GENETIC PROGRAMMING FOR MANUFACTURING OPTIMISATION

Thesis submitted for candidature for the degree of PhD

Christos Dimopoulos

August 2000

Department of Automatic Control and Systems Engineering

University of Sheffield

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 being concurrently submitted in candidature for any degree.

Candidate:____________________________

Christos Dimopoulos

Supervisor:____________________________

Neil Mort

To my mother

SUMMARY

A considerable number of optimisation techniques have been proposed for the solution of problems associated with the manufacturing process. Evolutionary computation methods, a group of non-deterministic search algorithms that employ the concept of Darwinian strife for survival to guide the search for optimal solutions, have been extensively used for this purpose.

Genetic programming is an evolutionary algorithm that evolves variable-length solution representations in the form of computer programs. While genetic programming has produced successful applications in a variety of optimisation fields, genetic programming methodologies for the solution of manufacturing optimisation problems have rarely been reported. The applicability of genetic programming in the field of manufacturing optimisation is investigated in this thesis. Three well-known problems were used for this purpose: the one-machine total tardiness problem, the cell-formation problem and the multiobjective process planning selection problem. The main contribution of this thesis is the introduction of novel genetic programming frameworks for the solution of these problems.

In the case of the one-machine total tardiness problem genetic programming employed combinations of dispatching rules for the indirect representation of job schedules. The hybridisation of genetic programming with alternative search algorithms was proposed for the solution of more difficult problem instances. In addition, genetic programming was used for the evolution of new dispatching rules that challenged the efficiency of man-made dispatching rules for the solution of the problem.

An integrated genetic programming – hierarchical clustering approach was proposed for the solution of simple and advanced formulations of the cell-formation problem. The proposed framework produced competitive results to alternative methodologies that have been proposed for the solution of the same problem. The evolution of similarity coefficients that can be used in combination with clustering techniques for the solution of cell-formation problems was also investigated.

Finally, genetic programming was combined with a number of evolutionary multiobjective techniques for the solution of the multiobjective process planning selection problem. Results on test problems illustrated the ability of the proposed methodology to provide a wealth of potential solutions to the decision-maker.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor Dr. Neil Mort for his guidance and support throughout the period of this research work and for accepting to take over my supervision after my initial supervisor Dr. Ali Zalzala moved to a different institution. I would also like to thank Dr. Zalzala for his guidance during the initial stages of this research and I would like to wish him the best in his new position.

I am grateful to Dr. D.I. Tseles of the Technological Educational Institute (TEI) of Piraeus for his constant encouragement and support.

I am extremely thankful to the Greek State Fund for providing the main financial support for this research. The same goes for the Department of Automatic Control and Systems Engineering for providing my tuition fees for the last year of my research.

I am grateful to my family for supporting me throughout this period of my studies and to all my friends for being there when I needed them.

This thesis concludes a personal educational cycle that started 22 years ago. I would like to take this opportunity to thank all my teachers during that period and I hope that I would be able to put the acquired knowledge in good use.

TABLE OF CONTENTS

LIST OF PUBLICATIONS

JOURNAL PUBLICATIONS

Dimopoulos, C., and Zalzala, A.M.S., "Recent developments in evolutionary computation for manufacturing optimisation: problems, solutions and comparisons", *IEEE Transactions in Evolutionary Computation*, vol.4, no.2, pp.93-113, 2000.

Dimopoulos, C., and Mort, N., "A hierarchical clustering methodology based on genetic programming for the solution of simple cell-formation problems", *International Journal of Production Research*, in press.

Dimopoulos, C. and Zalzala, A.M.S, "Investigating the use of genetic programming for a classic one-machine scheduling problem", *Advances in Engineering Software, Elsevier Journal Special Edition - Evolutionary and Adaptive Computing in Design and Manufacture 2000*, in press.

CONFERENCE PUBLICATIONS

Dimopoulos, C. and Zalzala, A.M.S, "Evolutionary computation approaches to cell optimisation", *Adaptive Computing in Design and Manufacture (ACDM '98)*, Parmee, I.C., (ed.), pp. 69-83, Plymouth, April 1998. Springer.

Dimopoulos, C. and Zalzala, A.M.S, "Optimisation of cell configuration and comparisons using evolutionary computation approaches", in *IEEE World Congress on Evolutionary Computation (ICEC '99)*, pp.148-153, Anchorage, May 1998. IEEE.

Dimopoulos, C. and Zalzala, A.M.S., "A genetic programming heuristic for the one-machine total tardiness problem", *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, vol.3, pages 2207-2214, Washington D.C., USA, July 1999. IEEE Press.

Dimopoulos, C. and Zalzala, A.M.S., "Evolving scheduling policies through a genetic programming framework", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99)*, Banzhaf, W., *et al.* (eds.), vol. 2, pp.1231, Orlando, Florida, USA, July 1999. Morgan Kaufmann.

Dimopoulos, C. and Mort, N., "Genetic programming for cellular manufacturing", in *Proceeding of the 2nd Workshop on European Scientific and Industrial Collaboration (WESIC '99)*, Roberts, G.N., and Tubb, C.A.J., (eds.), Newport, Gwent, September 1999. Mechatronics Research Centre.

Dimopoulos, C. and Mort, N., "A genetic programming-based hierarchical clustering procedure for the solution of the cell-formation problem ", *Adaptive Computing in Design and Manufacture (ACDM 2000)*, Parmee, I.C., (ed.), pp. 211-222, Plymouth, April 2000. Springer.

Dimopoulos, C. and Mort, N., "Evolving similarity coefficients for the solution of cellular manufacturing problems", in *Proceedings of the Congress on Evolutionary Computation (CEC 2000),* pp.617-624, San Diego, California, USA, July 2000. IEEE Press.

Dimopoulos, C. and Mort, N., "Solving cell-formation problems under alternative quality criteria and constraints with a genetic programming-based hierarchical clustering algorithm", to be presented at the *Sixth International Conference on Control, Automation, Robotics and Vision*, Singapore, 5-8 December, 2000.

Chapter 1

INTRODUCTION

1.1 Background

Manufacturing is the process of transforming raw materials through labour into a product of greater value that meets the designer's specifications (Singh, 1996).

The Industrial Revolution transformed the manufacturing process from a series of operations carried out collectively by a single man or few people, to a set of interrelated activities where each person involved is specialised in the completion of a certain task. As manufacturing processes grew rapidly in scale and complexity, the need for efficient production lines emerged.

Modern manufacturing companies aim to *continuously improve* the efficiency of their production lines in an attempt to reduce overall costs, increase productivity and address customer needs. The process of continuous improvement is critical in today's competitive market environment, since even a small difference in the overall production costs of rival products could have an impact on their commercial success.

The need for efficient production lines has been addressed by scientific research, which, especially the last fifty years, is continuously developing new optimisation techniques and procedures. The term *manufacturing optimisation* is usually employed for the description of problems associated with the optimisation of various stages of the manufacturing process. The scheduling of part operations, the balancing of machine workloads and the sequencing of assembly operations are some typical manufacturing optimisation problems that require optimal solutions in the day-to-day operation of a modern manufacturing plant. Manufacturing optimisation also considers problems from areas such as robot control, aggregate production planning and inventory control.

The design of modern manufacturing systems is usually based on philosophies that aim to eliminate or ease the solution of these optimisation problems. The principles of just-in-time production (JIT systems), flexibility (flexible manufacturing systems) and groupability (cellular manufacturing systems) have been used successfully for this purpose. However, the implementation of such systems gives rise to optimisation problems of different nature that can be just as difficult to solve, like the cell-formation problem, which is a subject of research in this thesis.

Most of the problems described above, when formulated mathematically, are combinatorial in nature and NP-hard (Garey and Johnson, 1979), i.e. the time complexity of solution methodologies increases exponentially with the size of the problem. This means that exhaustive enumeration of the solutions' space is only feasible for small problem instances with present computational resources.

The difficulty of manufacturing optimisation problems has attracted considerable research interest, not only for the practical issues associated with them on the manufacturing process level, but also for their benchmarking potentials in relation to solution methodologies. The main methodologies that have been used for the solution of manufacturing optimisation problems over the years are the following:

- Implicit enumeration algorithms (dynamic programming, branch and bound) (Bellman and Dreyfus, 1962))
- Classic optimisation methods (steepest descent, gradient search) (Beveridge, 1970)
- Mathematical programming (linear programming, non-linear programming) (Winston, 1995)
- Neural networks (Zurada, 1992)
- Fuzzy logic (Dubois and Prade, 1980)
- Meta-heuristics

Meta-heuristics is a group of search algorithms that have been mainly developed during the last two decades for the solution of difficult multimodal and combinatorial optimisation problems. The main feature of these algorithms is their ability to escape local optima by probabilistically accepting worse-performing solutions during an optimisation run, in contrast to classic local search or gradient search procedures. Evolutionary algorithms (EA's) (Holland, 1975), (Michalewicz, 1992), simulated annealing (Kirkpatrick et *al.*, 1985) and tabu search (Glover, 1990) are the most notable members of the meta-heuristics group. Meta-heuristics have been applied successfully on a wide range of optimisation problems where traditional solution methodologies could either handle only small problem instances (mathematical programming), or performed poorly due to the presence of local optima (local search or gradient search techniques).

Evolutionary computation methods proved to be particularly popular due to their added characteristic of being able to search the solutions' space not from a single point but from a population of points in parallel. Their search procedure employs the concept of the Darwinian strife for survival, where solutions performing better on the problem considered are given more chance of surviving during an optimisation run.

There are several variants of evolutionary algorithms that are mainly distinguished by the type of coding used for the representation of solutions, and the type of operators employed for the generation of new solutions in the search space. The optimisation methodology used in this thesis is genetic programming (GP) (Koza, 1992), one of the newest and most popular members of the evolutionary algorithms group. Genetic programming evolves solutions in the form of computer programs, i.e. structures that can be compiled either directly or with slight modifications by a computer. In that sense genetic programming is an automatic programming methodology. This efficient combination of evolutionary computation concepts with the principles of program induction has produced successful applications in a wide range of optimisation fields (Banzhaf *et al.*, 1998).

1.2 Objectives and contributions of the research

Manufacturing optimisation is one of the main application fields of evolutionary computation algorithms. The use of evolutionary computation techniques for the solution of manufacturing optimisation problems has grown rapidly during the last decade both in terms of number and range of applications (Dimopoulos and Zalzala, 2000).

However, genetic programming has not followed the development of the evolutionary computation research in the field of manufacturing optimisation. The applications of genetic programming in this field have been relatively few (McKay *et al.*, 1996), (Garces-Perez *et al.*, 1996), given the sharp rise in the popularity of the method and the number of publications during the last decade.

Motivated by this noticeable imbalance, this thesis investigates the applicability of genetic programming for the solution of manufacturing optimisation problems. It attempts to explore the applicability of genetic programming on a *range* of problems from various areas of manufacturing optimisation rather than focus on a group of similar problems from the same field, such as scheduling. In addition, the performance of the proposed methodologies is compared with the best alternative methodologies that have been reported so far for the problems considered.

This research aims to provide evidence on the benefits and disadvantages of employing genetic programming for the solution of manufacturing optimisation problems. At the same time it aims to produce quantitative arguments on the merits of using the proposed methodologies in relation to the state of the art on manufacturing optimisation.

The contributions of this research can be summarised with the help of the following points:

- Development of a genetic programming-based methodology for the solution of the one-machine total tardiness problem. The methodology employs combinations of dispatching rules for the indirect construction of job schedules. The same methodology is also combined with alternative search techniques such as local search and simulated annealing for the generation of near-optimal solutions in large problem instances.

- Evolution of a new dispatching rule for the solution of the one-machine total tardiness problem. The rule schedules jobs in a similar manner to ordinary dispatching rules; however, its form is a product of the evolutionary procedure rather than a formula based on the human understanding of the problem.

- Development of a genetic programming-based methodology for the solution of the cell-formation problem. The proposed methodology is designed to consider both simple and advanced formulations of the cell-formation problem. The latter refer to cases where information about design constraints, operation sequences of jobs, and machine workloads are explicitly considered in the formulation of the problem.
- Evolution of a similarity coefficient to be used in combination with the Single Linkage Cluster Analysis procedure (SLCA) (Sneath, 1957) for the solution of cell-formation problems. The coefficient employs identical inputs to the ones used by man-made coefficients for the calculation of similarity between machines; however, the form of the coefficient is determined by the evolutionary procedure.
- Development of a genetic programming-based methodology for the solution of the optimal process planning selection problem. The methodology uses the network representation of the problem to evolve a solution in the form of a network path. This path corresponds to a potential process plan for the product considered.
- Combination of genetic programming with various evolutionary multiobjective optimisation approaches that are based on the concept of Pareto optimality (Goldberg, 1989) for the solution of the multiple-objective version of the optimal process planning selection problem.

1.3 Outline of this thesis

The layout of this thesis is described as follows:

Following the introductory chapter, chapter 2 presents a review of evolutionary computation research on the solution of manufacturing optimisation problems. A considerable number of applications are surveyed from the fields of scheduling, process planning, cellular manufacturing, assembly lines and other related manufacturing optimisation areas.

Chapter 3 introduces genetic programming, the optimisation method that is the focus of this thesis. After briefly examining the mechanics of evolutionary computation, the genetic programming approach to problem solving and the relevant structures evolved are explained. A number of issues related to the application of genetic operators such as crossover and mutation are discussed. The chapter finishes with an illustrative example of the use of genetic programming on a symbolic regression problem.

Chapter 4 describes the application of genetic programming to the solution of the one-machine total tardiness problem. Initially the problem is defined and a survey of solution methodologies that have been proposed for its solution is presented. The genetic programming-based methodology for the solution of the problem is explained and compared with alternative solution methodologies on a wide range of test problems. The performance of hybrid approaches based on combinations of genetic programming with local search and simulated annealing algorithms is also investigated. Finally, genetic programming is employed for the evolution of a new dispatching rule for the solution of the one-machine total tardiness problem. Rules produced through the evolutionary procedure are compared with dispatching rules devised by human intuition.

Chapter 5 introduces the genetic programming-based approach for the solution of the cell-formation problem. The formulation of the problem and a literature survey of related solution methodologies are followed by the description of the genetic programming-based hierarchical clustering algorithm for the solution of simple binary cell-formation problems. The proposed approach is compared with alternative solution methodologies on a wide range of test problems taken from the literature. Results are also presented for advanced formulations of the cell-formation problem, which explicitly consider the operation sequences of jobs, constraints on the total number of cells and machines per cell, and machine workloads. Finally, genetic programming is employed for the evolution of a similarity coefficient that is used in combination with a hierarchical clustering procedure for the solution of cell-formation problems.

Chapter 6 focuses on the development of a genetic programming algorithm that explicitly considers a multiobjective manufacturing optimisation problem. The process planning selection problem is employed for this purpose. The principles of multiobjective optimisation are explained and a survey of relevant evolutionary computation techniques is presented. The application of genetic programming on the solution of the process planning selection problem is illustrated using an example of its operation on a single-objective case. Finally, combinations of genetic programming with evolutionary multiobjective optimisation techniques are applied to a group of randomly generated process planning selection problems. The evolutionary techniques that are employed in the experimentation are the Pareto-ranking approach (Goldberg, 1989), the Multi-Objective Genetic Algorithm approach (MOGA) (Fonseca and Fleming, 1993), and the Niched Pareto Genetic Algorithm (NPGA) (Horn and Nafpliotis, 1993). Results are presented to illustrate the ability of the proposed framework to provide a variety of potential solutions to the decision-maker.

Chapter 7 draws the conclusions of this research and gives recommendations for future work.

Finally, some reference data are provided in the Appendix of the thesis.

Chapter 2

EVOLUTIONARY COMPUTATION FOR MANUFACTURING OPTIMISATION

2.1 Introduction

From as early as the 1950's researchers have been using concepts based on Darwin's evolution theory for the solution of optimisation problems (Box, 1957). A considerable number of algorithms based on these concepts have been developed over the last thirty years. They are usually described by the term 'evolutionary computation methods'. The most notable members of this group are simple genetic algorithms (GA's) (Holland, 1975), evolution strategies (Rechenberg, 1973), evolutionary programming (Fogel, 1966), and genetic programming (Koza, 1992).

The basic operation of the algorithm is similar in all evolutionary computation methods: Given a certain optimisation problem, an initial population of appropriately coded solutions ('chromosomes') is generated randomly. The performance of each solution is evaluated and assigned with a 'fitness' value. Chromosomes selected from the 'old' population then create a 'new' population of solutions. The higher the 'fitness' of an individual solution, the better is its chance to be selected for the new population. Chromosomes exchange or alter their genetic material using specially designed genetic operators. The purpose of this operation is the best possible exploration of the solutions' search space. The procedure is repeated until desired 'fitness' values have been reached, or until a predefined number of iterations have been completed.

The use of intelligent techniques in the manufacturing field has been growing rapidly during the last two decades due to the fact that most of manufacturing optimisation problems are combinatorial and NP – hard, i.e. the time-complexity of the solution methodologies is not a polynomial function of the size of the problem.

Heuristic methods are often employed for the solution of these problems. A growing number of researchers have adopted the use of meta-heuristic techniques ('smart-heuristics') for large combinatorial problems. As it has been discussed in the previous chapter, evolutionary computation methods are meta-heuristics that are able to search large regions of the solutions' space without being trapped in local optima.

The aim of this chapter is to illustrate recent developments in the field of evolutionary computation for manufacturing optimisation. A wide range of optimisation problems is considered, from the classic job-shop and flow-shop scheduling problems, to assembly line balancing and aggregate production planning. The focus of this review is mainly on recent publications, but there are pointers to significant earlier approaches.

The terminology of evolutionary computation methods has not been standardised; thus the term 'evolutionary algorithms' (EAs) is used interchangeably to describe different evolutionary computation methods. The same convention will be followed throughout this chapter, unless otherwise stated.

The rest of this chapter is organised as follows: Section 2.2 examines recent evolutionary algorithms that have been proposed for the solution of the job-shop scheduling problem. The same procedure is followed in section 2.3 for the flowshop scheduling problem, in section 2.4 for the dynamic scheduling problem, in section 2.5 for the process planning problem, in section 2.6 for cellular manufacturing optimisation problems, and in section 2.7 for assembly optimisation problems. Section 2.8 overviews some recent developments in other manufacturing optimisation areas and section 2.9 draws the conclusions of this chapter.

2.2 The Job-Shop Scheduling Problem

2.2.1 Introduction and historical development

Considerable work in the field of evolutionary computation has been devoted to the solution of the job-shop scheduling problem (JSSP). The first attempt to solve the JSSP using evolutionary algorithms was made by Davis (1985), who employed the concept of preference lists for the coding of solutions (a discussion on the subject of solution representation will follow later in this chapter). A few years later Yamada and Nakano (1992) proposed a more natural representation, which was based on the completion times of operations. Since then, the number of alternative evolutionary computation approaches that have been proposed for the solution of the problem has been growing rapidly.

2.2.2 Formulation of the problem

The job-shop scheduling problem consists of ordering *n* jobs to be processed in *m* machines. Each job involves a number of different machining operations. The following conditions hold for the classic formulation of the JSSP:

- Each machine can process only one job at a time
- The sequence of operations for each job is predefined
- Two operations of the same job cannot be processed at the same time
- Pre-emption is not allowed (an operation cannot be withdrawn from a machine unless it is completed)
- Processing times are known in advance
- Transportation time between machines is zero

The quality criterion most often used for the JSSP is the minimisation of *makespan* (*Cmax*). Makespan is defined as the completion time of the final job to leave the system (Pinedo, 1995). Bierwirth *et al.*(1996) describe JSSP as a 'representative of constrained combinatorial problems'. Garey *et al.* (1976) have illustrated that it is NP-hard in the strong sense (proof by transformation of the 3-PARTITION problem to the associated JSSP decision problem). In this section the static version of the problem is considered, in which unexpected events are not taken in account. The dynamic version of the JSSP will be discussed in a following section.

Cao *et al.* (1997) argued that the classic formulation of the JSSP is unrealistic since it does not take in account a number of elements that are important in real-life scheduling, like set-up times, due dates and machine off-line times. Academic research has been criticised for considering scheduling problems that rarely appear in practice. As a result, many researchers in the field of evolutionary computation are increasingly using a variety of criteria for the evaluation of schedules. Minimisation of makespan is still used as an objective in many cases (Bierwirth *et al.*, 1996), (Dorndorf and Pesch, 1995), however, the trend in modern manufacturing optimisation systems is the minimisation of the overall production cost. Addressing the over-dominance of makespan-oriented work in the field, Fang *et al.* (1996) employed seven quality criteria for the evaluation of good schedules: maximum tardiness, average tardiness, weighted flow time, weighted lateness, weighted tardiness, weighted number of tardy jobs, and weighted earliness plus weighted tardiness. The last criterion is in accordance with the Just-In Time (JIT) principle of having a product made exactly when it is required, so that storage costs (earliness) and lateness fines (tardiness) are kept to a minimum. Similar objectives were used in (Croce *et al.*, 1995), (Kumar and Srinivasan, 1996). Due-dates and ready times of the products were pre-specified in these cases.

2.2.3 Encoding

A number of alternative representations for JSSP schedules have been proposed by researchers in the field of evolutionary computation. In the following paragraphs a classification of the most successful representations is attempted.

2.2.3.1 Direct representations

A natural representation for the solution of the JSSP problem is a data structure that can be used as a schedule itself. No decoding is needed to obtain the schedule; thus this type of representation is called '*direct*'. Burns (1993) was the first researcher to employ an EA with direct representation for the solution of a production scheduling problem. His representation explicitly defined the process plan for each job, machine assignment for each operation, and individual start-end times. Purpose-based genetic operators ensured that solutions remained valid throughout the evolutionary procedure.

An alternative approach is the use of an *m*-partitioned permutation (where *m* is the total number of machines), with each partition representing the complete schedule of an independent machine. This representation is especially popular in sequencing problems, where the solution is not partitioned, so a number of genetic operators that have been originally designed for the solution of the travelling salesman problem (Michalewicz (1992)) can be easily applied. Dagli and Sittisathancai (1995) employed this type of direct representation for the solution of the JSSP. They overcame feasibility problems by using legal schedules to initialise the population and an order-based crossover operator to preserve the precedence constraints of the problem. They also used a back-propagation neural network for the evaluation of schedules. Aizpuru and Usunariz (1995) adopted the same representation for their hybrid scheduling algorithm, which was based on evolutionary algorithms and tabu search. A knowledge-based system was employed for the generation of efficient scheduling strategies. The hybrid algorithm helped the system to induce knowledge about the scheduling procedure. Giffler and Thompson's (1969) algorithm generated initial actives schedules, and efficient operators maintained the precedence relations of the jobs.

2.2.3.2 Indirect representations

2.2.3.2.1 Job-based representations

This common type of indirect representation does not explicitly state the number of an operation, but instead, the job number is defined. The chromosome:

$$[J_1, J_2, J_1, J_3, J_2, J_1, ...]$$

indicates that the first operation of the first job should be scheduled first, followed by the first operation of the second job, the second operation of the first job, etc. It is obvious that a schedule builder is needed to transform this solution into a feasible schedule (for a discussion about schedule builders, see Cheng *et al.* (1996c)). Bierwirth *et al.* (1996) employed this method in their discussion of permutation representations for combinatorial problems. Their experimentation with various crossover operators led to the conclusion that the preservation of the absolute order of jobs and their associated operations was quite significant for the JSSP. They introduced a new operator called PPX (precedence preservation operator) which featured this useful characteristic. Fang *et al.* (1996) highlighted the superiority of a job-based GA over dispatching rules and stochastic hillclimbing on a variety of scheduling criteria. Shi (1997) built an EA scheduler that efficiently decoded the strings into active schedules and utilised genetic operators optimised for speed.

2.2.3.2.2 Dispatching rule representations

The use of dispatching rules for scheduling purposes is common manufacturing practice. The following definition of a dispatching rule is given by Blackstone *et al.* (1982): 'A dispatching rule is used to select the next job to be processed from a set of jobs awaiting service'. In the case of static scheduling, this selection is based on various job characteristics such as processing time, due date, etc.

Herrmann *et al.* (1995) proposed an efficient EA representation based on the concept of dispatching rules. The solution was encoded in the following form:

[EDD, SPT, FIFO,.........]

where:	EDD : Earliest Due Date rule
	SPT : Shortest Processing Time rule
	FIFO : First In First out rule

Each element (gene) corresponded to a particular machine, and the value of the element defined the dispatching rule that this machine employed for the scheduling of operations awaiting service. This type of representation did not suffer from feasibility problems and the application of conventional operators was straightforward. Fujimoto *et al.* (1995) employed the same representation for the scheduling of a flexible manufacturing system. In this case, each gene represented a decision making point in the plant, and the value of the gene specified the dispatching rule that would be used at this point. Kumar and Srinivasan (1996) used a circular string of dispatching rules as a scheduling policy, whenever a part was requested for processing.

Dorndorf and Pesch (1995) proposed an alternative use of the same representation for the JSSP, where each rule determined the next job to be scheduled among the conflict set of jobs created by Giffler and Thompson's algorithm. However, this method performed poorly in comparison with another algorithm presented in the same paper based on the shifting bottleneck heuristic, a well-known method for the solution of the JSSP. An EA was employed to control the selection of nodes in the enumeration tree created by this heuristic.

2.2.3.2.3 Preference-list representations

A popular way of encoding a solution of the JSSP is the preference-list representation. Preference lists are not actual schedules, but a preferable sequence of operations on each machine. Operations are scheduled according to this sequence unless they violate a precedence constraint. In that case the next operation in the preference list is scheduled. Croce *et al.* (1995) used the concept of preference lists for the encoding of solutions, together with a look-ahead evaluation method that generated non-delay schedules (a discussion on schedule types is given by Baker (1974)). Cao *et al.* (1997) addressed a complex JSSP problem with multiple objectives utilising a Hierarchical Evaluation (HE) model instead of a look-ahead evaluation. Their framework was able to generate feasible schedules and perform local optimisation at the same time, resulting in slightly better performance than Croce's algorithm. Kobayashi *et al.* (1995) and Ono *et al.* (1996) encoded the solution in the same preference-list form. They additionally introduced two purpose-based crossover operators, the subsequent exchange crossover (SSX) and job-order based crossover (JOX) respectively. JOX used the traditional Giefler and Thompson (GT) algorithm for the decoding of solutions into active schedules. The result was a much better performance both in terms of optimal and average values for Fisher & Thompson's benchmark problems (see section 2.2.4). Park and Park (1995a) (1995b) reported their preference list-based GA with the introduction of a crossover operator called active schedule constructive crossover (ASCX), which was based on the active schedule generation algorithm (Baker, 1974).

2.2.3.2.4 Alternative representations

A number of alternative schemes for the representation of schedules have been reported in the literature. The most successful of them was proposed by Kim and Lee (1995), (1996). Their schedule representation was a priority list of operation-machine assignment pairs, which corresponded to a certain priority rule. Schedules (and consequently the corresponding priority rules) were constructed with the help of a genetic reinforcement learning (GRL) procedure. Their method showed the best overall performance on Muth and Thompson's benchmark JSSP problems in comparison with any other evolutionary method included in this survey.

Yamada and Nakano (1995) employed a disjunctive-graph representation for the solution of the JSSP. Following the trend of enhancing the evolutionary process with local search techniques, they introduced a crossover operator called multi-step crossover (MSX), which was in effect a local search operator. Cho *et al.* (1996) presented the Total Operation Order Method (TOOM), where a solution was given in the form of a job operation matrix that defined the absolute order of all operations to be processed. A dynamic data structure called "hierarchical linked list" was utilised by Niemeyer and Shiroma (1996) in order to accommodate variable lengths of jobs and operations in a real manufacturing environment. Kim and Kim (1996) tackled the problem of infeasibility by using a random-keys (Bean, 1994) representation for the solutions. Finally, Gohtoh *et al.* (1995) applied a special EA with neutral mutations to some standard benchmark problems. An excellent analytical review of EA representations and hybrid methods that have been used for the solution of the JSSP, can be found in Cheng *et al.* (1996c), (1999).

2.2.4 Test problems and case studies

In recent years academic research has attempted to consider real-life scheduling problems, since the use of standard benchmark problems has not generated considerable interest in industrial circles. However, the famous Fisher and Thompson's (1963) and Lawrence's (1984) benchmark problems are still used by many researchers. Table 2.1 gives a summary of results that have been published recently for the three Fisher & Thompson problems. The best and average (wherever available) results of each method in terms of the total makespan are presented. Table 2.2 summarises the results published for some Lawrence's benchmark problems. A considerable number of recently published papers address real-life scheduling cases. Herrmann *et al.* (1995) described the development of a global scheduling system for a semiconductor test area. Niemeyer and Shiroma (1996) used EAs for the scheduling of factories of a multinational company. Hamada *et al.* (1995) approached a complex scheduling problem in a steel making company using a hybrid system based on EAs and expert systems. Finally, Shaw and Fleming (1997) and Kumar and Srinivasan (1996) proposed evolutionary computation methods for the solution of scheduling problems in companies that produce ready chill meals and defence products respectively.

PAPERS	*FT 6X6*		*FT 10X10*		*FT 20X5*	
	Best	*Aver.*	*Best*	*Aver.*	*Best.*	*Aver.*
Aizpuru *et al.* (1995)	-	-	930	951	-	-
Cao *et al.* (1997)	-	-	945	953.5	1176	1198.3
Cho *et al.* (1996)	55	-	943	-	-	-
Croce *et al.* (1995)	55	55	946	965.2	1178	1199
Dorndorf *et al.* (1995)	55	-	938	-	1178	-
Gen *et al.* (1994)	55	-	962	-	1175	-
Gohtoh *et al.* (1996)	-	-	930	935.36	1165	1180.34
Kim *et al.* (1995)	-	-	930	931.57	1165	1165.97
Kim *et al.* (1996)	-	-	930	930	1165	1165.27
Kobayashi *et al.* (1995)	-	-	930	934.3	1165	1217.4
Ono *et al.* (1996)	-	-	930	931.1	1165	1176.5
Park *et al.* (1995a)	-	-	936	949	1178	1185
Shi *et al.* (1997)	-	-	930	946.2	1165	-
Yamada *et al.* (1995)	-	-	930	934.5	1165	1177.3

Table 2.1: Published results (makespan) on Fisher & Thompson's benchmark problems. Optimal values: FT 6X6: 55, FT 10X10: 930, FT 20X5: 1165

	Aizpuru et al. (1995)		*Cao et al. (1997)*		*Kim et al. (1995)*		*Croce et al. (1995)*		*Park et al. (1995b)*	
TEST NO.	Best	Aver.	Best	Aver.	Best	Aver.	Best	Aver.	Best	Aver.
LA01			666*	666	666*	666	666*	666		
LA06			926*	926	926*	926	926*	926		
LA11			1222*	1222	1222*	1222	1222*	1222		
LA16			956	980	945*	945.4	979	989		
LA21	1056	-	1061	1083.6	1055	1055.8	1097	1113.6		
LA22					935	935.47			935	949
LA26			1227	1231.2	1218*	1218	1231	1248		
LA27	1255	-			1255	1264.9				
LA31			1784*	1784	1784*	1784	1784*	1784		
LA36			1337	1348			1305	1330.4		

Table 2.2: Published results on Lawrence's benchmark problems (makespan).(*) denotes optimal value

2.3 The Flowshop Scheduling Problem

2.3.1 Introduction

The permutation flowshop scheduling problem, or the job sequencing problem as it is often called, is another manufacturing optimisation problem that attracts particular research interest. It is relatively easy to apply evolutionary computation methods to this problem, since it can be formulated as a classic travelling salesman problem (TSP) with path representation (Michalewicz, 1992). This latter problem has been a subject of research from the early days of evolutionary computation. As a result, the efficient operators that have been developed for the travelling salesman problem are directly applicable to the flowshop scheduling problem.

2.3.2 Problem formulation

The permutation flowshop scheduling problem involves ordering *n* jobs to be processed in *m* machines. The difference between the job shop and the flowshop scheduling problem is that in the latter case each job undergoes the same machining sequence, and the sequence of operations is the same on each machine. This means that the solution of the problem can be represented as a permutation of all jobs to be processed:

$$[\, J_1, J_2, J_3, \ldots\ldots. J_n \,]$$

where *n* is the total number of jobs. The conditions that were introduced for the JSSP hold for the flow shop scheduling problem as well. The minimisation of makespan is usually employed as the objective of the scheduling algorithm (Braglia and Gentili, 1994), (Reeves, 1995), (Chen *et al.*, 1996a), (Murata *et al.*, 1996a). In the special case of m=1, the problem is described as the one-machine scheduling problem. Garey *et al.* (1976) have shown that the flowshop scheduling problem is NP-hard in the strong sense (proof by transformation of the 3-PARTITION problem to the associated flowshop scheduling decision problem).

In recent years, more complicated formulations of the problem have been considered with various alternative optimisation criteria included. Murata *et al.* (1996b) used their multi-objective GA approach for a flowshop scheduling problem, aiming to simultaneously minimise makespan, total tardiness, and total flowtime of the production. Minimisation of total tardiness was also employed as an optimisation criterion by Lam *et al.* (1996). Sikora (1996) attempted to minimise makespan, holding costs (earliness), and overtime (tardiness), in a flow-line with limited buffer capacity. Sannomiya and Iima (1995) also tried to minimise makespan, keeping at the same time the processing rate of each product as constant as possible. Their formulation of the problem considered the existence of a carrier that transferred products between the machines. Lee and Choi (1995) assigned earliness and tardiness penalty weights to schedules, for a one-machine scheduling problem. Lee *et al.* (1997b) presented an interesting formulation of the problem, introducing the concept of a flexible flow line with variable lot sizes. In this case jobs consisted of lots that could be split and an efficient EA was used to simultaneously optimise the ordering of jobs and the lot sizing. Gonzalez *et al.* (1995) considered the 'no-wait' version of the job sequencing problem, where once the processing of a job has started in the first machine of the production line, there must be no time delay between the consequent operations of the job at the following machines. An EA combined with heuristic methods was employed for the solution of the problem. Herrman and Lee (1995) described a class one-machine scheduling problem where jobs belonged to different classes, with each class having sequence-dependent set-up times. Their evolutionary algorithm generated different input conditions for a minimum waste heuristic algorithm, which accomplished the task of producing legal schedules. Finally, Karabati and Kouvelis (1997) addressed the flowshop scheduling problem with controllable processing times, i.e. the problem where the processing time of a part is not fixed, but can assume a number of different values. An EA was employed for the solution of large-scale problems of this type.

2.3.3 Encoding

The permutation representation is used in the majority of evolutionary computation approaches discussed in this section. A permutation is a natural representation for the solution of the problem, since there are many well-tested operators to ensure the feasibility of solutions and to enhance the evolutionary process.

There are, however, some exceptions to this rule. The most notable is that of Lam *et al.* (1996) who introduced a pigeon hole coding scheme. In this representation, the value of each gene corresponded to the index of job selected for scheduling, out of the list of unscheduled jobs. Each time a job was scheduled, the list of unscheduled jobs was re-indexed, and the value of the next gene defined the job selected out of the new set. Their representation allowed the use of traditional crossover and mutation operators without producing infeasible solutions. Some slight modifications in the encoding of solutions were also present in Sikora (1996) and Lee *et al.* (1997b), in order to accommodate the simultaneous lot sizing that was attempted by these algorithms. Finally, Kebbe *et al.* (1996) adopted the vibrational-potential method (VPM) for the solution of sequencing problems. VPM is an evolutionary computation method based on the concept of information propagation in nature, which employs different representation schemes.

2.3.4 Test problems and case studies

It is extremely difficult to compare the performance of different evolutionary algorithms in flowshop scheduling problems, since most researchers use their own instances of randomly generated test problems, i.e. problems where the processing times and due dates of jobs are selected randomly out of a uniform distribution. A comparison of results taken from this type of problems would not be valid.

Most of the papers referenced in this section use their own problem instances, or test problems not widely available. The only exceptions are Reeves (1995), and Ross and Tuson (1997) who presented results on standard benchmark problems taken from Tailard (1993). It is interesting to note that Lee *et al.* (1997b) and Sikora (1996) considered the scheduling of a manufacturing plant producing Printed Circuit Boards (PCB's) as a case study.

2.4 The Dynamic Scheduling Problem

2.4.1 Introduction

The cases discussed so far in job shop and flow shop scheduling were addressing static scheduling problems, i.e. problems where the dynamic nature of the scheduling decision is not examined. However, in practice, a scheduler often has to react to unexpected events. The main uncertainties encountered in a real manufacturing system, are the following:

- machine breakdowns including uncertain repair times
- increased priority of jobs
- change in due dates
- order cancellations

Whenever an unexpected event happens in a manufacturing plant, a scheduling decision must be made in real-time about the possible reordering of jobs. This process is known as 'rescheduling'. The main objective of rescheduling is "to find immediate solutions to problems resulting from disturbances in the production system" (Jain and Elmaraghy, 1997).

Until recently, evolutionary computation methods have rarely been used for dynamic scheduling, due to their inability to cope with real-time decision making. They were developed and tested on static scheduling problems that did not require real-time control. However, the last few years, EAs have been employed as parts of hybrid dynamic scheduling systems, which exploit their useful characteristics.

2.4.2 Machine learning methods

Machine learning is one of the methods that have traditionally been used in manufacturing environments to face uncertainties. Chiu and Yih (1995) proposed such a learning-based methodology for dynamic scheduling. They divided the scheduling process in a series of ordered scheduling points. An evolutionary algorithm examined which dispatching rules performed better for each of these points, given a set of plant conditions (system status). The chromosome was formed by a series of genes, each one representing a respective scheduling point and taking as a value one of the available dispatching rules. The performance of the algorithm was simulated under different plant conditions, forming a knowledge base that described the scheduling rules that were preferable in different cases. A binary decision tree was used to describe the gained knowledge. This method had the advantage of being able to modify its existing knowledge (new system conditions), without having to reconstruct the entire knowledge-base. Aytug *et al.* (1994) presented a different machine learning approach for dynamic scheduling, based on classifier systems (Booker *et al.*, 1989). In this case, an initial knowledge base was given, and an EA modified it, using results taken from the simulation of the production line. In that way the system learned to react to certain unexpected events. Jones *et al.* (1995) used a hybrid system based on neural networks, EAs, and an inductive learning algorithm to infer knowledge about the scheduling process. A back-propagation neural network selected a number of candidate dispatching rules out of a larger set of available rules. The schedules formed by these dispatching rules were used as the initial population of an EA that evolved an optimal schedule. The results taken from the simulation of the schedule helped the learning algorithm to create a set of rules that formed the knowledge - base. Lee *et al.* (1997a) also proposed a hybrid scheduling framework that consisted of an inductive learning system for job releasing in the plant, and an EA-based system for the dispatching of jobs at the machines.

2.4.3 Alternative methods

Fang and Xi (1997) presented a different rescheduling strategy based of the rolling horizon optimisation method. Scheduling was performed periodically on a predefined number of jobs that formed the 'job-window'. Rescheduling was initiated either by the elapse of a job-window or by the occurrence of an unexpected event. An EA evolved an optimal schedule for each planning horizon, considering the status of the system. Cartwright and Tuson (1994) employed the same concept of job-windows, in their attempt to dynamically control the scheduling of a chemical flowshop using an EA. Bierwirth *et al.* (1995) proposed a similar approach aiming to decompose a non-deterministic job-shop problem in a series of deterministic smaller ones. Each subproblem was then solved with the help of the static scheduling EA method that was described in the JSSP section (Bierwirth *et al.*, 1996).

Finally, Jain and Elmaraghy (1997) presented a steady-state EA-framework for the scheduling of an FMS system. Specially designed algorithms dealt with unexpected events like machine breakdowns and order cancellations. A series of test cases indicated the validity of the method for scheduling and rescheduling purposes.

2.5 Process Planning

2.5.1 Introduction

Process planning is one of the most complex phases in the manufacturing process. It comprises of a series of tasks that are heavily dependent on the type of product that is to be processed. Process planning takes as input the design characteristics of a product and gives as output its complete production plan. This plan determines the machining processes needed, the tools that are going to be used, and the sequencing of operations. If alternative plans exist, an optimal process plan should be selected in relation to the optimisation objective(s). Process planning can be fine-grained or course-grained, according to the processing requirements of a particular part.

Process planning is the link between the design and the manufacturing phase of a product. The design phase is highly automated nowadays with the introduction of state-of-the-art Computer Aided Design (CAD) programs. Research interest in the field of Computer Aided Process Planning (CAPP) is growing rapidly.

2.5.2 Operation sequencing

Operation sequencing is an important task of process planning. The planner must determine the machining sequence of parts, taking in account all the existing precedence constraints for the machining of features. These constraints are normally given in the form of a precedence graph. Usher and Bowden (1996) proposed an evolutionary computation approach for the solution of this problem, where the number of genes in the chromosome was equal to the number of features that must be machined. The authors introduced a special decoding procedure based on the feature precedence graph, which transformed any string into a feasible sequence of machining operations. This type of representation was first introduced by Yip-Hoi and Dutta (1996). The total number of set-ups, the continuity of motion and the loose precedence determined the quality of solutions. Takatori *et al.* (1994) adopted a TSP representation for the solution of the same problem, using a repair mechanism to cope with solutions that violated the constraints. The objectives of their algorithm were the minimisation of the total change cost, the machining cost and the non-machining cost.

Kamhawi *et al.* (1996) developed an elaborate feature sequencing system based on

EAs. The representation scheme was the same as the one used by Takatori *et al.* (1994), but the evaluation of solutions was based on rules and constraints about safety, quality, and minimisation of tool changes and tool travel. The user assigned a weight to each of these objectives, according to his preferences.

Norman and Bean (1997) discussed the problem of operation sequencing and tool allocation in Parallel Machine Tools (PMT's). A PMT is a machine capable of processing more than one part at a time, since it contains multiple spindles. A random-keys coded EA was proposed for the solution of the problem. The tool allocation task was dealt with the introduction of an integer part to the value of the genes. This part defined the machining unit (MU) that was responsible for a particular operation. The decimal part of the value determined the sequence of operations. The authors also proposed the enhancement of the algorithm with a heuristic method, presenting results that justified their decision. Yip-Hoi and Dutta (1996) tackled the same problem using an efficient solution representation based on feature precedence graphs, as was discussed earlier. The objective of their algorithm was the minimisation of the part's total processing time.

2.5.3 The process planning selection problem

The process planning selection problem is the task of selecting an optimal process plan out of a population of alternative plans. The problem is usually modelled with the help of flow-networks, i.e. constructions of arcs and nodes that determine alternative sequences of machining for a given product (figure 2.1).

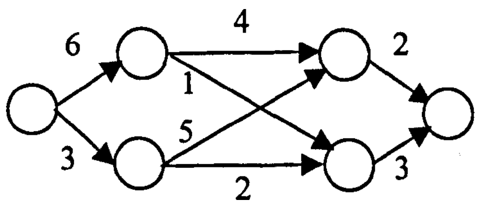


Figure 2.1: Flow network representation of the process planning selection problem

Each stage of this graph represents a machining operation and the nodes denote the number of alternative machines that are capable of performing this operation. The weighted arcs define the cost of following a particular machining sequence.

Awadh *et al.* (1995) presented one of the first evolutionary algorithms for the solution of the process planning selection problem. Each stage of a process plan was represented by a binary-coded matrix, where the occurrence of a bit with positive value denoted the presence of a connection between the corresponding nodes of the matrix. The authors warned that this representation could sometimes lead to the existence of multiple processing plans for a single chromosome solution. A decoding algorithm called 'Path Modifier' ensured that there was a '1 to 1' relationship between the genotype and the phenotype of each solution. The objective of their approach was the minimisation of the overall cost. Zhou and Gen (1997) noted that fast and efficient algorithms like the shortest path method and dynamic programming were capable of producing good solutions for single-objective process planning problems like the previous one. They argued that evolutionary computation methods would be ideal for the multiobjective version of the problem, which cannot be easily formulated as a shortest path or dynamic programming problem. They constructed an EA that used the same network flow model but employed an efficient integer solution representation that did not require the existence of additional operators like 'Path Modifier'. A non-aggregating approach facilitated the solution of the multiobjective version of the problem.

2.5.4 Advanced process planning methodologies

Concurrent Engineering (Singh, 1996) has received a lot of attention lately, as a modern approach to manufacturing optimisation. It is a manufacturing philosophy where the design and the related manufacturing processes of a product are integrated into one procedure. Process planning and scheduling are closely related manufacturing processes. One of the aspects of concurrent engineering is the integrated process planning (in terms of the optimal selection of a process plan) and scheduling of a product. McIlhaga *et al.* (1996) proposed an EA-based method for the simultaneous determination of planning and scheduling in a vehicle manufacturing company. They used parallel genetic algorithms with a diploid chromosome representation, which defined both the sequencing of operations and the use of alternative machines. A number of different optimisation objectives were considered by the algorithm, like the minimisation of makespan, flowtime and tardiness.

Bowden and Bullington (1996) created a hybrid system called GUARDS, based on unsupervised machine learning and EAs, in order to optimise the control of a manufacturing process. The system learned to select the optimal process plan according to the status of the plant. Horvath *et al.* (1996) described a complete process planning procedure, from the input of part specifications in the form of CAD files, to the optimisation of the constructed process plan. They used an object-oriented approach in the form of 'features'. A 'feature' was an object that defined specific operations and contained all the relative functional, geometrical and technological data. Knowledge - based reasoning was used for the generation of plans, which were subsequently optimised with the help of an evolutionary algorithm. Zhang *et al.* (1998) developed a similar complete CAPP system for parts manufactured in job shop environments. They adopted a direct solution representation originally introduced by Burns (1993). Each chromosome defined the sequencing of operations, machine-tool assignments and Tool Approach Directions (TAD's) for an individual process plan. In this way, the procedures of operation sequencing and process planning selection were integrated.

Dereli and Filiz (1999) indicated that process planning methodologies usually addressed specific parts of the overall planning procedure, making their integration with existing CAD and CAPP packages a difficult task. They introduced a framework for the process planning optimisation of prismatic parts that comprised of three evolutionary algorithms. Each of these algorithms was responsible for the optimisation of a specific planning problem, namely operation sequencing, tool-magazine positioning and cutting parameter selection. The advantage of the proposed methodology was its ability to be used both off-line and in combination with CAD and CAPP packages, giving as output the ready-to-use process plan of the part considered.

2.6 Cellular Manufacturing

2.6.1 Introduction

Cellular Manufacturing is the application of Group Technology (GT) in manufacturing systems. GT was first introduced in the former USSR by Mitrofanov (1966), and was popularised in the west by Burbidge (1975), who introduced Production Flow Analysis (PFA), the first scientific method for creating manufacturing cells. Cellular manufacturing is a philosophy that attempts to convert a manufacturing system into a number of cells. Each cell manufactures products with similar processing characteristics. Ideally, all the processing operations of a part should be completed within a cell. However, in realistic cases, intercell movements of parts are always present. Cellular manufacturing offers certain advantages to mid-variety, mid-volume production lines like the reduction of set-up and transfer costs, the minimisation of inventory, improved quality and significant savings in plant space.

