennet, Fonteix and Marc, 1995. *Proc. Artificial Evolution*, Brest, France, pp. 120 - 127, 1995.
- Whitley, D., Starkweather, T. and Fuquay, D., 1989. Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator, *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, ed. J. D. Schaffer. Morgan Kaufmann Publishers.
- Whitley, D., Starkweather, T., and Shaner, D., 1991. The Travelling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, Chapter 22, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.
- Willems, T. M. and Brandts, L. E. M. W., 1995. Implementing heuristics as an optimization criterion in neural networks for Job Shop Scheduling, *Journal of Intelligent Manufacturing*, vol. 6, pp. 377 - 387, Dec. 1995.
- Williams, W., 1996. *Email Correspondence*.
- Wogan, B., 1993. New Generations of Control, *Dairy Industries International*, January 1993.
- Wright, A. H., 1991. Genetic Algorithms for Real Parameter Optimization, *Foundations of Genetic Algorithms*, ed. J. E. Rawlins, Morgan Kaufmann, pp. 205 - 218.
- Yamada, T., and Nakano, R., 1992. A Genetic Algorithm applicable to Large Scale Job Shop Problems, *Parallel Problem Solving from Nature*, 2, 1992, pp. 281 - 291