There are three main phases in the design of a manufacturing cell: i) the grouping of machines into cells, better known as the cell-formation problem, ii) the layout of cells in the plant and iii) the layout of machines within the cells. The implementation of each of these stages is associated with difficult optimisation problems, where traditional optimisation methods are incapable of finding optimal solutions in reasonable time. In the following paragraphs some evolutionary methods that have recently been used to tackle optimisation problems associated with cellular manufacturing will be examined.

2.6.2 The cell-formation problem

Venugopal and Narendran (1992) were the first researchers to approach the cell-formation problem using EAs. Their objective was the minimisation of the intercell traffic and the balancing of load in the cells. A different population of solutions was employed for each of these objectives. The solution representation was simple and efficient. Each machine in the plant corresponded to a gene in the chromosome. The value of the gene defined the cell of the respective machine. The total number of cells in the plant was predefined, but the formulation of the problem also considered the processing time of parts, which was an additional feature in relation to conventional cell-formation methods. Gupta *et al.* (1996) enhanced this formulation by considering the intracell moves of parts and the intracell layout. Special care was taken to ensure that no cell remained empty during the evolutionary process.

Billo *et al.* (1996) adopted a direct solution representation, based on a two-part chromosome. The first part was a permutation of all parts to be processed, while the second part denoted the cut-off points of the first part. Each segment between cut-off points denoted a part-family. The objective of their algorithm was the maximisation of machines' similarity within the cells and the minimisation of the total number of cells. The advantage of this method was that the total number of cells was not predefined, however, the structure of the chromosomes was quite complex and computationally expensive. The algorithm performed well on a series of test problems, including some ill-structured machine-component matrices.

Joines *et al.* (1996) introduced a new efficient integer programming formulation of the problem, which reduced the search space significantly. An evolutionary algorithm was employed for the solution of the problem, with the variables of the mathematical formulation coded into the chromosome. Only the upper bound of the total number of cells needed to be specified. The objective of the algorithm was the minimisation of exceptional elements and voids (zero's in the diagonal blocks) in the machine-component matrix. The validity of the method was depicted by results on test problems taken from the literature. Su and Hsu (1996) used the classic Venugopal's solution representation, but their chromosome also accommodated the existence of multiple machines of the same type.

Cheng et al. (1998) noted that the reorganisation of rows and columns in a binary m/c matrix could be described as a permutation problem equivalent to the TSP, where the objective was the minimisation of some type of distance measure between columns or rows. They employed a real-coded EA with path-representation for the solution of the problem. The Minkowski metric was used as an indication of the distance between a pair of machines or parts. The performance of the algorithm was compared with that of a non-hierarchical clustering solution methodology (ZODIAC) (Chandrasekharan and Rajagopalan, 1987), on a wide range of problems taken from the literature and was found to be superior in most cases.

Gravel et al. (1998) considered a version of the cell-formation problem that allows the existence of alternative process plans for the parts. A double-loop EA was employed for the solution of the problem with the objective of minimising the volume of intercell moves and balancing the workload within cells. The external loop of the EA used Venugopal and Narendran's coding for the assignment of machine to cells. A second internal loop that determined the allocation of process plans to parts was used for the evaluation of solutions created in the external loop. Different multiobjective optimisation approaches were tested, including the epsilon-constraint approach and the weighted-sum approach.

2.6.3 Cell layout and machine layout optimisation methods

Once the configuration of cells has been determined, the designer must define the layout of machines inside the cells, and the layout of cells in the plant area. These optimisation problems belong to the general category of the Facility Layout Problem (FLP). The FLP is a well-known combinatorial problem. It has been formulated as a quadratic set covering problem, linear integer programming problem, mixed integer programming problem and graph theoretic problem. However, the Quadratic Assignment Problem (QAP) formulation is the most popular in the literature and since QAP is known to be NP-complete for most problem instances, efficient algorithms must be used for its solution.

2.6.3.1 Evolutionary computation methods for the solution of the facility layout problem

Several researchers have used evolutionary algorithms to tackle FLP problems in manufacturing. Cohoon *et al.* (1991) and Tam (1992) were the first researchers to approach the problem using evolutionary computation methods. In both cases, the layout was represented by a Slicing Tree Structure (STS) that can be easily decoded into a layout. A slicing tree is "a binary tree representing the recursive partitioning process of a rectangular area, through cuts. A cut specifies the relative position of departments through four distinguished branching operators" (Mavridou and Pardalos, 1997). Kado *et al.* (1995) investigated the combination of STSs with different clustering methods for the initialisation of the population, and different decoding methods for the creation of layout. Some of these combinations produced improved results on previously published test problems. Garces-Perez *et al.* (1996) refined these results by putting the slicing tree structures into a much more natural genetic programming framework, and by employing a variation of one of Kado's most successful decoding methods. The STS representation was also adopted by Cheng *et al.* (1995) in their EA framework. The authors additionally addressed the issue of the uncertainty of material flow between cells using a convex fuzzy number representation. Gau and Meller (1999) illustrated the deficiencies of the STS representation and proposed a number of modifications in order to improve the search power of the evolutionary algorithm. Their methodology allowed both the exchange of departments within the same tree structure and the change of structure of STSs by the introduction of 'dummy' departments. The methodology was tested on several test problems taken from the literature, producing satisfactory results.

Tate and Smith (1995) adopted the QAP formulation of the problem with the objective of minimising the sum of products of total material flow and rectilinear distances between the departments. They proposed a flexible-bay layout structure that accommodated unequal sizes for the departments. The plant was initially divided into a number of bays by end-to-end slices in one direction, which were subsequently split into departments by perpendicular slices. A permutation representation of the solution was used, which determined both the allocation of departments in the layout and the place of bay-divisions. Norman and Smith (1997) enhanced this representation by using a random-keys EA - thus avoiding feasibility constraints - and by incorporating uncertainty in the mathematical formulation of the problem. Material handling costs were expressed using expected values and standard deviations for the product volume over time. Suresh *et al.* (1995) adopted the permutation representation but used a much simpler grid-structure for the layout. Kazerooni *et al.* (1996) proposed an integrated approach for the design of manufacturing cells, which incorporated specific stages for the simultaneous determination of cell and machine layouts.

Banerjee *et al.* (1997) modelled the problem using a mixed-integer programming formulation. They proposed a graph solution representation based on nodes and edges. Nodes corresponded to input-output cell stations and edges corresponded to material flows between the stations. The layout structure was continuous; thus much more flexible than the grid and bay structures which restricted the shape of cells. Genetic search was employed as a part of the overall algorithm, aiming to transform the problem into a series of iterative linear programming problems. The robustness of this method was illustrated in a number of test cases taken from the literature, where it was shown to outperform traditional methods.

Conway and Venkataramanan (1994) considered an interesting version of the FLP, the dynamic FLP. In this case, the facility layout changes with time, and the algorithm must find the best allocation of facilities over an entire planning horizon. The authors introduced a multi-part chromosome representation for the layout, where each part corresponded to a planning period. The position of a gene corresponded to a fixed place in the layout, and the value of the gene denoted the facility that occupied this place for a particular period. The objective of the algorithm was the minimisation of layout rearrangements costs and materials flow costs over the entire planning horizon.

2.6.3.2 Special cases for the machine layout problem

The papers reviewed so far introduced methods that normally apply to the cell layout problem. The machine layout problem is a special type of FLP and it is usually addressed individually, since various assumptions that are made for the FLP are not valid for this problem, such as the equal-sized areas and the a-priori knowledge of facilities locations (Bazargan *et al.*, 1997).

Manufacturing practice usually restricts the search for an optimal intracell layout to a small number of fixed configurations, like the single-row layout, the multi-row layout, the semi-circled layout and the loop layout. Braglia and Sternieri (1996) utilised an EA in order to find the machine layout in a pre-fixed single-row structure. The objective of the algorithm was the minimisation of the distance travelled by the material-handling device of the cell. The solution was represented by a permutation of all machines in the row. This method performed well on large problem instances, in comparison with heuristic approaches. Cheng *et al.* (1996b) addressed the loop machine layout problem using two different objectives: the minimisation of the total number of reloads for all products (minsum problem) and the minimisation of the maximum number of reloads for all products (minmax problem). The layout was considered to be unidirectional and there was a single loading-unloading station. The solution was once again represented by a permutation of the available machines. Gen *et al.* (1995) introduced a hybrid fuzzy-GA approach for the solution of complex multi-row machine layout problems. The objective of the algorithm was the minimisation of travel cost between the machines, and the solution was represented by a multi-part chromosome that contained information about the total number of rows, the permutation of machines in each row, and the clearances between the machines. Fuzzy sets were used for the representation of the uncertainty that existed in the value of clearances.

2.7 Optimisation of Assembly Lines

2.7.1 Introduction

Assembly lines are widespread in manufacturing plants. A number of optimisation problems are associated with assembly lines, like the assembly sequence planning problem, the sequencing of mixed model assembly lines and the assembly line balancing problem. A variety of evolutionary computation methods have been proposed for the solution of assembly line optimisation problems.

2.7.2 The assembly sequence planning problem

The Assembly Sequence Planning Problem (ASSP) is the problem of finding an optimal sequence of assembling a product that consists of N parts, given its design characteristics. An assembly sequence is feasible if it does not violate the assembly rules and constraints that are defined by the designer. Sebaaly and Fujimoto (1996) proposed an evolutionary approach for the solution of this problem, where an individual chromosome was a randomly constructed sequence of parts. An efficient mapping procedure transformed any random assembly sequence into a feasible one. Gropetti and Muscia (1995) analysed the assembly planning procedure and used an EA in order to obtain a clear contact relational graph.

2.7.3 Sequencing in mixed model assembly lines

It is often the case that several products with similar characteristics (models) are assembled in a single line (mixed-model assembly lines). The sequencing of models in mixed-model assembly lines is an important task, especially if the JIT principle is to be applied in the production line. There are a number of objectives associated with this task, like the minimisation of line's length, the minimisation of total utility work, and the minimisation of the variability of parts' consumption (vpc). This latter objective is critical in JIT systems. Leu *et al.* (1996) addressed the problem of sequencing a mixed-model assembly line with the objective of minimising vpc in a JIT production system. An EA was used for the solution of the problem, with each chromosome representing a sequence of models to be assembled. The sequence was cyclic, and the number of individual models in each sequence was fixed. This method outperformed Toyota's Goal Chasing Algorithm (GCA), which is often used in JIT production systems, in a number of test problems. Kim *et al.* (1996) adopted the same representation for the sequencing of a mixed model assembly line, where the objective was the minimisation of the total length of the line.

2.7.4 The assembly line balancing problem

Another well-known assembly line optimisation problem is the assembly line balancing problem. Given *n* workstations and *m* parts to be assembled, the assignment of parts to workstations should be defined according to certain optimisation criteria. Two versions of the problem are usually considered: the first version aims to minimise the total number of workstations in the plant given a fixed cycle time, while the second version aims to minimise the cycle time, given a fixed number of workstations. Secondary objectives like the minimisation of balance delay and the minimisation of probability of line stoppage are also considered. Suresh *et al.* (1996) presented an excellent literature review on the assembly line balancing problem and proposed an evolutionary algorithm for the solution of a similar problem where the objective was mainly the minimisation of the smoothness index of balance delay. The solution was represented by a list of sets with length equal to the total number of workstations. Each set contained one or more processing jobs. All the initial solutions were feasible and special operators ensured the feasibility of solutions throughout the evolutionary procedure. The authors also presented an alternative version of the algorithm, where a number of infeasible solutions were allowed in the population. The latter version performed well on large problem instances. Rubinovitz and Levitin's (1995) representation was a permutation of all parts, divided into a number of sections equal to the total number of workstations. Random sequences were initially constructed, and special mechanisms were employed to reorder them according to the precedence constraints and to divide them in an appropriate number of sections. Tsujimura *et al.* (1995) presented an interesting EA-fuzzy logic method for the solution of the assembly line balancing problem, aiming to minimise the balance delay. A conventional permutation representation was employed, which considered all precedence constraints. The processing time of each job was not deterministic, but was defined by a fuzzy set. The allocation of jobs to workstations was accomplished using the EA sequence, the fuzzy sets, and a standard predefined maximum completion time: starting with the first job in the sequence, the fuzzy sets of processing times were added, until the upper limit of the sum of fuzzy sets became bigger than the predefined maximum completion time. The set of jobs that comprised the sum was assigned to the first workstation, and the procedure started again from the next job after this set in the sequence. Special mechanisms and operators ensured the feasibility of solutions.

2.8 Manufacturing-related Optimisation Problems

2.8.1 Introduction

In the previous sections recent papers in the field of evolutionary computation for some standard manufacturing optimisation problems were reviewed. However, these are not the only optimisation problems associated with the manufacturing process. The purpose of this section is to illustrate some recent evolutionary computation approaches in various manufacturing areas.

2.8.2 Design Optimisation Problems

Design is a complicated and time-consuming phase in the development of a product. Every design must be properly optimised, otherwise the result will be huge redesign costs. Enormous effort has been devoted to the development of efficient CAD systems in order to simplify and speed up the design process. Evolutionary computation methods have been applied successfully to complex design optimisation problems.

Cao and Wu (1997) adopted an evolutionary programming approach for the solution of a mechanical design optimisation problem: a number of design variables needed to be optimised, subject to certain constraints. Continuous, binary, integer and discrete variables were included in the mathematical model, a condition that made the optimisation procedure even harder. The solution was represented by a string of design variables initialised within the constraints, while a special mutation procedure was used for each type of variable. Two design problems were used to illustrate the method, the design of a gear train and the design of a pressure vessel. The algorithm performed equally well or better in comparison with other optimisation methods like the branch & bound algorithm and simulated annealing.

Rasheed *et al.* (1997) proposed an EA for the solution of a similar parameter optimisation problem that involved only continuous variables. The solution was a string of all parameters that needed to be optimised, initialised within their feasible regions. Feasibility problems were accommodated using a penalty function. The evolutionary process was enhanced with the introduction of two crossover operators namely line crossover and guided crossover, which produced an offspring on the line connecting the parent chromosomes, considering the solutions' search space. The algorithm was tested on two complex design optimisation problems, the design of a supersonic transport aircraft and the design of a supersonic missile inlet. The method performed much better on these problems than a classic binary-coded GA and a sequential quadratic programming method.

A discussion about the use of evolutionary computation techniques in the wider field of engineering design can be found in (Parmee, 1998).

2.8.3 Process model identification

The identification of process models is essential for the optimal control of manufacturing systems. Pohlheim and Marenback (1996) used genetic programming in order to identify the model of a manufacturing process. Common control engineering tools, like transfer function blocks were used for the creation of trees (programmes). In this way, the algorithm provided structured process models, giving the control engineer a useful insight on systems' internal configuration. Test problems validated the performance of the method and especially its ability to generalise. McCay *et al.* (1996) also employed genetic programming for system identification, constructing the trees with common mathematical functions. Reeves *et al.* (1996) proposed an interesting EA methodology where the solution was coded in terms of the radii and angles of poles and zeros of the transfer function. The values of these variables were constrained within the stability regions; thus the final solution was guaranteed to be stable.

2.8.4 Machine failure and maintenance

Some failure of machines in the plant is often inevitable. Shop-floor engineers aim to diagnose the failure of a machine as quickly as possible. They normally use a number of symptom parameters that are sensitive to changes of specific signals from the plant. Chen *et al.* (1996b) described some of these parameters and proposed an evolutionary approach for the determination of an optimal sequence of symptom parameters. Their method resembled genetic programming, in terms of the tree structures that were used as individual chromosomes. Guzman and Kramer (1994) developed a hybrid Bayesian networks - EAs system that performed on-line monitoring and failure diagnosis, based on data taken from the plant.

Maintenance scheduling is another important task in the shop-floor, since the disruption of the production process must be as little as possible, but on the same time the machines must work without failures for the longest time possible. Kim *et al.* (1994) proposed an interesting hybrid of EAs and simulated annealing for optimal maintenance scheduling. The acceptance probability of simulated annealing was used for the survival of the less fit offspring in the population.

2.8.5 Quality control

Quality control is an important aspect of modern manufacturing. The optimal allocation of inspection stations in the plant ensures that products are manufactured according to the quality criteria set by the management team. Viswanadham *et al.* (1996) addressed this problem in a multi-stage manufacturing system and employed an evolutionary algorithm to optimally allocate inspection stations. The solution was binary coded, with each gene representing a manufacturing stage. The presence of a station at a particular stage was denoted by a positive value. Patro and Kolarik (1997) designed a system that performed statistical processing control using neural networks and evolutionary computation. The neural network identified the process model, and the evolutionary algorithm adjusted the control parameters in order to obtain the desired quality performance. Lu *et al.* (1995) presented an EA-based system that optimised the motion of a co-ordinate measuring machine used in inspection systems. A permutation representation was employed for the solution of the problem, with each gene corresponding to a testing point that the measuring machine should visit. The algorithm aimed to find the optimal sequence of visiting points that minimised the total length of the inspection path.

2.8.6 Advanced manufacturing optimisation problems

In the following paragraphs, some advanced manufacturing optimisation problems that have been the subject of evolutionary computation research will be discussed. Mak and Wong (1995) considered the problem of designing an optimal integrated production-inventory-distribution system, aiming to minimise the overall costs, including inventory holding costs, delivery costs, manufacturing costs and shortage costs. An evolutionary algorithm was employed for the solution of the problem. An integer programming formulation of the problem was adopted, and the solution was represented using the variables of the model. Disney *et al.* (1997) addressed the problem of controlling a production and inventory system. Transfer functions were used for the modelling of the problem, illustrated in the form of block diagrams. The solution of the problem was represented by the variables of the transfer function, and a fitness measure was designed based on stock reduction, production robustness and inventory recovery.

A difficult decision that the marketing team often has to face is the location of inventory centres for the accommodation of department stores, and the allocation of an inventory centre to each of these stores. This difficult location-allocation problem was formulated as a non-linear mixed-integer programming problem and solved by Gong *et al.* (1996) using an evolutionary approach for the location task, and a Lagrangian relaxation method for the allocation task.

Aggregate production planning is a high level decision making procedure that takes as input product capacities and forecast demands, and produces aggregate production plans. Stockton and Quinn (1995) addressed this problem using a binary-coded GA. The algorithm determined the amount of resources needed each month in order to meet the demand. The resources were expressed in the form of overtime, subcontracts and stock. Wang and Fang (1997) formulated the same problem using a fuzzy linear programming model. They employed Zimmerman's tolerance approach to transform the problem into a linear programming model. The variables of the model formed the chromosome of an evolutionary algorithm that was used for the solution of the problem. Feng *et al.* (1997) addressed the problem of joint marketing/production decision making aiming to maximise the net profit of a company. The decision problem consisted of the promotion problem for the marketing department and the production problem for the manufacturing department. Each problem was formulated mathematically and a respective number of EAs were employed for their solution. The decision variables of the mathematical models were used for the representation of solutions. Garavelli *et al.* (1996) considered the production planning problem of a multinational company with multiple manufacturing plants around the world. Parameters like local market demands and independent capacities were taken in account in the formulation of the problem. An EA defined which plants would be activated for production and the timing of their activation.

The dynamic lot-sizing problem in a multi-stage, multi-item production system was described by Jinxing (1997). He proposed an evolutionary programming approach with binary representation for the solution of the problem. The objective was the minimisation of set-up, production and inventory costs.

2.8.7 Various applications

In a pull (JIT) production system, the demand must always be satisfied without the help of excessive stocks. The total number of kanbans in the plant and the corresponding production trigger values should be optimally defined in order to achieve this objective. Bowden *et al.* (1996) addressed this problem using an evolutionary algorithm seeded with the optimal solution of the Toyota equation. Zhao *et al.* (1996) addressed the problem of robot selection and workstation assignment in a Computer Integrated Manufacturing (CIM) system. A bin-packing formulation of the problem was proposed and an EA was employed for the solution of the problem. A diploid chromosome that accommodated both parts of the problem represented the solution. Finally, McIlhaga (1997) designed a framework for solving generic scheduling problems, i.e. scheduling problems of non-specific form. This framework was based on distributed genetic algorithms and was able to solve problems of this kind more efficiently than random search and dispatching rules. The parameters of the problem were defined by the user through a Scheduling Description Language (SDL).

2.9 Conclusions

It is obvious that the use of evolutionary computation methods for manufacturing optimisation is growing. The number of papers published is increasing rapidly, and research covers a wide range of manufacturing problems. The amount of work itself indicates that evolutionary computation methods have established themselves as a useful optimisation technique in the field.

Evolutionary computation research has been criticised for the consideration of artificial test problems that are much simpler than real-life manufacturing cases. This review shows that researchers have reacted to this criticism by considering realistic cases taken from manufacturing plants. This move has also been triggered by the low response of evolutionary computation in manufacturing practice. It is encouraging to report recent projects where companies have adopted evolutionary computation methods in their plants. The gap between academic research and manufacturing practice is not a problem restricted to the field of evolutionary computation. However, for the optimisation field considered in this thesis, there are a number of additional reasons that make this approach harder:

- The terminology of evolutionary computation is vague for the manufacturing engineer. Despite the fact that the concept of evolutionary algorithms is simple, the terminology inherited from genetics predisposes manufacturing engineers to think the opposite.
- Evolutionary computation is a relatively new technique, still in development. There are no universally accepted methods for the determination of technical parameters like population size, probability of applying operators etc. There is also no guarantee that an algorithm will converge to an optimal or near-optimal solution, except under specific problem conditions.
- There is no standard evolutionary computation toolkit that can be used easily by manufacturing people who are not familiar with evolutionary concepts.

Despite the previous considerations, evolutionary computation methods offer solutions that combine computational efficiency and good performance. This significant feature will certainly continue to attract the interest of engineers.

The robustness of evolutionary algorithms is greatly enhanced when they are hybridised with other optimisation methods like local search techniques, simulated annealing, tabu search, neural networks and expert systems. The number of papers introducing hybrid systems is growing, indicating that there is a trend towards this direction.

This review highlights the lack of genetic programming applications in the field of manufacturing optimisation. In the remainder of this thesis the possibility of using genetic programming for the solution of some standard manufacturing optimisation problems will be examined. An overview of the genetic programming optimisation method is presented in the following chapter.

Chapter 3

INTRODUCTION TO GENETIC PROGRAMMING

3.1 Introduction

During the last decade *genetic programming* has emerged as an efficient methodology for teaching computers how to program themselves.

Automatic programming algorithms have been proposed within the machine learning community from as early as the late 1950's (Friedberg, 1958). While a number of program induction techniques based on the principles of evolution were introduced during the 1980's (Cramer, 1985), (Fujiki and Dickinson, 1987), genetic programming was formally introduced by Koza (1992) as an extension of the popular genetic algorithm paradigm (Holland, 1975). The efficient integration of the evolutionary procedure with a simple scheme for the representation of computer programs proved to be successful in the solution of a number of alternative optimisation problems.

The original genetic programming algorithm was soon subject to proposed modifications both in its internal operations and the representation of genetic programs. Recently Banzhaf *et al.* (1998) argued that any search algorithm that employs a population of computer programs in its search and generates new variant programs using any form of non-deterministic procedure could be legitimately called genetic programming. This extension of the genetic programming definition was necessitated by the controversy caused within the evolutionary computation community about the optimality of the original genetic programming algorithm and more specifically the use of the crossover operator. This subject will be discussed in more detail in the following sections.

The focus of this thesis is the usefulness of the genetic programming model for the solution of manufacturing optimisation problems. The original form of the genetic programming algorithm, as introduced by Koza, was employed in the majority of the experiments.

The remainder of this chapter is organised as follows: In section 3.2 an overview of the genetic programming paradigm is presented. Section 3.3 deals with the representation, initialisation and computer implementation of the programs. The issues of fitness assignment and fitness-based selection are discussed in section 3.4. In section 3.5 the use of genetic operators and their implication on the theoretical foundations of genetic programming is examined. The preparatory steps for the run of the genetic programming algorithm are described in section 3.6. Section 3.7 describes the application of genetic programming on an example symbolic regression problem. The conclusions of this chapter are drawn in section 3.8.

3.2 Overview of genetic programming

Genetic programming is an evolutionary algorithm that employs the principle of Darwinian strife for survival for the creation of solutions in the form of computer programs. Koza (1992) claimed that there exists a large number of problems that can be re-formulated as program induction problems, i.e. their solution can be represented by a computer program that uses a number of problem-related input(s) to create the necessary output(s) (figure 3.1).

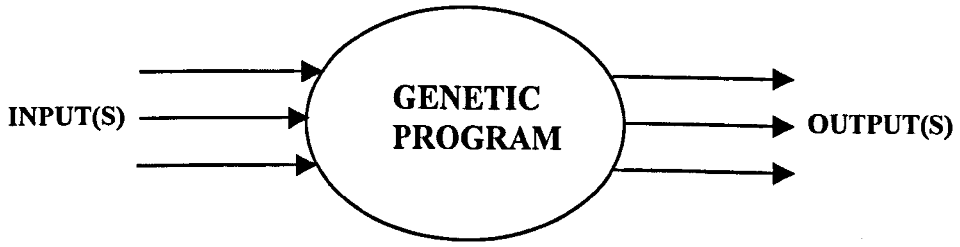


Figure 3.1: Genetic programming approach to problem-solving

Depending on the coding used for the representation of genetic programs (see section 3.3) and the particular optimisation problem considered, the search space for the optimal computer program can be quite substantial. Genetic programming employs the concept of genetic evolution to guide its search within the space of potential computer programs.

The operation of the genetic programming algorithm is presented in a pseudo-code form in figure 3.2.

```
Generate population of randomly created genetic programs
Evaluate performance of genetic programs on the problem considered
Assign fitness values to genetic programs
While termination criterion has not been satisfied
        While population size has not been exceeded
                Probabilistically select genetic operation (crossover, mutation, or
                        reproduction)
                Select genetic programs to participate in genetic operation based on
                        their fitness
                Perform genetic operation
                Insert offspring genetic programs in the new generation
        end
        Evaluate performance of genetic programs on the problem considered
        Assign fitness value to genetic programs
end
Report best solution found
```

Figure 3.2: Genetic programming algorithm in pseudo-code form

Initially, a population of genetic programs is randomly generated and their performance is evaluated on the solution of the problem considered. A fitness value is associated with each genetic program, as a measure of its quality in solving the problem. There is a possibility that the optimal solution of the problem will be generated form the initial population of programs. However, this rarely happens in practice, unless a problem is trivial or small-sized.

The next step of the genetic programming procedure is the selection of genetic programs that will form the new population of solutions. What makes the operation of genetic programming (and evolutionary computation algorithms in general) different to a random search procedure is that these programs are selected probabilistically according to their measured fitness. That is, genetic programming aims to exploit the programs that perform better on the solution of the problem. However, genetic programming is not a deterministic hill-climbing procedure, since the probabilistic selection step might favour the reproduction of a program that is not as fit as other individuals of the same population. In that way, genetic programming is able to escape from local-optima.

One of the necessary conditions for evolution to occur, both in nature and in algorithms that attempt to mimic its operation, is genetic diversity. While the computer programs of the initial generation are quite variable in their structure (in fact the designer can enforce them to be different from each other), it is certain that if only simple reproduction of fit programs was to be used as the method of generating the new population of solutions, the best programs would overwhelmingly dominate the population within few generations, and the search would stagnate (*premature convergence*). Instead, genetic programming, in accordance with all other evolutionary computation methods, employs specially designed genetic operators for the creation of genetic diversity. These operators are used to either force the exchange of genetic material between programs (crossover) or to introduce new pieces of genetic code in them (mutation). The operation of genetic operators and the controversy that surrounds their use is discussed in section 3.5.

The new population of solutions is constructed through simple reproduction and genetic modification. In both cases, probabilistic selection of solutions according to their measured fitness is required. Once the new population of solutions has been formed, their performance is evaluated once again and new fitness values are assigned to them.

The same procedure is repeated until a user-defined termination criterion has been reached. This criterion can be the generation of a program with fitness equal to the optimal value for the problem considered. However, since the optimal solution is not always known in advance, genetic programming runs normally terminate after a predefined number of generations. The end product of the evolutionary procedure would ideally be a computer program that is able to adequately solve the problem considered.

The form of the program structures initialised, evolved and translated by a genetic programming algorithm is discussed in the following section.

3.3 Representation, initialisation and computer implementation of genetic programs

Koza (1992) defined genetic programs as variable-length structures that can be compiled as they are, or with slight modifications by a computer. The shape and the size of these programs are not predefined and can change dynamically during the course of the evolutionary procedure.

3.3.1 Basic program elements

Genetic programs consist of a set of inputs that provide problem-specific information and a set of functions that manipulate these inputs. In genetic programming linguistics, the terms 'terminals' and 'functions' are used to describe these elements respectively.

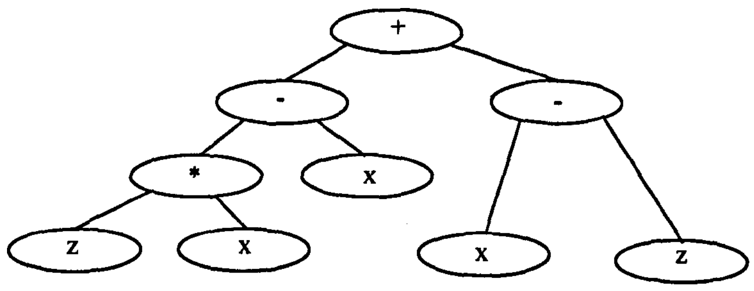
The terminal set may contain variables, constants, or functions that require no arguments. The function set may contain any function that the designer of the algorithm considers to be relevant to the solution of the problem. This definition does not only include standard functions (arithmetic, boolean, etc.) but user-defined functions as well, i.e. functions that have been constructed by the designer and have a specific meaning for the problem considered. This feature of genetic programming is a useful tool in the hands of the designer, especially when standard functions are unable to form a solution for the problem considered. Koza (1992) indicated that a function set consisting of the four basic arithmetic operations (addition, subtraction, multiplication, and division) could solve a considerable number of optimisation problems.

The choice of terminals and functions for the construction of the respective sets is not straightforward. The minimum requirement for the selected terminals and functions is that they should be able to create a solution for the problem considered (*sufficiency property*). A rule of thumb states that the size of these sets should be kept to the minimum possible. The evolutionary procedure has the ability to disregard any function that is irrelevant to the solution of the problem, however, when extraneous functions are included in the function set, the performance of the system is generally degraded. There are cases where the relevance of specific inputs to the solution of the problem is not known in advance. Statistical methods (correlation coefficients etc.) can provide some indication of their importance. However, in the overwhelming majority of genetic programming applications that have been reported in the literature, this decision is left to the evolutionary procedure (see for example Gilbert *et al.* (1998)).

Another consideration during the construction of the function and terminal sets is the maintenance of the *closure property*. This property states that each function included in the set should be able to accept as argument any value that might be generated by any other function or terminal in the system. The division function is a typical example of a potential violation of the closure property. If the denominator of the division function was to assume the value of '0', the computer would come to a halt (although some modern programming languages prevent this from happening using built-in protection procedures). In a genetic programming algorithm this situation is usually contained by employing the protected division function, which returns the value of '1', when the value of the denominator is '0'. Similar precautions must be taken for square-root and logarithmic functions, or functions that contain conditional branches.

3.3.2 Representation of genetic programs

The representation of genetic programs is mainly an issue of convention (Banzhaf *et al.*, 1998). The way that the designer chooses to represent and interpret genetic programs and the way that these programs are actually stored in the memory of the computer are not necessarily the same. The most popular form of representation for genetic programs is the parse-tree representation, originally suggested by Koza (1992) in his pioneering book. The intuition behind Koza's choice was that some computer languages use this type of representation to store and interpret programs. A parse-tree is a collection of terminal and function nodes interpreted in a depth-first, left-to-right postfix manner. This practically means that the interpretation of the program starts with the leftmost function for which all inputs are available. The interpretation is characterised as postfix since the operators appear after the operands. An example of a genetic program in a parse-tree form and its representation is presented in figure 3.3.



output = (z*x) – x + x - z

Figure 3.3: An example of a parse tree and its interpretation

While the parse-tree coding and representation of genetic programs has been dominant among GP researchers, it is by no means the only one that has been suggested. The register-based GP-machine of Nordin (1994) and the graph-based PADO GP-system (Teller and Veloso, 1996) are examples of alternative representations that have received considerable attention.

3.3.3 Initialisation of genetic programs

There are three main methods of initialising parse-tree genetic programs, all of them originally introduced by Koza (1992).

When the 'grow' initialisation method is used, genetic programs are constructed by randomly selecting functions and terminals from the union of the respective sets. The root of the tree is always selected from the function set, ensuring that a single-terminal genetic program is never created. Every time a function is selected, a corresponding number of arguments are selected randomly from the union set. If one of these arguments turns out to be a terminal, the subtree terminates at this point. If the selected argument is a function then the procedure continues in the same fashion. There is a user predefined maximum depth of genetic programs that should not be exceeded (where *depth* is defined as the maximum non-backtracking distance from an end node to the root of the tree). When the maximum depth level for a tree has been reached, selection of nodes is restricted to the terminal set only. Trees initialised using the 'grow' method are characterised by uneven depth since terminal nodes can be selected at any initialisation stage.

The 'full' method proceeds by restricting the choice of possible nodes, for depths less than the prespecified maximum depth, from the function set only. Maximum depth nodes can only be selected from the terminal set, as in the case of the 'grow' method.

The 'ramped half-and-half' method combines efficiently the characteristics of the 'grow' and 'full' methods constructing genetic programs that are structurally quite dissimilar. Initially, the population of genetic programs is divided into a number of depth levels according to the maximum depth constraint. If, for example, the maximum depth allowed for a genetic program is equal to 6, then the population is divided into equal sets of programs for each depth level between 2 and 6. For a population size of 500 programs, 100 programs are assigned to each of the levels 2, 3, 4, 5 and 6. Then, half of the programs in each set are initialised using the 'grow' method, while the remaining half is initialised using the 'full' method. The 'ramped half-and-half' method is used as the standard initialisation method in the genetic programming applications presented in this thesis.

3.3.4 Computer implementation of genetic programs

Koza employed the LISP programming language for the computer implementation of genetic programs, mainly due to its unique interpretation of computer code as both program code and data. However, several alternative computer implementations of the genetic programming algorithm have been proposed. The genetic programming system used in this thesis is based on a computer implementation that exploits the pointer utility of the C++ programming language. This implementation stores genetic programs as collections of node classes. Each node contains information about the function or terminal that it represents. It also carries pointers to the addresses of the function and/or terminal nodes that are necessary for the evaluation, or follow the evaluation of the specific node.

3.4 Fitness assignment and selection methods

3.4.1 Fitness functions

As it has already been discussed the quality of evolved genetic programs on the solution of the problem considered is mirrored in their fitness values. The form of the function used for the assignment of fitness to genetic programs is problem-dependent. In symbolic regression problems the fitness of a genetic program will normally be equal to the sum of the squared differences between the predicted value of the dependent variable and its actual value, over the entire set of input-output training examples. Cost, time and parsimony are just some of the alternative forms of fitness functions that have been used in genetic programming algorithms.

The measured value of fitness is known as the *raw fitness* of the genetic program. In some cases it is preferable to express fitness in a form where lower values of fitness correspond to better performance than higher ones, with 'zero' being the fitness of the best possible individual program. This type of fitness is usually referred to as the *standardised fitness* of the genetic program. The choice of fitness representation is critical for the operation of the genetic programming algorithm, since even subtle differences in the performance of genetic programs should be reflected in their assigned fitness values.

The calculation of fitness requires in many problems the evaluation of their performance on a number of test cases (*fitness cases*) representative of the problem considered. An important decision that the designer of the genetic programming algorithm has to face is the choice of the number and type of cases that will form the training set. Large training sets generally produce better performance, but require significant computational power. The type of training cases is also important since they must be representative of as many instances of the problem considered as possible. The result of employing an inappropriate set of fitness cases for the evaluation of genetic programs is poor generalisation. A genetic program that fails to perform satisfactory on problem instances that were not included in its training set has failed to generalise, i.e. it has failed to capture information that is relevant to the solution of the problem. The cause of poor generalisation is the overfitting of data. This phenomenon occurs when the algorithm, in its attempt to reduce the error between the predicted input-output relationship and the actual relationship of the data, fits the noise that is inherent in the training set since only a sample of all possible fitness cases is employed. In the experiments presented in this thesis, a validation set is used for the assessment of the solution's generalisation, wherever this is appropriate. This set comprises of problem instances that were not used for the training of the evolved program.

3.4.2 Selection methods

A significant step in the operation of genetic programming is the selection of computer programs that will participate in genetic mating or will be used for reproduction and mutation purposes. As has already been discussed, individual programs are selected probabilistically based on their fitness values. The selection operation ensures that fitter individuals have more chance of surviving in the next generation. At the same time, the non-deterministic nature of the procedure allows the potential survival of less-fit individuals, a condition that makes the algorithm less vulnerable to the existence of local optima in the solutions' search space.

The selection procedure is independent of the representation scheme employed in the evolutionary procedure. As a result, genetic programming has inherited the selection methods that were originally designed for alternative evolutionary algorithms. The most popular of these methods is the 'fitness-proportionate' or 'roulette-wheel' selection method. It is based on the calculation of the relative fitness of each individual in respect to the fitness of the entire population. The value of the relative fitness indicates the probability of survival of the individual into the next generation. Specific individuals are then selected with the help of a pseudo-random number generator. The 'roulette wheel' selection method is simple but computationally expensive since it requires a centralised calculation of fitness. In addition, the existence of individuals with extremely high levels of fitness ('super individuals'), leads to the premature convergence of the algorithm since these individuals are assigned with a very high probability of survival.

The 'ranking' selection method addresses the shortcomings of the 'roulette-wheel' selection method by assigning ranks to individual programs according to their fitness, with the best individual receiving the highest rank and the worse individual having the lowest rank. The probability of survival for an individual is based on its ranking, however, since the scale of ranking is fixed, 'super individuals' do not overcrowd the population and thus the evolutionary procedure does not easily stagnate.

During the last few years 'tournament' selection has become the standard choice of selection method, at least within the genetic programming community. Tournament selection proceeds by randomly selecting a number of individual programs from the population. The exact number of programs is predefined by the designer of the algorithm and is referred to as the *tournament size*. Selected individuals 'compete' with each other, i.e. their fitness values are compared and the best individual is selected. Tournament selection owns its popularity to its computational efficiency and simplicity. In addition, it provides the designer with the flexibility of adjusting the selective pressure of the algorithm by changing the tournament size.

The importance of stochastic steps in the operation of evolutionary algorithms has already been discussed (section 3.2). While this type of non-determinism ensures that these algorithms are not just parallel hill-climbing heuristics, the use of the pseudo-random number generator is a controversial issue. The quality of the generator has been reported to affect the quality of the results in genetic programming applications (Daida *et al.*, 1997). Moreover, researchers rarely provide detailed information about the type of generator used in their experiments, thus making the task of comparing published results less reliable.

3.5 Genetic operators

Genetic operators are assigned with the task of preserving good genetic material for future generations, exchanging genetic material between evolved genetic programs and introducing diversity to the population of programs. Researchers in the field of genetic programming have proposed several forms of genetic operators. The operators employed in the majority of cases are crossover, mutation and reproduction.

3.5.1 Crossover

The crossover operator aims to emulate the process of sexual recombination in nature. In genetic programming terms, and for the parse-tree representation, the operation of the tree-based crossover is quite simple (figure 3.4). Initially, two genetic programs are selected probabilistically according to their fitness. Then, a node is selected randomly in each of the programs and the subtrees defined by these nodes are swapped. The resulting genetic programs are inserted to the new population.

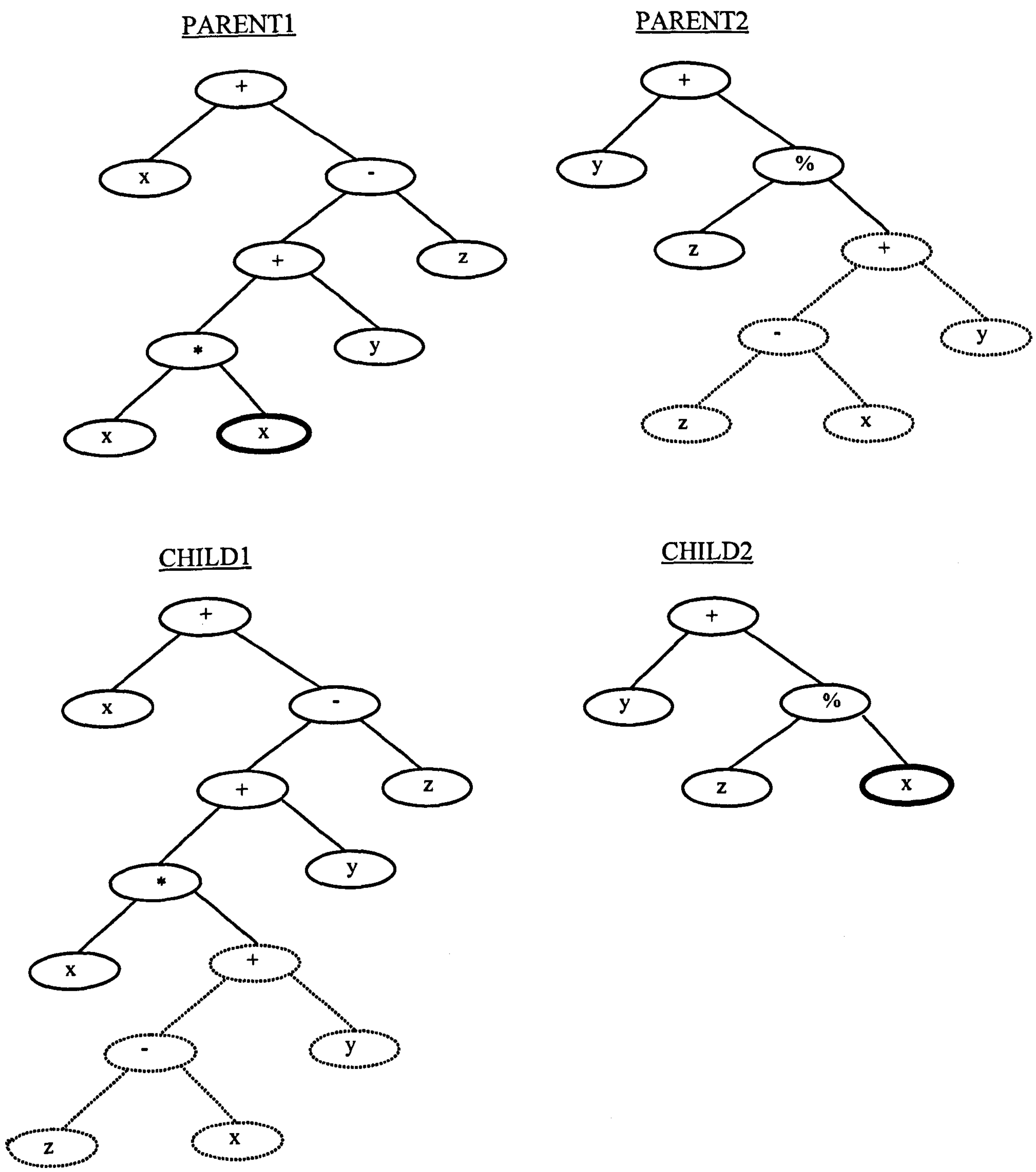


Figure 3.4: Example of the crossover operation

3.5.1.1 The schema theorem in genetic programming

Koza considered crossover as both essential for the operation of genetic programming and responsible for its success. In fact, in his genetic programming implementation

90% (on average) of the genetic programs in each generation were created using tree-based crossover. However, the justification of its use and its real contribution to the evolutionary procedure is fiercely debated within the evolutionary computation and machine learning communities. The theoretical foundations for the use of the crossover operator are based on the existence of a theorem that would explain how its application improves the fitness of genetic programs during the evolutionary procedure. Koza defined a genetic programming 'schema' as any tree or subtree whose presence within a computer program improves its fitness. He claimed that schemata increased exponentially during the evolutionary procedure with the help of the crossover operator, acting as building blocks for the construction of increasingly fit individuals (*building block hypothesis*). Koza's ideas were an extension of the well-known *schema theorem* originally introduced by Holland, which explains the operation of simple genetic algorithms (Holland, 1975). Koza's schema theorem was not expressed in mathematical form, however, subsequent genetic programming researchers have presented schema theorems for genetic programming algorithms that use specific types of crossover and mutation operators (O'Reily and Oppacher, 1995), (Langdon and Poli, 1997). It has been indicated that all the above theorems do not guarantee the exponential increase of good schemata during a GP run, thus further research is needed for the establishment of sound mathematical foundations (Banzhaf *et al.*, 1998). Empirical results have not been able to clarify this issue any further. If crossover was to be mathematically responsible for the success of the GP algorithm, then its omission should degrade the performance significantly. However, it has been debated that the use of alternative genetic operators yield results that are at least as good or even better (Angeline, 1997). The assessment of these studies is difficult, since experiments are usually conducted on a limited number of test problems, while it is a known fact that the performance of evolutionary algorithms is problem-dependent and sensitive to the combination of various parameters of the experimental run.

The operation and the performance of the crossover operator are not a subject of research in this thesis. Unless otherwise stated, genetic programming experiments were conducted using the tree-based crossover operator as introduced earlier. However, the significance of the crossover operator is still a controversial issue within the evolutionary computation and machine learning communities.

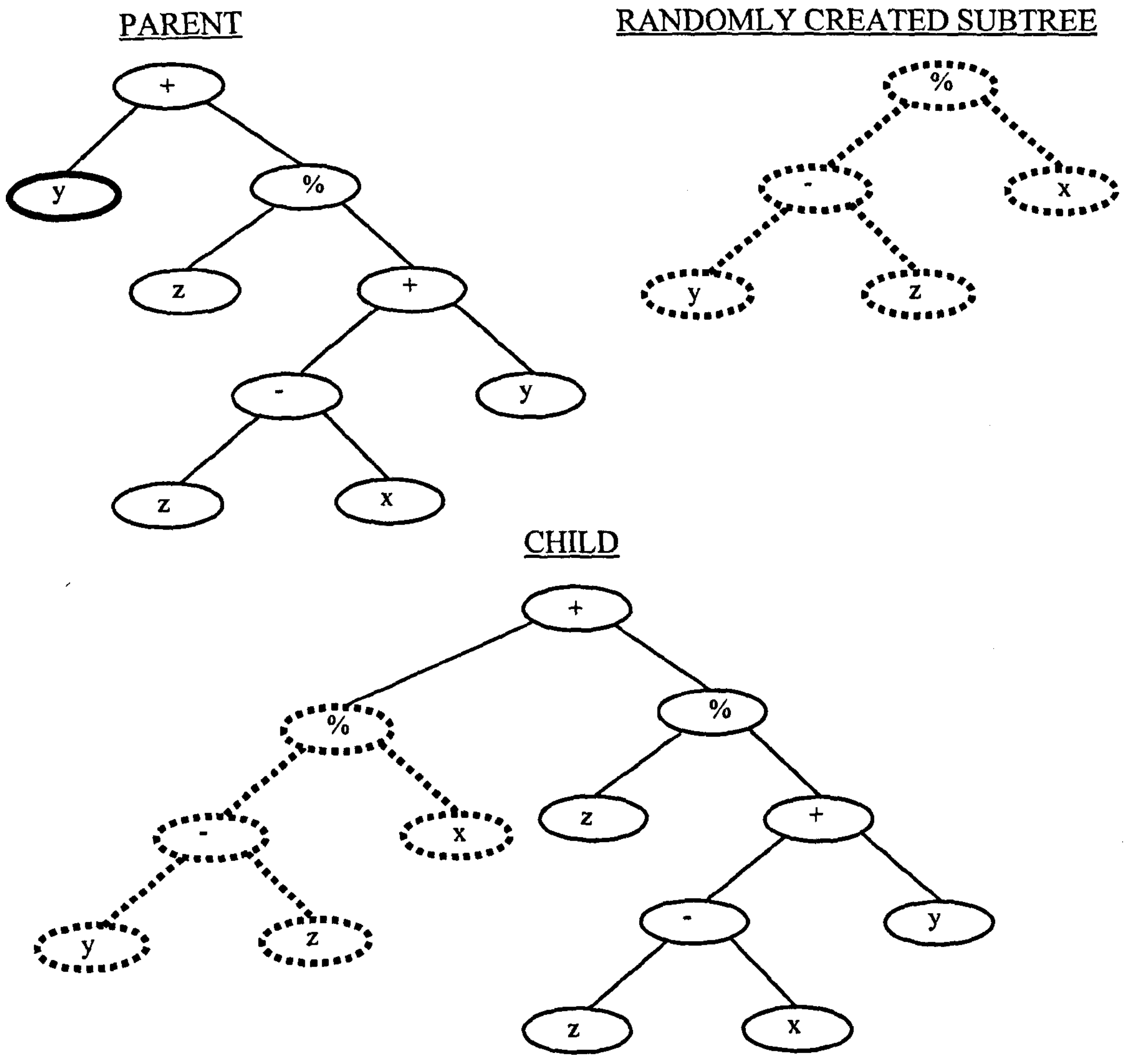


Figure 3.5: Example of the mutation operation

3.5.2 Mutation

Until recently, mutation was considered to be a minor operation in evolutionary algorithms, with the exception of *evolutionary programming* (Fogel *et al.*, 1966) that employs it as the sole genetic operator. A number of researchers have published results which indicate that the mutation operator might be more beneficial to a genetic programming algorithm than originally thought, especially when small populations of genetic programs are used (Luke & Spector, 1997), (Fuchs, 1998).

The operation of the tree-based mutation operator is described in figure 3.5. Initially, a genetic program is selected probabilistically according to its fitness. Then, a node within the tree is selected randomly and the corresponding subtree is deleted. In its place a new subtree is generated, following the same procedure described in the initialisation section (3.3.3). In the majority of genetic programming experiments conducted in this thesis the tree-based mutation operator was applied with a probability of 10% to individual computer programs.

3.5.3 Reproduction

Reproduction proceeds by copying into the new generation the genetic programs that have been chosen through the selection algorithm, without any modification. While it preserves fit individual programs without altering their structure and operation, it does not introduce genetic diversity to the population, thus forcing the premature convergence of the algorithm. The reproduction operator was employed in a small number of experimental genetic programming runs that were conducted in this thesis.

3.6 Design of the genetic programming algorithm

The design of a genetic programming algorithm for the solution of an optimisation problem, is a multi-step procedure that requires the determination of various parameters. Initially, the problem is redefined in a program induction form. Then, the main preparatory steps for the run of the algorithm are followed:

I. *Definition of the functions that will participate in the function set*

II. *Definition of the terminals that will participate in the terminal set*

III. *Definition of a fitness measure for the evaluation of the performance of genetic programs*

IV. *Definition of a number of additional parameters that are necessary for a valid genetic programming run*

The first three preparatory steps have been discussed in detail in the previous sections. The fourth preparatory step requires the determination of the following parameters:

- Population size
- Number of generations
- Initialisation method

- Maximum depth allowed for initial genetic programs
- Selection method
- Probability of applying genetic operators (crossover, mutation, reproduction)
- Maximum depth allowed for genetic programs after the application of operators
- Termination criterion

Once all preparatory steps have been completed, an experimental run of the genetic programming algorithm can be conducted. However, since genetic programming is a non-deterministic search procedure, it is recommended that multiple runs (at least 20) of the algorithm should be conducted for each set of fitness cases.

3.7 An illustration of the genetic programming algorithm

The genetic programming approach to the solution of optimisation problems will be described in this section with the help of a symbolic regression problem taken from Banzhaf *et. al* (1998). In symbolic regression problems the algorithm aims to uncover the function that describes the behaviour of a system using a set of input-output numeric values as the learning domain. In the example problem this set corresponds to the function described in equation 3.1.

$$y = \frac{x^2}{2} \tag{3.1}$$

Initially, a redefinition of the problem in a program induction form is required. In this case the solution of the problem is expressed as a computer program that takes as input the value(s) of the independent variable(s), and produces as output(s) the value(s) of the dependent variable(s). The quality of the evolved computer program is determined by its performance on the set of fitness cases.

The design of the genetic programming algorithm is based on the preparatory steps described in the previous section:

I. *Definition of the functions that will participate in the function set*

The function set comprises of the four main arithmetic operations (addition, subtraction, multiplication, and division). The set of functions that is sufficient for the solution of the problem is not always known in advance. The designer of the algorithm needs to examine the problem carefully and experiment with alternative configurations in order to find a set with satisfactory performance. Note that a protected form of the division function was always used in the experiments conducted in this thesis, for reasons explained in section 3.3.1.

II. *Definition of the terminals that will participate in the terminal set*

The terminal set comprises of the independent variable x, and a set of integer constants with values ranging from (–5) to (5). These particular constants were included for reasons of consistency with the experimental approach followed in Banzhaf *et. al* (1998). However, a solution to the problem can be created without including any constant in the terminal set.

III. *Definition of a fitness measure for the evaluation of the performance of genetic programs*

The fitness of evolved computer programs is calculated using the root mean squared error between the predicted output and the actual output on the set of fitness cases described in table 3.1. Then, the fitness value is adjusted using the formula presented in equation 3.2, so that a higher fitness value will correspond to a better individual.

$$\textit{adjusted fitness} = \frac{1}{1 + \textit{raw fitness}} \tag{3.2}$$

	INPUT	OUTPUT
Fitness case 1	0.000	0.000
Fitness case 2	0.100	0.005
Fitness case 3	0.200	0.020
Fitness case 4	0.300	0.045
Fitness case 5	0.400	0.080
Fitness case 6	0.500	0.125
Fitness case 7	0.600	0.180
Fitness case 8	0.700	0.245
Fitness case 9	0.800	0.320
Fitness case 10	0.900	0.405

Table 3.1: Fitness cases for the example problem

IV. *Definition of a number of additional parameters that are necessary for a valid genetic programming run*

Parameters	Values
Objective:	identification of the function that corresponds to the data of fitness cases
Terminal set:	x, (integer constants from –5 to 5)
Function set:	+, -, ×, % (protected division function)
Population size:	600
Subtree crossover probability:	0.9
Subtree mutation probability:	0.05
Reproduction probability:	0.05
Selection:	Tournament selection, size 4
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Table 3.2: Koza tableau for the GP methodology

The set of additional parameters that need to be defined for the valid run of the genetic programming algorithm is described in table 3.2.

As it has already been discussed, genetic programming is a non-deterministic procedure, thus multiple runs are required for a valid estimation of the performance of the algorithm on the problem considered. 20 runs were conducted in total for the problem discussed in this section. Genetic programming was able to correctly identify the required function in 12 of them. In the following paragraphs a successful genetic programming run will be presented, describing the general performance of the algorithm and the structures evolved.

The best program from the initial generation of randomly created individuals is depicted in figure 3.6. This program corresponds to the function described in equation 3.3.

$$y = \frac{2}{5} \cdot x^2 + \frac{4}{25} \cdot x \tag{3.3}$$

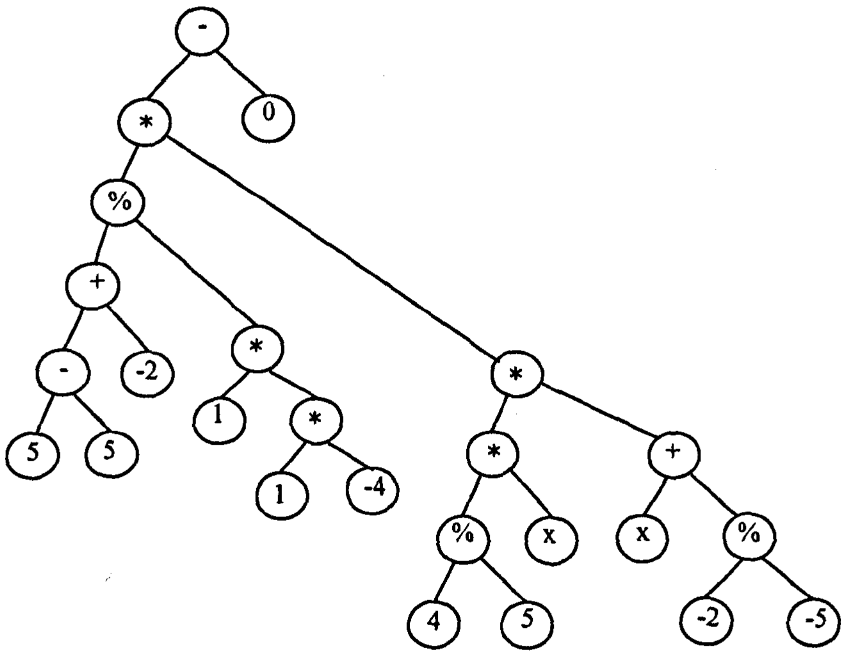


Figure 3.6: Best individual of initial generation

In generations 1, 2 and 3 the evolutionary procedure began to take effect, however, the algorithm did not manage to find the 100% correct solution. The best individuals of these generations and the corresponding functions are presented in figures 3.7, 3.8 and equations 3.4, 3.5 respectively.

$$y = \frac{2x^2 + \frac{x}{9}}{5} \tag{3.4}$$

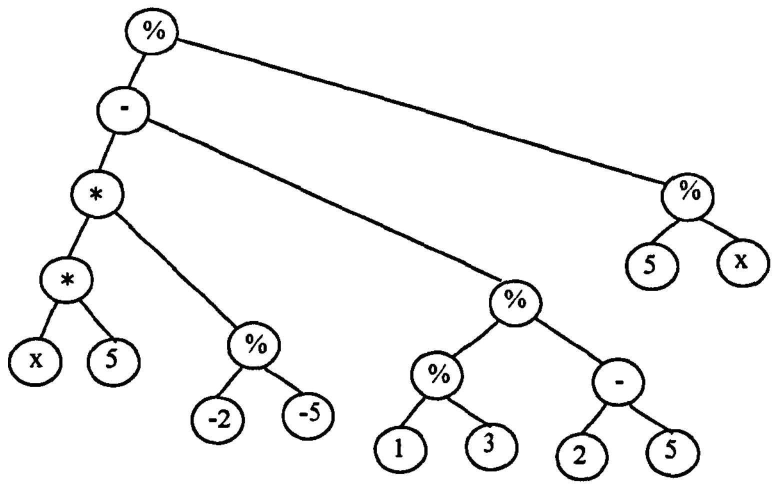


Figure 3.7: Best individual of generation 1

$$y = \frac{x}{3} \tag{3.5}$$

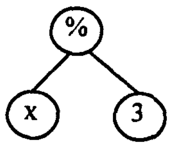


Figure 3.8: Best individual of generations 2 and 3

In generation 4, the algorithm was able to find the 100% correct individual, i.e. the computer program that produces a root mean squared error of '0' between the predicted and the actual output value. The evolved program (figure 3.9) corresponds to the test function from which the fitness cases had been sampled (equation 3.1).

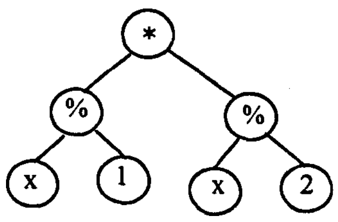


Figure 3.9: Best individual of generation 4 (100% correct)

Figure 3.10 illustrates the dynamic characteristics of this particular run in terms of the mean adjusted fitness of the population, the mean complexity (length) of the population and the adjusted fitness of the best individual over the course of 50 generations.

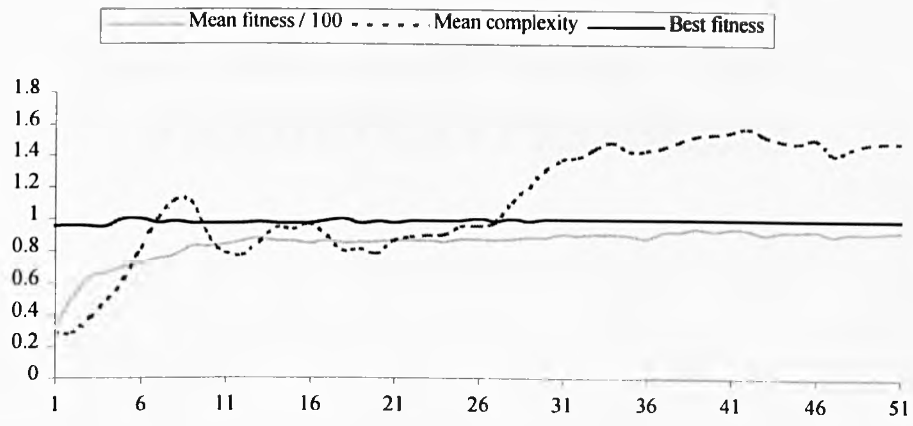


Figure 3.10: Dynamic characteristics of the run

3.8 Conclusions

Genetic programming is a non-deterministic algorithm that combines efficiently the concepts of evolutionary computation and automatic programming. It requires the redefinition of problems in program induction form. An evolutionary procedure is then assigned with the task of finding a satisfactory solution within the space of potential computer programs. Research interest in genetic programming has developed rapidly since its introduction in the early 1990's. In the next chapters the use of genetic programming for the solution of manufacturing optimisation problems will be investigated, starting with the one-machine total tardiness scheduling problem.

Chapter 4

THE ONE-MACHINE TOTAL TARDINESS PROBLEM

4.1 Introduction

One of the possible reasons for the lack of genetic programming applications in the field of manufacturing optimisation is the difficulty of evolving a direct permutation through a GP algorithm. Most solutions of manufacturing optimisation problems - especially in scheduling – can be represented by permutations. While in a classic evolutionary algorithm a permutation can be easily coded as a fixed-size string chromosome and the feasibility of solutions is guaranteed by the application of specially designed operators, a similar GP structure would suffer feasibility problems from the application of subtree-crossover and mutation operators.

In this chapter, the potential use of traditional and modified-GP algorithms for the solution of a well-researched scheduling problem, the one-machine total tardiness problem, is investigated. The proposed algorithms use the traditional manufacturing concept of dispatching rules for the indirect construction of job schedules, thus avoiding feasibility problems.

Dispatching rules are employed in two alternative forms. First, combinations of existing dispatching rules are evolved for the solution of individual tardiness problems. Then, a genetic programming framework is employed for the construction of a new formula of a dispatching rule that challenges man-made dispatching rules on the problem considered.

Potts and Van Wassenhove (1982) created an algorithm that is able to find optimal solutions for one-machine total tardiness problems within acceptable computational times. This algorithm allows the realistic evaluation of the performance of all GP-generated methods introduced in this chapter. However, while Potts and Van Wassenhove's algorithm has no other known applicability apart from the one-machine total tardiness problem, all the methods described in the following sections can be used - in principle - for the solution of any other one-machine scheduling problem.

The rest of this chapter is organised as follows: In section 4.2 the one-machine total tardiness problem is defined and a review of the solution methodologies that have been proposed for its solution is presented. Section 4.3 introduces the GP–based methodology for the solution of individual one-machine total tardiness problems. In section 4.4 genetic programming is employed for the evolution of new dispatching rules that can be used for the solution of all instances of the problem. The conclusions of this chapter are drawn in section 4.5.

4.2 Minimising total tardiness in a single-machine environment

4.2.1 Problem definition

One of the main objectives of the scheduling procedure is the completion of all jobs before their agreed due dates. Failure to keep that promise has negative effects on the credibility of the company.

If lateness of job i is defined as the difference between its completion time C_i and the corresponding due date d_i, then the tardiness of the job is calculated from equation (4.1).

$$T_i = \max(0, C_i - d_i) \tag{4.1}$$

In other words, tardiness represents the positive lateness of a job. In a single-machine environment, the total tardiness problem is defined as follows:

A number of jobs J_1, J_2,...,J_n are to be processed in a single facility. Each job is available for processing at time zero, and its processing time p_i and due date d_i are known in advance. The aim is to find the processing sequence that minimises the sum of tardiness of all jobs (4.2).

$$\sum_{i=1}^{n} \max(0, C_i - d_i) \tag{4.2}$$

where C_i is the completion time of job i. If for each job i, an associated weight (penalty) w_i exists, the total tardiness is calculated using (4.3).

$$\sum_{i=1}^{n} w_i \{\max(0, C_i - d_i)\} \tag{4.3}$$

The objective of the weighted total tardiness problem is the minimisation of (4.3). If (4.2) or (4.3) was to be divided by the total number of jobs n, the objective would become the minimisation of mean tardiness. However, since a division by a constant does not alter the nature of the objective, the problems are essentially the same.

The total tardiness problem is a special case of the weighted total tardiness problem. Both problems are not easy to solve, especially for large values of n. The complexity of the weighted total tardiness problem has been established by Lawer (1977). He proved that the associated decision problem is NP-complete by reduction from the 3-partition problem. The complexity of the unweighted total tardiness problem remained unestablished until 1989, when Du and Leng (1989) proved that the associated decision problem is NP-complete by reduction from a restricted version of the Even-Odd Partition problem.

In the following paragraph the solution methodologies that have been proposed for the solution of the one-machine total tardiness problem will be discussed.

4.2.2 Literature review

4.2.2.1 Introduction

The research for the solution of both versions of the one-machine total tardiness problem spans a period of four decades. From the early stages it became apparent that complete enumeration of all permutations of jobs was not practical, since the total number of all possible schedules is (n!), where n is the total number of jobs in the problem. Two main lines of research were followed during these forty years. In the early stages researchers focused on the development of efficient implicit enumeration algorithms, mainly dynamic programming and branch and bound. While these algorithms are analytical, their application is restricted to relatively small-sized problems due to computational and memory requirements.

Dynamic Programming (DP) (Bellman and Dreyfus, 1962) is much faster than complete enumeration. However, it has obvious limitations in terms of memory requirements (2^n values must be stored before the construction of an optimal schedule). Branch and bound methods are quite unpredictable in their computational requirements. Their success depends heavily on the calculation of sharp lower bounds, which result on the quick elimination of subtrees, speeding up the procedure considerably.

In recent years, researchers have focused on the development of fast and efficient heuristic algorithms. While these algorithms perform much better than implicit enumeration techniques in terms of computational requirements, the optimality of their solutions is not guaranteed. In the following paragraphs the research on the one-machine total tardiness problem as it has evolved during the last forty years will be presented.

4.2.2.2 Early approaches

The earliest investigation of the total tardiness problem was given by McNaughton (1959), who presented some theorems for scheduling independent tasks in a single machine, with associated penalties for missing the deadlines. McNaughton showed that the set of permutation schedules is dominant for this objective, and that the Weighted Shortest Processing Time (WSPT) rule produces optimal schedules when no job can be finished on time. Schild and Fredman (1961) extended the use of dispatching rules on some other cases and proposed a general methodology for the solution of the problem with no guaranteed optimality.

Held and Karp (1962) were the first researchers to propose the use of dynamic programming for the solution of sequencing problems. The principle of optimality for a scheduling problem of this type stated that "in an optimal schedule, the first k jobs must form an optimal schedule for the reduced problem based on these k jobs alone".

The recursive equations of DP were formed based on this principle. Lawler (1964) utilised dynamic programming for the solution of the weighted total tardiness problem. No results were presented, however the author indicated the computational drawbacks of his method.

Emmons (1968) published a breakthrough paper on the one-machine total tardiness problem without associated weights. He proved a series of theorems that established relations between jobs in an optimal sequence. Emmons also proposed an algorithm that utilised these theorems in order to reduce the size of the problem and employed branching when no further relations between jobs could be found. These theorems formed the basis of a number of solution methodologies over the years. Emmons also generalised the cases where EDD or SPT sequencing yielded optimal schedules.

A few years later Srinivasan (1971) introduced a hybrid method based on Emmons' theorems and dynamic programming. In this three-step algorithm, Emmons' theorems were initially used to reduce the size of the problem and to introduce precedence relations between jobs. Dynamic programming was then employed to solve the reduced problem. Results from randomly generated test problems showed that his method exhibited substantial gain on computational efficiency in comparison with complete enumeration and conventional DP. Srinivasan also investigated the effect that the change of the parameters of the problem had on the computational requirements. He reported that problems representing shops 60% tardy on average were computationally hard. This observation was later confirmed by alternative researchers (Russel and Holsenback, 1992).

The same year Wilkerson and Irwin (1971) presented the first heuristic technique for the solution of the total tardiness problem. Their algorithm employed a decision rule and adjacent pairwise interchanges for the construction of schedules. Their method was substantially faster than complete enumeration, however, the optimality of solutions was not guaranteed.

4.2.2.3 Development of implic it enumeration algorithms during the 70's and 80's

Rinnooy Kan *et al.* (1975) proposed an improved brand and bound approach for the solution of the weighted version of the problem. They focused their research on the development of dominance theorems and the calculation of strong lower bounds. The authors warned that the implementation of their theorems could lead to the creation of precedence cycles and proposed a method for avoiding this deadlock situation. Despite the fact that their algorithm was faster than previous algorithms for problems with $15 \leq n \leq 20$, they underlined the need for sharper lower bounds, and even better dominance theorems.

Fisher (1976) recognised this need and produced an algorithm that was able to find extremely sharp lower bounds for branch and bound algorithms. His method was based on a dual formulation of the problem. He introduced a subalgorithm that solved efficiently the Langrangian problem formed by the dual variables. The solution of this problem provided both sharp lower bounds and good feasible solutions. In this latter way the algorithm could be utilised as an efficient heuristic procedure. The extremely sharp lower bounds allowed the solution of large sized problems (n=50) in moderate computational times. Fisher's method presented considerable advantages over the alternative methods that had been proposed until that time for the same problem.

Lawler (1977) presented a 'pseudopolynomial' algorithm for the solution of the total tardiness problem with agreeable weights (of which the unweighted total tardiness problem is a special case). Lawler's algorithm, based on dynamic programming, had a worst case running time of complexity $O(n^4 P)$, where

$$P = \sum_{i=1}^{n} p_i$$

and p_i is the processing time of job i. The existence of Lawler's 'pseudopolynomial' algorithm meant that the unweighted total tardiness problem is NP-hard in the ordinary sense.

It has already been pointed out that dynamic programming approaches to the total tardiness problem had limited applicability due to the extensive computer memory requirements. However, Baker and Shrage presented two dynamic programming - based methods that were able to solve large-sized problems (n=50) faster than branch and bound algorithms. The first method (Baker and Shrage, 1978) was an investigation into the application of dynamic programming to sequencing problems with precedence constraints. The authors argued that when chain-like relations between jobs existed (jobs with only one direct predecessor and only one direct successor), the number of feasible subsets that needed to be enumerated was reduced, making the DP procedure much faster. In the case of the total tardiness problem, where no precedence relations exist a priori, Emmon's theorems could be utilised to create them a posteriori. A few months later Shrage and Baker (1978) introduced a powerful dynamic programming implementation for problems of the same type. Their method involved an enumerative procedure for all feasible subsets, and a labelling procedure for storing and retrieving efficiently the values of these subsets. Depending on the precedence constraints of the problem, the labelling procedure could reduce considerably the storage requirements of the algorithm, allowing the application of dynamic programming in large sized problems. Indeed, Shrage and Baker showed that their method could solve 50-job test problems much faster than the algorithms of Fisher and Rinnoy Kan *et al.*

In the early 80's Potts and Van Wassenhove (1982) presented a new methodology for the solution of the total tardiness problem, which combined features from earlier approaches. The algorithm started by constructing a precedence relations graph based on Emmons' theorems. The labelling scheme of Shrage and Baker was then employed to address all feasible subsets. In the case of a very large number of labels (>35000), the decomposition algorithm of Lawer (1977) was utilised to break up the problem in a number of smaller and easily tackled subproblems. Each of these subproblems was solved optimally by the dynamic programming approach of Shrage and Baker. The efficiency and the speed of this algorithm enabled the solution of medium to large problems ($50 \leq n \leq 100$) in reasonable computational times. Especially for problems with $n \leq 70$, the performance of the algorithm was impressive. In larger problems the algorithm was slightly slower, but still no optimal results on problems of this size had been reported from any researcher until that time. The authors indicated that their method did not generalise for the case of the weighted total tardiness problem. A few years later, they proposed an alternative branch and bound method for the solution of the latter problem (Potts and Van Wassenhove, 1985). The lower bounds were obtained using Langrangian relaxation. The multiplier adjustment method was employed for the calculation of Langrangian multipliers. Some dominance features of dynamic programming were also utilised for the elimination of as many nodes of the solution tree as possible. Potts and Van Wassenhove's method was able to solve problems much larger than those reported in earlier branch and bound approaches. The authors concluded that the existence of a very sharp lower bound was not as important as it was considered to be by previous researchers. They claimed that a feature of quick enumeration of feasible subsets - like the one that they used in their algorithm - could enhance the search for an optimal schedule considerably.

Another implicit enumeration algorithm was proposed by Sen *et al.* (1983) who utilised Emmon's theorems and corollaries in order to establish precedence relations between jobs. They reported that their seven-step algorithm outperformed Shrage and Baker's (1978) dynamic programming algorithm in terms of computational efficiency for large-sized problems (n>50).

4.2.2.4 Recent developments

In the last decade, researchers have turned their attention to the implementation of efficient heuristic methods, which are generally faster and much easier to implement. The optimality of a heuristic solution is not guaranteed, however, it is easy to test its efficiency by calculating the optimal values using an implicit enumeration algorithm.

A typical local search procedure for permutation problems such as the one-machine total tardiness problem incorporates Adjacent Pairwise Interchanges (API's) of jobs in order to find optimal or near optimal sequences. The quality of an API depends both on the initial sequence of the algorithm ('seed'), as well on the search strategy that is employed. Fry *et al.* (1989) presented a heuristic approach that utilised the best of nine Adjacent Pairwise Interchange (API) strategies. These strategies were produced by the combination of three different initialising procedures (EDD, SPT and Minimum Slack Time (MST)), with a same number of search strategies (front to back - restart when local optimum found, back to front - restart when local optimum found, all adjacent pairwise interchanges - restart from the best found). A comparison of their method with Wilkerson and Irwin's algorithm showed that API's constituted a very fast and reliable optimisation technique for the one-machine total tardiness problem.

Some years later, Holsenback and Russel (1992) introduced their powerful Net Benefit Of Relocation (NBR) heuristic procedure. Emmons stated in one of the corollaries of his theorems that an EDD sequence is optimal if the tardiness of each job is less or equal than its processing time. The NBR heuristic was based on this observation. Starting from the last job of the EDD sequence the algorithm found the first job which possessed 'reducible' tardiness (i.e. $T_i > p_i$). For each job preceding job *i* in the sequence and having largest processing time, the Net Benefit of Relocation was calculated, i.e. the benefit in tardiness units of exchanging the positions of these jobs. The job with the higher NBR (subject to NBR>0) exchanged its position with job *i*. The same procedure was repeated on the *remaining i-1 jobs*. NBR produced high quality results even for large problem instances (n=100). The deviations from the optimal values - obtained using Potts and Wassenhove's method - were relatively small, while the computational times where excellent (around one second of CPU time for problems with n=100).

A year later Panwalkar *et al.* (1993) introduced the PSK heuristic, which performed impressively on a number of different cases. PSK was a simple, effective procedure that started from an SPT sequence and constructed a schedule by making n-passes, one for each job. The algorithm used two sets of jobs, the SPT set of unscheduled jobs {U}, and the set of the of jobs that had already been scheduled {S}. In each pass, a job from {U} was considered to be active, starting from the leftmost one. If the scheduling of this job was considered necessary since it was already tardy or on time, or because the successive job would make it tardy, then it was removed from {U}and it was placed on {S}. Otherwise the next job in {U} became active and the same set of comparisons was performed. The algorithm was tested on a wide range of test problems producing satisfactory results. However, Russel and Holsenback (1996) questioned the validity of these results, claiming that in their own experimentation with the same problems used in (Panwalkar *et al.*, 1993), the NBR was generally superior to PSK. In any case, they noted that PSK was particularly suitable for a specific set of problems characterised by high values of tardiness factor and range of due dates.

Recently, Russel and Holsenback (1997) introduced some modifications to the NBR heuristic, which improved the performance significantly, especially in the case of large - sized problems (n=100). They also proposed the composite use of the modified NBR and PSK heuristics, since both methods were extremely fast, easy to implement and complementary in nature.

Meta-heuristics (simulated annealing, tabu search, genetic algorithms etc.) have gained a considerable research interest during the last decade. All of these techniques have been applied to a wide range of scheduling problems. Simulated annealing (SA) is a non-deterministic heuristic algorithm developed by Kirkpatrick *et al.* (1983). The main operation of SA is similar to a hill-climbing local search procedure. SA however, allows probabilistic jumps to neighbourhood solutions that perform worse than the starting solution. In that way the algorithm has the ability to escape local optima. The probability of selecting a worse solution depends on a 'temperature' value T, in a procedure that resembles the annealing of metals to a minimum energy state. Temperature is initially high, but its value drops exponentially with the number of iterations. Thus, selection of a worse individual is more likely during the initial phases of the algorithm.

Matsuo *et al.* (1989) were the first researchers to employ simulated annealing for the solution of the weighted total tardiness problem. A few years later Potts and Van Wassenhove (1991) presented a simulated annealing algorithm for the unweighted case of the same problem, adopting the adjacent pairwise interchange strategy for the creation of neighbourhood schedules, and a special interweaving procedure which performed local search at certain stages of the algorithm. A much simpler simulated annealing method was proposed a few years later by Ben-Daya and Al-Fawzan (1996). Random job interchanges were used for the creation of neighbourhood schedules and no form of local search was employed. Their method outperformed the heuristic approaches of Fry *et al.* (1989) and Holsenback and Russel (1992) in a wide range of test problems. However, while Holsenback's heuristic was extremely fast, Ben-Daya's algorithm suffered from slow convergence, a well-known disadvantage of simulated annealing.

Ibaraki and Nakamura (1994) presented another dynamic programming - based approach for the solution of the weighted total tardiness problem called Successive Sublimination Dynamic Programming (SSDP). It aimed to reduce the number of subsets that need to be enumerated in a dynamic programming procedure. The algorithm utilised upper and lower bounds to eliminate as many states of the scheduling tree as possible. Experimentation showed that Ibaraki's approach was much faster than conventional dynamic programming. However, as the authors indicated, the method had limited applicability to problems characterised by certain levels of tardiness.

Finally, Tansel and Sabuncuoglu (1997) have recently presented an interesting investigation on the total tardiness problem, utilising geometric representations in order to analyse and prove Emmons' theorems. Their research introduced a number of theorems that identified 'easy' or 'hard' problem instances based on the graphic representation of the problem's data. They noted that a particularly 'hard' family of problems (the one where Emmons' theorems cannot be initiated), is difficult to be created using a random number generator, thus the reliability of the randomly generated test problems is not guaranteed.

4.3 A GP - heuristic for the solution of the one-machine total tardiness problem

4.3.1 Introduction

This section describes the design of a genetic programming algorithm for the solution of individual one-machine total tardiness problems. The aim of the algorithm is the generation of schedules for specific instances of the problem rather than the evolution of scheduling policies for the general total tardiness problem. This is achieved by employing the instance of the problem considered as the only fitness case during the training phase of the algorithm. The first step for the design of the genetic programming algorithm is the redefinition of the problem in a program induction form: "Find a computer program that takes as input information about the characteristics of the unscheduled jobs, and produces as output a complete schedule that minimises the total tardiness of all jobs in the system".

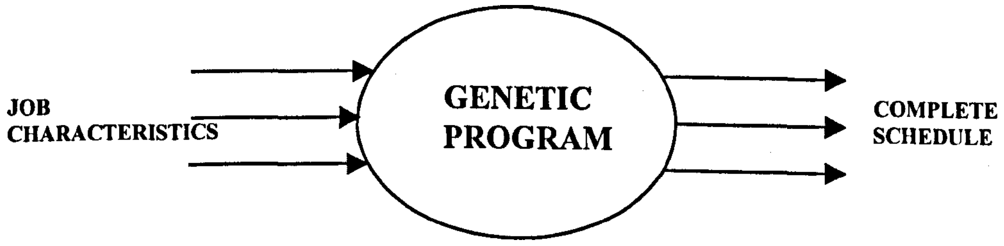


Figure 4.1: Genetic programming approach to schedule generation

4.3.2 Design of the algorithm

4.3.2.1 Schedule representation

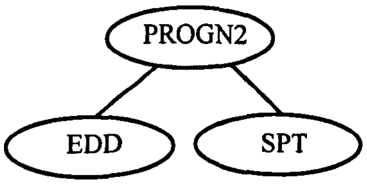
A natural representation for the solution of the one-machine total tardiness problem is a permutation of all jobs to be scheduled. Evolutionary computation researchers have extensively used permutation representations for flowshop and one-machine scheduling problems like the one discussed in this chapter. Specially designed genetic operators (originally created for the solution of the travelling salesman problem) ensure the feasibilty of solutions throughout the evolutionary procedure.

The representation of a permutation within a conventional GP framework is not straightforward due to the variable length of the structures evolved. A direct representation of a schedule through a tree-like program structure would suffer from feasibility problems by the application of subtree crossover and mutation operators.

In the implementation proposed in this section, common manufacturing dispatching rules are employed as an indirect way of representing a permutation through a genetic programming framework. A dispatching or priority rule is a method of determining the next job to be scheduled out of a set of unscheduled jobs (section 2.2.3.2.2). The decision is based on certain job characteristics like processing times, due dates etc. There is a wide variety of dispatching rules available, especially for dynamic scheduling problems (Blackstone *et al.*, 1982).

The idea of using combinations of dispatching rules for the solution of scheduling problems is not new. In a well-known scheduling textbook, Fisher and Thompson (1963) proposed the probabilistic learning of scheduling rules as a method for tackling job-shop scheduling problems. A number of researchers have employed combinations of dispatching rules for the solution of similar problems (section 2.2.3.2.2). The majority of these approaches were based on the idea of utilising different dispatching rules on individual machines or alternative scheduling points in the plant.

In the algorithm presented in this section, a sequence of jobs is constructed indirectly through an associated sequence of dispatching rules. While dispatching rules can be easily represented in a GP framework as terminal nodes, there are still a number of issues that need to be addressed before a GP run can take place. First, a connection mechanism between the dispatching rules within the genetic program needs to be defined. Koza has already suggested a way of sequencing two or more functions or terminals by employing the function PROGN (the notation is taken from the LISP - equivalent function). PROGN takes as arguments two or more function or terminal nodes and operates as a connection point between these arguments. Figure 4.2 portrays the operation of PROGN function and its role in the indirect generation of schedules.



Equivalent code:

schedule a job according to EDD rule
then,
schedule a job according to SPT rule

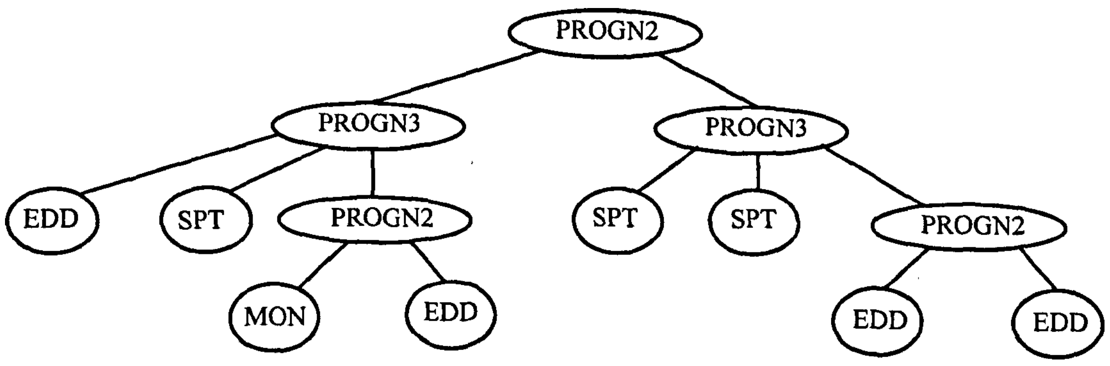
Figure 4.2: Operation of the PROGN function

The number of jobs in the tardiness problems considered in the experimentation is always fixed. However, the application of operators creates offspring of unequal size. The standard GP algorithm has to be modified in order to take in account the following cases: (i) the case where evolved individuals contain more dispatching rules than the number needed to generate a schedule and (ii) the case where programs contain fewer dispatching rules than the number needed for the creation of a complete schedule. The former case is easily accommodated by considering only the first *n* dispatching rules (where *n* is the total number of jobs in the problem) during the evaluation phase of the algorithm. The latter case is accommodated using a function that penalises any program that produces an incomplete schedule with a very high value of tardiness.

Figure 4.3 illustrates the translation of an evolved computer program to a valid job schedule. The data of the example problem (table 4.1) have been taken from Baker (1974). A description of the dispatching rules used in this example program is given in section 4.3.2.3.

Job no.	1	2	3	4	5	6	7	8
p_i	121	147	102	79	130	83	96	88
d_i	260	269	400	266	337	336	683	719

Table 4.1: Data of the example problem for the illustration of schedule representation



Dispatching rules sequence:	EDD-SPT-MON-EDD-SPT-SPT-EDD-EDD
Corresponding job sequence:	1 - 4 - 6 - 2 - 8 - 7 - 5 - 3
Total Tardiness:	1014

Figure 4.3: Illustration of schedule representation using the GP-heuristic

4.3.2.2 Function set

Two variations of the PROGN function were employed in the experimentation, the PROGN2 and PROGN3 functions that require two and three arguments for their execution respectively. While the PROGN2 function was sufficient for small sized problems, PROGN3 enhanced the performance of the algorithm in large-sized problems. Tests were also made with PROGN4 and PROGN5 functions without any significant change on the performance of the algorithm. As a result, the function set of the algorithm contained only the connecting functions PROGN2 and PROGN3.

4.3.2.3 Terminal set

The terminal set comprised a number of dispatching rules that can be considered as good building blocks for the generation of fit schedules. Three rules were selected to take part in the terminal set. The first was the Earliest Due Date rule (EDD), which sequences jobs in non-decreasing order of their due date. The second was the Shortest Processing Time rule (SPT), which sequences jobs in non-decreasing order of their processing time. Both these rules are known to perform optimally or near-optimally in specific cases; the SPT rule produces an optimal schedule when no job in the resulting schedule can be completed on time, while the EDD rule is optimal when at most one job in the resulting schedule is tardy. More general cases for the optimality of EDD and SPT scheduling are given by Emmons (1968). Based on these theorems the SPT rule is expected to perform better at problems with high levels of tardiness, and the EDD rule to be ideal for the inverse case. The last rule of the terminal set was originally introduced by Montagne (1969) for the solution of the weighted total tardiness problem. This rule (which will be identified as 'MON' from this point onwards) sequences jobs in non-decreasing order of the following ratio:

$$\frac{p_i}{w_i} \cdot \frac{1}{\left[1-\left(\frac{d_i}{\sum_{i=1}^{n} p_i}\right)\right]} \qquad (4.4)$$

where: p_i is the processing time of job i

d_i is the due date of job i

w_i is the associated penalty for job i

By setting all weights equal to one, the ratio used for the unweighted version of the problem is obtained:

$$\frac{p_i}{\left(\sum_{i=1}^{n} p_i\right)-d_i} \qquad (4.5)$$

(the summation term missing from the numerator of the ratio has no effect on the operation of the rule). The fact that the MON rule has been designed specifically for the solution of the one-machine total tardiness problem, means that it produces a good overall performance. If, for example, a due date of a job is close to the makespan of all jobs, then the ratio becomes larger, thus the job is likely to be scheduled on a later stage. Conversely, jobs with early due dates are given extra priority.

For this particular terminal set, there are 3^n possible combinations of dispatching rules. In principle, GP should be able to find a combination that performs at least as good as the best dispatching rule of the terminal set. Unless a combination of dispatching rules is able to produce a better result, the algorithm should be able to create an individual that is constructed only from terminals of the best dispatching rule for the problem considered.

4.3.2.4 Fitness measure

The fitness measure used for the evaluation of solutions was the level of tardiness produced by the resulting schedules on the problem considered. The higher the level of tardiness, the lower was the chance of individual combinations of dispatching rules of surviving into the next generation. The raw and stanardised fitness of the programs was the same, since a smaller value of fitness corresponded to better individuals. Note that since only one problem was considered as fitness case during the evaluation phase of the algorithm, the proposed algorithm acted as an optimiser for specific problem instances.

4.3.2.5 Genetic operators

The traditional crossover and mutation operators were employed in the experimentation described in this section together with a point-mutation operator, which, within the context of this problem, operated as a conventional EA mutation operator. Whenever an individual genetic program was probabilistically selected for the point-mutation operation, a terminal point of the program was selected randomly, and was replaced by another terminal selected randomly from the terminal set.

Different combinations of these operators were tested during the experimentation phase, and results indicated that the best performance was given by a combination of the tree-mutation and point-mutation operators. Configurations involving the crossover operator performed slightly worse than the proposed setting. However, since this is a modified version of a genetic programming algorithm, there is nothing to suggest a generalisation of this statement.

4.3.2.6 Additional parameters

The values of additional parameters that need to be defined for the valid run of the GP algorithm together with a summary of the proposed methodology are described in figure 4.4.

Parameters	Values
Objective:	Evolve a combination of dispatching rules for the solution of individual total tardiness problems
Terminal set:	EDD, SPT, MON
Function set:	PROGN2, PROGN3
Population size:	200
Tree mutation probability:	.8
Point mutation probability:	.2
Selection:	Tournament selection, size 4
Number of generations:	50
Maximum depth for crossover:	17
Maximum depth for individual generated for mutation:	2 (*n*=12), 3 (*n*=25), 4 (*n*=50,*n*=100)
Initialisation method:	Ramped half and half

Figure 4.4: Koza tableau for GPC heuristic

4.3.3 Experimental basis

The experimental basis employed in this section followed the guidelines suggested by the majority of researchers investigating the one-machine total tardiness problem. A number of test problems were generated randomly for particular settings of *n*. The processing times for each job were drawn out of the uniform distribution [1..100]. Due dates were drawn out of a uniform distribution defined as follows:

$$[P(1-T-(R/2)), P(1-T+(R/2))]$$

where:

P: is equal to the sum of processing times of all jobs.

T: is the tardiness factor. It defines the percentage of jobs that are expected to be tardy on average. If, for example, T=0.2, 20% of jobs are expected to be tardy.

R: is the range of due dates. It defines the tightness of due dates around the makespan of all jobs. Generally speaking problems with tight due dates are more difficult to solve.

Table 4.2 describes the combinations of R and T that were used in the experimentation and the number of replications for each of these combinations:

	T=0.2	T=0.4	T=0.6	T=0.8
R=0.2	5 problems	5 problems	5 problems	5 problems
R=0.4	5 problems	5 problems	5 problems	5 problems
R=0.6	0	5 problems	5 problems	5 problems
R=0.8	0	5 problems	5 problems	5 problems
R=1.0	0	0	5 problems	5 problems

Table 4.2: Configuration settings for the test problems

Problems with T=1.0 were not included in the experimental setup since the levels of tardiness in this case are unacceptable in practice. The remaining combinations of T and R that were not included in the set are considered to represent trivial cases.

This set of combinations was tested on four different settings of *n* (12, 25, 50 and 100). The GP heuristic was tested on 320 problems in total, a fairly representative set of benchmark problems. The optimal values for these problems were generated using Potts and Van Wassenhove's algorithm, which has already been described in section 4.2.2.3. The modified version of Holsenback and Russel's NBR heuristic (Russel and Holsenback, 1997) was employed as a comparison heuristic. This method has shown outstanding performance in published results. In addition, a simulated annealing algorithm was designed and tested on the same benchmark problems. This implementation of the SA algorithm was based on two previous successful approaches of Potts and Van Wassenhove (1991) and Ben-Daya and Al-Fawzan (1996). Table 1A in the Appendix describes the characteristics of the SA approach that was used in this section. The search strategy of the algorithm was systematic in the form of adjacent pairwise interchanges, but temperature was decreased according to the schedule proposed by Ben-Daya and Al-Fawzan.

Three versions of the GP-heuristic described in this section were tested during the experimental phase. Apart from the simple version (which will be identified as GPC from now on), two local search methods were employed as optimisers for the best individual evolved by the GP algorithm. The first method used the best individual schedule found as the starting sequence of a local adjacent pairwise procedure (GLS). In the second method, the GP-heuristic ran for a small number of generations (five), and the best evolved individual program provided the initialising sequence of the SA algorithm already described (GSA). There are several reasons for hybridising GP with other local search techniques in this particular problem:

- Local search has been used successfully to enhance the evolutionary procedure in sequencing problems (Murata *et al.*, 1996a).
- The function and terminal sets employed by GPC are not always sufficient for the creation of an optimal schedule. The introduction of a local search algorithm at the end of the evolutionary procedure increases the probability of reaching an optimal solution.
- A number of researchers have reported that dispatching rules provide good initial schedules for local search techniques (Panwalkar *et al.*, 1993). A combination evolved by GPC should provide, in principle, a better starting point than individual dispatching rules for a local search procedure.

A number of student's t-tests were used in many cases for the calculation of a statistical measure of the difference between the algorithms compared. Note that these tests were performed on the sets of penalties produced by each method in relation to the optimal values, as was suggested in the experimental approach of Russel and Holsenback (1997).

4.3.4 Results

4.3.4.1 GPC vs. other dispatching rules

The hypothesis for the introduction of combinations of dispatching rules (4.3.2.3) suggested that they would perform at least as good or better than individual dispatching rules on the problems considered. Tables 4.3-4.6 support this hypothesis by highlighting the superiority of GPC on a variety of cases. Table 2A in the Appendix explains in detail the statistical terms used in the tables.

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
EDD	10	357.4	39.12811	126.75
SPT	1	215.7875	410.7046	14200
MON	5	97.2625	22.68682	169.44444
GPC	33	25.275	2.250583	15.562565

Table 4.3: GPC vs. EDD, SPT and MON (*n*=12)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
EDD	4	1775.538	52.57428	131.42857
SPT	0	994.45	413.1591	19483.333
MON	3	502.05	29.03441	203.125
GPC	9	230.275	7.039146	23.329558

Table 4.4: GPC vs. EDD, SPT and MON (*n*=25)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
EDD	6	6668.163	55.095	124.15254
SPT	0	4372.938	1035.89	30030
MON	4	2197.275	54.53038	1129.5238
GPC	5	1254.25	12.0713	43.808651

Table 4.5: GPC vs. EDD, SPT and MON (*n*=50)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
EDD	5	28229.94	57.86391	92.885164
SPT	0	17231.99	9109.769	318066.66
MON	5	8689.35	49.79771	1126.8691
GPC	6	6418.888	18.4861	140.47619

Table 4.6: GPC vs. EDD, SPT and MON (*n*=100)

In comparison with EDD the difference of GPC in terms of MADO started from 1314% for *n*=12, fell to 671% for *n*=25, and further decreased to 431% and 339% for *n*=50 and *n*=100 respectively. GPC performed better than SPT as well, especially in problems with small levels of tardiness, as expected from the theory of the problem.

The best performance of individual dispatching rules was achieved by MON, the only purpose-based dispatching rule included in this report. The penalties imposed by MON in terms of MADO were 284% higher than GPC for n=12, 212% higher for n=25 and 175% higher for n=50. The difference in penalties dropped significantly for n=100 (only 35% higher), showing that as the size of the problem increased and the algorithm struggled to find a suboptimal solution in the enormous search space (3^{100}), the MON rule became dominant in the combination of dispatching rules of the GPC solutions. In terms of optimal values GPC had a significant advantage only in very small problem instances. EDD outperformed GPC for n=50 due to its strength in problems with low levels of tardiness (as discussed earlier, an EDD sequence is optimal if it produces schedules where at most one job is tardy). However, Table 4.7 illustrates that when GPC was compared with the cumulative performance of the three individual dispatching rules, it outperformed them in all problem sizes.

	Number of times GPC was better than individual dispatching rules	**Number of times individual dispatching rules were better than GPC**	**Number of times GPC had the same performance with individual dispatching rules**
n=12	64	0	16
n=25	77	0	4
n=50	69	5	6
n=100	64	11	5

Table 4.7: GPC vs. dispatching rules in terms of non-dominated solutions

4.3.4.2 M-NBR vs. GLS and GSA

The fact that GPC employs dispatching rules for the scheduling of jobs, indicates that there are only a limited number of schedules that can be generated. The combination of GP with local search procedures allows the exploration of regions of the solutions' space that the GPC algorithm is unable to reach. In this section the performance of these methods is compared with the M-NBR heuristic, which is a well-tested and well-documented optimisation method for the one-machine total tardiness problem. Tables 4.8-4.11 illustrate the statistical performance of these three algorithms on the same experimental basis that was used in the previous section.

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
M-NBR	75	0.8	0.12705	5.1136363
GLS	63	2.425	0.246131	5.9431524
GSA	64	1.7125	0.509119	19.277108

Table 4.8: M-NBR vs. GLS and GSA (*n*=12)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
M-NBR	61	9.75	0.352129	8.3456790
GLS	25	22.375	1.201713	11.059907
GSA	57	5.1375	0.988265	29.761904

Table 4.9: M-NBR vs. GLS and GSA (*n*=25)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
M-NBR	32	65.675	1.012977	9.2341356
GLS	9	78.575	1.682691	23.728813
GSA	53	9.3125	0.286414	4.2517006

Table 4.10: M-NBR vs. GLS and GSA (*n*=50)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
M-NBR	13	494.8625	2.438429	19.846647
GLS	8	454.8125	3.543847	78.971962
GSA	42	67.8	0.571563	23.809523

Table 4.11: M-NBR vs. GLS and GSA (*n*=100)

GLS performed significantly better than its parent method GPC. However, GLS could not reach the level of M-NBR performance in small and moderate problem sizes. GLS imposed 203% higher penalties for *n*=12, 129% for *n*=25 and 19% for *n*=50. However, the performance of GLS seemed to drop slower than M-NBR as the problem-size increased. For *n*=100 M-NBR had a slightly worse MADO than GLS. In addition, while t-tests for *n*=12 and *n*=25 suggested a difference between the two methods, (t=1.75, p<0.0417 and t=2.53, p<0.006 respectively), the same test on larger problem instances resulted in different conclusions (t=0.97, p<0.16 for *n*=50 and t=0.61, p<0.27 for *n*=100). On the other hand, Table 4.12 shows that M-NBR consistently found better solutions than GLS over the whole range of problems. In addition, M-NBR produced a considerably larger number of optimal solutions especially for n=25 and n=50. The consideration of these data leads to the conclusion that M-NBR performed better than GLS on the set of one-machine total tardiness problems considered in the experimentation. If computational efficiency was also to be considered, the result would significantly favour M-NBR since it required at worst (n=100) a few milliseconds of CPU time for the generation of schedules.

	Number of times GLS was better than M-NBR	**Number of times GLS was worse than M-NBR**	**Number of times GLS was equal to M-NBR**
n=12	5	13	62
n=25	11	46	23
n=50	25	46	9
n=100	29	45	6

Table 4.12: M-NBR vs. GLS in terms of non-dominated solutions

The superiority of M-NBR over GLS was not replicated with the other stochastic optimiser, GSA. In the smallest problem size considered (n=12), M-NBR had the advantage since GSA produced on average 114% higher penalties. As the problem size increased, the performance of M-NBR dropped significantly faster than GSA. For n=25 M-NBR imposed 89% higher penalties than GSA on average. Tables 4.8-4.11 illustrate the increasing difference in performance for increasing values of n. Table 4.13 contains t-test results on the null hypothesis for GSA and M-NBR. From these data it can be concluded with high confidence that GSA was significantly different from M-NBR for large values of n.

(M-NBR) – GSA		
n	t	$p(T \leq t)$
12	1.26	0.105
25	1.25	0.106
50	5.03	$1.48*10^{-6}$
100	6.34	$6.44*10^{-9}$

Table 4.13: t-test results between M-NBR and GSA

In addition, the data in table 4.14 highlight the superiority of GSA in moderate and large problem instances in terms of non-dominated solutions.

	Number of times GSA was better than M-NBR	Number of times GSA was worse than M-NBR	Number of times GSA was equal to M-NBR
n=12	5	15	60
n=25	16	21	43
n=50	10	43	27
n=100	66	7	7

Table 4.14: M-NBR vs. GSA in terms of non-dominated solutions

4.3.4.3 SA versus GSA

From the results presented in the previous paragraph, it can be concluded that GSA had the best overall performance on the set of benchmark problems used in the experimentation. While its superiority from the other methods was evident, a comparison between GSA and the simple SA algorithm, i.e. the GSA procedure without the GP seed, could not bring any safe conclusions.

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
SA	68	2.325	0.735869	15.286624
GSA	64	1.7125	0.509119	19.277108

Table 4.15: SA vs. GSA (*n*=12)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
SA	60	3.1875	0.401984	9.7142857
GSA	57	5.1375	0.988265	29.761904

Table 4.16: SA vs. GSA (*n*=25)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
SA	45	21.675	0.765335	25
GSA	53	9.3125	0.286414	4.2517006

Table 4.17: SA vs. GSA (*n*=50)

	OPTIMAL SOLUTIONS	MADO (UNITS)	MRDO (%)	MAX(RDO) (%)
SA	36	86.525	0.298231	4.3756145
GSA	42	67.8	0.571563	23.809523

Table 4.18: SA vs. GSA (n=100)

Differences in terms of MADO were relatively small as tables 4.15-4.18 indicate. SA seemed to perform slightly better in small problem instances, while GSA had an advantage for large values of n. Pairwise t-tests suggested that the null testing hypothesis should not be easily rejected (Table 4.19).

SA - GSA		
n	t	$p(T \leq t)$
12	0.64	0.26
25	1.58	0.05
50	2.4	0.009
100	1.45	0.07

Table 4.19: t-test results between SA and GSA

The comparison of performance between SA and GSA in terms of non-dominated solutions (Table 4.20) showed that the number of equal solutions remained close to 50% of the total number of problems even for n=100. This figure suggests that there was a high correlation on the performance of SA and GSA. In other words there was not enough evidence to suggest that the introduction of a five-generation evolved GPC-seed significantly improved the performance of SA.

	Number of times GSA was better than SA	Number of times GSA was worse than SA	Number of times GSA was equal to SA
n=12	8	13	59
n=25	11	17	52
n=50	25	9	46
n=100	26	18	36

Table 4.20: SA vs. GSA terms of non-dominated solutions

A different point of view can be established if the computational requirements of the algorithms are to be considered. Despite the fact that the SA algorithm used in this report was not as slow as GP, it was still much slower than M-NBR. The combination of GPC and SA resulted in small improvements in terms of computational efficiency. This was due to the initialisation of the SA procedure from a better starting point in the solutions' search space. However, the running time of the algorithm was very much problem-dependent and unpredictable.

4.4 Evolving dispatching rules using Genetic Programming

4.4.1 Introduction

Results presented in section 4.3.4.1 suggest that MON dispatching rule produces a good overall performance in a variety of one-machine scheduling problems, due to its unique design that takes in account both the processing time and due dates of individual jobs, as well as the makespan of all jobs. These three parameters are combined in Montagne's formula (see section 4.3.2.3) and create a ratio that defines the scheduling urgency of each job. Montagne constructed this formula based on his understanding of the problem. However, there is a possibility that another formula exists – perhaps more complex – that is able to utilise a priori knowledge of the problem in a way that creates better scheduling ratios than the MON dispatching rule. In this section the possibility of evolving a dispatching rule formula through a GP-framework for the solution of the one-machine total tardiness scheduling problem is investigated. The algorithm is supplied with problem-specific information and trained on various sets of tardiness problems, aiming to evolve a dispatching rule that will perform at least as good as dispatching rules produced by human intuition.

The problem can be described in the same program induction form that was presented in paragraph 4.3.1. However, while in the algorithm described in the previous section scheduling information was used indirectly through the combination of dispatching rules, in this case the algorithm directly processes raw scheduling data through the formula of the evolved dispatching rule.

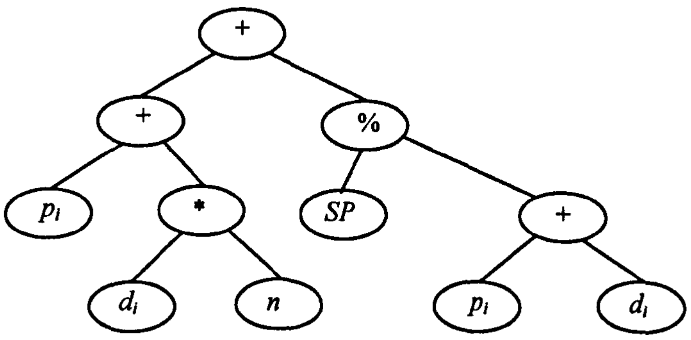
4.4.2 Design of the algorithm

4.4.2.1 Schedule representation

The proposed algorithm employs the same procedure for the generation of job schedules as the one used by dispatching rules designed by human intuition. Evolved dispatching rules comprise of combinations of variables and constants that provide scheduling information. For each job in the system, the respective scheduling data are fed in the formula of the dispatching rule, which calculates an urgency value. When all jobs have been considered, the job schedule is generated by ordering jobs in a non-decreasing order of their urgency values. Note that since the formula of the dispatching rule is decided by the evolutionary procedure, the choice of increasing or decreasing order of the urgency values is purely an issue of designer's choice and does not affect the operation of the algorithm.

In the algorithm presented in this section the traditional subtree crossover and mutation operators can be employed without the danger of producing infeasible solutions. At the same time, the size of tardiness problems that can be considered is unlimited since, once evolved, the dispatching rule operates independently of the number of jobs included in the problem.

Figure 4.5 and table 4.21 illustrate the scheduling of jobs by a potentially evolved dispatching rule on the example problem described in section 4.3.2.1. A description of the terminals used in the example program is given in section 4.4.2.3.



$$p_i + (d_i \cdot N) + \frac{SP}{p_i + d_i}$$

Figure 4.5: An example of an evolved dispatching rule

Job no.	1	2	3	4	5	6	7	8
Priority value	2203.22	2301.03	3303.68	2209.45	2827.81	2773.01	5561.08	5841.04

Corresponding job sequence: 1 - 4 - 2 - 6 - 5 - 3 - 7 - 8

Total Tardiness: 859

Table 4.21: Priority values of jobs for the example dispatching rule and corresponding schedule

4.4.2.2 Function set

The function set comprises of the four main mathematical operations: addition, subtraction, multiplication and division (+, -, *, %). The '%' symbol corresponds to the protected division function that returns the value of '1' when the value of the denominator is equal to '0'. These functions help the algorithm to create a formula of a dispatching rule for the scheduling of jobs.

4.4.2.3 Terminal Set

The terminal set of the algorithm is mainly constructed from the parameters that form Montagne's dispatching rule:

- p_i: processing time of job i
- d_i: due date of job i
- *SP*: sum of the processing time of all jobs in the problem ('makespan')

Two additional parameters were included in the terminal set:

- *SD*: sum of the due dates of all jobs in the problem
- n: total number of jobs in the problem

There was no a priori knowledge about the suitability of the additional terminals for the evolution of an optimal formula. As has already been discussed (section 4.3.2.3), the GP algorithm should, in principle, be able to disregard any terminal that is not related to the solution of the problem and create a fit program using only the relative terminals. In this case, the algorithm should be able to converge at least to the formula of MON rule, since all its elements are included in the function and terminal sets.

4.4.2.4 Fitness measure and fitness cases

The quality of evolved programs is once again measured by the amount of tardiness produced by the resulting schedules. However, since the aim of the algorithm is the evolution of a generic dispatching rule for all problem instances, a variety of test problems are used for the training of the programs. Thus, the objective of the algorithm becomes the minimisation of the sum of tardiness over the entire set of test problems that are used as fitness cases. Tardiness is measured by scheduling jobs in non-decreasing order of their priority value, as this is calculated using the formula of the evolved dispatching rule.

A set of twenty tardiness problems was employed for the training of dispatching rules in each individual GP run. Nine different sets of test problems (table 4.22) were used in the experimentation. In the first four of them the value of *n* in the training set was kept fixed (n=12, n=25, n=50 and n=100). The remaining sets comprised of twenty problems, five for each value of *n*. All problems in training sets were generated using the method described in section 4.3.3. Careful consideration was given so that different levels of tardiness (T) and tightness of due dates (R) were included in each experimental set.

Name	*n*	FITNESS CASES (PROBLEMS) PER SET-UP
SETUP12	12	20
SETUP25	25	"
SETUP50	50	"
SETUP100	100	"
SETVAR1	5×(n=12)+ 5×(n=25)+ 5×(n=50)+ 5×(n=100)	"
SETVAR2	"	"
SETVAR3	"	"
SETVAR4	"	"
SETVAR5	"	"

Table 4.22: Training sets

4.4.2.5 Additional parameters

The values of additional parameters needed for the run of the GP algorithm and a summary of the proposed methodology are described in figure 4.6.

Parameters	**Values**
Objective:	Evolve a formula of a dispatching rule for the solution of total tardiness problems
Terminal set:	p_i, d_i, *SP*, *SD*, *n*
Function set:	+, -, *, %
Population size:	200
Mutation probability:	.5
Crossover probability:	.5
Selection:	Tournament selection, size 4
Number of generations:	50
Maximum depth for crossover:	17
Maximum depth for individual generated for mutation:	4
Initialisation method:	Ramped half and half

Figure 4.6: Koza tableau for evolution of dispatching rules

4.4.3 Results

The GP framework evolved different dispatching rules for each training set that was used. The individual and cumulative performance of each rule is illustrated in table 4.23. The outlined cells in this table indicate the performance of the corresponding dispatching rule on the set of test problems that was used for its training. The rest of the cells in the same column illustrate the performance of the dispatching rule on the previously unseen test problems (validation set). The results on the latter set of problems allow us to assess the generalisation of the evolved dispatching rule since these problems were not used for its training. From table 4.23 it can be concluded that in most cases the algorithm was able to evolve dispatching rules with better overall performance than MON rule and much better performance than the EDD and SPT rules. Most evolved rules performed quite well in a very large set of validation problems (160 in total). Based on this observation it can safely be concluded that these rules did not just fit the data of the fitness cases but they contained information that was relevant to the solution of the problem.

	DISP.RULE SETUP 12	DISP.RULE SETUP 25	DISP.RULE SETUP 50	DISP.RULE SETUP 100	DISP.RULE SETVAR1	DISP.RULE SETVAR2	DISP.RULE SETVAR3	DISP.RULE SETVAR4	DISP.RULE SETVAR5	EDD	SPT	MON
SETUP12	13189	20094	16357	17073	17068	13858	14812	13795	15676	17598	17044	14916
SETUP25	68440	54380	58734	68101	69135	56292	57228	57101	57915	77523	69698	60657
SETUP50	263231	242878	200935	227332	245993	202102	203149	213328	206380	286042	264225	226565
SETUP100	1047471	1128759	868513	790374	779475	808824	806609	828507	852978	1158371	1093840	886600
SETVAR1	596043	613494	489485	452603	440238	457345	452723	453549	490575	643558	501370	472004
SETVAR2	295745	317238	248306	250549	262970	238507	239999	253411	245431	336645	355541	263629
SETVAR3	478636	493293	399584	384416	380885	380406	377472	383781	396611	522807	449199	420317
SETVAR4	487201	495474	392716	377739	377760	377886	378137	377781	397891	526316	427636	393561
SETVAR5	219631	235816	188116	184426	190791	180388	182291	193274	183361	248575	266219	204599
TOTAL	3469599	3601451	2862796	2752713	2764315	2715608	2712420	2774527	2846818	3817435	3444772	2942848

Table 4.23: Tardiness results for all evolved dispatching rules and comparison with dispatching rules produced by human intuition

However, the formulas of these dispatching rules were not as simple as the formula of the MON rule. Table 3A in Appendix presents the mathematical formulas of the nine evolved dispatching rules. All these formulas were cleared from introns (segments of code that have no effect on the outcome of the problem) and were also simplified wherever that was possible.

In order to compare the performance of a GP-evolved rule with the traditional dispatching rules used in this chapter, the rule evolved from the experimental set-up SETVAR3 (figure 4.7), which produced the best overall performance, was selected.

$$\left(\frac{\left(SD \cdot d_i \cdot (SP - n)\right) \cdot \left(\frac{n}{d_i} + \frac{SP}{n}\right)}{n \bullet SP}\right) - \left(\left(\left(4 \cdot SP\right) - \left(2 \cdot d_i\right) - \left(\frac{p_i}{d_i^2}\right) - n\right) \cdot d_i\right) + \left(\left(p_i \cdot SP\right) + \left(n^2 - d_i\right)\right) \cdot \left(p_i + (2 \cdot n)\right)$$

Figure 4.7: Dispatching rule evolved from set-up SETVAR3

This particular rule was constructed from three main terms. The first and the third term operated in favour of EDD and SPT scheduling respectively. The second term acted as a control segment that shifted the operation of the rule towards EDD or SPT scheduling according to the values of the parameters of the problem. When, for example, the due date of a job was small in comparison with the makespan, the second term of the formula produced a significant negative result that decreased the value of the ratio and therefore assigned urgency to the job. In the inverse case the value of the term was becoming less significant, thus the two big positive terms controlled the ratio.

In table 4.24 SETVAR3 is compared with EDD, SPT and MON dispatching rules in all test problems used in this section. The improvement in overall performance by using SETVAR3 was significant. MON imposed 81% higher penalties in terms of MADO, while the t-test between the two rules rejected the null hypothesis with a very high probability ($t=5.62$, $p<3.49\times^{10-8}$).

	OPT	EDD	SPT	MON	SETVAR3
TOTAL TARDINES (UNITS)	2430198	3817435	3444772	2942848	2712420
MADO (UNITS)		7706.872	5636.522	2848.056	1567.9
OPTIMAL SOLUTIONS		39	0	4	25

Table 4.24: Comparative performance of SETVAR3 for all test problems

At least 77% of the solutions produced by SETVAR3 were better or equal than those produced by alternative dispatching rules (Table 4.25).

	Number of times SETVAR3 was better	**Number of times SETVAR3 was worse**	**Number of times SETVAR3 was equal**
EDD	115	40	25
SPT	164	8	8
MON	147	30	3

Table 4.25: Performance of SETVAR3 on non-dominated solutions (all test problems)

As expected, EDD performed well in the set of problems identified by small levels of tardiness and wide range of due dates. However, when the scheduling problems in the plant were evenly distributed in terms of T and R, EDD scheduling produced the worst performance over the available dispatching rules.

The dispatching rule evolved using the experimental set-up SETVAR2, which performed almost identical to SETVAR3, is illustrated in figure 4.8.

$$\left(n \cdot p_i \cdot \left(\left(n^3 \cdot SP\right) + n - p_i\right)\right) + \left(d_i \cdot \left(SD^2 - n\right)\right) + p_i^2 + \\ + \left(2 \cdot p_i \cdot SD^2\right) - \left(\frac{d_i}{\left(SP + n\right)}\right)$$

Figure 4.8: Dispatching rule evolved from set-up SETVAR2

While there were terms with similar operation between the two rules (the second one favouring EDD scheduling and the first, third and fourth favouring SPT scheduling), there were no other easily observed similarities. An interesting feature in the operation of SETVAR2 was the control nature of the SD value in the second term. When the sum of due dates was large in comparison with the sum of the processing times, EDD scheduling was favoured, since the problem was likely to be less tardy. In the inverse case, the SPT terms became more significant, thus SPT scheduling was favoured.

The examples presented in the previous paragraphs illustrate the potential of employing genetic programming for the evolution of dispatching rules in scheduling problems. The fact that genetic programming evolves solution representations in the form of computer programs means that it is possible to understand the operation of their solution mechanism. This feature can be exploited by human experts, who may be able to identify the underlying relationships of the data and thus get significant insights on the theory of the problem. Efficient solution algorithms can be constructed as a result of this process.

As Banzhaf *et al.* (1998) indicated, it is not always possible to explain the operation of a genetic program on the problem considered. The methodology described in this chapter was able to find individual programs that outperformed conventional dispatching rules on a wide range of test problems, however, the analysis of their operation was not a straightforward task, as the previous examples have illustrated. This phenomenon raises the issue of *transparency* of evolved genetic programs, i.e.

the understanding of their operation in solving the problem considered. The conventional subtree crossover operator that was used in the previous experiments causes the *semantic disruption* of evolved genetic programs since it does not consider either the length of the exchanged subtrees or their usefulness in solving the problem considered (Watson and Parmee (1996), (1998)). As a result, subtrees that contribute highly to the fitness of a genetic program can easily be disrupted. At the same time, evolved programs can grow either so large that their evaluation process is slowed down considerably, or so small that their efficiency in solving the problem considered is limited. Watson and Parmee (1998), proposed the use of the Constrained Complexity Crossover (CCC) operator within their Distributed, Rapid, Attenuated Memory Genetic Programming (DRAM-GP) framework, which only allowed the exchange of genetic material between subtrees of similar complexity, thus restricting the length of evolved genetic programs. While their method produced more parsimonious structures, it did not have any significant effect on the transparency of evolved programs, as indicated in (Parmee and Watson (2000)).

4.5 Conclusions

In this chapter the potential use of genetic programming for the solution of the one-machine total tardiness problem was investigated. To the best of the author's knowledge, no previous effort has been made to solve static scheduling problems in a GP-framework, in contrast with alternative evolutionary computation techniques that have been extensively used for the solution of similar problems. It is difficult to evolve a permutation representation without producing infeasible solutions when subtree crossover and mutation are utilised. The variable length of genetic programs is an additional disadvantage for the evolution of a fixed-size schedule.

Two alternative ways of approaching the problem were tested during the experimental stage. First, a combination of dispatching rules was utilised as an indirect way of representing a permutation through a GP-framework. The problem of variable length was dealt by considering only the part of the program that was significant for the solution of the problem. This configuration was employed as a heuristic procedure for the solution of individual problems. While it outperformed conventional dispatching rules, near-optimal solutions were only reached when alternative heuristic procedures were employed to enhance the outcome of the GP algorithm. A particular combination of a few generations evolved GP individual and SA produced high quality solutions over the entire range of problem parameters and sizes.

In the second approach, a conventional GP-framework was employed for the evolution of a dispatching rule for the solution of the one machine total tardiness problem. A number of the evolved dispatching rules outperformed Montagne's dispatching rule in the experimental basis used in this chapter. The conclusions of this investigation can be summarised in the form of the following points:

- While the evolution of a direct permutation representation through a GP-framework is not a straightforward task, an indirect representation can be sometimes constructed based on the problem considered.
- Evolved combinations of dispatching rules can outperform individual dispatching rules in the solution of one-machine total tardiness problems.
- The hybridisation of GP with alternative search techniques can yield significant improvements in the performance of the algorithm.
- While it is possible to employ GP as a heuristic procedure for the solution of the one-machine total tardiness problem, the algorithm is quite slow in comparison with alternative search methods.
- The evolution of dispatching rule formulas is a procedure much more natural to the GP-framework. While high quality results can be achieved, the evolved individuals might be able to provide human experts with significant insights to the nature of the problem. This process can lead to the development of improved solution algorithms for individual scheduling problems.

In the following chapter, genetic programming will be employed for the solution of another difficult manufacturing optimisation problem that has received considerable research attention, the cell-formation problem.

Chapter 5

THE CELL-FORMATION PROBLEM

5.1 Introduction

A modern manufacturing system should be cost-effective, easily controllable and human friendly. Group Technology (GT) (Mitrovanov, 1966) addresses these issues comprehensively by grouping together objects that are bound by some form of similarity on all levels of the corporate structure. The implementation of GT at shop-floor level is traditionally referred to as *cellular manufacturing.*

Cellular manufacturing targets mid-volume, mid-variety production lines which are widespread in today's versatile market environment. The intuition behind cellular manufacturing is an attempt to bring the benefits of the mass-production flow-line manufacturing to batch production lines. The implementation of cellular manufacturing has been reported to result in significant benefits for the manufacturing process (Singh, 1993). Some of these benefits are:

- reduced set-up times
- reduced work-in-progress inventory
- reduced throughput times
- reduced material handling costs
- simplified scheduling
- simplified flow of products
- improved quality

The implementation of a cellular manufacturing system is a multi-stage procedure that requires the analysis of material flow in the plant (Burbidge, 1975). A specific part of this procedure, the problem of forming machine cells and part families, has attracted considerable research attention. Numerous methodologies have been proposed for its solution over the last thirty years. Many of these approaches have been successful in handling particular versions of the problem. However, the trade-off between modelling an accurate version of the manufacturing process and the resulting computational complexity of the algorithm, means that there is always a research interest in finding more efficient solution methodologies.

In this chapter, a novel genetic programming approach for the solution of simple and advanced formulations of the cell-formation problem is introduced. In addition, the possibility of evolving new similarity coefficients, which - in combination with hierarchical clustering procedures – can be used for the solution of cell-formation problems, is investigated.

The remainder of this chapter is organised as follows: In section 5.2 the cell-formation problem is defined, and some of the issues involved in its solution are discussed. A literature survey of the problem is given in section 5.3. The genetic programming approach for the solution of binary cell-formation problems is illustrated in section 5.4. An extension of the proposed methodology for the solution of advanced formulations of the cell-formation problem is presented in section 5.5. Genetic programming is employed for the evolution of new similarity coefficients in section 5.6. The conclusions are summarised in section 5.7.

5.2 Formulation of the cell-formation problem

5.2.1 Simple binary matrix formulation

The most important step in the development of a cellular manufacturing system is the creation of machine cells and associated part families. There are numerous alternative problem formulations depending on the objective of optimisation and the level of manufacturing data incorporated in the solution procedure.

In this chapter the simplest version of the cell-formation problem is mainly considered, which is usually illustrated with the help of the machine-component (m/c) matrix, $A[n \times m]$, where:

n: total number of machines in the plant
m: total number of parts in the plant

Each position in the matrix can assume two values, '0' and '1'. An entry, '1', indicates that the part of the corresponding column has an operation on the machine of the corresponding row. A '0' entry indicates the opposite ('0' entries are mostly omitted from the matrix for ease of illustration). The information provided by the m/c matrix is illustrated with the help of the following example:

It is assumed that a plant produces 5 parts using 3 machines. By analysing information from the route cards of parts, the m/c matrix illustrated in figure 5.1 is obtained.

	p1	p2	p3	p4	p5
m1	0	1	1	0	1
m2	1	0	0	1	0
m3	0	1	1	0	1

Figure 5.1: An example of a m/c matrix

The value of $A_{2,4}$ is equal to '1', thus part 4 needs an operation on machine 2. In contrast, part 4 does not need an operation in machine 1 since $A_{1,4}$ is equal to '0'.

Once the m/c matrix has been obtained, the cell-formation problem is transformed to the problem of finding a configuration with all positive entries arranged inside blocks along the *main diagonal* of the *m/c matrix*. A *diagonalised matrix* allows the easy identification of machine cells and corresponding part families. Figure 5.2 shows the diagonalised version of the example matrix that resulted by rearranging its rows and columns.

	p4	p1	p5	p3	p2
m2	1	1	0	0	0
m1	0	0	1	1	1
m3	0	0	1	1	1

Figure 5.2: The diagonalised m/c matrix

By observing the matrix it is easy to identify two independent cells, the first one comprising of machine 2 and parts 1 and 4, and the second one comprising of machines 1 and 3 and parts 2, 3 and 5. The main objective of a cell-formation algorithm in this simple version of the problem is the construction of completely independent cells, i.e. cells where the parts included in a part family are processed only within the corresponding machine cell. However, this is a case rarely encountered in practice. Figure 5.3 illustrates a situation on a different m/c matrix where the cells that have been formed are not independent.

	p4	p1	p5	p3	p2
m2	1	1	0	1	0
m1	0	0	0	1	1
m3	0	0	1	1	1

Figure 5.3: m/c matrix with intercell moves

The reason for this inefficiency is part 3, which requires an operation on a machine that belongs to a different cell (machine 2). It is customary in cellular manufacturing terminology to describe part 3 as an '*exceptional part*' and machine 2 as a '*bottleneck machine*'. The handling of bottleneck machines and exceptional parts is a significant consideration in cellular manufacturing research. Some of the approaches that have been proposed for its solution over the years are described in the following section.

When completely independent cells cannot be formed, the objective of the solution methodologies is usually expressed in terms of the minimisation of intercell moves or the minimisation of material handling costs in general. However, in the case of binary m/c matrices it is common to employ a grouping measure to assess the quality of block diagonalisation. Several grouping measures have been introduced over the years, with grouping efficiency and grouping efficacy being the ones that have been used by the majority of researchers. A detailed explanation of these measures is given in section 5.4.2.6.

The binary m/c matrix representation of the cell-formation problem is a simple and efficient representation but captures only a limited amount of manufacturing data. However, even in its simplest form, the cell-formation problem is a difficult combinatorial problem. Lee and Garcia-Diaz (1993) indicate that the number of *p*-sized non-empty partitions of *n* objects is:

$$S(n,p) \approx \frac{p^n}{p!} \tag{5.1}$$

where $S(n,p)$ is the Stirling number of the second kind. For *n*=20 the number of non-empty subsets of size 5 is approximately 7.94×10^{11}. If the number of clusters is not pre-specified, the total number of partitions becomes equal to:

$$\sum_{j=1}^{n} S(n,j)$$

This formula rules out the complete enumeration of solutions. The manual manipulation of rows and columns as a method of diagonalising the m/c matrix is only feasible in small problem instances.

5.2.2 Advanced formulations

The binary m/c matrix representation of the problem has been extensively used in cell-formation research, mainly because it was introduced and utilised by Burbidge in the first scientific method for creating manufacturing cells, namely Production Flow Analysis (Burbidge, 1971). This representation suffers from serious deficiencies that limit its ability to represent realistic manufacturing environments. More specifically, a binary m/c matrix can capture only a limited amount of manufacturing data, ignoring in that way information that can be critical in creating the appropriate cell configuration. Advanced formulations of the problem (mainly mathematical programming and graph-based models) are capable of incorporating a wide range of production data like:

- processing times
- product demands

- machine capacities
- alternative process plans for parts
- batch sizes
- limits on cell sizes and total number of cells
- operation sequences
- multiple machines of the same type
- tooling considerations

The larger the amount of data included in a formulation of the cell-formation problem, the more computationally intractable the model becomes. The objective of an advanced formulation can be as simple as the minimisation of intercell moves and as complex as the minimisation of the total costs associated with the production process over a specified period of time. The application of the GP–based methodology in a number of advanced formulations of the cell-formation problem is illustrated in section 5.5.

5.3 Literature review

5.3.1 Introduction

Research effort for the solution of the cell-formation problem spans a period of thirty years. A considerable number of papers have been published during this time making the task of surveying the field and classifying the solution approaches extremely difficult.

In this section a number of solution methodologies that are considered to be important in cellular manufacturing literature will be examined. The list of publications is by no means complete. The aim of this survey to illustrate the state-of-the-art in cell-formation research so that meaningful comparisons can be made in later sections.

There is no standard way of classifying cell-formation methods. A coarse-grained classification would result in the following categories:

- Visual inspection methods
- Coding and classification methods
- Production-based methods

Visual inspection methods or simply 'eye-balling' methods rely on the visual identification of machine cells and part families. Considerable experience is required in this process even in small problem cases. However, as the size of the problem increases the task becomes almost impossible.

In coding and classification methods the design characteristics of the parts are used for the formation of part families. Each part is assigned a multi-digit code according to its shape, size, or production requirements, and a classification system is used to group parts according to their code. While coding systems are widely used by companies, very few cell-formation methods are based on them.

The core of the cell-formation methodologies belongs to the category of production-based methods. In general, production based methods analyse the information found on the route cards of parts and bring together parts with similar processing requirements and/or machines that process similar parts. The genetic programming methodology presented in this chapter belongs to the family of production-based methods. A fine-grained classification of these methods result in the following categories:

- Array-based methods
- Hierarchical clustering methods
- Non-hierarchical clustering methods
- Graph-based methods
- Mathematical programming methods
- Meta-heuristics, fuzzy logic and neural networks

In the following paragraphs a number of important papers in each category will be discussed. The classification system used in this section is illustrated in figure 5.4.

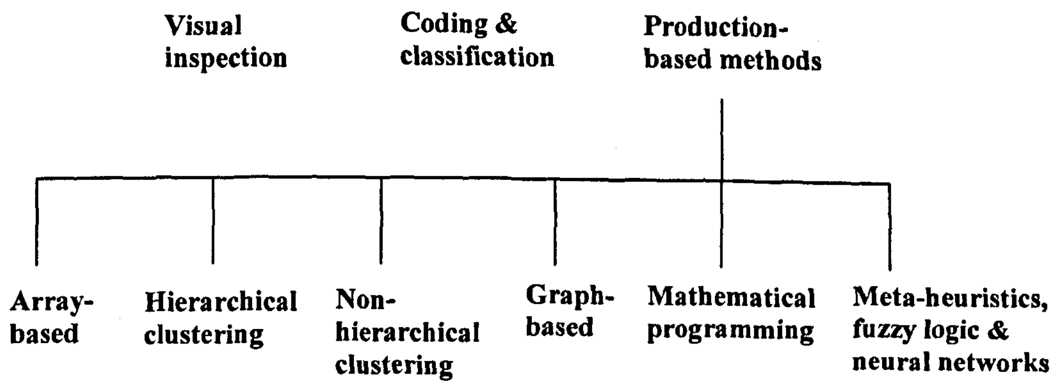


Figure 5.4: Classification of cell-formation approaches

5.3.2 Array-based methods

Array-based methods manipulate the rows and the columns of the m/c matrix aiming to obtain visible groupings of machines and components. This is usually achieved by constructing group diagonals that include as many positive entries as possible.

The first array-based method for obtaining machine-component groups was part of the Production Flow Analysis (PFA) procedure for the implementation of a cellular manufacturing system (Burbidge, 1971, 1975). PFA comprised four main steps:

- Factory flow analysis

  This step was necessary in large industries and aimed to decompose the factory in a number of independent 'major' departmental groups, making the implementation of the group analysis step easier.

- Group analysis

  This step started with the construction of the m/c matrix using information obtained from the route cards. A manual manipulation of rows and columns created machine-component groups. Burbidge believed that these groups existed naturally and it was up to the designer to unveil them. He also claimed that groups could be obtained manually even in large m/c matrices. Burbidge later presented a seven-step method for obtaining cells that was based on the concept of 'nucleus' machines. The algorithm started by eliminating 'immaterial' machines from consideration, i.e. machines that performed secondary operations, such as washing etc. Then, the machines processing the smallest number of parts were identified as 'nucleus' machines and primitive 'modules' were built around them. In the latter stages of the algorithm, final groups were identified by combining or dividing primitive modules. Burbidge proposed a number of alternative methods for the elimination of exceptional elements and the balancing of workload between the cells.

- Line analysis

  After the grouping of machines and components, the layout of machines within the cells was chosen based on the flow of parts between machines.

- Tool analysis

  During this step families of tools that processed similar parts were identified and the optimal loading sequence of tools in the machines was decided.

Group analysis received considerable research interest. Subsequently many researchers attempted to improve the efficiency of diagonalisation by introducing algorithms that were able to manipulate the rows and columns of the m/c matrix using a pre-programmed logic.

King (1980) devised a diagonalisation method called Rank Order Clustering (ROC). ROC was based on the ranking of rows and columns according to the binary word represented by the '0' and '1' entries for each of them. Rows and columns were rearranged in decreasing order of their ranking. The process was iterative and continued until no further change could be achieved. Chandrasekharan and Rajagopalan (1986) argued that the algorithm tended to gather as many positive entries as possible in the top-left hand corner of the m/c matrix, while the rest of the matrix was left highly disorganised. This tendency resulted in the erroneous identification of bottleneck machines. King and Nakornchai (1982) introduced a modified version of ROC, called ROC2. ROC2 utilised linked lists to store the data of the matrix. Linked lists enabled the use of fast and efficient sorting procedures, which resulted in an overall algorithm with linear time complexity. They also presented an interactive algorithm that combined ROC2 with specialised procedures for dealing with exceptional elements and bottleneck machines.

Chandrasekharan and Rajagopalan (1986) noted that the application of ROC always resulted in one clear-cut machine-component group that was located on the top left-hand corner of the m/c matrix. They proposed an extension of ROC called MODROC, which took advantage of this characteristic. MODROC started with the execution of two iterations of the ROC algorithm. Then, the group of machines and components that had been formed on the top left-hand corner of the matrix was recorded and the components were removed from the matrix. The procedure was repeated until no more components were left. This algorithm created mutually independent part families, but the identified machine cells intersected most of the time. The authors employed a hierarchical clustering procedure to create the final plant configuration.

Chan and Milner (1982) introduced the Direct Clustering Algorithm (DCA), a fast and efficient method for diagonalising m/c matrices. DCA employed a systematic procedure for the manipulation of rows and columns of the matrix. The aim of the procedure was to move the rows with the 'left-most' positive entries to the top of the matrix and the columns with the 'top-most' positive entries to the left of the matrix. The procedure was iterative and continued until no further improvement could be achieved. The main advantage of this method over ROC was that the initial configuration of the matrix did not affect the resulting partition. This was achieved by a pre-processing stage where the columns and rows were ranked according to the number of their positive entries. In this way, the input to the main phase of the algorithm was always the same.

A cell-formation approach that attracted considerable attention is the Cluster Identification Algorithm (CIA) of Kusiak and Chow (1987). CIA was based on a cutting algorithm, originally introduced by Iri (1968). CIA was able to identify machine cells and part families by drawing vertical and horizontal lines on the m/c matrix. The authors also introduced the Cost Analysis Algorithm (CAA), an extension of CIA that explicitly considered the cost of subcontracting parts that caused intercellular moves.

Boe and Cheng (1991) presented the Close Neighbour Algorithm, a simple but efficient heuristic procedure for the clustering of machines and parts in a binary m/c matrix. A measure of similarity for each pair of machines in the plant was calculated based on the 'closeness' of their part routings. The rows of the m/c matrix were then reorganised by bringing 'similar' machines closer together. The rearrangement of columns through a simple heuristic procedure resulted in the diagonalisation of the intermediate matrix. The authors compared the performance of the algorithm with some well-known clustering procedures on a number of test problems taken from the literature and concluded that its performance was always equal or better in terms of the grouping efficiency measure (Chandrasekharan and Rajagopalan, 1986b).

In a recent paper Da Silveira (1999) presented a methodology for the practical implementation of a cellular manufacturing system. The procedure was illustrated for the case of a toy manufacturing plant in Brazil. The benefits of the implementation in terms of reduction in scrap, rework, work-in-progress, stock, and delivery times were quite significant. Boe and Cheng's Close Neighbour Algorithm was used for the creation of machine cells and part families.

5.3.3 Hierarchical clustering methods

Hierarchical clustering methods employ some form of similarity or dissimilarity measure between machines or parts in order to create machine cells or part families. Solutions are generated by either progressively breaking down a single cell or part family to individual machines or parts (divisive methods) or by progressively merging individual machines or parts until a single cell or family has been formed (agglomerative methods).

The first author to introduce hierarchical clustering for the solution of the cell-formation problem was McAuley (1972). Since his methodology formed the basis of the genetic programming approaches introduced later in this chapter, it will be described in more detail with the help of the example problem presented in figure 5.5.

	p1	p2	p3	p4	p5
m1	1		1		
m2		1		1	1
m3	1		1		
m4	1	1		1	

Figure 5.5: Example m/c matrix for the illustration of McAuley's algorithm

The algorithm comprised of two main stages. Initially, a similarity coefficient was calculated for each pair of machines that were available in the plant. The value of the coefficient represented the similarity of machines in terms of the common operations performed. McAuley employed Jaccard's (1908) similarity coefficient which, for this particular problem, was defined as follows:

$$S_{ij} = \frac{a_{ij}}{a_{ij} + b_{ij} + c_{ij}} \tag{5.2}$$

where: S_{ij} : similarity between machines *i* and *j*

a_{ij} : number of parts processed by both machines *i* and *j*

b_{ij} : number of parts processed by machine *i* but not by machine *j*

c_{ij} : number of parts processed by machine *j* but not by machine *i*

The value of the similarity coefficient ranged from 0 to 1. For the above example the similarities were calculated as follows:

$$S_{1,3} = \frac{2}{2+0+0} = 1 \qquad S_{1,4} = \frac{1}{1+1+2} = 0.25 \qquad S_{1,2} = \frac{0}{0+2+3} = 0$$

$$S_{2,4} = \frac{2}{2+1+1} = 0.5 \qquad S_{3,4} = \frac{1}{1+1+2} = 0.25 \qquad S_{2,3} = \frac{0}{0+3+2} = 0$$

Thus, there is total similarity between machines 1 and 3, and no similarity between machines 1 and 2. The above values were used for the construction of the similarity matrix (figure 5.6).

	m1	m2	m3
m2	0	*	*
m3	1	0	*
m4	0.25	0.5	0.25

Figure 5.6: Similarity matrix for the example problem

In the second stage, the calculated similarity matrix was used for the creation of a pictorial representation of solutions, known as 'dendrogram'. Single Linkage Cluster Analysis (Sneath, 1957) was employed for the construction of the dendrogram. SLCA assumed that all machines were initially ungrouped. Then, the pair of machines having the highest value in the similarity matrix was recorded and grouped at this level of similarity. In the previous example, machines 1 and 3 were grouped at the similarity level of 1. The next highest similarity level was then found and the associated machines were merged at this level. In this case, machines 2 and 4 were merged at the similarity level of 0.5. At this point the highest similarity level was 0.25 between machines 1 and 4. Since both machines had already been grouped, their associated groups were merged as well. Thus, at the similarity level of 0.25 all machines had formed a single cell and consequently there was no reason to examine the remaining similarity coefficient values. The constructed dendrogram is illustrated in figure 5.7:

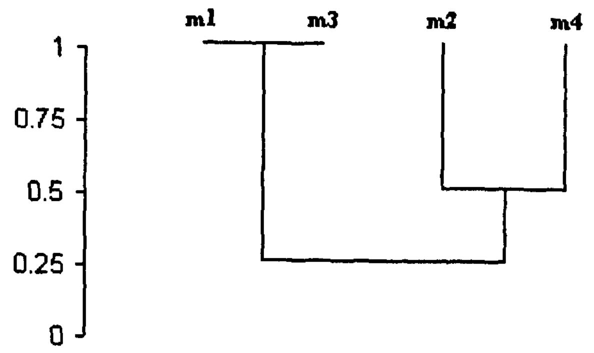


Figure 5.7: Dendrogram of solutions for the example problem

The above dendrogram contains a number of alternative solutions, depending on the choice of the threshold level (T). More specifically:

- **Solution 1** (initial) *cell 1*: m_1
  cell 2: m_2
  cell 3: m_3
  cell 4: m_4
- **Solution 2** (T=1) *cell 1*: m_1, m_3
  cell 2: m_2
  cell 3: m_4
- **Solution 3** (T=0.5) *cell 1*: m_1, m_3
  cell 2: m_2, m_4
- **Solution 4** (T=0.25) *cell 1*: m_1, m_2, m_3, m_4

The objective of McAuley's algorithm was the minimisation of the sum of material handling costs, which was calculated by adding the intercellular and intracellular handling costs under a pre-specified layout. However, since the output of the algorithm was a partition of machines into a number of cells, the algorithm could be used in conjunction with any desired objective.

The main disadvantage of SLCA was the occurrence of the 'chaining' phenomenon during the grouping procedure. Since the existence of a single linkage between machines or groups of machines was enough for the approval of merging, the algorithm could bring together machines with low similarity. In our example problem, machines 1 and 2 were grouped together at the similarity level of 0.25, while their similarity coefficient was 0. McAuley, as well as other researchers (Gupta and Seifoddini, 1990) suggested the use of alternative clustering methods like Average Linkage Cluster Analysis (ALCA) and Complete Linkage Cluster Analysis (CLCA) in order to overcome this problem. ALCA calculated the average of similarity coefficients between groups of machines. CLCA worked in the opposite way of SLCA by assigning the lowest and not the highest similarity coefficient between groups of machines. However, both ALCA and CLCA required the recalculation of the similarity matrix after each individual merging step, resulting in greater computational complexity.

Gupta and Seifoddini (1990) extended the applicability of the coefficient-based hierarchical clustering methods by introducing an enhanced version of McAuley's similarity coefficient. The authors argued that the main disadvantage of the simple similarity coefficient was the limited amount of manufacturing information considered by it. They introduced the production-based similarity coefficient that explicitly considered the production volume of parts, the part routing sequences and the processing times. The superiority of the production-based similarity coefficient over Jaccard's similarity coefficient was illustrated on test problems taken from the literature. Gupta (1993) later introduced an improved version of the coefficient that explicitly considered the existence of alternative process plans for the parts produced. Seifoddini and Djassemi (1995) also compared the performance of these two coefficients and concluded that the production-based similarity coefficient was able to reduce total material handling costs more efficiently than Jaccard's coefficient.

Vakharia and Wemmerlov (1990) presented a methodology for the creation of manufacturing cells, which was based on the identification of part families rather than machine cells. A similarity measure for parts was introduced, which considered not only the machines visited by each part, but the operation sequences as well. The merging procedure was interactive, with the designer having the power to approve or object to merging, based on the information about the resulted skip moves, backtracking moves, etc. After the part families had been created, the designer dealt with backtracking and single-operation parts that had been initially removed from consideration, and decided where to allocate 'key' equipment (equipment required by many parts in the plant). At the final stage of the procedure more general objectives were considered, like the minimisation of investment and the respect of cell-size constraints. The procedure was highly interactive since it required decisions from an experienced human operator.

The hierarchical clustering methods described so far employed agglomerative procedures for the creation of machine cells or part families: individual machines or parts were merged into progressively larger cells or families until a single cell or family was obtained or until a size constraint had been reached. Stanfel (1985) proposed a simple divisive hierarchical procedure for the creation of machine cells that was not based on the calculation of similarity coefficients. The algorithm started with an initial cell comprised of all available machines. An iterative procedure followed with each machine leaving the cell to either form a new cell or to join one that had already been formed. All the solutions were evaluated in terms of the resulting intercell moves and the number of extraneous transitions caused by the presence of machines within a cell that were not processing all the family parts.

5.3.4 Non – hierarchical clustering methods

Non-hierarchical clustering methods also employ a measure of similarity or dissimilarity for the grouping of machines or parts. However, machine cells or part families are formed around a number of seed points that are selected by a pre-specified procedure. The main drawback of these methods is that they usually require the specification of the total number of manufacturing cells in advance.

Chandrasekharan and Rajagopalan (1986b) introduced the first non-hierarchical clustering algorithm for the solution of the cell-formation problem. A modified version of MacQueen's k-means method (MacQueen, 1967) was employed for the initial clustering of machines and components. Since the k-means method required the pre-specification of the total number of clusters, a formula was derived for the calculation of the maximum number of independent cells that could be formed for a specific problem. During the second stage of the algorithm, part families were allocated to machine groups according to their efficiency factor, an indicator of the within-cell utilisation for each part family. The output of this stage was used for the determination of ideal-seed clustering points in a perfectly diagonalised matrix. These points initialised a new run of the k-means clustering algorithm that resulted in the elimination of singleton clusters. In the same paper Chandrasekharan and Rajagopalan introduced *grouping efficiency*, a qualitative measure of matrix diagonalisation. While grouping efficiency has been criticised as being inadequate to assess the performance of cell-formation algorithms, it has been used by a considerable number of researchers over the years.

The same authors (Chandrasekharan and Rajagopalan, 1987) introduced an extension of the ideal-seed clustering algorithm called ZODIAC (Zero-One Data: Ideal-seed Algorithm for Clustering). The algorithms were quite similar with the exception of the initialisation phase where ZODIAC considered a choice of different seeding procedures.

Srinivasan and Narendran (1991) illustrated some of the deficiencies of ZODIAC in terms of the choice of initial seeds and the use of the city block distance measure as the clustering criterion. They introduced a new non-hierarchical clustering procedure called GRAFICS (GRouping using Assignment method For Initial Cluster Seeds). GRAFICS identified initial machines for seeding by solving the assignment problem as introduced by Srinivasan *et al.* (1990). The maximum density rule was employed as the clustering criterion. The main algorithm progressed by alternatively clustering machines and parts until no improvement could be made in terms of the number of exceptional elements and voids (zeros inside the block diagonal matrices). GRAFICS did not allow the existence of singleton clusters. GRAFICS was tested against ZODIAC on a considerable number of problems taken from the literature and was found to be superior in most cases. Srinivasan (1994) later extended GRAFICS by using a minimum-spanning tree algorithm for the creation of initial seeds. The modified GRAFICS algorithm performed better than simple GRAFICS and ZODIAC on a wide range of problems.

Nair and Narendran (1998) presented a non-hierarchical clustering method for the creation of manufacturing cells called CASE (Clustering Algorithm for SEquence Data). They introduced a new similarity metric that explicitly considered the sequence of operations for each part, multiple visits to machines, and part demands. The metric was used for the identification of initial seeds in a non-hierarchical clustering algorithm. Nair and Narendran (1999) later presented an enhanced version of CASE called ACCORD (A bicriterion Clustering algorithm for Cell-formation using Ordinal and Ratio-level Data). ACCORD combined the similarity coefficient used in CASE with a new similarity coefficient that captured the workload similarity between any pair of machines in the plant.

5.3.5 Graph-based approaches

Graph-based methods employ a graph or network representation of the cell-formation problem and use corresponding techniques for the creation of manufacturing cells.

One of the first graph-based approaches for the solution of the cell-formation problem was introduced by Rajagopalan and Batra (1975). Their method combined graph theory and similarity coefficients. The problem was modelled with the help of a vertex-edge map. Vertices represented machines in the plant. Jaccard's similarity coefficients were calculated for each pair of machines. A pair of vertices was connected by an edge if the value of the similarity coefficient for the corresponding pair of machines was larger than a pre-specified threshold level. All the cliques (complete maximal sub-graphs) of the graphs were identified and subsequently used for the formation of hybrid machine cells. At this point machines could be present in more than one cell, thus a procedure was needed for the creation of mutually independent cells. A new graph was constructed with each vertex representing a cell and each connecting edge representing intercellular moves between the hybrid cells. The graph was partitioned with the help of a standard graph-partitioning procedure introduced by Kernighan and Lin (1970). The objective of the algorithm was the minimisation of the total number of intercell moves. The resulted partitions corresponded to the final configuration of cells in the plant.

(De Witte, 1980) combined the same combination of the hierarchical clustering procedure of McAuley with the graph-partitioning approach of Rajagopalan and Batra (1975). The machines in the plant were initially divided in three main types: primary machines (machine types of which only one unit was available), secondary machines (machine types of which multiple units were available) and tertiary machines (machine types of which enough units were available to cover every cell in the plant). A machine-to-machine combination matrix was then created based on the routings of parts and their required quantity. This matrix was utilised for the calculation of three different similarity coefficients that were fed as input to Rajagopalan and Batra's graph partitioning procedure for the creation of manufacturing cells. Primary, secondary and tertiary cells were created in a sequential manner. Finally, secondary and tertiary cells were added to primary cells to obtain the final design.

Vannelli and Kumar (1986) focused on the development of a method for finding the minimal number of bottleneck machines or parts when creating manufacturing cells. They showed that the problem was equivalent to finding the minimal cut-nodes of a bipartite graph while disconnecting it to a number of subgraphs. Since the problem was NP-complete, a heuristic procedure (Lee *et al.*, 1979) was employed for its solution. The same authors later extended their methodology by introducing the concept of weighted graphs (Kumar and Vannelli, 1987). The improved model was able to tackle cost-based problems by simply assigning costs as weights for each part in the graph.

Askin *et al.* (1991) proposed a diagonalisation method based on the Hamiltonian path representation of the problem. Initially, the distance matrix for all machine and parts in the plant was calculated. A suitably modified version of Jaccard's similarity coefficient was employed as the distance metric. The problem of rearranging the rows and columns of the m/c matrix was modelled as a graph-based Travelling Salesman Problem (TSP) with the objective of finding the shortest tour of all vertices. TSP required a cyclic solution, thus the associated Hamiltonian Path Problem (HPP) had to be considered since it did not require a return tour to the starting vertex. Graph heuristic procedures where used for the solution of both problems.

Ng (1993) introduced a minimum spanning tree methodology for the solution of the binary version of the problem. The nodes of the tree represented the rows of the matrix and the connecting arcs denoted the distance between them, i.e. the level of their dissimilarity. K machine cells were obtained by deleting the (k-1) largest arcs from the tree. A procedure was also presented for the re-assignment of parts to machine cells aiming to improve the derived partitions. A worst-case analysis of the algorithm was performed in terms of the grouping efficiency and grouping efficacy

(Kumar and Chandrasekharan, 1990) measures. Ng illustrated the deficiencies of these grouping measures and proposed the *weighted grouping efficacy* measure for the evaluation of cell-formation solutions.

The concept of minimum spanning trees was also employed by Lin et al. (1996) in an attempt to solve a more complex version of the cell-formation problem. The objective of their cost-based mathematical model was the minimisation of the sum of intercell processing costs, intracell processing costs and total cell-balance delay costs. Since the formulation was computationally intractable, a minimum spanning tree heuristic was employed for its solution. Disconnected subgraphs were created by progressively deleting arcs until no configuration could be found that resulted in lower overall costs. The method was compared with some conventional array-based methods on test problems taken from the literature and produced excellent results. In addition, a case-study application of the method was presented for a company that manufactured irrigation products.

5.3.6 Mathematical programming

Mathematical programming formulations of the cell-formation problem are capable of considering a wide range of manufacturing data. Several types of integer programming formulations have been proposed over the years, especially the last decade. Most of these models suffer from computational intractability and require the a priori specification of the total number of manufacturing cells.

Kusiak (1987) was one of the first researchers to propose the use of mathematical programming for the solution of the cell-formation problem. He introduced a p-median zero-one integer programming model for the formation of part families. The objective of the model was the maximisation of similarity of parts within the part families in terms of the common machines used. A standard integer programming package (LINDO) was employed for the solution of the problem. Kusiak additionally presented a generalised zero-one integer programming formulation that considered the existence of alternative process plans for each part. Shtub (1989) proposed a Generalised Assignment Problem (GAP) formulation that was equivalent to Kusiak's p-median formulation.

Choobineh (1988) presented a two-stage procedure for the design of a cellular manufacturing system. First, a hierarchical clustering algorithm was employed for the creation of part families. The similarity between parts was calculated through an enhanced version of Jaccard's similarity coefficient. The existence of alternative process plans was also addressed during this stage of the procedure. After the creation of part families, a linear integer programming formulation of the problem was presented. The objective of the model was the minimisation of the sum of production costs and the costs of acquiring and maintaining machine tools.

Wei and Gaither (1990) introduced a zero-one integer programming formulation that explicitly considered the available capacity of machines. The objective of the model was the minimisation of opportunity costs in the form of subcontracting costs for exceptional parts. The authors indicated a number of alternative objectives that could be used in conjunction with the proposed mathematical formulation.

Boctor (1991) presented a simple zero-one integer programming formulation of the problem that considered only the data available from the binary m/c matrix. The objective was the minimisation of the total number of exceptional elements. An efficient procedure for the linearisation of the objective function was proposed. In addition, Boctor showed that a large number of integrality constraints could be relaxed without affecting the binary outcome of the solution. Even with these modifications the model was computationally intractable for large problem instances. Boctor proposed the use of simulated annealing for these cases.

Zhu et al. (1995) introduced a zero-one integer programming formulation of the problem with the objective of maximising the opportunity costs associated with all the parts manufactured within the system, i.e. the parts that do not need to be subcontracted. The authors showed that their formulation resulted in a smaller number of variables and constraints and in faster computational times than the corresponding formulation of Wei and Gaither (1990).

Finally, Cheng *et al.* (1996a) proposed a simple zero-one quadratic assignment formulation of the problem based on the information available from the binary m/c matrix. The objective of the model was the minimisation of the sum of Hamming distances between machines within the cells. A truncated-tree heuristic algorithm was employed for the solution of the problem. The authors extended their formulation to allow for the existence of multiple machines of the same type.

5.3.7 Meta-heuristics, fuzzy logic and neural networks

The evolutionary computation research for the solution of the cell-formation problem has been described in detail in section 2.6.2. In this section a number of solution methodologies that are based on alternative forms of meta-heuristics, fuzzy logic and neural networks will be described.

Sofianopoulou (1997) introduced an efficient mathematical programming formulation of the problem that did not require the pre-specification of the total number of cells in the plant. The objective of the model was the minimisation of the total intercell traffic. A simulated annealing algorithm was employed for the solution of the problem. Sofianopoulou (1999) later extended the use of simulated annealing to cell-formation problems with alternative process plans and duplicate machines. She presented non-linear mathematical programming formulations for the machine allocation and part allocation problems respectively. Since the models were computationally intractable, a novel simulated annealing procedure was proposed for their solution. The procedure had the ability to move simultaneously in two different search dimensions. The algorithm started with a random allocation of parts to process plans. Then, the algorithm searched for the machine-cell configuration that minimised the number of intercell moves, given the part-process allocation. Once the termination criterion had been reached, a new part-process allocation was randomly created and the same procedure was repeated until a global termination criterion was reached. The efficiency of the proposed methodology was illustrated on several test problems taken from the literature.

Aljaber *et al.* (1997) modelled the cell-formation problem using a pair of shortest spanning path problems, one for the machines (rows) and one for the parts (columns) of the m/c matrix. A modified version of Jaccard's similarity coefficient was employed for the calculation of distances between pairs of machines or parts. The authors introduced a tabu search methodology for the solution of both problems. The algorithm was able to accommodate the consideration of additional manufacturing data with a suitable modification of the distance measure used.

Vakharia and Chang (1997) presented both a simulated annealing and a tabu search methodology for the solution of a detailed version of the cell-formation problem. A considearble amount of manufacturing data were included in the formulation of the problem, such as transportation costs and processing times. The objective was the minimisation of machine investment and material handling costs. A comparison of the proposed algorithms on a number of problems showed that simulated annealing outperformed tabu search both in terms of solution quality and computational complexity.

Chu and Hayya (1991) indicated that there was a degree of uncertainty in the allocation of a part to a specific part family. This uncertainty could be expressed with the help of fuzzy sets. For each part processed a degree of membership was defined in relation to each part family. The authors employed the generalised fuzzy c-means algorithm for the clustering of parts. The advantage of this methodology was that it provided the designer with a number of alternative solutions.

Kao and Moon (1998) employed the concept of memory association for the solution of the cell-formation problem. The intuition behind their approach was to simulate the association procedure that takes part in the memory of a production engineer who is faced with the task of creating manufacturing cells. The methodology was comprised of two main stages: First, an autoassociative neural network formed part families by considering the characteristics (features) of the parts. Then, a heteroassociative neural network created machine cells by considering the relation between machines and part features. In addition, an extension of the latter network was introduced that was able to create groups for other important GT domains, like tool sets, canned cycles etc.

5.4 A genetic programming-based methodology for the solution of binary cell-formation problems

5.4.1 Introduction

In this section a novel methodology for the solution of binary cell-formation problems is presented. The methodology is based on the combination of genetic programming and the Single Linkage Cluster Analysis algorithm described in section 5.3.3 (the proposed framework will be identified as 'GP-SLCA' from this point onwards). GP-

SLCA acts as an optimiser for specific cell-formation problems, i.e. only one fitness case is considered during the evaluation phase of the algorithm.

The application of genetic programming for the solution of any optimisation problem requires the restatement of the problem in a program induction form. In the cell-formation case, the problem can be restated in the following form: "Find a computer program that takes as input information about the similarity of processing operations between machines and produces as output a grouping of machines and parts that minimises a pre-specified objective" (figure 5.8).

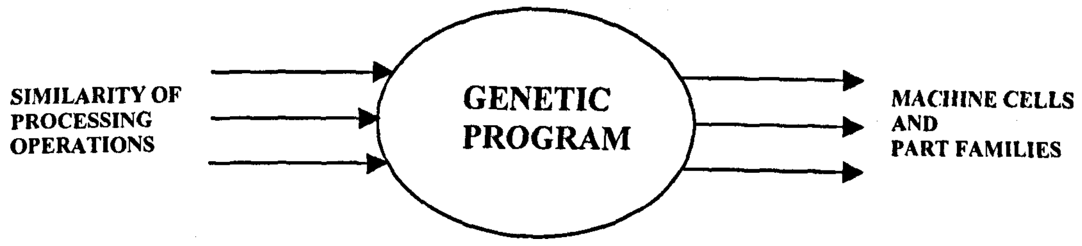

SIMILARITY OF PROCESSING OPERATIONS
GENETIC PROGRAM
MACHINE CELLS AND PART FAMILIES

Figure 5.8: Genetic programming approach to cell-formation

This definition forms the basis of the solution methodology presented in the following paragraphs.

5.4.2 Design of the algorithm

5.4.2.1 Generation of machine-cells and part-families using the GP-SLCA algorithm

The application of McAuley's SLCA procedure for the solution of cell-formation problems is inflexible since the same dendrogram of solutions is always produced irrespective of the optimisation objective. In addition, as Sarker (1996) has indicated, the performance of similarity coefficients varies significantly with both the problem domain and the optimisation objective.

The proposed methodology replaces Jaccard's coefficient with a GP algorithm that generates a variety of similarity coefficients and consequently a variety of potential solutions. The evolutionary part of GP ensures that this procedure is not just a random search through the population of potential coefficients, since coefficients that produce promising machine groupings (in relation to the optimisation objective) are more likely to survive in subsequent generations. Genetic programming evolves similarity coefficients in the form of computer programs that take as input similarity information for the problem considered and output similarity values for pairs of machines to the SLCA procedure. A detailed description of the interaction between GP and the SLCA algorithm is presented in figure 5.9.

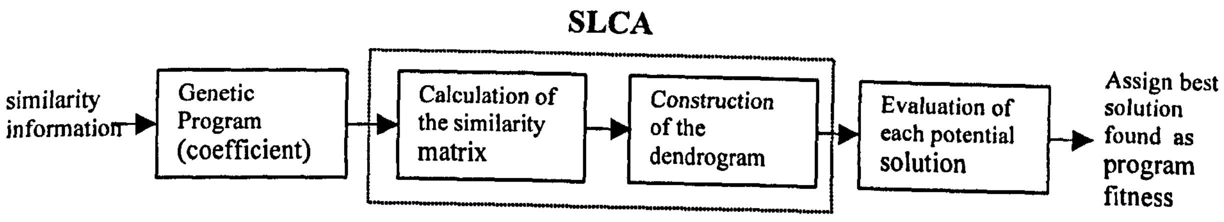

SLCA
similarity information
Genetic Program (coefficient)
Calculation of the similarity matrix
Construction of the dendrogram
Evaluation of each potential solution
Assign best solution found as program fitness

Figure 5.9:Illustration of the GP-SLCA procedure

The operation of the GP-SLCA procedure in pseudo-code form is presented in figure 5.10.

```
Procedure Main
initialise population of randomly created similarity coefficients
run procedure SLCA for each coefficient
loop
    loop
        select individuals for crossover or mutation
        apply genetic operators and form new coefficients
    until a new generation has been formed
    run procedure SLCA for each coefficient
until termination criterion is true

Procedure SLCA
compute similarity matrix
construct dendrogram
loop
    create machine cells for the highest level of similarity coefficient
    assign parts to machine cells
    calculate the fitness value of the cell configuration
    if solution is the best recorded so far, best=current solution
until a single cell has been formed
assign the best solution found as fitness of the individual
```

Figure 5.10 : GP-SLCA procedure in pseudo-code form

5.4.2.2 Allocation of parts

The methodology described in this section belongs to the category of cell-formation methods that group machines into cells and not parts into families. Thus, for each machine-cell configuration created from the dendrogram of potential solutions, the corresponding part families must be formed in order to be able to calculate the value of the objective function. Since no information is available about the sequencing of operations (backtracking, skips, etc.), parts are assigned to the cell where the majority of their operations take place. In case of a tie, the part is assigned to the smallest of the candidate cells. In that way, the number of voids created by the assignment procedure is always minimal. If there is still a tie, the part is assigned randomly to one of the candidate cells. If the allocation of parts to machine cells results in the creation of an empty cell (a cell that processes no parts), then the fitness of the solution is set to zero. However, there is no limit on the size of machine cells, and consequently no limit on the total number of cells in the plant. If required, the algorithm has the ability to explicitly consider size constraints, as it will be illustrated in section 5.5.

5.4.2.3 Function set

The set of the four standard arithmetic operations {+, -, ×, %} is sufficient for the production of a wide range of formulas of similarity coefficients. Note that the division operator '%' corresponds once again to the protected division function that returns the value of '1' if the denominator of the division is equal to '0'. In that way the closure property of the set is not violated.

5.4.2.4 Terminal set

The following four terminals were used for the construction of potential similarity coefficients:

a_{ij} : number of parts processed by both machines i and j

b_{ij} : number of parts processed by machine i but not by machine j

c_{ij} : number of parts processed by machine j but not by machine i

d_{ij} : number of parts processed by neither machine i nor machine j

With the exception of d_{ij} the same variables are present in the formula of Jaccard's similarity coefficient. McAuley did not include this independent variable in his SLCA algorithm since its value is usually quite high, thus it created very small values of the coefficient. In the GP-SLCA case, the structure of evolved similarity coefficients is not known in advance and neither is the significance of d_{ij} in the construction of a fit partition. In principle, the evolutionary procedure should be robust enough to leave out of the final solution any terminal (variable) that is irrelevant to the solution of the problem. The range of values for the evolved similarity coefficient is not known in advance. However, this does not change the operation of the SLCA procedure, since the latter does not require a specific range of similarity values. For ease of illustration a function that normalises the values of the similarity matrix within the region [0, 1] was included in the algorithm. The operation of SLCA on either matrix yields the same result.

5.4.2.5 Genetic operators

The subtree crossover and subtree mutation operators were employed for the creation of genetic diversity in the population of coefficients. These operators were applied with a probability of 90% and 10% respectively in each generation.

5.4.2.6 Objective function

Two different fitness measures were used for the evaluation of the similarity coefficients. Both measures assess the quality of block diagonalisation and have been extensively used by researchers to illustrate the performance of cell-formation algorithms. The following notation is essential for understanding the calculation of the fitness measure (notation taken from Ng (1993)):

n: total number of columns (parts)
m: total number of rows (machines)
e: total number of non-zero entries in the m/c matrix
e_1: total number of non-zero entries inside the diagonal blocks
e_0: total number of non-zero entries outside the diagonal blocks
(exceptional elements)
e_v: total number of zero entries inside the diagonal blocks (voids)

d_1: total number of elements inside the diagonal blocks

d_0: total number of elements outside the diagonal blocks

The grouping efficiency, η, of a diagonalised matrix is calculated using formula (5.3) (Chandrasekharan and Rajagopalan, 1986b).

$$\eta = q \cdot (e_1/d_1) + (1-q) \cdot \left[1 - \left(\frac{e_0}{d_0}\right)\right] \tag{5.3}$$

where: $0 \leq q \leq 1$

In the case of a single cell configuration d_0 is equal to 0, thus (5.4) should be used instead.

$$\eta = \frac{q \cdot e}{(m \cdot n)} + (1-q) \tag{5.4}$$

Grouping efficiency has two main drawbacks:

- If the value of the weight q used in the calculation of (5.3) and (5.4) is set to 0.5, it can be shown that the elimination of voids becomes much more important than the elimination of exceptional elements (Ng, 1993). However, in practical situations, exceptional elements are more costly to handle. A remedy that has been suggested is to set the value of the weight to 0.2, however, the majority of reported results have been calculated with q=0.5. For this reason the same value was employed in our experimental set-up.
- The value of grouping efficiency is always greater than 75%, independent of the structure of the diagonalised matrix. It is thus not a good reflection of the real quality of diagonalisation.

The grouping efficacy measure, Γ, is calculated using the formula (5.5) (Kumar and Chandrasekharan, 1990).

$$\Gamma = 1 - \frac{e_v + e_0}{e + e_v} = \frac{e - e_0}{e + e_v} \tag{5.5}$$

Grouping efficacy has been used by a considerable number of researchers, since it is considered to be a better grouping measure than grouping efficiency. However, as Ng (1993) proved, it assigns excessive importance to the elimination of exceptional elements. He suggested a weighted version of the measure (weighted grouping efficiency, γ), which is calculated using formula (5.6).

$$\gamma = \frac{q \cdot (e - e_0)}{q \cdot (e + e_v - e_0) + (1 - q) \cdot e_0} \tag{5.6}$$

where: $0 \le q \le 1$

If q is set to 0.5, then $\gamma = \Gamma$. Ng showed that if q is set to 0.2, γ assigns realistic importance to the existence of exceptional elements and voids in the m/c matrix.

The proposed methodology is flexible enough to work with any objective function chosen by the user (for a review of the grouping measures in cellular manufacturing, see (Sarker and Mondal, 1999)). The only consideration should be that the available information is sufficient for the calculation of the objective value.

5.4.2.7 Additional parameters

The values of the additional genetic programming parameters and a summary of the proposed methodology are presented in the Koza-tableau of figure 5.11.

Parameters	**Values**
Objective:	maximisation of the grouping efficiency or grouping efficacy of a diagonalised matrix
Terminal set:	a, b, c, d (defined earlier)
Function set:	+, -, ×, %
Population size:	500
Subtree crossover probability:	.9
Subtree mutation probability:	.1
Selection:	Tournament selection, size 7
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Figure 5.11: Koza tableau for the GP-SLCA methodology

5.4.2.8 Illustration of the GP-SLCA procedure using an example problem

The purpose of this section is to illustrate the operation of the methodology with the help of a well-known binary cell-formation problem, the (16×43) m/c component matrix, originally introduced by Burbidge (1975), which is presented in figure 5.12.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
1																																					1					1	
2		1								1																		1				1					1	1		1		1	
3							1										1																	1	1	1							
4					1				1					1					1		1		1						1														
5					1			1	1					1	1	1			1		1		1						1				1								1		1
6	1	1				1	1	1				1	1	1			1		1				1									1	1	1			1		1	1		1	1
7	1												1												1																		
8	1	1	1					1	1		1	1			1				1	1	1		1	1			1	1			1						1	1			1		1
9		1		1						1								1										1				1					1	1		1		1	
10	1											1	1												1	1					1								1				
11			1						1											1				1			1			1													
12											1											1		1			1			1													
13			1																					1																			
14		1				1											1																		1								
15					1									1					1		1												1								1		1
16		1					1			1								1														1					1	1				1	

Figure 5.12: 16×43 example cell-formation problem

Almost every algorithm for the solution of binary cell-formation problems has been applied to this test problem.

20 runs of the GP-SLCA methodology were performed on the problem for each of the objectives. The results of the experiments are summarised in tables 5.1 and 5.2:

Best value of efficiency recorded	**0.935761**
Number of times this value was found	**2**
Mean best value of efficiency per run	**0.932657**
Standard deviation	**0.003324**

Table 5.1: GP-SLCA grouping efficiency results for Burbidge's (16×43) test problem

Best value of efficacy recorded	**0.567901**
Number of times this value was found	**5**
Mean best value of efficacy per run	**0.566063**
Standard deviation	**0.003658**

Table 5.2: GP-SLCA grouping efficacy results for Burbidge's (16×43) test problem

The similarity coefficient that produced the best value of grouping efficacy (when fed into the SLCA algorithm), was evolved in the form presented in figure 5.13 (in LISP symbolic language with prefix order of execution). The equivalent formula for the similarity coefficient is illustrated in figure 5.14.

```
((-(+(/(d)(b))(-(+(d)(a))(/(a)(b))))(-(-(*(+(a)(a))(-
(b)(a)))(/(*(a)(d))(+(b)(c))))(/(*(+(a)(a))(-(b)(a)))(-(+(*(-
(+(a)(a))(*(c)(/(+(/(*(a)(b))(/(d)(b)))(d))(*(a)(a)))))(d))(/(+(/(-
(+(d)(a))(/(/(a)(b))(-
(d)(b))))(/(+(a)(b))(d)))(a))(a)))(/(+(+(d)(c))(d))(*(a)(a)))))))
```

Figure 5.13: Similarity coefficient that produced the best value of grouping efficacy (in computer program form)

$$a+d+\frac{d}{b}-\frac{a}{b}-2a(b-a)+\left(\frac{ad}{b+c}\right)+\frac{2a(b-a)}{d\left[2a-\dfrac{c\left(\dfrac{ab}{d/b}+d\right)}{a^2}\right]-\dfrac{2d+c}{a^2}+\dfrac{\left[a+\dfrac{a+d-\left(\dfrac{a/b}{d-b}\right)}{(a+b)/d}\right]}{a}}$$

Figure 5.14: Formula of the similarity coefficient that produced the best value of grouping efficacy

It is not expected that genetic programming will evolve programs which human operators will find easy to understand how and why they work. However, in the case presented in figure 5.14 it can be seen that the values of a and d, which indicate a similarity of processing operations between a pair of machines, dominate the outcome of the formula. High values of a and d will result in high values for the similarity coefficient. It should be noted that the evolved coefficient is not necessarily ideal for the solution of any other cell-formation problem, since its evolution was based solely on its performance on the particular Burbidge's test problem.

The diagonalised matrix is derived from the evolved coefficient through the Single Linkage Clustering Analysis procedure, which proceeds as follows: The formula of figure 5.14 is used for the calculation of similarities for each pair of machines in the plant, resulting in the similarity matrix of figure 5.15. The corresponding normalised similarity matrix, in which all similarity values have been scaled to the region [0, 1], is presented in figure 5.16.

	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12	m13	m14	m15
m2	56.6724														
m3	54.0000	33.7500													
m4	51.0000	31.5000	37.2000												
m5	42.0000	24.7500	30.0000	169.7049											
m6	36.8271	57.9114	44.1750	24.2409	11.1468										
m7	57.0000	36.0000	42.0000	37.7143	29.0769	-31.5418									
m8	45.1000	30.5500	21.6000	41.5745	84.2238	-26.7635	30.9998								
m9	51.2545	300.7515	33.6000	29.7143	21.5385	-61.2140	40.0000	-71.3682							
m10	51.0000	31.5000	37.2000	33.1429	24.7692	-58.3231	84.0851	-58.5406	28.6000						
m11	52.5000	32.6250	38.4000	29.8161	7.4648	18.9474	45.3333	-64.3697	29.7000	34.2857					
m12	54.0000	33.7500	39.6000	35.4286	26.9231	20.0000	46.6667	-56.1498	30.8000	35.4286	69.6667				
m13	58.5000	37.1250	43.2000	38.8571	30.1538	23.1579	50.6667	-34.6101	34.1000	38.8571	58.2819	48.3946			
m14	55.5000	28.6247	59.7232	36.5714	28.0000	483.5966	48.0000	-13.1068	20.7166	36.5714	38.5000	40.8000	55.5000		
m15	51.0000	31.5000	37.2000	76.5809	89.8009	-58.3231	44.0000	-67.7242	28.6000	33.1429	35.0000	37.2000	51.0000	40.0000	
m16	56.6724	149.8667	36.3173	32.0000	23.6923	-54.6544	42.6667	-59.8156	159.2070	32.0000	33.8333	36.0000	49.5000	42.5328	32.000

Figure 5.15: Similarity coefficient matrix

	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12	m13	m14	m15
m2	0.2307														
m3	0.2259	0.1894													
m4	0.2205	0.1654	0.1956												
m5	0.2043	0.1732	0.1827	0.4344											
m6	0.1950	0.2330	0.2082	0.1723	0.1487										
m7	0.2313	0.1935	0.2043	0.1966	0.1810	0.0718									
m8	0.2099	0.1836	0.1675	0.2035	0.2804	0.0804	0.1845								
m9	0.2210	0.6705	0.1891	0.1821	0.1674	0.0183	0.2007	0.0000							
m10	0.2205	0.1654	0.1956	0.1883	0.1732	0.0235	0.2801	0.0231	0.1801						
m11	0.2232	0.1874	0.1978	0.1823	0.1421	0.1627	0.2103	0.0126	0.1621	0.1904					
m12	0.2259	0.1894	0.2000	0.1924	0.1771	0.1646	0.2127	0.0274	0.1841	0.1924	0.2541				
m13	0.2340	0.1955	0.2064	0.1986	0.1829	0.1703	0.2199	0.0662	0.1900	0.1986	0.2336	0.2158			
m14	0.2286	0.1802	0.2362	0.1945	0.1791	1.0000	0.2151	0.1050	0.1659	0.1945	0.1980	0.2021	0.2286		
m15	0.2205	0.1854	0.1956	0.2668	0.2904	0.0235	0.2079	0.0066	0.1801	0.1883	0.1917	0.1956	0.2205	0.2007	
m16	0.2307	0.3986	0.1940	0.1863	0.1713	0.0301	0.2055	0.0208	0.4155	0.1863	0.1896	0.1935	0.2178	0.2052	0.1863

Figure 5.16: Normalised similarity coefficient matrix

Either of these matrices can be used as input to the SLCA algorithm, since they produce the same dendrogram of potential solutions. Figure 5.17 illustrates the resulting dendrogram, which has been cut at the similarity level of 0.2340. The cell configuration for this instance of the dendrogram (after the allocation of parts to families through the procedure described in 5.4.2.2.), corresponds to the diagonalised matrix of figure 5.18, which has a grouping efficacy value of 0.5679.

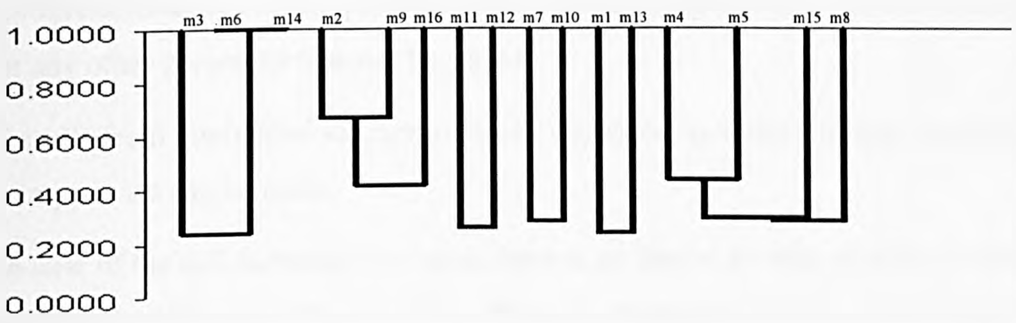
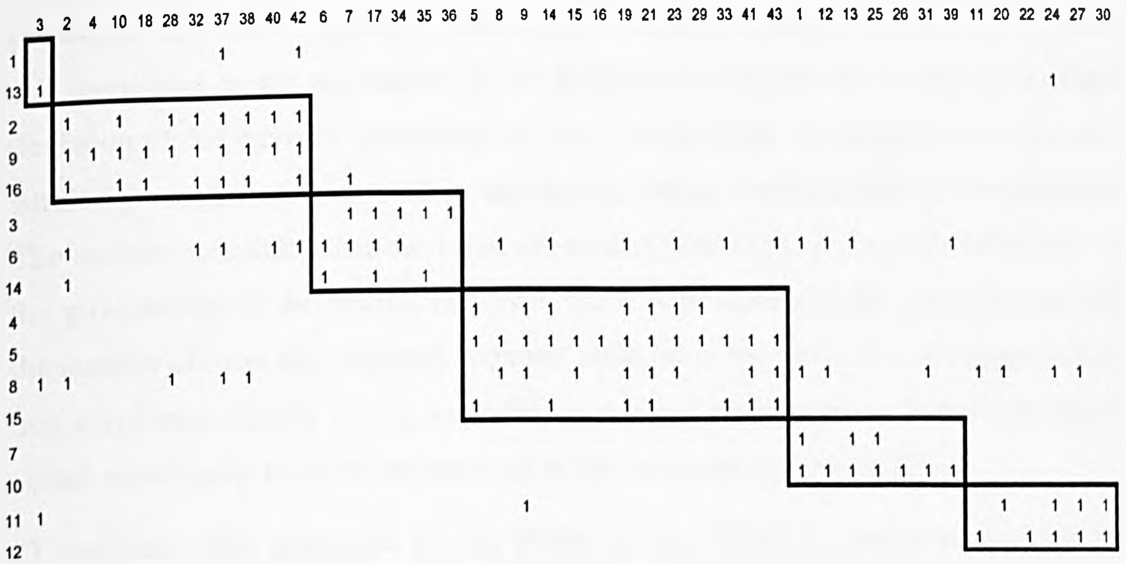


Figure 5.17: Dendrogram cut at the similarity level 0.2340



	3	2	4	10	18	28	32	37	38	40	42	6	7	17	34	35	36	5	8	9	14	15	16	19	21	23	29	33	41	43	1	12	13	25	26	31	39	11	20	22	24	27	30
1								1			1																																
13	1																																								1		
2		1		1		1	1	1	1	1	1																																
9		1	1	1	1	1	1	1	1	1	1																																
16		1		1	1		1	1	1		1		1																														
3													1	1	1	1	1																										
6		1					1	1		1	1	1	1	1	1				1		1			1		1		1		1	1	1	1				1						
14		1										1		1		1																											
4																		1		1	1			1	1	1	1																
5																		1	1	1	1	1	1	1	1	1	1	1	1	1													
8	1	1				1		1	1										1	1		1		1	1	1			1	1	1	1				1		1	1		1	1	
15																		1			1			1	1			1	1	1													
7																															1		1	1									
10																															1	1	1	1	1	1	1						
11	1																			1																			1		1	1	1
12																																						1		1	1	1	1

Figure 5.18: Diagonalised matrix (grouping efficacy)

Comparative results presented in section 5.4.4 indicate that this value of grouping efficacy is one of the best that have been reported in the literature for the particular problem considered.

In the following paragraphs the performance of the proposed methodology is illustrated on a considerable number of test problems taken from the literature.

5.4.3 Experimental basis

Finding a set of suitable problems for the evaluation of the performance of an optimisation method is always a difficult task. The main requirements that a representative set of test problems should fulfil are the following:

- Different instances of the problem should be included in terms of size, difficulty or any other parameter that can be varied
- Results from alternative solution methods should be available, so that meaningful comparisons can be made.

In the case of the cell-formation problem, there is no formal definition of the difficulty for a particular instance of the problem. The only attempt to express the difficulty of binary cell-formation problems has been made by Chandrasekharan and Rajagopalan (1989). In their investigation, they employed ZODIAC (Chandrasekharan and Rajagopalan, 1987) for the solution of a set of progressively more difficult test problems. The value of grouping efficiency for each of these problems was recorded and associated to the parameters of the matrices. The mean value and the standard deviation of the pairwise similarities of rows and columns as measured by Jaccard's similarity coefficient were used as the discriminating characteristics of the matrices. The authors concluded that the value of standard deviation was a good indication of the groupability of the matrix. However, since other discriminating parameters, like the number of rows and columns, were not included in the study, it was suggested that this conclusion should not be regarded as nothing more than an indication of the actual relationship between the structure of the matrix and its groupability.

27 problems were employed for the testing of the GP-SLCA methodology. All the problems were taken from the cellular manufacturing literature and results from alternative cell-formation methods were available. The size of the problems ranged from 10×15 to 40×100. All these problems along with their characteristics and their corresponding references are described in table 5.3. The number located in the first column of the table will be used for the identification of these problems from this point onwards. For readers interested in using the same test problems for their own comparisons, it should be noted that problems 1-8 correspond to problems 1-6, 8, 9 in the order presented by Boctor (1991), and problems 16-21 correspond to problems 1-3, 5-7 in the order presented by Chandrasekharan and Rajagopalan (1989).

No.	Reference	Size	e
1	Boctor (1991)	16×30	121
2	"	16×30	106
3	"	16×30	92
4	"	16×30	111
5	"	16×30	107
6	"	16×30	101
7	"	16×30	114
8	"	16×30	118
9	Boe and Cheng (1991)	20×35	153
10	Burbidge (1975)	16×43	126
11	Carrie (1973)	20×35	136
12	Chan and Milner (1982)	10×15	46
13	Chandrasekharan and Rajagopalan (1987)	40×100	420
14	Chandrasekharan and Rajagopalan (1986)	8×20	91
15	"	8×20	61
16	Chandrasekharan and Rajagopalan (1989)	24×40	131
17	"	24×40	130
18	"	24×40	131
19	"	24×40	131
20	"	24×40	131
21	"	24×40	130
22	Kumar et al. (1986)	23×20	113
23	Kumar and Vannelli (1987)	30×41	128
24	Seifoddini (1989)	11×22	78
25	Stanfel (1985)	14×24	61
26	"	30×50	154
27	"	30×50	167

Table 5.3: Test problems used for the evaluation of the GP-SLCA methodology

The GP-SLCA framework was applied to all test problems using each of them as an individual fitness case. Two different optimisation objectives were utilised, grouping efficiency and grouping efficacy. Twenty runs of GP-SLCA were conducted for each problem and each individual objective.

5.4.4 Results

Cumulative results are presented in two parts. First, in table 5.4, detailed results of the GP-SLCA procedure are illustrated. Then, in tables 5.5 and 5.6, the best solution evolved by GP-SLCA is compared with a number of solutions that have been produced by alternative cell-formation methodologies for the same test problems.

Pr. No.	max η	$\overline{\eta}$	σ (s.dev)	e_0	e_v	No. of cells	max Γ	$\overline{\Gamma}$	σ (s.dev)	e_0	e_v	No. of cells
1	0.917	0.909	0.0051	71	0	10	0.509	0.503	0.0063	40	38	5
2	0.935	0.903	0.0146	56	0	10	0.618	0.618	0	22	30	6
3	0.952	0.949	0.0024	44	0	10	0.7	0.7	0	8	28	4
4	0.926	0.924	0.0007	64	0	10	0.496	0.493	0.0029	31	40	6
5	0.930	0.926	0.0032	52	1	9	0.727	0.727	0	11	25	4
6	0.938	0.933	0.0023	54	0	11	0.782	0.782	0	18	18	5
7	0.930	0.926	0.0020	60	0	10	0.595	0.590	0.0063	23	39	4
8	0.927	0.913	0.0058	54	1	9	0.774	0.774	0	12	19	4
9	0.930	0.915	0.0107	90	0	13	0.568	0.568	0	40	46	5
10	0.940	0.933	0.0033	77	0	13	0.568	0.566	0.0037	34	36	6
11	0.944	0.916	0.0133	62	1	11	0.767	0.766	0.0016	11	27	6
12	0.96	0.96	0	0	4	3	0.92	0.92	0	0	4	3
13	0.964	0.956	0.0029	88	17	13	0.840	0.840	0	36	37	10
14	0.788	0.788	0	51	0	4	0.587	0.587	0	27	18	2
15	0.958	0.958	0	9	0	3	0.852	0.852	0	9	0	3
16	1	1	0	0	0	7	1	1	0	0	0	7
17	0.967	0.964	0.0040	31	3	10	0.851	0.851	0	10	11	7
18	0.953	0.940	0.0067	51	3	12	0.735	0.735	0	20	20	7
19	0.961	0.960	0.0011	70	0	16	0.533	0.531	0.0029	50	21	11
20	0.961	0.957	0.0042	70	0	16	0.479	0.476	0.0020	63	11	13
21	0.930	0.919	0.0045	78	3	18	0.437	0.435	0.0016	61	28	11
22	0.846	0.784	0.0291	66	8	9	0.490	0.453	0.0179	43	30	5
23	0.975	0.968	0.0079	59	0	19	0.607	0.607	0.0011	46	7	16
24	0.917	0.917	0	28	1	6	0.731	0.731	0	10	15	3
25	0.957	0.954	0.0008	26	0	10	0.718	0.718	0.0010	10	10	7
26	0.963	0.960	0.0063	70	2	20	0.594	0.583	0.0063	53	16	14
27	0.966	0.944	0.0205	96	0	22	0.5	0.488	0.0044	75	17	15

Table 5.4: Performance of GP-SLCA on individual test problems

Pr.No.	GP-SLCA	ZODIAC *(Chandr. & Raj., 1987)*	GRAFICS *(Sriniv.& Naren., 1991)*	MST-GRAFICS *(Srinivasan et al., 1994)*	MST *(Ng, 1993)*	GA-TSP *(Cheng et al., 1998)*
1	0.917	0.643	0.772	0.846	-	-
2	0.935	0.795	0.816	0.810	-	-
3	0.952	0.858	0.869	0.858	-	-
4	0.926	0.586	0.764	0.730	-	-
5	0.930	0.881	0.901	0.881	-	-
6	0.938	0.896	0.908	0.891	-	-
7	0.930	0.636	0.791	0.799	-	-
8	0.927	0.907	0.907	0.907	-	-
9	0.930	0.776	-	-	-	0.796
10	0.940	0.802	0.794	0.776	-	0.794
11	0.944	0.878	0.878	-	0.945	0.878
12	0.96*	0.96	0.96	-	0.96	0.96
13	0.964*	0.951	0.951	-	0.974	0.951
14	0.788*	0.719	0.763	-	-	0.719
15	0.958*	0.958	0.958	-	0.958	0.958
16	1*	1	1	-	1	1
17	0.967	0.952	0.952	-	0.975	0.952
18	0.953	0.908	0.912	-	-	0.908
19	0.961	0.773	0.789	0.856	-	0.836
20	0.961	0.724	0.791	0.833	-	0.853
21	0.930	0.693	0.791	0.761	-	0.811
22	0.846	0.670	0.762	0.721	-	0.814
23	0.975	0.681	0.823	0.865	-	0.824
24	0.917	0.878	0.878	-	-	-
25	0.957	0.839	0.839	-	-	0.841
26	0.963	0.754	0.852	-	-	0.860
27	0.966	0.629	0.856	-	-	0.822

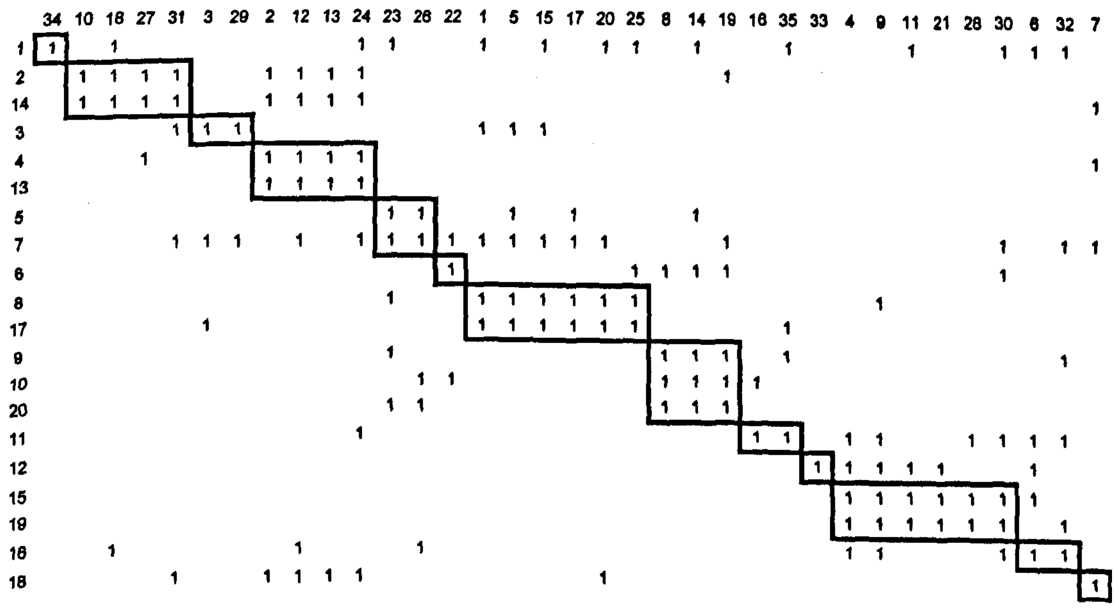
Table 5.5: Comparison with alternative cell-formation methods (grouping efficiency) (* indicates that the evolved GP-SLCA solution contained no singleton clusters)

Pr.No.	GP-SLCA	ZODIAC *(Chandr. & Raj., 1987)*	GRAFICS *(Sriniv.& Naren., 1991)*	MST-GRAFICS *(Srinivasan et al., 1994)*	MST *(Ng, 1993)*	GA-TSP *(Cheng et al., 1998)*
1	0.509	0.349	0.481	0.447	-	-
2	0.618	0.586	0.534	0.508	-	-
3	0.7*	0.686	0.675	0.644	-	-
4	0.496	0.267	0.449	0.407	-	-
5	0.727*	0.727	0.691	0.727	-	-
6	0.782	0.764	0.771	0.760	-	-
7	0.595*	0.320	0.579	0.530	-	-
8	0.774*	0.774	0.774	0.774	-	-
9	0.568	0.511	-	-	-	0.551
10	0.568*	0.538	0.544	0.471	-	0.539
11	0.767	0.751	0.751	-	0.767	0.753
12	0.92*	0.92	0.92	-	0.92	0.92
13	0.840*	0.839	0.839	-	0.831	0.840
14	0.587*	0.583	0.581	-	-	0.583
15	0.852*	0.852	0.852	-	0.852	0.852
16	1*	1	1	-	1	1
17	0.851*	0.851	0.851	-	0.851	0.851
18	0.735*	0.730	0.735	-	-	0.730
19	0.533	0.204	0.433	0.446	-	0.494
20	0.479	0.182	0.445	0.439	-	0.447
21	0.437	0.176	0.417	0.335	-	0.425
22	0.490*	0.387	0.494	0.436	-	0.466
23	0.607	0.337	0.554	0.559	-	0.538
24	0.731*	0.731	0.731	-	-	-
25	0.718	0.656	0.656	-	-	0.674
26	0.594	0.461	0.563	-	-	0.566
27	0.5	0.211	0.480	-	-	0.459

Table 5.6: Comparison with alternative cell-formation methods (grouping efficacy) (* indicates that the evolved GP-SLCA solution contained no singleton clusters)

5.4.5 Discussion

Results from tables 5.4, 5.5 and 5.6 indicate that GP-SLCA is an efficient algorithm for the solution of binary cell-formation problems. More specifically, for the grouping efficiency measure GP-SLCA dominated the performance of ZODIAC, GRAFICS, MST-GRAFICS and the GA-TSP heuristic. However, while the value of grouping efficiency was always higher than 0.9, the resulting diagonalised matrices were not ideal for the implementation of a cellular manufacturing system. Figure 5.19 illustrates the diagonalised matrix for problem 9, which had a grouping efficiency value of 0.944.



	34	10	16	27	31	3	29	2	12	13	24	23	26	22	1	5	15	17	20	25	8	14	19	18	35	33	4	9	11	21	28	30	6	32	7
1	1		1								1	1			1		1		1	1		1			1				1			1	1	1	
2		1	1	1	1			1	1	1	1												1												
14		1	1	1	1			1	1	1	1																								1
3					1	1	1								1	1	1																		
4				1				1	1	1	1																								1
13								1	1	1	1																								
5												1	1			1		1				1													
7					1	1	1		1		1	1	1	1	1	1	1	1	1				1									1		1	1
6														1						1	1	1	1									1			
8												1			1	1	1	1	1	1								1							
17						1									1	1	1	1	1	1					1										
9												1									1	1	1		1									1	
10													1	1							1	1	1	1											
20												1	1								1	1	1												
11											1													1	1		1	1			1	1	1	1	
12																										1	1	1	1	1			1		
15																											1	1	1	1	1	1	1		
19																											1	1	1	1	1	1		1	
16			1						1				1														1	1				1	1	1	
18					1			1	1	1	1								1																

Figure 5.19: Diagonalised matrix for test problem 9 (Boe and Cheng, 1991)

The proposed grouping of machines and parts illustrated in figure 5.19 is not a practical cell-formation solution since its implementation would rather disrupt the manufacturing process. However, this is not due to the inefficiency of GP-SLCA, since the aim of the algorithm was to maximise the desired objective. It is the maximisation of grouping efficiency that does not necessarily correspond to suitable solutions for the implementation of a cellular manufacturing system.

The deficiencies of the grouping efficiency measure have been discussed in detail in section 4.2.5.2. Figure 5.19 illustrates the result of assigning excessive importance to the minimisation of voids in comparison to the minimisation of exceptional elements.

It is obvious that the algorithm attempted to minimise the number of voids by creating small compact matrices of positive elements along the main diagonal of the m/c matrix. Table 5.4 provides numerical evidence that all evolved solutions focused mainly on the elimination of voids from the diagonalised matrix. The GP-SLCA solutions that did not include singleton clusters for both the grouping efficiency and efficacy measures are illustrated in tables 5.5 and 5.6.

The unusual configuration of the diagonalised matrices could be the reason for the slightly worse performance of GP-SLCA over the MST algorithm in some test problems. When a large number of small-sized cells is formed, the optimal assignment of parts to machines becomes almost a random search procedure since there are many candidate cells that satisfy both allocation criteria. As it was discussed in section 5.4.2.2, double ties in the part-assignment algorithm of GP-SLCA are broken randomly. In contrast, MST uses a special procedure for maximising the grouping measure by reassigning parts and machines after the initial machine cell – part family configuration has been created (section 5.3.5). This procedure is particularly useful in this situation where the assignment of parts to cells is not a straightforward task. However, partitions resulting from the MST algorithm are even more impractical than the ones already described. Ng (1993) reports a final solution containing 15 cells in a 20-machine problem (problem 9). An increase in the population size of GP-SLCA could result in similar solutions, simply because more random assignments of parts to machines would be generated. In any case, it is obvious that the grouping efficiency measure is inadequate for judging the quality of cell-formation solutions.

The inefficiency of the grouping efficiency measure should also be blamed for the poor performance of all non-hierarchical clustering methods in comparison to the evolutionary algorithms and the MST algorithm. The former methods do not allow the formation of singleton clusters (cells containing only one machine) in the potential cell configuration. However, as the previous results indicate, an optimal cell configuration in terms of the grouping efficiency measure will almost certainly involve singleton cells, thus the solutions of these algorithms are mostly sub-optimal.

The performance of the GP-SLCA procedure was even better when the maximisation of grouping efficacy measure was used as the objective of optimisation. Evolved solutions were always equal or better than those reported for all the comparing cell-formation methodologies (with the exception of problem 22, where GRAFICS produced a marginally better result). In addition, a considerable number of the evolved solutions did not include singleton clusters, so in these cases the algorithm exhibited a genuinely equal or better performance than the non-hierarchical clustering algorithms (ZODIAC, GRAFICS and the Assignment Algorithm).

Tables 5.5 and 5.6 indicate that GP-SLCA and MST produced the best overall performance for the specific grouping objectives used in the experimentation. The algorithms were further compared using the maximisation of the weighted grouping efficacy as the optimisation objective. As suggested by Ng (1993), the weight value of 0.2 was used for the calculation of the objective function. Cumulative results are presented in table 5.8. The weighted grouping efficiency value reported for GP-SLCA, was the best value found in twenty runs of the algorithm.

Problem number	GP-SLCA	MST
11	0.732	0.732
12	0.92	0.92
13	0.680	0.680
15	0.591	0.591
16	1	1
17	0.702	0.702

Table 5.8: Comparison of GP-SLCA for the MST method (weighted grouping efficacy, q=0.2)

As can be seen, both algorithms produced identical results. However, a more thorough investigation on the relative performance of the algorithms is required, since only a limited number of published results were available for comparison. Irrespective of the above results, it can be argued that an increase in the population size and the number of generations per run could further enhance the performance of GP-SLCA. Fine-tuning of the parameters could also result in better solutions. MST cannot be further improved in that way since there are no variable parameters that could determine the outcome of the run.

5.5 Advanced formulations of the cell-formation problem

5.5.1 Introduction

In the previous section the application of genetic programming for the solution of simple binary cell-formation problems was described. This formulation of the problem is considered to be insufficient, since only a limited amount of production-related information can be included in a binary machine-component matrix (see section 5.2.2). Information such as operation sequences of parts, processing times, machine capacities and opportunity costs is essential for the practical implementation of a cellular manufacturing system. The designer of the system might also impose constraints on the size of the machine cells and part families formed by the algorithm.

Ideally, a solution methodology should be able to simultaneously consider all these factors and produce an optimal configuration according to the desired settings. However, while a number of detailed models of cellular manufacturing systems have been developed, these are usually computationally intractable (see for example Selim et al. (1998)). In most cases cell-formation methodologies focus on the optimisation of models that are balanced between an accurate description of a manufacturing system and computational efficiency.

While the majority of binary cell-formation methods are either incapable of solving advanced formulations of the problem, or exhibit poor performance on them, the genetic programming methodology introduced in the previous section is capable of considering a variety of additional production-based information. GP-SLCA employs the following methods (individually or combined) for dealing with advanced formulations of the cell-formation problem:

- Modification of the terminal set
- Use of penalty function for solutions that violate designer's constraints
- Modification of the objective function

In the following paragraphs it will be shown how GP-SLCA can be modified to explicitly consider realistic formulations of the cell-formation problem. Three example problems taken from the literature will be used for the illustration of the proposed methodology. Note that results presented in the following paragraphs are only indicative of the performance of the algorithm in comparison with alternative solution methodologies that have been proposed for the same version of the problem. A comparative analysis requires performance results of the competing methodologies on a large number of test problems. However, unlike the case of the binary cell-formation problem described in the previous section, there are only a limited number of test problems and published results for each of the advanced formulations of the problem. Thus, the main aim of this section is to illustrate the application of GP-SLCA on advanced cell-formation problems and at the same time give an approximate indication of its performance in comparison with other solution methodologies for the version of the problem considered.

5.5.2 Operation sequences

The simple binary formulation of the cell-formation problem does not contain any information about the operation sequences of parts. However, this information is essential for the determination of the real cost of a cellular manufacturing configuration. Figure 5.20 illustrates the machine cells and part families produced by a potential solution algorithm.

	p2	p1	p5	p4	p3
m3	1	1	0	0	1
m1	0	0	1	1	1
m2	0	0	1	1	1

Figure 5.20: Potential cell-formation solution

Part 3 in this configuration is identified as the exceptional part, and machine 3 as the bottleneck machine. This simple interpretation of the proposed solution can be enhanced by replacing the positive entries in the m/c matrix with integer values indicating the operation sequence of a part on a particular machine, as shown in figure 5.21.

	p2	p1	p5	p4	p3
m3	1	1	0	0	2
m1	0	0	2	1	3
m2	0	0	1	2	1

Figure 5.21: m/c matrix indicating operation sequences

From this figure it can be deduced that the proposed configuration results in 2 intercell moves, since the operation required outside the block diagonals is neither the first nor the last of the processing sequence. This configuration leads to increased material handling costs and more complicated flow of parts within the system.

The consideration of the operation sequences of parts is therefore essential for the realistic evaluation of a cell-formation configuration. The simple GP-SLCA algorithm can be modified to explicitly consider the sequencing of parts during the evaluation phase of the algorithm.

The original GP-SLCA can be used unchanged for the solution of the advanced version of the problem, as it will be illustrated later in this section. However, it is clear that the similarity information fed to the algorithm through terminal *a* is noisy, since no distinction can be made between consecutive and non-consecutive operations of parts on pairs of machines. Thus, this terminal has to be modified accordingly, in order to explicitly consider the additional sequencing information. The value of the new terminal, a_s, is calculated as follows:

$$a_{s_{ij}} = 1 \text{ if part has non-consecutive operations on machines } i \text{ and } j$$

$$a_{s_{ij}} = 2 \text{ if part has consecutive operations on machines } i \text{ and } j$$

In this way, the similarity value of machines that processed parts in sequence is increased.

The objective of the optimisation algorithm should also be modified to reflect the quality of the optimisation in relation to the additional information that is now available. Nair and Narendran (1998) proposed *bond efficiency, β*, as a quality measure for the evaluation of cell-formation methodologies. Bond efficiency is able to consider both the compactness and the number of intercell moves produced by the proposed cell configuration. Equation (5.7) is used for the calculation of the bond efficiency of a diagonalised matrix.

$$\beta = q \cdot \frac{(I-U)}{I} + (1-q) \cdot \frac{\sum_{k=1}^{c} TOTOP_k}{\sum_{k=1}^{c} (TOTOP_k + NOP_k)} \tag{5.7}$$

where:

$$I = \sum_{j=1}^{n} (r_j - 1) \tag{5.8}$$

$$U = \sum_{j=1}^{n} \sum_{k=1}^{(r_j - 1)} xl_{jk} \tag{5.9}$$

$\frac{(I-U)}{I}$: Group technology efficiency (Harhalakis *et al.*, 1990)

$\frac{\sum_{k=1}^{c} TOTOP_k}{\sum_{k=1}^{c} (TOTOP_k + NOP_k)}$: Compactness measure

I : Possible number of intercell moves in the system

U : Number of intercell moves generated by the potential solution

r_j : Number of operations on part j

$xl_{jk} = 0$ if operations k, k+1 are performed in the same cell

$xl_{jk} = 1$ if operations k, k+1 are not performed in the same cell

$TOTOP_k$: Number of operations within cell k

NOP_k : Number of non-operations within cell k

$TOTOP_k + NOP_k$: Possible number of operations within cell k

n : Total number of parts in the system

c : Total number of cells in the system

q : Weighting factor ($0 \le q \le 1$)

There are no other changes in the operation of the GP-SLCA algorithm. A summary of the proposed implementation is presented in the Koza tableau of figure 5.22:

Parameters	Values
Objective:	maximisation of the bond efficiency of the cell configuration
Terminal set:	a_s, b, c, d (defined earlier)
Function set:	+, -, ×, %
Population size:	500
Subtree crossover probability:	.9
Subtree mutation probability:	.1
Selection:	Tournament selection, size 7
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Figure 5.22: Koza tableau for the GP-SLCA methodology that explicitly considers operation sequences

The proposed GP-SLCA algorithm was tested on the example problem introduced by Nair and Narendran (1998) and illustrated in figure 5.23. Initially, 20 runs of the original GP-SLCA algorithm were conducted and the results are summarised in table 5.9. The value of the maximum bond efficiency produced was lower than the one reported by Nair and Narendran $(\beta = 0.7163)$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1		2										1																								2				
2		3													4																					3				
3			2						3			4	3	4								3																		
4	5				3		3									1										2				4										
5								1						1	3								2	1					3		2									
6																									1				2						2					2
7	3						2									3	1													2							3			
8										2	2								1		1	4						2										2		
9										1									3		3	2						1										3		
10	1						5								5		3		2		2							3												
11			3						4				2	2																			1			4				
12				1	2	3																				3	1							2			2		1	
13											3																					2								
14																		3														1								
15																		2							3	4						3			4					3
16	4					2	4	3							1	2							3	2						3										
17		4										3										1									1					1				
18	2				1		1									4	2													1										
19								2							2								1								3									
20			1						1				1	3																			3							
21											1														2		3		1						1					1
22	6																	1									2					4			3			1		
23				2		1														1						1								1			1			
24												2																						3						
25		1							2	3		5																					2							

Figure 5.23: Nair and Narendran's (25x40) test problem

Best value of bond efficiency recorded	**0.701984**
Number of times this value was found	**3**
Mean best value of bond efficiency per run	**0.690819**
Standard deviation	**0.006823**

Table 5.9: Bond efficiency results for the original GP-SLCA algorithm

In contrast, the modified GP-SLCA algorithm that explicitly considered the operation sequences of parts was able to find the best reported value of bond efficiency in 7 out of the 20 runs (table 5.10).

Best value of bond efficiency recorded	**0.71633**
Number of times this value was found	**7**
Mean best value of bond efficiency per run	**0.708852**
Standard deviation	**0.007145**

Table 5.10: Bond efficiency results for the modified GP-SLCA algorithm

The cell configuration that corresponds to this particular value of bond efficiency is illustrated in figure 5.24.

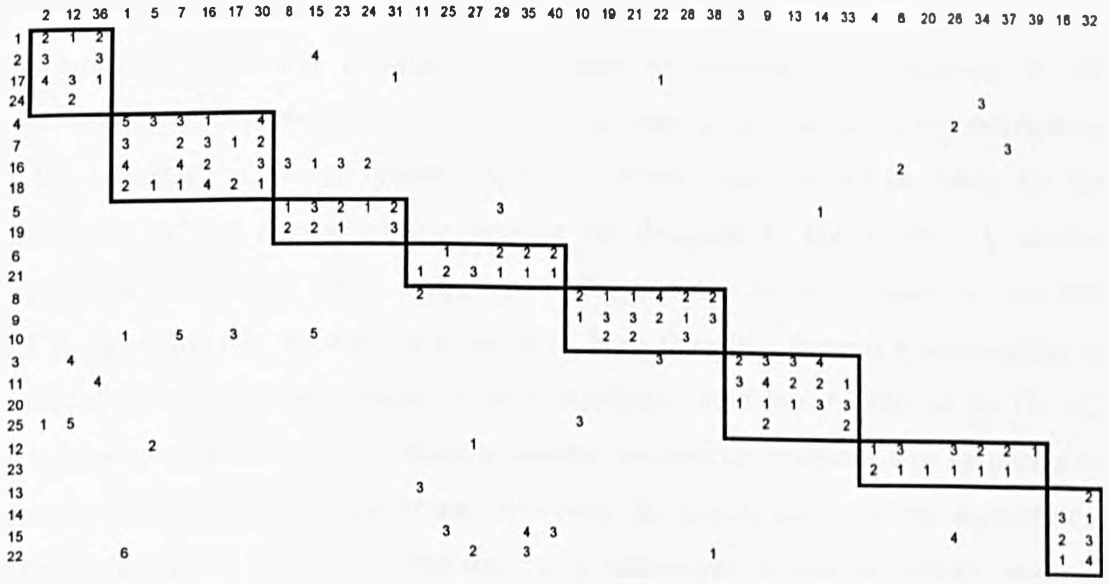


Figure 5.24: Best solution found by the modified GP-SLCA algorithm (β=0.71633)

The previous example demonstrates the ability of GP-SLCA to consider cell-formation problems where information about the operation sequences of parts is included in the formulation of the problem.

5.5.3 Minimisation of opportunity costs under design constraints

Any part not totally processed within its corresponding cell is identified as an exceptional part. Generally speaking, there are two alternative ways of dealing with the presence of exceptional parts within a cellular manufacturing system:

- Exceptional parts are subcontracted to a third party
- Bottleneck machines associated with exceptional parts are duplicated

It is obvious that there are specific costs associated with each of these decisions. The minimisation of opportunity costs of exceptional parts is the objective in a number of cell-formation solution methodologies (Kumar and Vanelli, 1987), (Wei and Gaither, 1990), (Zhu et *al.*, 1995). The problem becomes more complex in the case of imposed constraints by the designer of the system on the total number of cells allowed in the plant and the maximum number of machines allowed within a cell. These constraints might be introduced due to practical considerations about the size of the plant and the existing layout of machines.

The original GP-SLCA methodology is able to consider the objective of the minimisation of opportunity costs for the exceptional parts, without any modification of the terminal set of the system. However, special care should be taken for the satisfaction of the constraints imposed by the designer of the system. A penalty function is introduced, which assigns zero fitness value to any solution of the GP-SLCA algorithm that violates the constraints. Note that since there is a dendrogram of potential solutions corresponding to each similarity coefficient evolved by the GP evolutionary procedure, a considerable number of feasible solutions will be produced in every generation of the algorithm. However, the search power of the algorithm is certainly degraded by the fact that there is a substantial amount of genetic material that is ignored by the evolutionary procedure because it produces infeasible solutions.

The GP methodology employed for the solution of cell-formation problems with associated opportunity costs for the exceptional parts and/or design constraints is presented in the Koza tableau of figure 5.25.

Parameters	Values
Objective:	minimisation of the opportunity costs of the exceptional parts in the system under specific constraints
Terminal set:	*a, b, c, d* (defined earlier)
Function set:	+, -, x, %
Population size:	500
Subtree crossover probability:	.9
Subtree mutation probability:	.1
Selection:	Tournament selection, size 7
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Figure 5.25: Koza tableau for the GP-SLCA methodology which explicitly considers opportunity costs of exceptional parts and design constraints

Kumar and Vanelli's (30x41) test problem (figure 5.26) was employed for the testing of the proposed methodology. The total number of cells in the proposed configuration was equal to 2, with no more than 20 machines allowed within each cell. Note that the last row in figure 5.26 denotes the opportunity cost associated with the part of the corresponding column. The cumulative results of the 20 runs of the algorithm conducted on the example problem are presented in table 5.11.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
1										1																					1	1	1						1		1
2											1	1						1														1	1						1		1
3												1																			1	1							1	1	
4							1								1								1												1						
5					1												1							1	1			1						1		1					
6						1																																			
7																1										1								1		1					
8	1							1	1					1													1														
9			1										1								1								1												
10		1										1								1			1																1		
11										1																							1						1		1
12											1	1							1	1			1	1															1	1	
13																			1						1													1			
14				1	1													1								1											1				
15																	1																				1				
16						1																						1													
17							1									1											1									1					
18														1		1											1							1							
19	1		1																		1	1								1										
20	1		1																		1	1																			
21										1		1																			1	1	1							1	
22										1		1											1								1	1							1	1	
23		1										1											1								1								1		
24																																						1			
25				1																																					
26																1											1							1		1					
27								1							1																										
28								1																					1						1						
29	1		1						1				1								1									1											
30	1		1																		1	1																			
	15	50	19	71	88	48	47	380	28	166	232	229	22	33	121	132	34	36	48	164	21	26	80	18	91	45	88	110	47	42	49	55	249	44	80	41	35	48	118	64	82

Figure 5.26: Kumar and Vanelli's (30x41) test problem

Best value of cost recorded	**$210**
Number of times this value was found	**19**
Mean best value of cost per run	**$210.9**
Standard deviation	**$4.024922**

Table 5.11: Opportunity cost results for the GP-SLCA algorithm

The best solution found by the GP-SLCA procedure corresponds to a total cost of $210, and is illustrated in figure 5.27. While the evolved solution was better than the one reported by Kumar and Vanelli ($217), Wei and Gaither's analytical mathematical programming approach was able to produce a cell configuration where the opportunity cost associated with the exceptional parts was equal to $146.

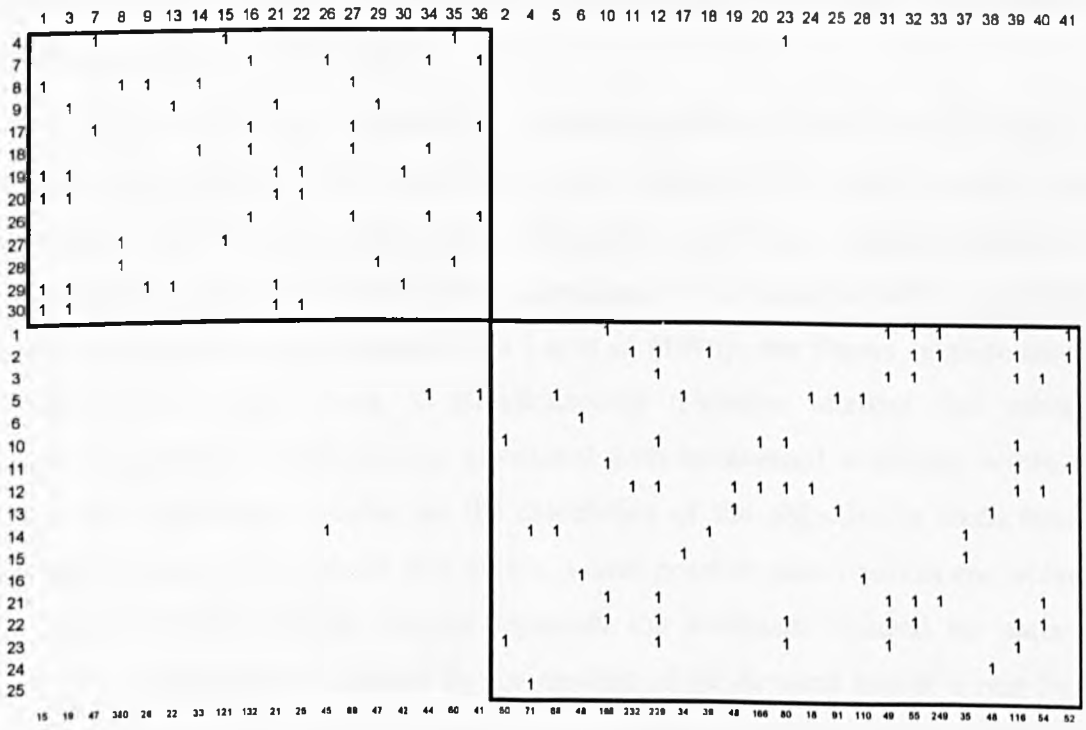


Figure 5.27: Best solution found by the GP-SLCA algorithm (cost=$210)

The cell-formation problem examined in this section illustrates the ability of the simple GP-SLCA methodology to deal with constraints imposed by the designer of the system. While penalising infeasible solutions clearly degrades the performance of the system, GP-SLCA is still able to produce high-quality cell configurations that meet the specifications of the designer.

5.5.4 Balancing the workload within the cells

The binary cell-formation formulation does not take into account the balancing of workload within the designed cells. This is because the calculation of workload requires information about the processing times and demand rates of parts in particular machines. This type of information is not available through the binary m/c matrix representation of the problem.

Balancing the workload within machine cells results in significant benefits both from the production and the operator's point of view. It means that the flow of parts within the cells is smooth and the work-in-progress inventory (WIP) is reduced. It also means that the operators will work at a steadier pace, a condition that, generally speaking, increases job satisfaction.

GP-SLCA is capable of solving cell-formation problems where the balancing of workload is explicitly considered, without any alteration on the basic structure of the algorithm. However, the optimisation objective should be suitably modified to incorporate additional information that is available. In the example problem presented in this section (which was adopted from Lin et *al.* (1996)), the fitness measure takes a multiobjective form, aiming to simultaneously minimise intercell and intracell processing costs, as well as costs associated with unbalanced workload within the cells. The information needed for the calculation of the objective is taken from a modified version of the binary m/c matrix, where positive binary values are replaced by integer weights. These weights represent the workload induced by parts on particular machines as calculated by the product of the demand rate of a part by its processing time on the respective machine. Formally, the problem can be described with the following non-linear integer programming formulation (Lin et al., 1996):

$$\text{Min } Z = aW_a + eW_e + dW_d \tag{5.10}$$

subject to:

$$W_a = \sum_{h=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ih} y_{jh} w_{ij} \tag{5.11}$$

$$W_e = W - W_a \tag{5.12}$$

$$W_d = \sum_{h=1}^{K}\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ih} y_{jh} \left| \overline{W_j} - w_{ij} \right| \tag{5.13}$$

$$\overline{W_j} = \sum_{h=1}^{K} y_{jh} \sum_{i=1}^{m} \left[x_{ih} w_{ij} \Big/ \sum_{i=1}^{m} x_{ih} \right], \qquad j = 1, \ldots, n \tag{5.14}$$

$$\sum_{h=1}^{K} x_{ih} = 1, \qquad i = 1, \ldots, m \tag{5.15}$$

$$\sum_{h=1}^{K} y_{jh} = 1, \qquad j = 1, \ldots, n \tag{5.16}$$

$$K = \sum_{h=1}^{K^f} \max_{i=1,\ldots,m} (x_{ih}) \tag{5.17}$$

$$K \leq K^f \tag{5.18}$$

$$x_{ih} = 0 \text{ or } 1, \; i = 1, \ldots, m, \quad h = 1, \ldots, K,$$

$$y_{jh} = 0 \text{ or } 1, \; j = 1, \ldots, n, \quad h = 1, \ldots, K$$

$$W_a \geq 0, W_e \geq 0, W_d \geq 0, K \geq 0$$

where:

a : unit time intracell processing cost

e : unit time intercell processing cost

d : unit time balance delay cost

W : sum of weights in matrix

K^f : maximum number of cells specified by the designer

w_{ij} : workload (demand rate × standard processing time) of part j on machine i

K : total number of machine cells

W_a : sum of intracell weights

W_e : sum of intercell weights

W_d : sum of balance delay weights

$\overline{W_j}$: average intracell workload for part j

x_{ih}= 1 if machine j is assigned to cell h, 0 otherwise

y_{jh}= 1 if part j is assigned to family h, 0 otherwise

Lin et al. (1996) applied their minimum spanning tree solution methodology (section 5.3.5) on a number of binary test problems taken from the literature, with part weights generated randomly from a uniform distribution. Since these data were not published in detail, comparison with the GP-SLCA procedure was not possible. However, the authors also presented an application of their methodology on a case study of a company that manufactured irrigation products. The company decided to transform their functional layout into a cellular manufacturing layout. After the collection of the necessary data, the resulted cell-formation problem was summarised with the help of a (22×62) machine/component matrix (figure 5.28). The heuristic procedure of Lin *et al.* was applied on this cell-formation problem, and resulted in an overall cost of 617 units (note that the multiobjective function was calculated with the cost weights set at the following values: a=1, e=3, d=0.5).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	3	3	3	2	2																										
2	3	3	3	2	2																										
3						4	4																								
4	4	4	4	2	2																										
5						4	4			4	4	2	2																		
6																															
7																															
8																															
9						3	3	3	3																						
10								5	5																						
11						3	3	3	3	3	3	2	2																		
12														3	3	3	3	3	3	3	3	3	3								
13			5	3	3																			4	3	3	4	3	3	4	3
14																															
15																															
16	3	3	3	2	2																										
17																															
18	2	2	2	1	1																										
19														1	1	1	1														
20																															
21	2	2	2	1	1																										
22														2	2	2	2														

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
1																															
2																															
3																															
4																															
5																															
6											3	2	2															4	4		
7		6	5	5										8	9	6	5	6	9	9											
8														8	9	6	5	6	9	9											
9																															
10																															
11																															
12																															
13	3	4	3	3	4	3	3	4	3	3	4	3	3																		
14																					3	3	2	2	2						
15														3	3	2	2	2	3	3											
16																															
17																					2	2	1	1	1	2	2				
18																															
19																															
20																												3	3	3	3
21																															
22																															

Figure 5.28: (22×62) test problem (Lin et al., 1996)

The GP-SLCA methodology was applied to the same problem without any modification on the basic structure of the algorithm. The optimisation objective used was the minimisation of the sum of material handling costs and balanced delay costs, as calculated by equation (5.10). The same set of weights was employed for the scaling of each cost factor (a=1, e=3, d=0.5).

A summary of the proposed methodology is presented in the Koza tableau of figure 5.29.

Parameters	Values
Objective:	minimisation of the sum of intercell, intracell, and balance delay costs
Terminal set:	*a, b, c, d* (defined earlier)
Function set:	+, -, x, %
Population size:	500
Subtree crossover probability:	.9
Subtree mutation probability:	.1
Selection:	Tournament selection, size 7
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Figure 5.29: Koza tableau for the GP-SLCA methodology that explicitly considers balance delay costs

The cumulative results of 20 runs of the algorithm are presented in table 5.12.

Best value of cost recorded	**606.601**
Number of times this value was found	**20**
Mean best value of cost per run	**606.601**
Standard deviation	**0**

Table 5.12: Total cost results for the GP-SLCA algorithm

GP-SLCA was able to find a solution that had a lower cost value than the one reported by Lin *et al.* (617) in all experimental runs. This solution is illustrated in figure 5.30.

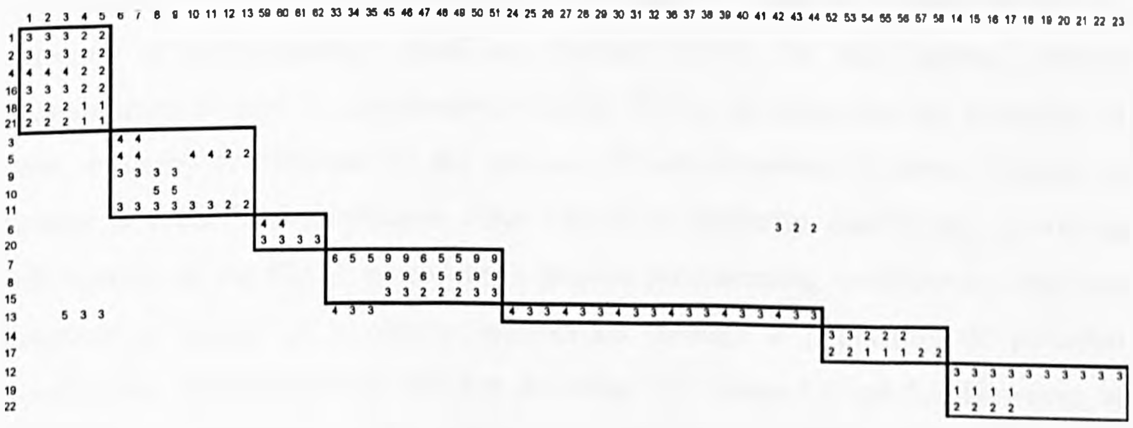


Figure 5.30: Best solution found by the GP-SLCA algorithm (cost=606.6 units)

The previous example illustrates the ability of the GP-SLCA algorithm to consider cell-formation problems where the balancing of workload is explicitly defined in the formulation of the problem.

5.5.5 Discussion

Results presented in sections 5.5.2, 5.5.3 and 5.5.4, suggest that GP-SLCA has the ability to consider advanced formulations of the cell-formation problem with only a limited number of modifications in the basic structure of the algorithm.

These results are only indicative of the performance of the GP-SLCA on advanced formulations of the cell-formation problem. While GP-SLCA produced competitive results to alternative solution techniques on the problems considered, a larger experimental basis is needed in order to establish a statistical performance measure of the efficiency of the algorithm. However, this experimental basis only exists for binary cell-formation problems, since published results for advanced formulations of the problem are usually restricted to a single illustrative example for each proposed methodology.

5.6 Evolution of similarity coefficients for the solution binary cell-formation problems

5.6.1 Introduction

A variety of similarity coefficients have been used in clustering algorithms for the solution of cell-formation problems (Sarker, 1996). In this section genetic programming is used in combination with the SLCA algorithm for the evolution of new similarity coefficients for the solution of cell-formation problems. Instead of having a predefined coefficient (like Jaccard's similarity coefficient) providing information to the SLCA procedure, a genetic programming evolutionary machine proposes a variety of similarity information through a population of potential coefficients, in the same way that was described in sections 5.4 and 5.5. However, in this case, the performance of evolved coefficients is not evaluated only on a specific instance of the problem, but on a set of test cases instead. The end product of this procedure is hoped to be a similarity coefficient that is at least as good as man-made coefficients in the solution of simple cell-formation problems, when used in combination with a hierarchical clustering procedure like SLCA.

In genetic programming terms, the problem is stated as follows: "Find a program that takes as input similarity information between machines and produces as output a similarity coefficient that maximises a pre-specified grouping measure of any binary cell-formation problem, when used in combination with a hierarchical clustering procedure" (figure 5.31).

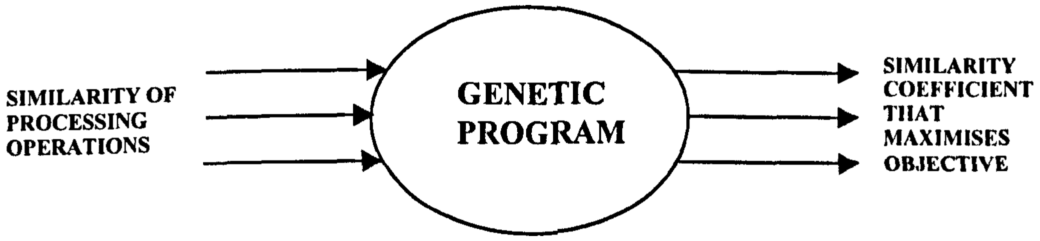

SIMILARITY OF PROCESSING OPERATIONS
GENETIC PROGRAM
SIMILARITY COEFFICIENT THAT MAXIMISES OBJECTIVE

Figure 5.31: Genetic programming approach to the evolution of similarity coefficients for the solution of binary cell-formation problems

5.6.2 Design of the algorithm

In the previous sections GP-SLCA was employed for the evolution of similarity coefficients that were specific to the problem considered, i.e. the test problem was used as the only fitness case of the evolutionary procedure. The method was not focused in the evolution of similarity coefficients as such, but rather exploited this procedure for the diagonalisation of binary m/c matrices.

In this section the possibility of evolving a similarity coefficient that can be used for the solution of any binary cell-formation problem, in a similar way to Jaccard's similarity coefficient, is examined. Evolved coefficients are evaluated through the SLCA algorithm on a pre-specified number of test problems that are used as fitness cases. The fitness of a solution is calculated by adding the value of the objective function from each individual test problem. In that way similarity coefficients that produce good overall performance will prevail during the evolutionary procedure. The proposed methodology is illustrated in figure 5.32.

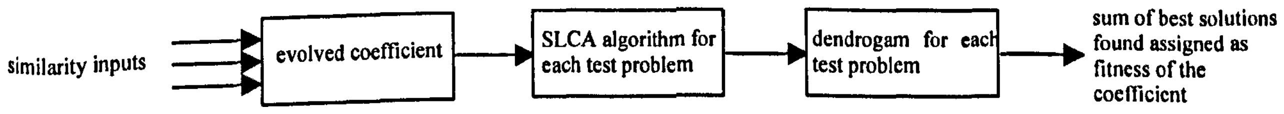

similarity inputs
evolved coefficient
SLCA algorithm for each test problem
dendrogam for each test problem
sum of best solutions found assigned as fitness of the coefficient

Figure 5.32: Illustration of the GP-SLCA procedure for the evolution of similarity coefficients

The basic operation of the GP-SLCA algorithm in terms of the function and terminal sets employed remains the same. The maximisation of grouping efficacy was used as the objective of the evolutionary procedure.

Selecting the test problems for the construction of the set of fitness cases was not a straightforward procedure. A representative set of fitness cases should include different instances of the problem in terms of size, difficulty or any other parameter that can be varied. In this way the outcome of the evolutionary procedure would be more likely to generalise to previously unseen problems. However, in the case of cell-formation problems, grouping difficulty cannot be easily described in terms of parameters (see section 5.4.3). At the same time the evaluation function of the algorithm is computationally expensive, especially for large-sized problems, thus the number of test cases must be kept within certain limits depending on the available computational power.

In the experimental set-up of this section, ten different combinations of test problems were employed, all taken from the batch of 27 test problems described in table 5.3. All sets comprised different problems in terms of their characteristics (size, grouping difficulty as it has been reported in published results, etc.) in an attempt to fulfil the requirements described earlier. Table 4A in the Appendix, illustrates the configuration of these sets. The additional parameters that were necessary for the valid run of the genetic programming algorithm and a summary of the proposed methodology are presented in the Koza tableau of figure 5.33.

Parameters	**Values**
Objective:	Evolution of a similarity coefficient that maximises grouping efficacy in binary cell-formation problems when SLCA is used as the clustering procedure
Terminal set:	*a, b, c, d* (defined earlier)
Function set:	+, -, x, %
Population size:	500
Subtree crossover probability:	.9
Subtree mutation probability:	.1
Selection:	Tournament selection, size 7
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Figure 5.33: Koza tableau of the GP-SLCA methodology for the evolution of new similarity coefficients for the solution binary cell-formation problems

5.6.3 Results

Twenty runs of the GP-SLCA algorithm were conducted for each experimental set-up, as the probabilistic nature of genetic programming requires. The cumulative results of the best coefficients evolved for each set-up are presented in table 5.13. The outlined problems in each column denote the test problems that were used as training (fitness) cases for the evolution of the corresponding similarity coefficient. The rest of the test problems in each column comprised the validation set for the evolved coefficient.

For comparison reasons, the performance of eleven different similarity coefficients (table 5A in Appendix) in combination with the SLCA algorithm was measured on the same set of test problems (table 5.14). The list of coefficients was retrieved from the review paper of Sarker (1996).

Pr.no.	SET1	SET2	SET3	SET4	SET5	SET6	SET7	SET8	SET9	SET10
1	0.5	0.471	0.451	0.471	0.5	0.467	0.438	0.438	0.490	0.467
2	0.615	0583	0.586	0.586	0.618	0.571	0.588	0.586	0.611	0.601
3	0.7	0.698	0.698	0.7	0.7	0.7	0.7	0.7	0.7	0.7
4	0.474	0.459	0.240	0.455	0.489	0.470	0.409	0.231	0.467	0.475
5	0.727	0.727	0.727	0.727	0.727	0.727	0.727	0.727	0.727	0.727
6	0.752	0.752	0.752	0.752	0.752	0.752	0.752	0.752	0.752	0.742
7	0.579	0.579	0.579	0.579	0.570	0.579	0.579	0.238	0.579	0.568
8	0.773	0.773	0.773	0.773	0.773	0.773	0.748	0.748	0.774	0.774
9	0.568	0.412	0.554	0.568	0.520	0.562	0.568	0.568	0.568	0.568
10	0.544	0.556	0.367	0.568	0.545	0.383	0.543	0.552	0.568	0.545
11	0.760	0.757	0.760	0.760	0.757	0.757	0.760	0.760	0.767	0.757
12	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92
13	0.840	0.840	0.840	0.840	0.840	0.840	0.840	0.840	0.840	0.84
14	0.569	0.569	0.569	0.587	0.569	0.587	0.587	0.569	0.587	0.587
15	0.852	0.852	0.852	0.852	0.639	0.852	0.852	0.852	0.852	0.852
16	1	1	1	1	1	1	1	1	1	1
17	0.851	0.851	0.851	0.851	0.851	0.851	0.851	0.581	0.851	0.851
18	0.735	0.735	0.735	0.735	0.735	0.735	0.735	0.735	0.735	0.735
19	0.443	0.532	0.149	0.465	0.503	0.523	0.522	0.136	0.513	0.507
20	0.454	0.472	0.194	0.466	0.458	0.477	0.309	0.136	0.295	0.453
21	0.410	0.429	0.330	0.429	0.41	0.431	0.203	0.41	0.203	0.429
22	0.283	0.246	0.430	0.246	0.246	0.246	0.479	0.246	0.385	0.337
23	0.520	0.525	0.6	0.528	0.558	0.530	0.6	0.543	0.516	0.585
24	0.709	0.677	0.731	0.682	0.682	0.650	0.731	0.731	0.720	0.682
25	0.671	0.7	0.718	0.671	0.699	0.710	0.710	0.706	0.666	0.696
26	0.558	0.570	0.571	0.567	0.558	0.482	0.468	0.573	0.521	0.561
27	0.176	0.298	0.484	0.244	0.161	0.224	0.165	0.479	0.147	0.479

Table 5.13: Cumulative results of evolved coefficients on test problems

Pr.no	COEF1	COEF2	COEF3	COEF4	COEF5	COEF6	COEF7	COEF8	COEF9	COEF10	COEF11
1	0.471	0.467	0.471	0.393	0.393	0.393	0.438	0.471	0.393	0.438	0.471
2	0.571	0.571	0.571	0.585	0.585	0.585	0.571	0.571	0.585	0.571	0.571
3	0.7	0.7	0.7	0.688	0.688	0.688	0.471	0.7	0.688	0.471	0.7
4	0.474	0.453	0.474	0.394	0.394	0.394	0.231	0.453	0.394	0.434	0.468
5	0.727	0.727	0.727	0.71	0.71	0.71	0.727	0.727	0.71	0.727	0.727
6	0.752	0.752	0.752	0.738	0.738	0.738	0.738	0.752	0.738	0.738	0.752
7	0.579	0.579	0.579	0.568	0.568	0.568	0.47	0.579	0.568	0.579	0.579
8	0.774	0.774	0.744	0.774	0.774	0.774	0.774	0.774	0.774	0.774	0.774
9	0.568	0.568	0.568	0.519	0.519	0.519	0.24	0.541	0.519	0.556	0.568
10	0.544	0.54	0.544	0.257	0.257	0.257	0.183	0.548	0.257	0.183	0.546
11	0.757	0.757	0.757	0.757	0.757	0.757	0.757	0.757	0.757	0.757	0.757
12	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92
13	0.840	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84	0.84
14	0.569	0.587	0.569	0.569	0.569	0.569	0.569	0.569	0.569	0.569	0.587
15	0.852	0.853	0.853	0.853	0.853	0.853	0.853	0.853	0.853	0.852	0.852
16	1	1	1	1	1	1	1	1	1	1	1
17	0.851	0.851	0.851	0.851	0.851	0.851	0.667	0.851	0.851	0.851	0.851
18	0.735	0.735	0.735	0.735	0.735	0.735	0.585	0.735	0.735	0.735	0.735
19	0.517	0.501	0.517	0.167	0.167	0.167	0.167	0.516	0.167	0.491	0.517
20	0.199	0.392	0.199	0.382	0.382	0.382	0.28	0.461	0.382	0.34	0.199
21	0.232	0.429	0.232	0.208	0.208	0.208	0.242	0.429	0.208	0.203	0.24
22	0.246	0.246	0.246	0.246	0.246	0.246	0.246	0.246	0.246	0.272	0.417
23	0.552	0.587	0.552	0.414	0.414	0.414	0.121	0.556	0.414	0.525	0.533
24	0.682	0.682	0.682	0.682	0.682	0.682	0.667	0.682	0.682	0.667	0.682
25	0.671	0.697	0.671	0.699	0.699	0.699	0.679	0.693	0.699	0.679	0.694
26	0.565	0.509	0.565	0.511	0.511	0.511	0.5	0.509	0.511	0.53	0.565
27	0.389	0.426	0.389	0.119	0.119	0.119	0.111	0.163	0.119	0.42	0.393

Table 5.14: Performance of similarity coefficients produced by human intuition

5.6.4 Discussion

Results from tables 5.13 and 5.14 indicate that the GP-SLCA framework was able to evolve coefficients that generalised over the entire set of problems.

	SET1	SET2	SET3	SET4	SET5	SET6	SET7	SET8	SET9	SET10
$\overline{\Gamma}$	0.629	0.629	0.610	0.630	0.621	0.622	0.622	0.584	0.621	0.646

	COEF1	COEF2	COEF3	COEF4	COEF5	COEF6	COEF7	COEF8	COEF9	COEF10	COEF11
$\overline{\Gamma}$	0.620	0.635	0.619	0.577	0.577	0.577	0.520	0.626	0.577	0.597	0.627

Table 5.15: Mean value of grouping efficacy for evolved and man-made coefficients

Table 5.15 illustrates that the mean value of grouping efficacy produced by all evolved coefficients was similar to the one produced by man-made coefficients.

Coefficients SET4 and SET10 performed particularly well on the entire set of problems, producing an increase of 1% and 2.6% respectively on average grouping efficacy in comparison to Jaccard's similarity coefficient (COEFF1). Since the difference in performance was relatively small, further research was needed in order to establish if SET10 could be distinguished from Jaccard's coefficient. A winner-takes-all comparison of their relative performance on the test problems is presented in table 5.16.

	Jaccard's coefficient better	**Jaccard's coefficient worse**	**Jaccard's coefficient equal**
SET10	5	10	12

Table 5.16. Jaccard's coefficient vs. SET10 in terms of non-dominated solutions

It is obvious that there are a large number of problems where the same level of grouping efficacy was achieved by both coefficients, thus we cannot safely reject the hypothesis that the coefficients are actually the same (null hypothesis). The Analysis Of Variance (ANOVA) between the two sets of values confirms this statement (table 5.17).

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
SET10	27	17.438	0.645852	0.027259
JACCARD	27	16.737	0.619889	0.041745

ANOVA

F	*P-value*	*F crit*
0.263753	0.609729	4.026631

Table 5.17: ANOVA for SET10 and Jaccard's coefficient (α=0.05)

The man-made similarity coefficient that produced the best performance in terms of the mean value of grouping efficiency was Yule's coefficient (COEFF2), a coefficient employed in psychological research. The winner-takes-all comparison between SET10 and Yule's coefficient produced a significant number of equal solutions (table

5.18). In addition, the ANOVA test indicated that the null hypothesis could not be rejected (tables 5.19).

	Yule's coefficient better	Yule's coefficient worse	Yule's coefficient equal
SET10	5	8	14

Table 5.18. Yule's coefficient vs. SET10 in terms of non-dominated solutions

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
YULE	27	17.143	0.634926	0.032474
SET10	27	17.438	0.645852	0.027259

ANOVA

F	*P-value*	*F crit*
0.05396	0.817223	4.026631

Table 5.19: ANOVA for SET10 and Yule's coefficient (α=0.05)

It is interesting to take a closer look at the structure of the evolved similarity coefficients. Coefficient SET4 is illustrated in figure 5.34:

$$d\left(d+c+\frac{b}{c}+a^2\right)-\left(\frac{d}{\frac{b}{c}-a}\right)$$

Figure 5.34: Similarity coefficient SET4

Notice that genetic programming evolved structures that did not follow the elegant form of Jaccard's coefficient, but were just as effective in the solution of the test problems. From the above formula it is clear that the value of the coefficient is proportional to the values of *a* and *d*. This is expected since these values are indicative of the similarity of parts processed between a pair of machines.

The structure of coefficient SET10 is much more complicated than SET4, as figure 5.35 depicts:

$$4d+3b-b^2-ab+ad+\left[\left(\frac{FACTOR-d-b-(d+b)cb+bc+ab}{d(a+b)}\right)\right]-\left(\frac{c-c}{d+b}\right)-\left(\frac{ad+d+b}{b^2c^2-a-b^2}\right)$$

where $FACTOR =$

$$\frac{a}{-a+(bc-db-cd-2a+b)\left[d-\left(\frac{(2a-3b)(a-b)+ab+db-dc-a+c+\left(\frac{ad}{a}\right)}{a}\right)+\left(\frac{ad}{\left(\frac{a-b}{d-a}\right)\Big/ab^2c}\right)\right]}$$

Figure 5.35: Similarity coefficient SET10

The size and complexity of the evolved coefficient makes the task of explaining its operation quite difficult. This is not an unusual situation in GP, since the application of genetic operators leads to quick growth of programs up to the pre-specified maximum depth constraint (the 'bloat' effect). However, there is still the issue of the transparency of evolved genetic programs, as was discussed in section 4.4.3. It is evident that the value of the coefficient is proportional to the values of a and d, however, a number of control terms are also present which seem to fine tune its value in particular fitness cases. Note, that there are two terms that according to common algebra should have been simplified:

$$\frac{c-c}{d+b} \quad \text{and} \quad \left(\frac{ad}{a}\right)$$

However, due to the operation of the protected division function and the postfix order of program execution, these expressions will evaluate to '1' if the denominator is equal to '0', which is not an unlikely case. Thus, they should be considered in this form during the calculation of the coefficient value.

The generalisation of this coefficient was quite good. SET10 appeared to have captured information that is relevant to the solution of the problem. In problems 19-21, where the m/c matrices have been custom designed to be difficult for grouping, Jaccard's coefficient failed to find fit partitions. On the same problems SET10 created cell configurations with much higher levels of grouping efficacy. On problem 27, where alternative evolved coefficients either produced poor results, or their good performance was not mirrored in the set of validation problems, SET10 produced an excellent level of grouping efficacy. While the difference in performance between SET10 and man-made coefficients could not be mathematically confirmed, results on specific test problems indicated that SET10 might be able to handle ill-structured matrices in a more efficient way.

From the above results it can be safely concluded that the GP-SLCA algorithm was able to evolve similarity coefficients that performed at least as good as the similarity coefficients that have been devised by human intuition. The advantage of the proposed methodology is that while man-made coefficients produce the same clustering outcome independent of the clustering objective, the proposed framework can be used to evolve purpose-based coefficients by simply altering the objective function, or introducing constraints in the evolutionary procedure.

The main disadvantage of evolved coefficients is their non-parsimonious structure. A solution to this problem could be the explicit consideration of parsimony in the objective function of the evolutionary procedure by the penalisation of long programs, as it has been suggested in genetic programming literature (Koza, 1992). However, this modification would lead to greater computational costs. Another disadvantage of the proposed methodology was the use of the SLCA clustering procedure, which has been reported to result in suboptimal groupings in comparison to the Average Linkage Clustering procedure, or the Complete Linkage Clustering procedure (Gupta and Seifoddini, 1990). The choice of SLCA was again necessitated by computational requirements, since it did not require the recalculation of the similarity matrix for every similarity level in the dendrogram.

5.7 Conclusions

In this chapter the use of genetic programming for the solution of simple and advanced formulations of the cell-formation problem was investigated. McAuley' Single Linkage Cluster Analysis (SLCA) algorithm was used as basis for the development of the proposed methodology. SLCA employs Jaccard's similarity coefficient for the creation of a pictorial representation of solutions in the form of a 'dendrogram'. Choosing a particular similarity level in the dendrogram can create a variety of cell configurations.

Genetic programming was utilised in two different ways. First, similarity coefficients were evolved for the solution of specific binary cell-formation problems. Coefficients were fed into an SLCA procedure, which returned the best solution found in the dendrogram of potential solutions, in relation to the desired objective. The methodology was tested on a number of published test problems, performing at least as good as alternative cell-formation algorithms. In addition, the application of the proposed methodology on the solution of advanced formulations of the cell-formation problem was illustrated with the help of some test problems taken from the literature.

Genetic programming was also employed for the evolution of new similarity coefficients that can be used in combination with a hierarchical clustering procedure for the solution of binary cell-formation problems. The proposed framework was able to evolve coefficients that performed at least as good as the coefficients that have been devised by human intuition, when SLCA is employed as the clustering procedure.

The proposed GP-SLCA methodology is quite flexible since it can be used with a variety of grouping objectives without altering its main operation. On the other hand, as the size of the problem increases, the evaluation function becomes computationally expensive since the value of the coefficient is calculated for every pair of machines in the plant and any potential solution from the constructed dendrogram has to be evaluated.

The research into the applicability of genetic programming for the solution of manufacturing optimisation problems concludes in the next chapter, with the consideration of a problem from the field of multiobjective manufacturing optimisation, the multiobjective process planning selection problem.

Chapter 6

MULTIOBJECTIVE MANUFACTURING OPTIMISATION

6.1 Introduction

In previous chapters single objective formulations of manufacturing optimisation problems were addressed. However, manufacturing practice usually involves decision-making procedures where multiple objectives need to be simultaneously optimised.

When the objectives considered are conflicting in nature, a single optimal solution that simultaneously optimises both objectives does not generally exist. However, there are solutions that perform better than any other solution in the search space for at least one of the objectives. This set of solutions is usually referred to as the *Pareto front*, or the set of *non-dominated solutions*. A formal declaration of Pareto optimality will be presented in the following section. Conventional optimisation methodologies cannot readily cope with this situation since they are designed to search for a single optimal solution within the search space. A compromise solution is usually found by aggregating the conflicting objectives into one optimisation function (see section 5.5.4). Weights are employed for the assignment of partial importance to individual objectives.

The fact that evolutionary algorithms evolve a population of potential solutions makes them ideal for the case of multiobjective optimisation, since they are able to search for all non-dominated solutions in a single optimisation run. In addition, there exist several multiobjective fitness assignment methodologies that help guide the population towards the area of the Pareto front, instead of converging to a single solution.

In this chapter the application of genetic programming to a multiobjective manufacturing optimisation problem is presented. The process planning selection problem is used for this purpose. While the single objective version of the problem can be handled efficiently by flow network optimisation techniques, only evolutionary algorithms have been able to consider non-trivial multiobjective instances of the problem. The proposed methodology employs a number of alternative multiobjective evolutionary techniques to facilitate the search for Pareto-optimal solutions.

The rest of this chapter is organised as follows: In section 6.2 the basic concepts of multiobjective optimisation are discussed and a survey of the main evolutionary multiobjective techniques is presented. Section 6.3 introduces the multiobjective process planning selection problem. The genetic programming based methodology for the generation of process plans is introduced in section 6.4. The application of the proposed methodology in combination with evolutionary multiobjective techniques on a number of test problems is illustrated in section 6.5. The conclusions of this chapter are discussed in section 6.6

6.2 Evolutionary multiobjective optimisation

6.2.1 Introduction to multiobjective optimisation

A problem where a number of non-commensurable objectives need to be simultaneously optimised is defined as a multiobjective optimisation problem. Formally, a general multiobjective optimisation problem can be described as follows (Zitler and Thielle, 1999):

min/max $y = f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$ (6.1)

subject to: $x = (x_1, x_2, \ldots, x_n) \in X$ (6.2)

$y = (y_1, y_2, \ldots, y_n) \in Y$ (6.3)

where: x is the decision vector

y is the objective vector

X is the parameter space

Y is the objective space

The main characteristic of a multiobjective optimisation problem with conflicting objectives is the non-existence of a single decision vector that simultaneously optimises all objectives. Instead, there is a set of solutions for which the performance cannot be further improved in relation to one of the objectives without degrading the performance in relation to one or more of the remaining objectives. These solutions constitute the set of *Pareto-optimal* solutions, or else the *Pareto-front*. All multiobjective solution methodologies seek to find solutions that lie within this set. Pareto-optimal solutions are also known as *non-dominated solutions* since no other possible solution is better than them in terms of all objectives considered.

Formally, and assuming a maximisation problem for all objectives (without loss of generality), the concepts of Pareto-dominance and Pareto-optimality can be defined as follows:

Pareto dominance: Given two decision vectors $a = (a_1, a_2, \ldots, a_n) \in X$ and $b = (b_1, b_2, \ldots, b_n) \in X$, a dominates b (or $a \succ b$) iff:

$$\forall \ i \in \{1,2,\ldots,n\} \quad f_i(a) \geq f_i(b) \quad \wedge \quad \exists \ j \in \{1,2,\ldots,n\} \quad f_j(a) > f_j(b) \qquad (6.4)$$

Pareto optimality: A decision vector $a = (a_1, a_2, \ldots, a_n) \in X$ is Pareto-optimal if there is no other decision vector $b = (b_1, b_2, \ldots, b_n) \in X$ such that $b \succ a$.

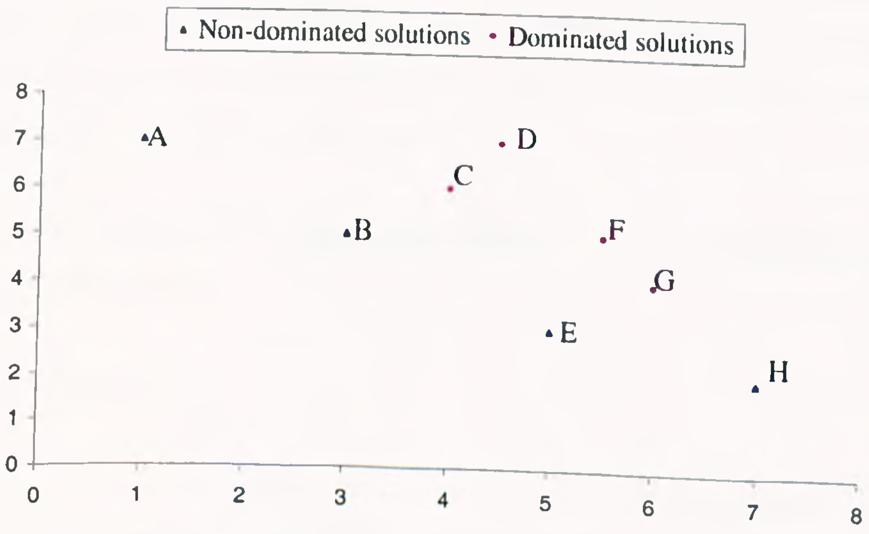


Figure 6.1: Illustration of the concept of Pareto optimality

	A	B	C	D	E	F	G	H
X	1	3	4	4.5	5	5.5	6	7
Y	7	5	6	7	3	5	4	2

Table 6.1: Objective values for the solutions of figure 6.1

These principles are illustrated in figure 6.1. The potential solutions of a multiobjective minimisation problem are represented by points A-H that are illustrated in table 6.1. Solutions A, B, E and H are non-dominated since no other solution is better than they are, when both objectives are considered. These four solutions constitute the Pareto-front for this example problem. Conversely, the remaining solutions (C, D, F, G) are dominated, since there are solutions that perform better than them in respect to both objectives. The Pareto-front provides the Decision-Maker (DM) with a wealth of potential solutions. At the same time it indicates the relative compromises that can be made with respect to the objectives. However, finding the actual Pareto-front is not an easy task. Conventional multiobjective optimisation methods usually attempt to transform the vector optimisation function into a scalar function, using aggregating techniques such as objective weighting and distance functions (Srinivas and Deb, 1994). The aggregation of the objectives means that these methods produce a single compromise solution; thus the whole length of the Pareto-front is not explored, at least in a single optimisation run. Alternative weight assignments have to be made in order to focus on different regions of the front.

Evolutionary computation techniques offer an alternative approach to multiobjective optimisation based on the foundations laid by Goldberg (1989) in the form of the Pareto-ranking approach. In the next section the main evolutionary multiobjective techniques will be discussed in more detail.

6.3 Evolutionary computation for multiobjective optimisation

6.3.1 Introduction

The ability of evolutionary algorithms to conduct a search in the solutions' space from a population of points in parallel is particularly useful for the solution of multiobjective optimisation problems. Evolutionary computation is perhaps the only optimisation procedure that has the natural ability to provide a set of potential solutions in a single optimisation run. At the same time, it provides a computationally feasible approach for the solution of large instances of multiobjective optimisation problems. A number of significant evolutionary multiobjective techniques that have been proposed over the years will be reviewed in the following paragraph.

6.3.2 Review of evolutionary multiobjective techniques

Schafer (1985) was the first researcher to propose the use of evolutionary algorithms for the solution of multiobjective optimisation problems. His Vector Evaluated Genetic Algorithm (VEGA) operated as a normal genetic algorithm. However, in the selection step of the algorithm each generation was divided into a number of parts equal to the total number of objectives considered. Each part was then filled with individuals selected based only on their performance on the respective objective. This approach was unable to explore the whole length of the Pareto front since the alternate selection scheme was promoting solutions that occupied the extreme regions. For that reason the evolutionary procedure had the tendency to neglect compromise solutions.

The need to include Pareto dominance information in the fitness assignment procedure was addressed by Goldberg (1989) (pages 197-201) who proposed, without actually implementing, an efficient scheme for the promotion of non-dominated solutions. Goldberg's scheme was based on the assignment of ranks to individual solutions according to their dominance level. The procedure started by identifying the non-dominated solutions of the entire population and assigning them with rank one. The search for non-dominated solutions was repeated, however, the solutions that had already been ranked were not considered in the process. A new set of non-dominated solutions was found and assigned with the next rank. The procedure was repeated, until all solutions in the population had been associated with a rank. Fitness was then assigned to individuals according to their rank.

Goldberg's ranking scheme, while unveiling the partial ordering of the solutions' space, provided the evolutionary algorithm with the necessary tools to promote non-dominated solutions without focusing on a specific area of the Pareto front. However, as in single-objective multimodal optimisation, genetic drift, the tendency of the evolutionary algorithm to converge to one of the equally ranked Pareto-optimal solutions, still limited the search to a fraction of the Pareto front. In order to maintain the diversity of the Pareto-front, Goldberg suggested the use of *niching*, a technique that had been employed successfully to fight genetic drift in multimodal optimisation.

Niching aims to form stable sub-populations of solutions across the length of the Pareto front. *Fitness sharing* is the niching technique employed by the majority of researchers in evolutionary computation. It is based on the idea that all individual solutions within the same niche have to share the same resources. In other words, the fitness of individual solutions is degraded in proportion to the number of solutions that belong to the same niche (*niche count*). In that way, the algorithm does not suffer from premature convergence, since the fitness of any solution that attempts to dominate the population is immediately degraded through the sharing scheme.

A *sharing function* is used for the calculation of the niche count of individuals within the same rank. The triangular sharing function is usually employed for this purpose:

$$Sh(d) = \begin{cases} 1 - \left(\dfrac{d}{\sigma_{share}} \right) & if\ (d < \sigma_{share}) \\ 0 & otherwise \end{cases} \tag{6.5}$$

where: d is the distance between two individual solutions

σ_{share} is the *niche radius*

For each individual solution, the value of the sharing function is calculated with respect to the solutions belonging to the same rank. The fitness of the solution is degraded in proportion to the sum of its sharing values. The distance between solutions can be defined either in the parameter space (decision vector) or the objective space. The parameter σ_{share} determines the size of the niche. The shape of the niche depends on the metric used for the calculation of distance between individual solutions (Horn and Nafpliotis, 1993). Deb and Goldberg (1989), Fonseca and Fleming (1993) and Horn and Nafpliotis (1993) have proposed guidelines for the selection of the σ_{share} value. The latter suggestions were used for the calculation of the σ_{share} value in the experiments conducted in this chapter. Alternative techniques for the avoidance of genetic drift include the crowding scheme (De Jong, 1975) and the mating restriction scheme (Deb and Goldberg, 1989).

Goldberg did not present a practical implementation of his ideas. However, subsequent researchers extended the concept of Pareto ranking and presented applications on multiobjective optimisation problems. Fonseca and Fleming (1993, 1998) introduced the Multiobjective Genetic Algorithm (MOGA). MOGA employed a modified ranking scheme, in which a solution's rank was proportional to the number of solutions by which it was dominated. This scheme produced a more fine-grained classification of solutions in comparison to Goldberg's approach, since a greater number of individual ranks existed within the population. Fitness sharing was performed in the objective space between individuals belonging to the same rank.

In the same year Horn and Nafpliotis (1993) presented an evolutionary multiobjective technique called Niched Pareto Genetic Algorithm (NPGA). The main feature of this algorithm was the use of tournament selection as the driving force of the evolutionary procedure. The authors employed a modified version of the conventional tournament selection scheme, which explicitly considered Pareto dominance information. Initially, two individuals were selected randomly from the population. Another set of individuals was also selected from the population for comparison purposes. The user could determine the selective pressure by changing the size of this comparison set. Each of the two individuals was compared with each individual in the comparison set. If one of the individuals was not dominated by solutions in the comparison set and the other was, then the former was selected for the potential genetic operation. If neither or both individuals were non-dominated solutions, then the individual with the lowest niche count was selected.

As in MOGA, fitness sharing was implemented in the objective space, between individuals having the same rank. However, the authors indicated that the combination of tournament selection with fitness sharing could lead the algorithm to exhibit chaotic behaviour (Oei, Goldberg and Chang, 1991). Instead, a modified ranking scheme was proposed, originally suggested by Oei, Goldberg and Chang, in which the calculation of niche counts was not based on the current population of solutions, but on the partially filled next generation. The sampling of the latter population was suggested, in an attempt to reduce the computational overhead induced by the updated sharing scheme.

The Non-dominated Sorting Algorithm (NSGA) (Srinivas and Deb, 1994) followed closely the guidelines proposed by Goldberg. Initially, the first set of non-dominated solutions was found and each solution was assigned with a large 'dummy' fitness value. Fitness sharing was implemented within the rank; however, the procedure took place in the parameter rather than the objective space. The first set of non-dominated solutions was removed from consideration and the second set of non-dominated solutions was found. These solutions were assigned a 'dummy' fitness value that was smaller than the minimum fitness value of the individuals included in the first set of non-dominated solutions, after fitness sharing had been implemented. This technique ensured that individual solutions were always assigned with higher fitness values than those having a lower rank.

Recently, Zitzler and Thiele (1999) presented the Strength Pareto evolutionary multiobjective technique that attempted to combine the positive characteristics of the previous methods. Their algorithm always maintained two sets of solutions in each generation; the evolved population *P* and the off-line set *P'* that contained the up-to-date non-dominated solutions. Any new solution generated was compared with the set of solutions in *P'*. If it was not dominated, then it was inserted in *P'*, while the solutions covered by it in *P'* were removed from the set. Both *P* and *P'* took part in the selection process.

The main feature on this methodology was its fitness assignment procedure. Each solution in *P'* was assigned with a strength value that was proportional to the number of solutions that it dominated in *P*. At the same time, the fitness of solutions in *P* was proportional to the sum of the strength values of the solutions in *P'* that dominated them. The intuition behind this technique was an attempt to promote dominated solutions that were covered only by a small number of non-dominated solutions and penalise dominated solutions that contained a large number of neighbourhood non-dominated solutions. This form of niching was based only on dominance information; there was no need to calculate the distance between individual solutions. The authors also suggested reducing the size of *P'* by means of clustering. The average linkage clustering method was employed for this purpose. The reduced set *P'* was generated by choosing the centroid of each cluster as a representative solution.

Fonseca and Fleming (1995) and Zitzler and Thiele (1999) discuss in detail the issues associated with evolutionary multiobjective optimisation and present analytical reviews of relative techniques that have been proposed over the years.

6.4 The multiobjective process planning selection problem

One of the main stages of the process planning procedure is the selection of the optimal process plan from the set of the potential process plans that exist for the product considered. The problem is usually referred to as the process planning selection problem. A formal declaration of the problem and a review of the evolutionary computation approaches that have been proposed for its solution are presented in section 2.5.3.

When the objective of the optimisation is the minimisation of a single objective, then the process planning selection problem is equivalent to the Shortest Path Problem (SPP). The SPP can be solved efficiently by network algorithms such as Dijkstra's (Jensen and Barnes, 1980). However, these algorithms are unable to handle the multiobjective version of the problem. Conventional optimisation techniques such as goal programming (Cohon, 1978), linear programming (Swaragi *et al.*, 1985) and dynamic programming (Sniedovich, 1985) have been used for this purpose, however, their application has been illustrated only in small-sized networks.

Evolutionary computation is the only optimisation method that has been employed for the solution of medium to large size instances of the multiobjective process planning selection problem. More specifically, Awadh *et al.* (1995) combined their binary encoded genetic algorithm (section 2.5.3) with a weighted-sum approach for the solution of the multiobjective version of the problem. The objective of their algorithm was the simultaneous minimisation of cost and maximisation of quality of the process plans. However, their simple aggregating multiobjective approach provided only a single compromise solution between the objectives, without considering the concept of Pareto optimality.

Zhou and Gen (1997) combined their real value-coded evolutionary algorithm (section 2.5.3) with a multiple criteria decision technique (Chankong and Haimes, 1983). This approach differed from the one proposed by Awadh *et al.*, in the sense that it evolved solutions across the length of the Pareto front and not a single compromise solution. However, not enough experimental evidence was presented to assess the efficiency of the proposed algorithm.

While both these approaches constituted a significant advance in the solution of the multiobjective process planning selection problem, their search for the set of Pareto optimal solutions was inefficient. In the next section a new methodology is presented, which addresses this issue by combining a novel genetic programming approach for the generation of potential process plans with a number of established evolutionary multiobjective techniques.

6.5 A genetic programming-based methodology for the solution of the multiobjective process planning selection problem

6.5.1 Introduction

In this section a novel methodology for the solution of the multiobjective process planning selection problem is introduced. The proposed framework consists of two semi-independent parts: A genetic programming algorithm is responsible for the generation of potential process plans. The search for Pareto-optimal solutions is implemented through a number of alternative evolutionary multiobjective techniques.

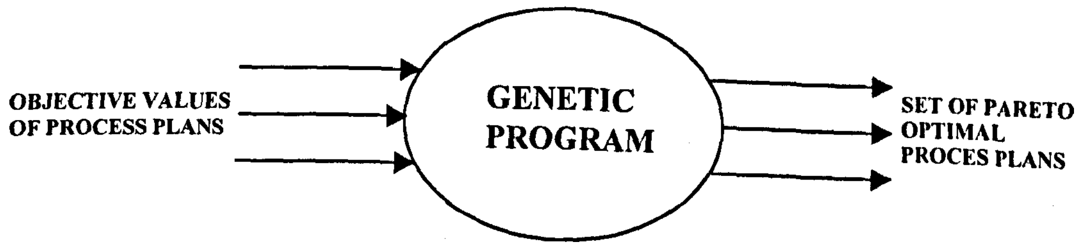

OBJECTIVE VALUES OF PROCESS PLANS
GENETIC PROGRAM
SET OF PARETO OPTIMAL PROCES PLANS

Figure 6.2: Genetic programming approach to the generation of Pareto-optimal process plans

As has already been discussed the design of the genetic programming algorithm requires the redefinition of the problem considered in a program-induction form. For the case of the multiobjective process planning selection problem, this definition takes the following form: "Find a computer program that takes as input information about the objective values associated with each process plan and produces as output a set of

Pareto-optimal process plans". Figure 6.2 illustrates the genetic programming approach for the solution of the multiobjective process planning selection problem.

6.5.2 Design of the algorithm

6.5.2.1 Representation of process plans

The process plan representation schemes employed by evolutionary techniques that have been proposed for the solution of the general process planning selection problem, are based on the network formulation of the problem. These algorithms evolve strings of binary or real-coded values that correspond to valid process plans for the problems considered (section 2.5.3).

The genetic programming methodology introduced in this chapter exploits the same formulation in its attempt to generate potential process plans. Since the network formulation transforms the process planning selection problem into a routing problem, the genetic programming algorithm evolves computer programs that aim to guide the product through the required processing stages in a way that simultaneously optimises all objectives considered.

More specifically, the product moves through the arcs and nodes of the network with the help of navigating instructions that are evolved by genetic programming in the form of a computer program. Any program that manages to find a complete path through the network corresponds to a unique process plan for the product considered.

The product is placed at the first node of the network and its navigation starts from an arc specified by the user. In the applications presented in this chapter, the right-most arc of the node constitutes the starting point of the navigation (figure 6.3). However, this is only an issue of convention and alternative initialisation schemes may also be used.

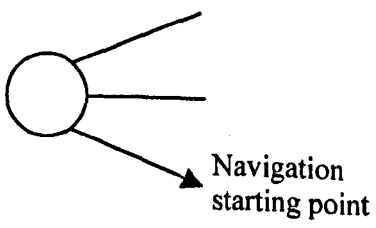


Figure 6.3: Initialisation of the navigation procedure

In each node the computer program can move between arcs (i.e. alternative process plans) using the navigation commands RIGHT and LEFT, which move the product to the node placed immediately left or right of the present node respectively (Figure 6.4).

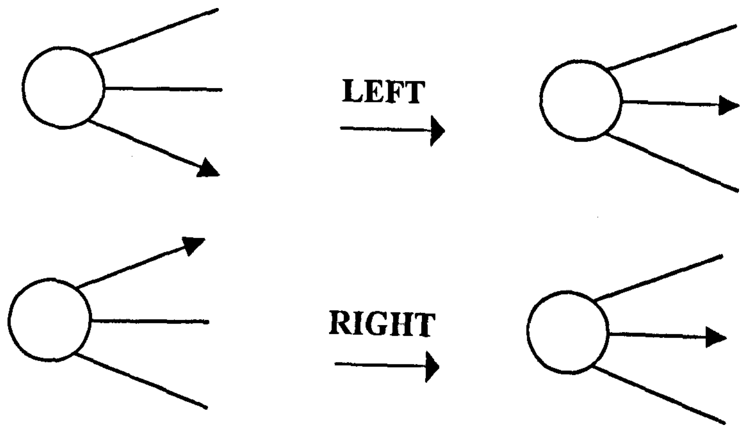


Figure 6.4: Effect of LEFT and RIGHT commands on the navigation procedure

The operation of these commands assumes that there are no borders at the extreme arcs of the network nodes. The application of the LEFT command on the left-most arc of the node and the application of the RIGHT command on the right-most arc of the node, positions the product on the right-most arc and the left-most arc of the node respectively (Figures 6.5-6.6).

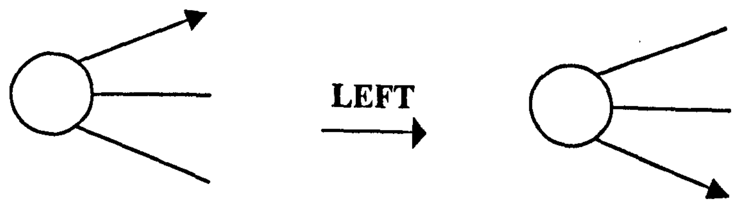


Figure 6.5: Effect of the LEFT command on left-most node of the arc

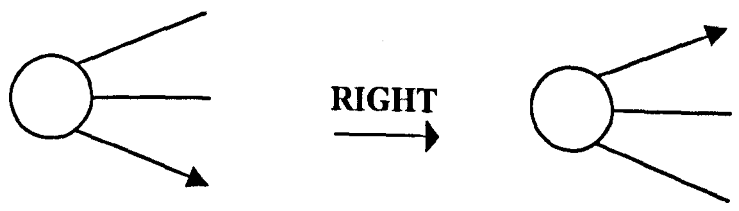


Figure 6.6: Effect of the RIGHT command on the right-most node of the arc

The product moves to the next processing stage with the help of the MOVE command. This command places the product on the node that is pointed by the current arc. The relative positioning of the product on the new node follows the convention of the right-most arc (figure 6.7).

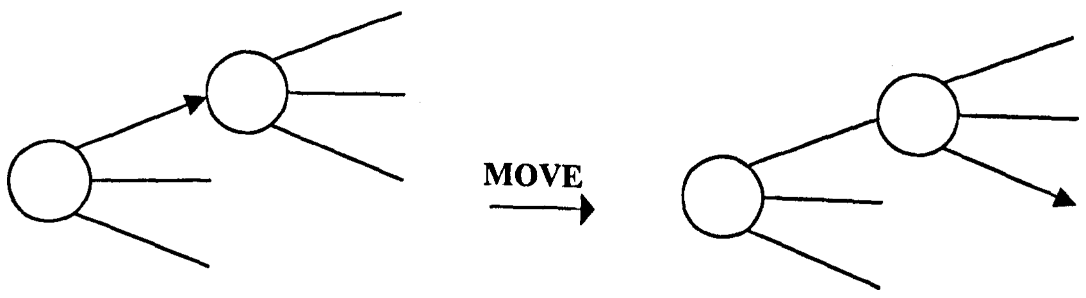


Figure 6.7: Effect of the MOVE command on the navigation procedure

The navigation commands are executed sequentially with the help of the PROGN function (a function that allows the sequential execution of two or more function or terminal nodes, section 4.3.2.1, Figure 4.2). The complete navigation program moves the product through the various processing stages.

The genetic programming approach for the generation of process plans will be illustrated with the help of an example problem taken from Awadh *et al.* (1995). The network flow model of a process planning selection problem is illustrated in figure 6.8.

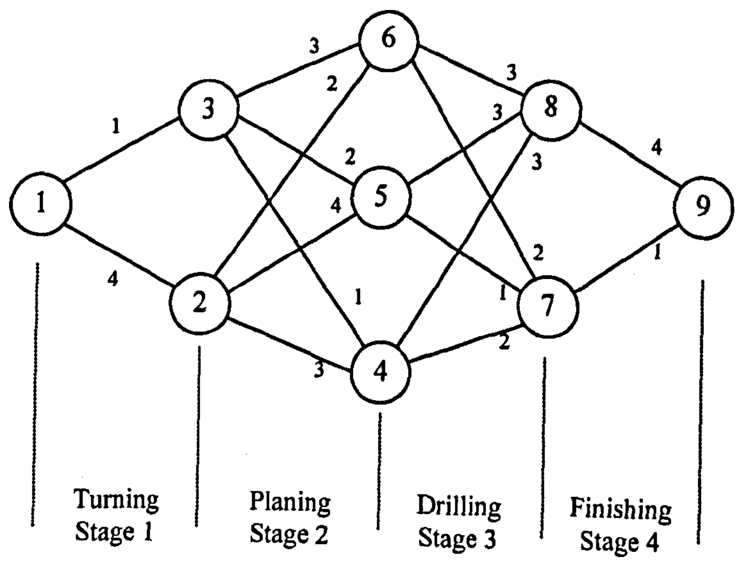


Figure 6.8: Network flow model of the example process planning selection problem

Each arc in the network is associated with a value that represents the cost of following a specific sequence of operations. The objective in this problem is the minimisation of the overall cost of the product's process plan. Figure 6.9 illustrates a computer program that generates the optimal process plan (cost=5) for the example problem. The detailed navigation of the product is presented in figure 6.10.

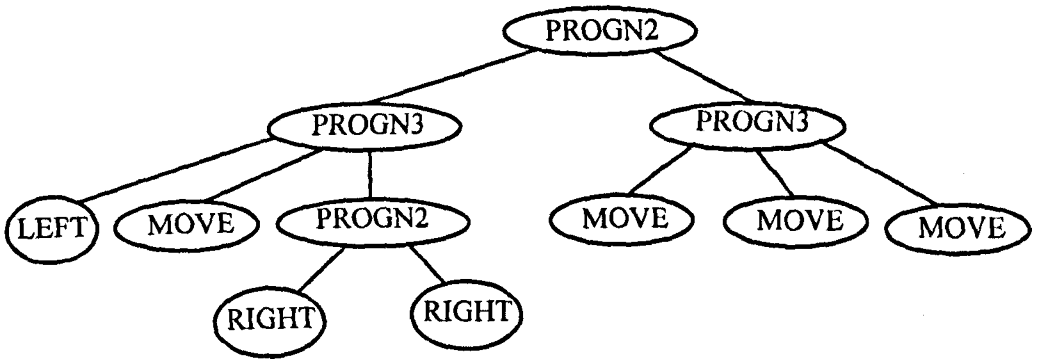


Figure 6.9: Evolved computer program for the generation of the minimum cost process plan

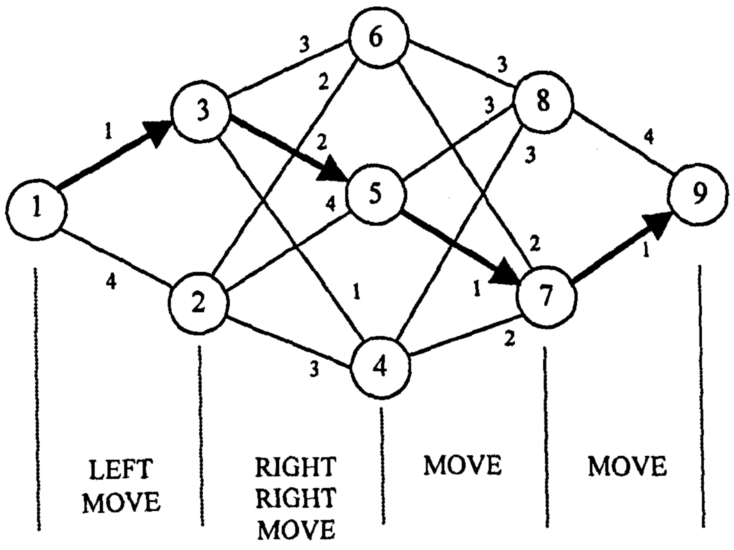


Figure 6.10: Step-by-step generation of the minimum cost process plan

6.5.2.2 Function set

The PROGN function executes sequentially two or more function or terminal nodes that form its set of arguments. As in the scheduling application presented in chapter 4, the PROGN2 and PROGN3 functions formed the function set of the genetic programming algorithm. The valid operation of these functions requires the specification of argument sets of size 2 and 3 respectively.

6.5.2.3 Terminal set

The three navigational commands LEFT, RIGHT and MOVE formed the set of terminals for the genetic programming algorithm. The LEFT and RIGHT commands alternated the choice of the potential processing sequence of a product for a specific processing stage. The MOVE command confirmed the processing sequence choice and initialised the product position for the next processing stage.

6.5.2.4 Genetic operators

The subtree crossover and mutation operators were employed by the evolutionary procedure for the exchange of genetic material between individual solutions and the generation of diversity within the population. The probability of applying the crossover and mutation operators was set to 90% and 10% respectively.

6.5.2.5 Objective function

The minimisation of processing cost and the maximisation of product quality were chosen as the objectives to be simultaneously optimised by the evolutionary procedure. The same objectives were employed in the multiobjective process planning selection applications presented by Awadh *et al.* (1995) and Zhou *et al.* (1997).

The proposed methodology performed a non-aggregating evaluation of generated process plans using three alternative evolutionary multiobjective techniques (section 6.3.2). The first approach (which will be identified as 'Pareto' from this point onwards) followed the guidelines proposed by Goldberg for ranking individual solutions according to their dominance information. A ranking fitness assignment methodology introduced by Fonseca and Fleming (1993) was employed for the calculation of fitness values. Individual solutions were selected to participate in genetic operations with the help of the Stochastic Universal Sampling (SUS) technique (Baker, 1987).

The Multiobjective Genetic Algorithm (MOGA) of Fonseca and Fleming (1993) was also employed as an alternative approach to the multiobjective evaluation of potential process plans. The implementation of the algorithm followed closely the guidelines set by the authors in terms of the rank assignment, fitness assignment and selection methods (SUS).

The final evolutionary multiobjective algorithm used in the experiments was the Niched Pareto Genetic Algorithm (NPGA) of Horn and Nafpliotis (1993). The size of the comparison set for the tournament selection procedure, as well as the sampling rate for the calculation of niche counts (see section 6.3.2) was set to 20% of the population size used in the experiments.

6.5.2.6 Additional parameters

The Koza tableau of table 6.2 illustrates the value of additional parameters that are necessary for the valid run of the genetic programming algorithm and summarises the multiobjective process planning selection solution methodology presented in this chapter.

Parameters	**Values**
Objective:	Simultaneous minimisation of cost and maximisation of quality of potential process plans for individual products.
Terminal set:	LEFT, RIGHT, MOVE
Function set:	PROGN2, PROGN3
Population size:	500-1000 (depending on problem size)
Evolutionary multiobjective technique:	Pareto, MOGA, NPGA
Selection:	Stochastic universal sampling (Pareto, MOGA), tournament (NPGA)
Subtree crossover probability:	0.9
Mutation probability:	0.1
Number of generations:	50
Maximum depth for crossover:	17
Initialisation method:	Ramped half and half

Table 6.2: Koza tableau for the multiobjective process planning genetic programming algorithm

6.5.3 Experimental basis

The experimental basis employed in this chapter comprised of ten multiobjective process planning selection problems with sizes ranging from (7 stages × 24 nodes) to (15 stages × 89 nodes). The test problems employed by Awadh *et al.* (1995) and Zhou *et al.* (1997) featured the same characteristics in terms of number of stages and number of nodes. However, while both authors were contacted, they were unable to provide their experimental basis and the respective results. For this reason, a new set of multiobjective process planning selection problems had to be randomly generated. The cost and quality values associated with each alternative processing sequence were randomly chosen from the uniform distribution [1, 50]. The minimum cost and maximum quality values for each of these problems were found through exhaustive enumeration. These values represented the solutions lying on the extreme regions of the Pareto front. They were also used for the calculation of the σ_{share} value in the fitness sharing procedure of the evolutionary multiobjective techniques. A summary of the experimental basis employed in this chapter is presented in table 6.3.

Problem no.	Number of stages	Number of nodes	Number of possible plans	Minimum cost value	Maximum quality value
1	7	24	2025	75	242
2	7	27	4320	83	292
3	8	38	86400	52	354
4	9	37	58800	52	337
5	10	47	756000	66	382
6	11	53	6.174×10^{6}	99	473
7	12	63	4.97×10^{7}	96	521
8	13	72	4.57×10^{8}	95	579
9	14	79	5×10^{9}	129	624
10	15	89	5.48×10^{10}	97	655

Table 6.3: Experimental basis for the multiobjective process planning selection problem

20 runs of the genetic programming algorithm were conducted for each evolutionary multiobjective technique on the process planning selection problems included in the experimental basis. For each technique, the off-line set of non-dominated solutions over the batch of 20 runs was recorded. The population size was set to 500 for problems 1-5, and 1000 for the larger problems 6-10. The results of the experimental phase are presented in the following section.

6.5.4 Results

The off-line sets of non-dominated solutions that were evolved by the combination of genetic programming with the evolutionary multiobjective techniques are presented in figures 6.11-6.20. For ease of illustration the non-dominated solutions of each technique have been connected with continuous coloured lines.

6.5.5 Discussion

The results presented in figures 6.11-6.20 illustrate that the combination of the genetic programming procedure for the generation of process plans with evolutionary multiobjective techniques provide a variety of potential solutions for the DM. However, the relative performance of the proposed methodology cannot be sufficiently evaluated for the following reasons:

- No comparative results exist on the same set of problems from alternative solution methodologies.
- The actual Pareto-front of the problems considered is not known in advance.

An indication of the algorithm's performance can be provided by the visual comparison of the evolved sets with the extreme values of the Pareto front, which were calculated through exhaustive enumeration. Based on this information, it can be said that for the small-sized problems (1 and 2) there is an indication that the algorithm evolved the actual Pareto-front because all evolutionary multiobjective techniques produced the same set of non-dominated solutions. However, since the actual Pareto-front is not known, no positive conclusions can be reached. While some extreme Pareto-front values (minimum cost and maximum quality) were evolved for bigger test problems as well (problems 3 and 4), the algorithm faced difficulties in reaching extreme Pareto regions as the size of the problem increased.

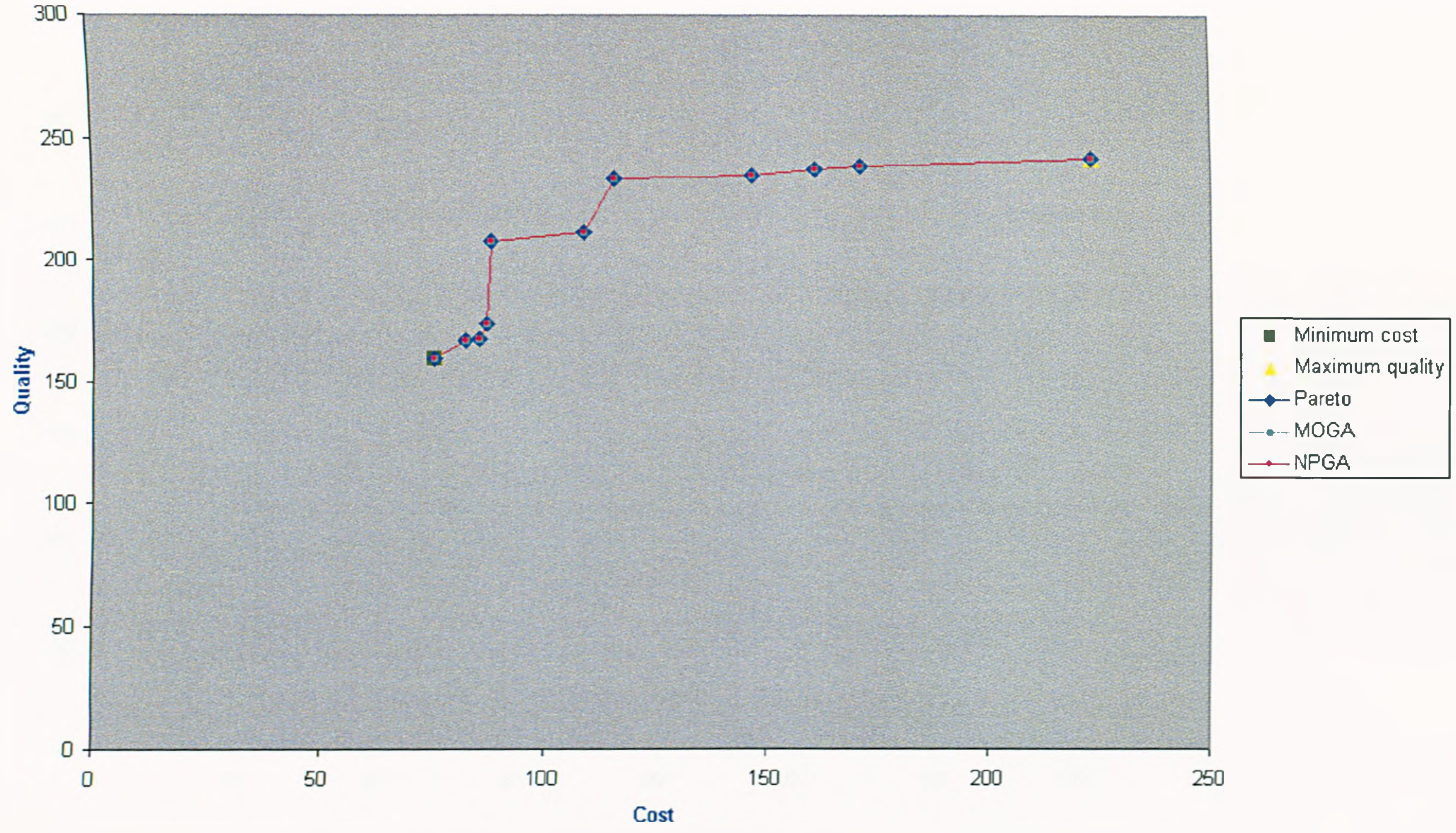


Figure 6.11: Evolved solutions for test problem 1

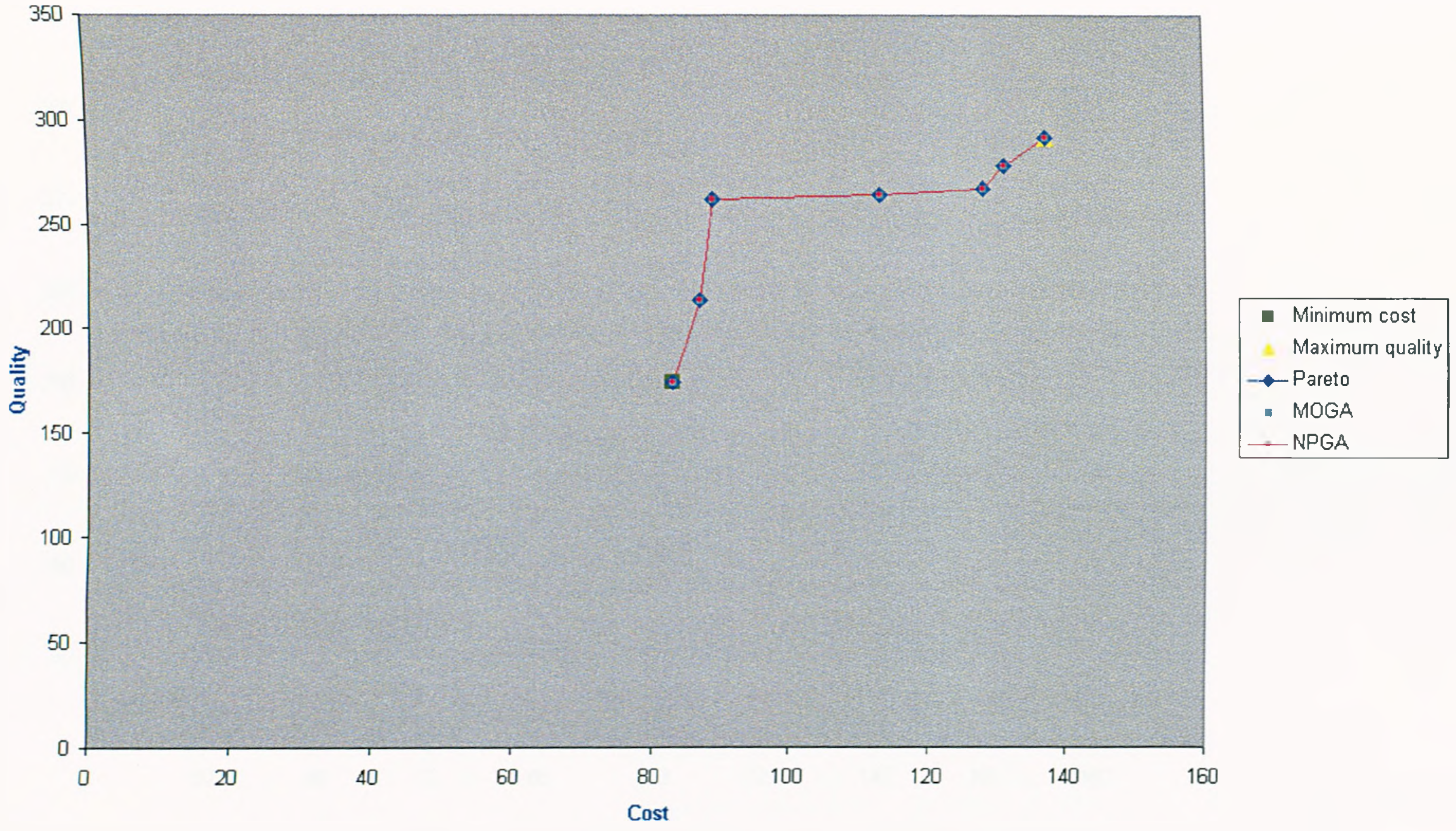


Figure 6.12: Evolved solutions for test problem 2

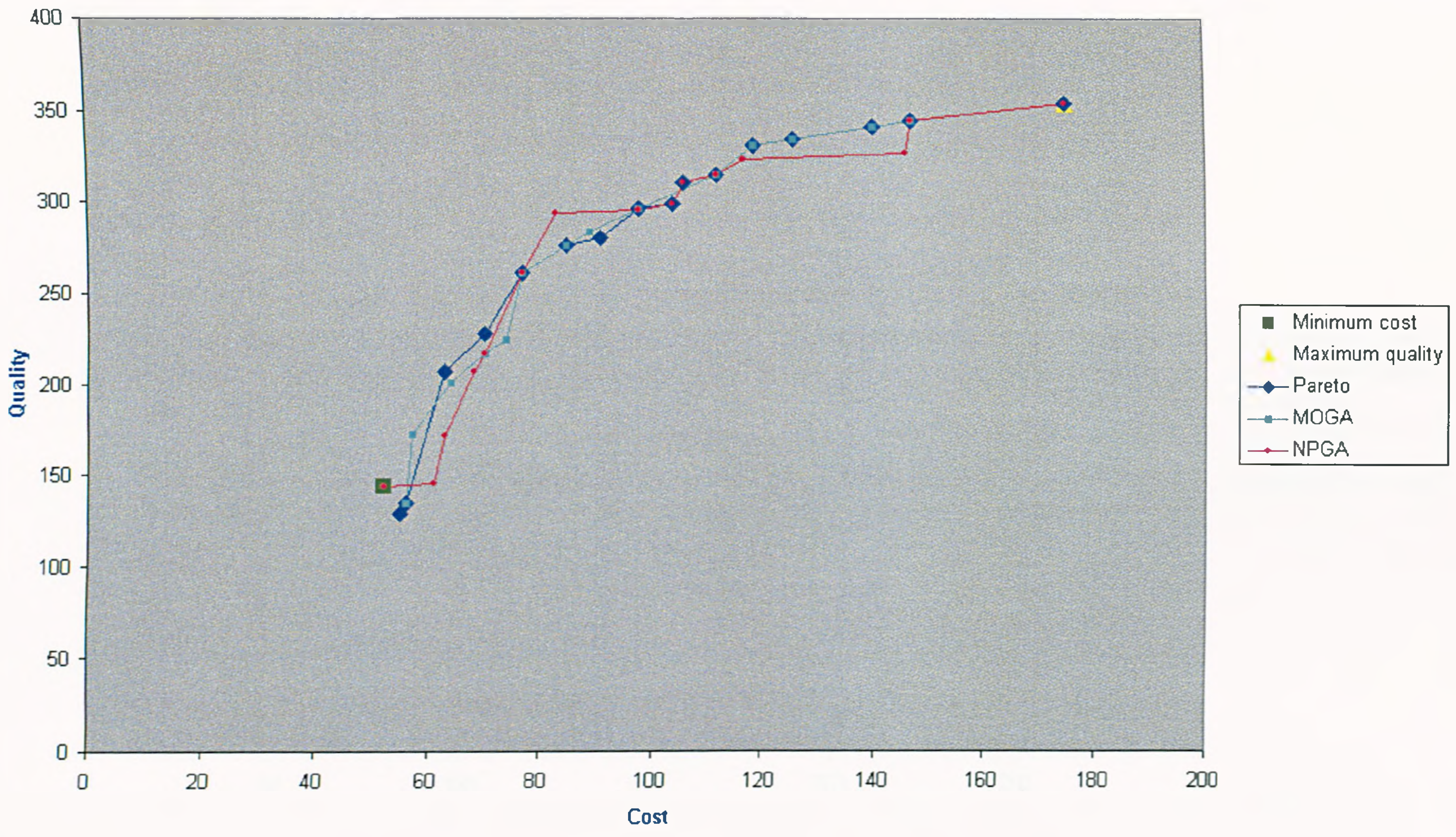


Figure 6.13: Evolved solutions for test problem 3

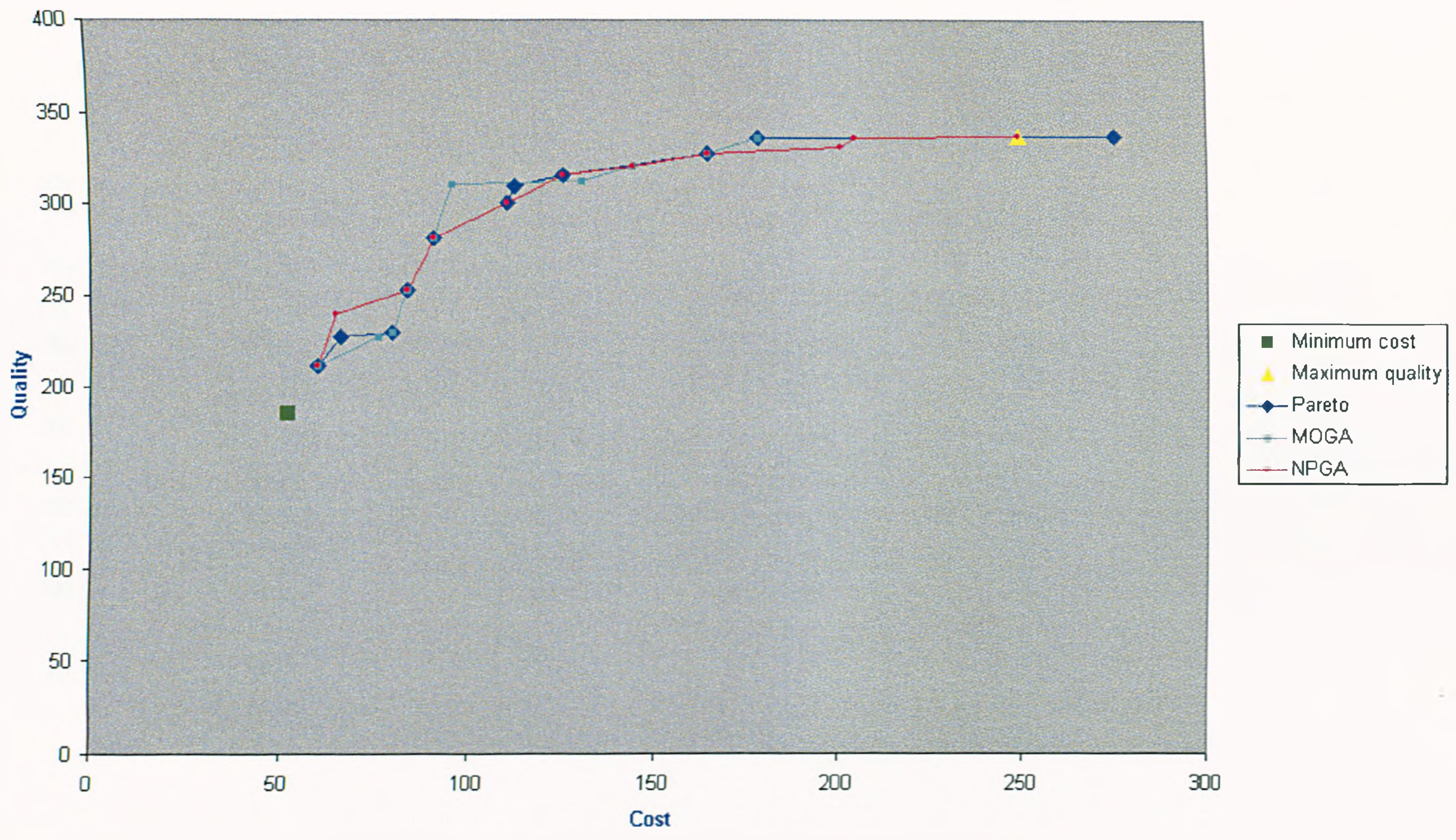


Figure 6.14: Evolved solutions for test problem 4

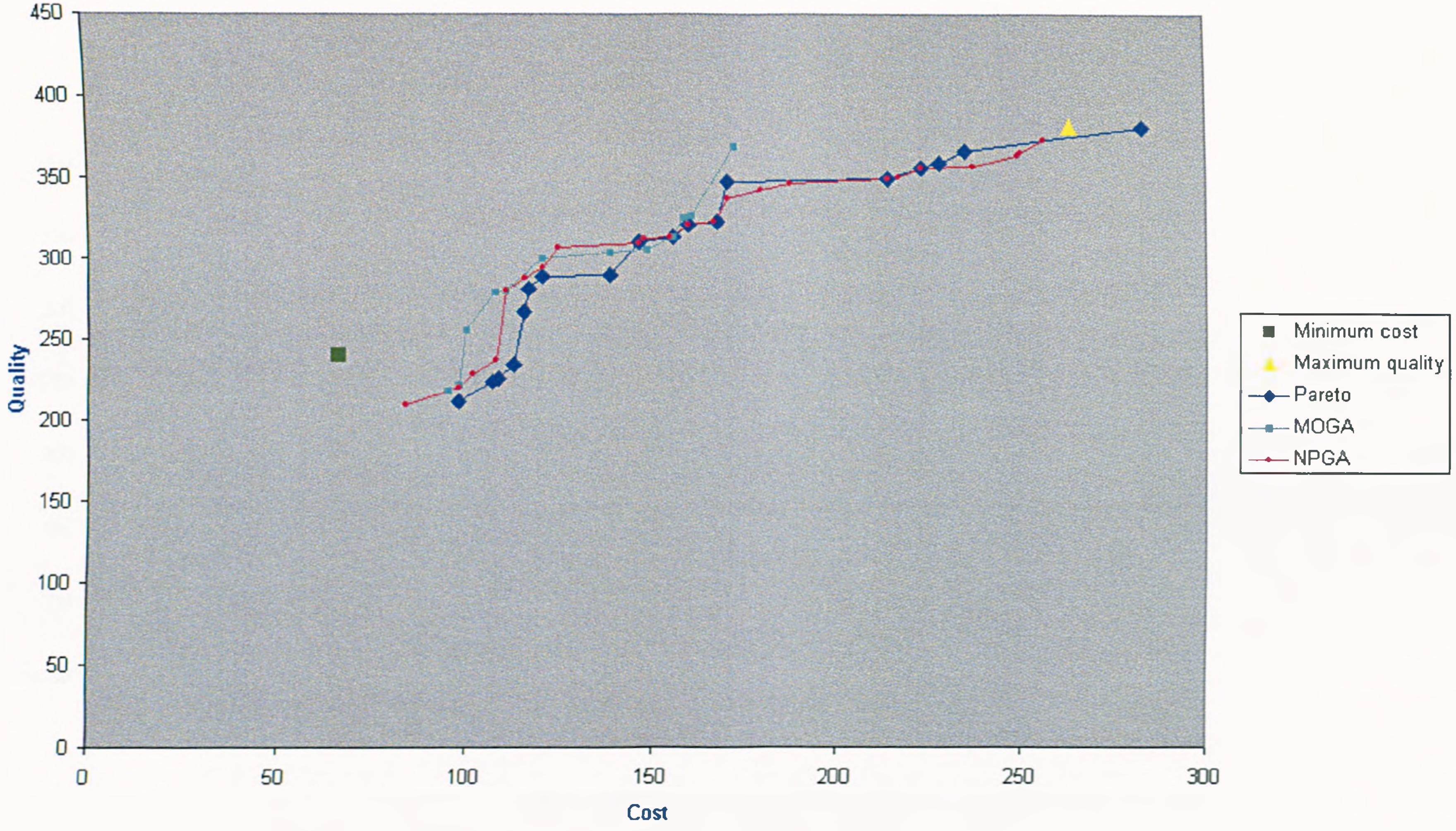


Figure 6.15: Evolved solutions for test problem 5

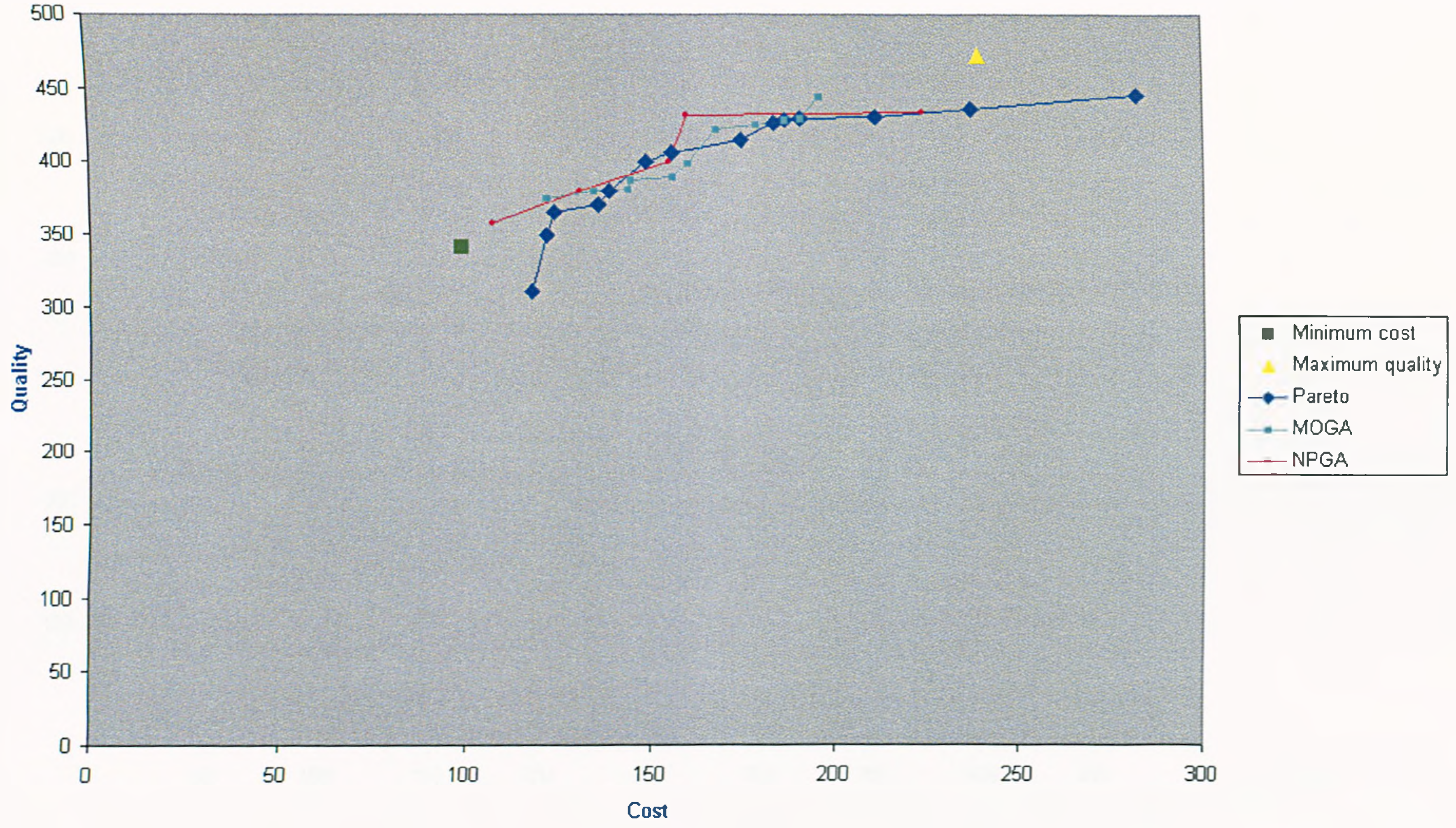


Figure 6.16: Evolved solutions for test problem 6

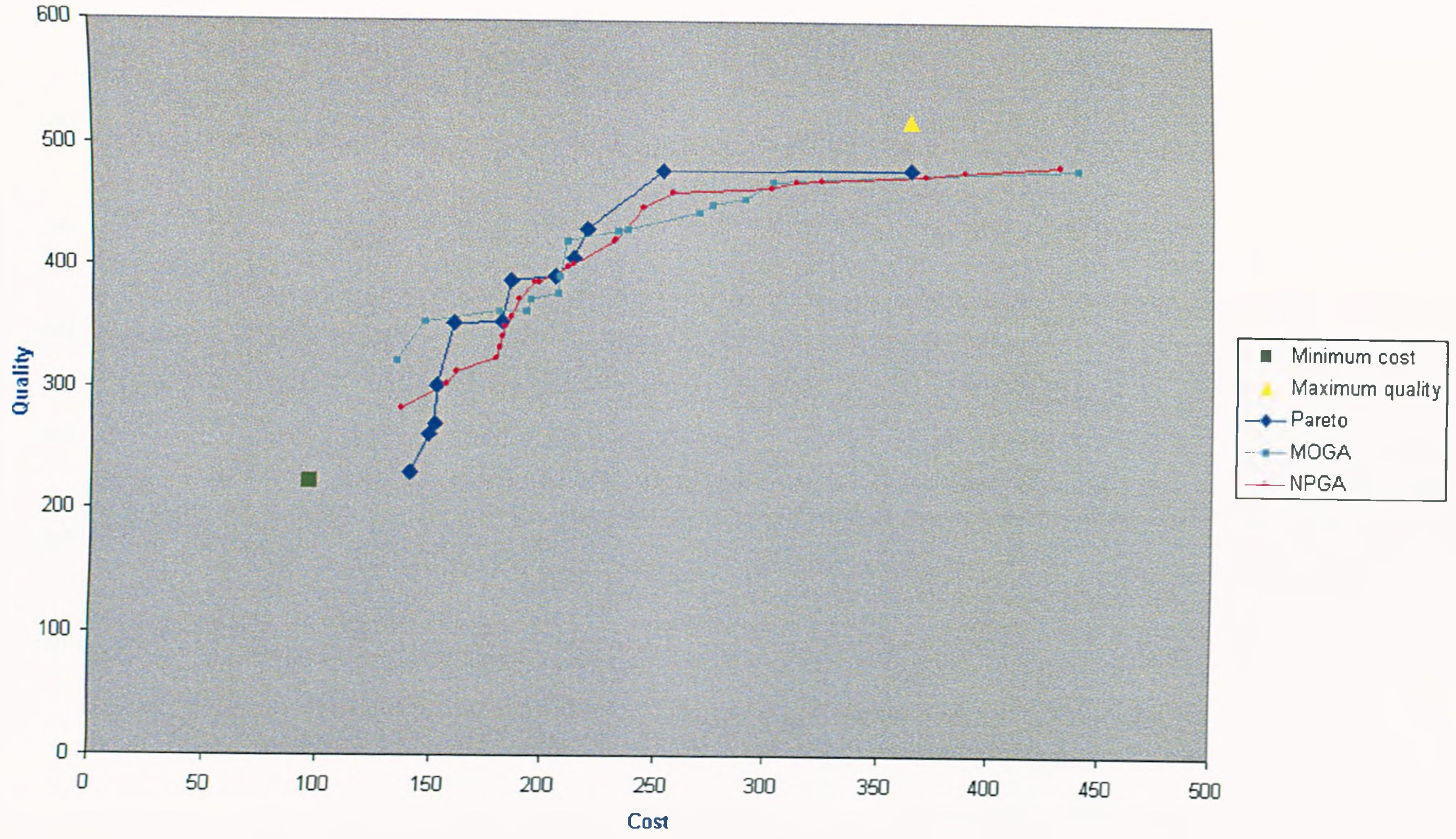


Figure 6.17: Evolved solutions for test problem 7

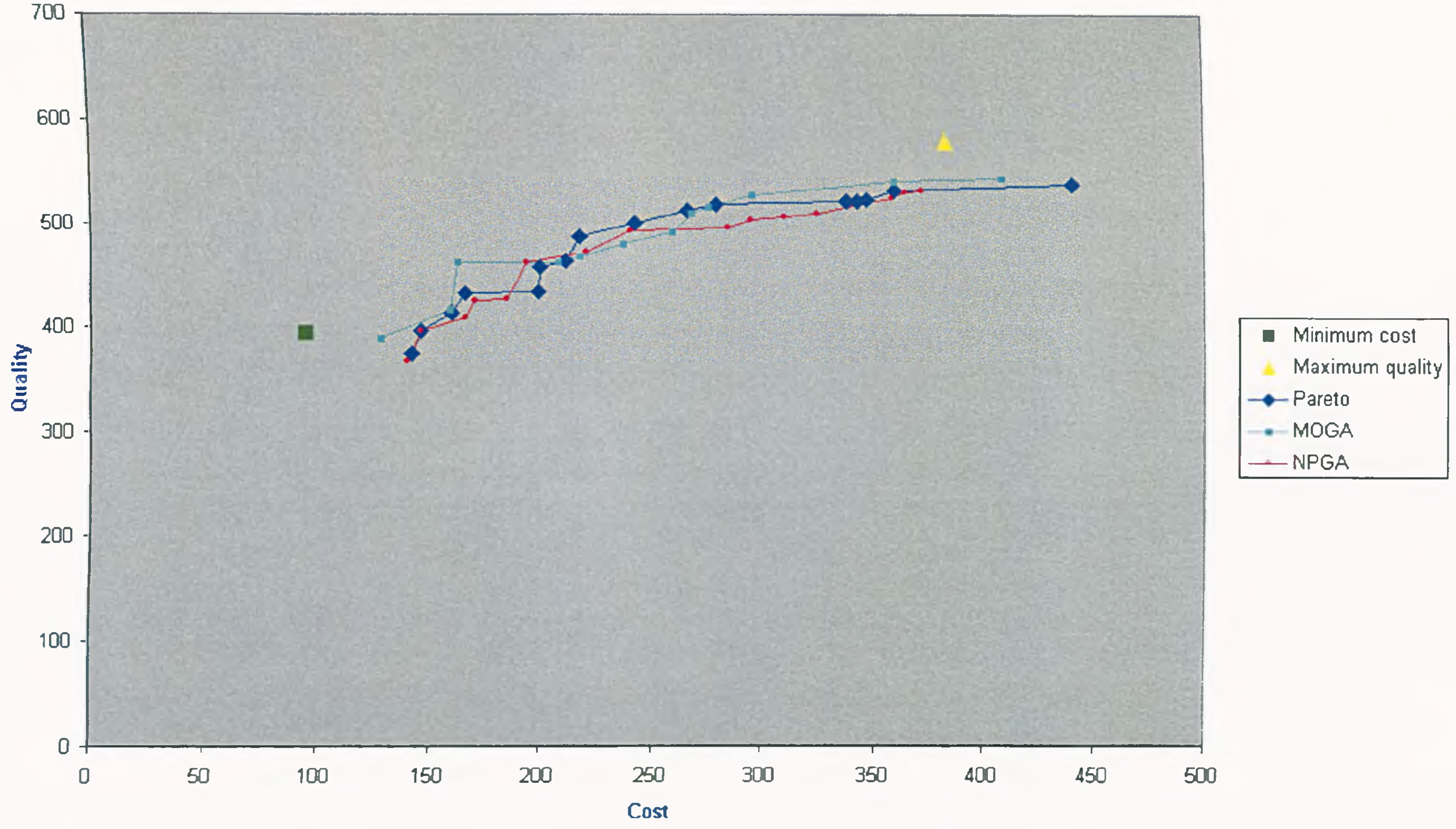


Figure 6.18: Evolved solutions for test problem 8

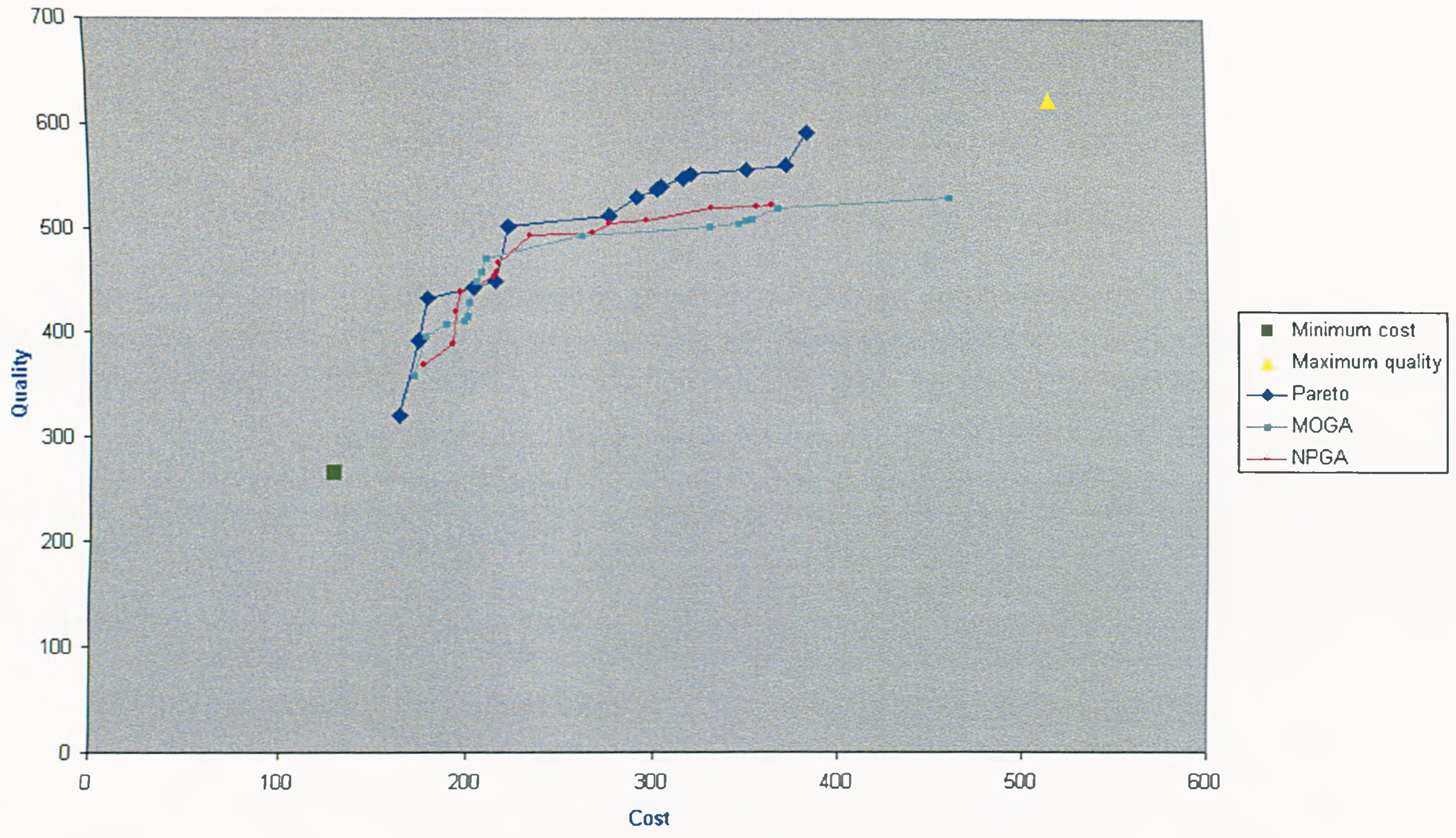


Figure 6.19: Evolved solutions for test problem 9

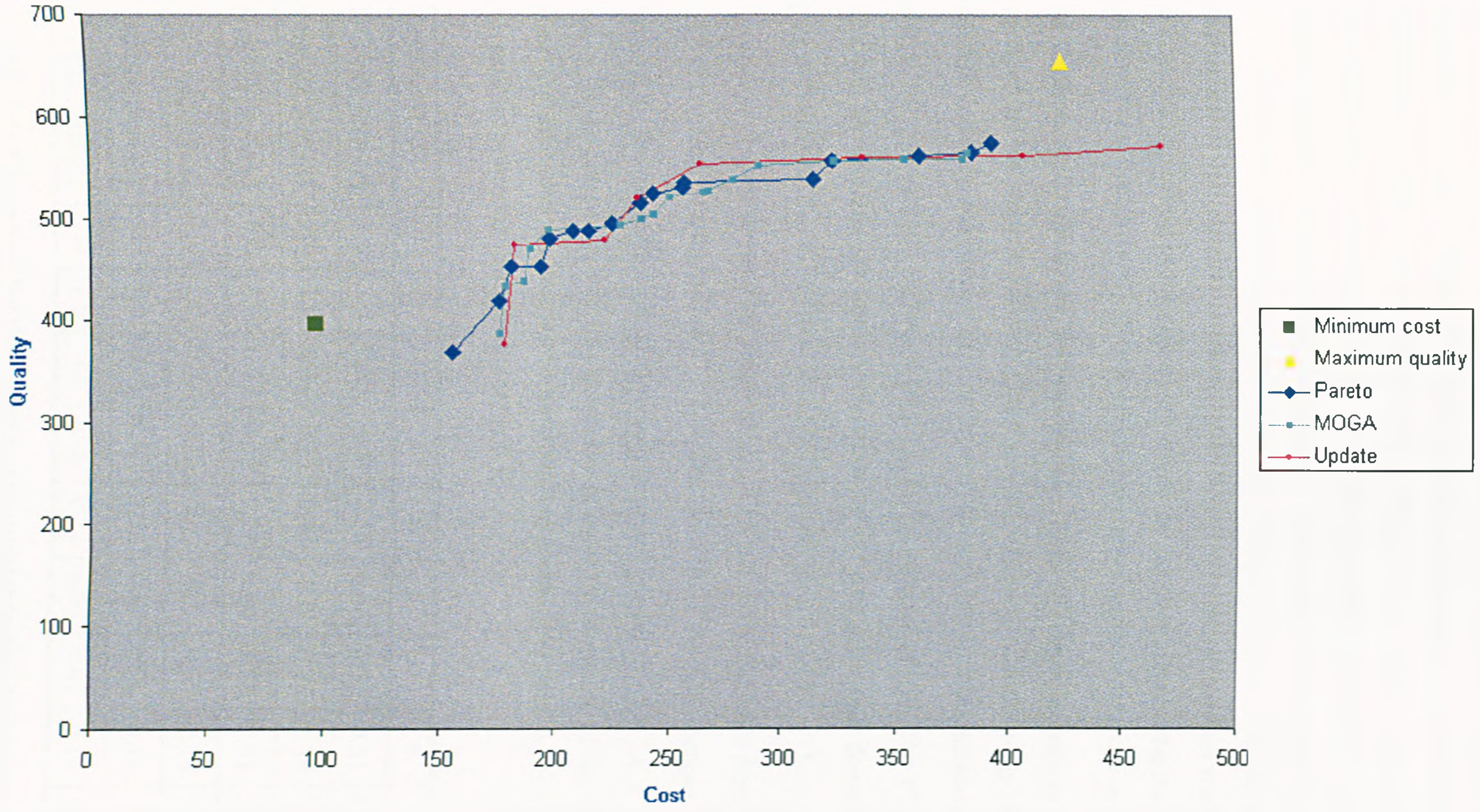


Figure 6.20: Evolved solutions for test problem 10

The performance of the algorithm degraded as the size of the problems increased. While for a number of problems there were indications that evolved solutions were situated close to the actual Pareto-front (problems 5, 6, 7 and 9), the Pareto-front evolved for some of the large-sized problems (problems 8 and 10) was probably entirely dominated by the actual Pareto-front. This is not an unexpected result since the number of potential process plans for these cases is considerable for a multiobjective optimisation problem (table 6.3). The attempt that was made to offset the massive expansion in the size of the solutions' space by increasing the population size of GP was insufficient, and thus additional computational resources are required.

The relative performance of the evolutionary multiobjective techniques that were employed in the experimentation phase can be assessed quantitatively through the coverage measure introduced by Zitzler and Thiele (1999). This measure maps an ordered pair of decision vectors $C(X',X'')$ to the interval [0, 1]. The C measure is calculated by dividing the number of points in X'' that are either dominated by or equal to points in X', to the total number of points in X''. $C(X',X'')=1$ means that all points in X'' are covered by points in X'. $C(X',X'')=0$ indicates that no solution in X'' is covered by solutions in X'. Note that since the C function defines an ordered relationship, the value of $C(X',X'')$ is not necessarily equal to $C(X'',X')$. Both cases need to be considered in order to assess the relative performance of the sets. The C values for all three evolutionary multiobjective techniques that were used in the experimental phase are illustrated in table 6.4:

PROBLEM NO.	1	2	3	4	5	6	7	8	9	10
$C(Pareto,MOGA)$	1.0	1.0	0.85	0.8	0.15	0.36	0.53	0.31	0.65	0.59
$C(Pareto,NPGA)$	1.0	1.0	0.73	0.73	0.43	0.2	0.78	0.75	0.71	0.5
$C(MOGA,Pareto)$	1.0	1.0	0.63	0.73	0.77	0.64	0.58	0.63	0.07	0.35
$C(MOGA,NPGA)$	1.0	1.0	0.53	0.73	0.7	0.2	0.52	0.75	0.29	0.38
$C(NPGA,Pareto)$	1.0	1.0	0.63	0.82	0.72	0.64	0.25	0.19	0.07	0.18
$C(NPGA,MOGA)$	1.0	1.0	0.5	0.8	0.22	0.64	0.47	0.15	0.53	0.59

Table 6.4: *C* measure for the evolutionary multiobjective techniques in the experimental set-up

These results indicate that the performance of all multiobjective techniques in small-sized problems was similar, since a considerable number of equal solutions were produced. As the size of the problem increased, different sets of non-dominated solutions were evolved for each technique. However, a consistent pattern in terms of the coverage measure did not emerge. Each technique produced better results than the alternative ones in some problems, but none of them was able to dominate the others on the entire set of test problems.

NPGA was able to produce a considerable number of extreme Pareto-front values but its performance deteriorated more rapidly as the size of the problem increased. MOGA produced the most consistent pattern of solutions; however, at no point it totally dominated the performance of both alternative techniques. The 'Pareto' technique managed to produce the best results in the majority of large-sized problem, but its performance was inconsistent over the entire set of problems.

It can be concluded that no significant differences existed between the performance of the three evolutionary multiobjective techniques that were employed in the experimental phase. The overall performance of the proposed framework was dictated by the search power of the genetic programming algorithm rather than the choice of a particular multiobjective technique for the assignment of fitness to individual solutions. For that reason it is suggested that future research should concentrate on the enhancement of the genetic programming search engine, which will make possible the consideration of large instances of multiobjective process planning selection problems.

6.6 Conclusions

In this chapter a novel methodology for the solution of the multiobjective process planning selection problem was introduced. The backbone of the methodology was a genetic programming algorithm for the generation of potential process plans. Multiobjective optimisation was implemented through a number of non-aggregating evolutionary multiobjective techniques.

The proposed methodology was tested on a number of variable-sized randomly generated problems. All evolutionary multiobjective techniques were able to produce a variety of potential solutions for the decision-maker to consider. The evaluation of the relative performance of the proposed methodology was not possible due to the lack of comparative results and the unknown configuration of the actual Pareto-front.

This chapter concludes the research conducted in this thesis concerning the use of genetic programming for the solution of manufacturing optimisation problems. The conclusions of this research and suggestions for further work in the same area are presented in the following chapter.

Chapter 7

CONCLUSIONS

This chapter summarises the conclusions drawn from the application of genetic programming to a number of manufacturing optimisation problems. A detailed description of these applications has been presented in chapters 4, 5, and 6.

The aim of this research was to investigate the possibility of using genetic programming for the solution of a range of manufacturing optimisation problems and at the same time assess the quality of the proposed methodologies in relation to the state-of-the-art solution techniques on the problems considered. Three well-known manufacturing optimisation problems, the one-machine total tardiness problem, the cell-formation problem and the process planning selection problem were used for this purpose. The first two cases were representative examples of combinatorial sequencing and clustering problems respectively. The multiobjective process planning selection problem highlighted a case which traditional solution methodologies were unable to address sufficiently in a single optimisation run.

The summary of conclusions starts with the evaluation of the performance of the proposed methodologies on the individual manufacturing optimisation problems that were used in this research. Based on these observations, general comments are made on the advantages and drawbacks of using genetic programming for the solution of manufacturing optimisation problems.

These conclusions lead naturally to the consideration of future perspectives in this research. The last section of this chapter examines ways in which this research can be extended in relation to both the genetic programming methodologies presented in this thesis and alternative manufacturing optimisation cases.

7.1 The one-machine total tardiness problem

A considerable number of manufacturing optimisation problems are formulated as sequencing problems. Evolutionary algorithms are particularly suited for the solution of sequencing problems, since a potential solution can be easily coded into a fixed-size chromosome. However, since genetic programming evolves variable-length computer programs, it has never been regarded as a suitable algorithm for the solution of these types of problems.

The applicability of genetic programming for the solution of sequencing problems was investigated in this thesis with the help of a representative problem, the one-machine total tardiness problem. A novel genetic programming methodology was introduced, which employed the concept of dispatching rules for the indirect generation of job schedules. The proposed methodology succeeded in producing schedules with lower tardiness levels than individual dispatching rules on the set of problems used in the experimentation phase. In addition, combinations of the proposed genetic programming algorithm with local search and simulated annealing techniques were able to produce competitive results to one of the leading heuristic algorithms that has been proposed for the solution of this problem.

The main drawback of the proposed methodology was that it evolved a sequence of dispatching rules for a fix-sized problem within a variable-length genetic programming environment. This feature clearly limited the efficiency of the algorithm, since it did not utilise the positive characteristics of genetic programming. The results of this application indicated that if genetic programming was to be employed for the solution of sequencing problems, a solution representation natural to its program-induction framework was necessary.

This issue was addressed by the second genetic programming application introduced for the solution of the one-machine total tardiness problem. In this application genetic programming created a variety of formulas of potential dispatching rules using the scheduling information available for the problem considered. These rules were trained on a set of representative test problems. Nine new dispatching rules were evolved during the experimental phase. A number of these rules were not only able to generalise on the entire set of validation problems, but also outperformed the corresponding man-made dispatching rules.

The main drawback of this methodology was the complexity of the formulas of evolved dispatching rules, which made the explanation of their operation a difficult task. However, the fact that the solution representation was natural to the problem considered meant that it could provide significant insights to the solution of the problem. At the same time, computational efficiency was not an issue in this application, since the process of evolution only took place during the training phase. Once the potential dispatching rule had been evolved, it could be used independently of the evolutionary procedure as a fast computer program for the scheduling of jobs.

The above considerations indicate that the evolutionary construction of dispatching rules in the form of computer programs is a promising application of genetic programming in the field of manufacturing optimisation.

7.2 The cell-formation problem

The solution of the cell-formation problem constitutes one of the most significant phases in the implementation of a cellular manufacturing system. It is a combinatorial NP-hard clustering problem that has been extensively researched during the last thirty years. A considerable number of solution methodologies from a wide range of optimisation fields have been proposed for its solution, since the complexity of the problem creates a benchmarking potential for any new optimisation technique.

While several evolutionary computation algorithms had been employed for the solution of this problem, it had never been the subject of genetic programming research. A novel genetic programming methodology for the solution of simple binary cell-formation problems was introduced in this thesis. The algorithm processed information about the similarity of operations between machines using a coefficient evolved through a genetic programming machine. The front-end of the procedure was a hierarchical clustering algorithm that produced the final groupings of machines into cells.

The methodology was tested on a wide range of test problems taken from the literature and was found to produce results that were at least as good as the ones that have already been published for the objectives and the problems considered. The proposed methodology combined the efficiency of the hierarchical clustering procedure with the flexibility of the evolutionary algorithm. While a stand-alone clustering procedure created a set of promising cell configurations, its outcome was always independent of the optimisation objective. The replacement of the standard similarity coefficient with a genetically evolved population of coefficients provided the necessary flexibility, since each coefficient was either promoted or demoted by the evolutionary procedure depending on the quality of the groupings that it produced.

The successful application of the proposed methodology on the solution of the binary cell-formation problem was mostly significant from the genetic programming perspective since the binary cell-formation problem represents a fairly simple and unrealistic modelling of the real problem. Fortunately, one of the main features of the proposed framework was its ability to address more complex formulations of the cell-formation problem by modifying either the type of similarity information evolved through the genetic programming machine or the objective function of the algorithm. This process was illustrated with the help of some example problems taken from the literature.

While the proposed framework was tested on a limited number of advanced formulations of the cell-formation problem, it can be said that its ability to consider a variety of cell-formation models constituted a significant advantage in relation to alternative non-evolutionary solution techniques.

Genetic programming was additionally employed for the evolution of general-purpose similarity coefficients that could be used in combination with clustering procedures for the solution of cell-formation problems. Evolved coefficients were found to generalise over the entire set of validation problems. However, their performance was similar to the one produced by man-made similarity coefficients. There were indications that some of the genetically evolved coefficients were able to handle ill-structured machine/component matrices more efficiently, however, no statistical verification of this hypothesis was obtained.

7.3 The multiobjective process planning selection problem

Multiobjective optimisation is one of the main application areas for evolutionary computation algorithms due to their natural ability to search for all Pareto-optimal points in a single optimisation run. Manufacturing optimisation is a field where multiple objectives, often conflicting in nature, need to be simultaneously optimised. In this thesis, a novel genetic programming methodology for the solution of the multiobjective process planning selection problem was presented. Genetic programming was applied on a set of randomly generated test problems in combination with a number of evolutionary multiobjective techniques.

The performance of the proposed framework was not sufficiently evaluated since test problems and comparative results from alternative evolutionary algorithms were not available. A visual analysis of evolved Pareto-fronts on the problems considered indicated that in small-to-medium sized problems the framework was able to produce some Pareto-optimal process plans in the extreme regions of the front (minimum cost - maximum quality). However, the performance of the algorithm degraded as the size of the problem increased. In all cases the proposed framework was able to provide the decision-maker with a wide range of potential process plans, instead of a single compromise solution that would have been produced by a traditional aggregating multiobjective technique. None of the evolutionary multiobjective techniques that were employed in the experimentation was able to dominate the performance of the alternative ones.

It can be concluded that the integration of genetic programming with the evolutionary multiobjective techniques for the solution of the multiobjective process planning selection problem was able to produce a wealth of potential solutions near the surface of the actual Pareto-front. Unfortunately the level of proximity could not be estimated due to the unknown position of this front. At the same time, since no comparative results were available, the performance of the algorithm in relation to alternative evolutionary algorithms could not be assessed. For these reasons further research is needed in order to establish the full significance of the proposed methodology.

7.4 Comments on the use of genetic programming for the solution of manufacturing optimisation problems

The main conclusion of this research is that genetic programming can be used for the solution of various types of manufacturing optimisation problems, producing competitive results to alternative optimisation techniques that have been proposed for the solution of these problems.

The most promising use of genetic programming in manufacturing optimisation is the off-line generation of computer programs that can subsequently be used independently of the evolutionary procedure as stand-alone optimisers, like the evolution of dispatching rules for the one-machine total tardiness problem and the evolution of similarity coefficients for the cell-formation problem. The advantages of this approach are the following:

- The computational complexity of the algorithm is no longer a critical issue. The training phase of the algorithm requires a number of genetic programming runs on some pre-specified fitness cases, however, once the best computer program has been evolved, it can be used independently of the evolutionary procedure on an unlimited number of problems.
- Evolved computer programs are constructed from building blocks that are natural elements of the problem considered. For that reason it is possible to explain the operation of a program, depending on its size and complexity. This feature can provide insights on optimisation problems where little or no theoretical background exists, and lead to the development of robust solution methodologies. The transparency of evolved computer programs can be enhanced by using techniques such as the Constrained Complexity Crossover (CCC) of Watson and Parmee (1998) and Automatically Defined Functions (ADFs) of Koza (1994).

The methodologies presented in the previous chapters illustrate the point that successful genetic programming applications in the field of manufacturing optimisation are possible. However, the practical implementation of the proposed methodologies in real manufacturing cases requires the consideration of additional parameters:

- The computational complexity of the algorithms, the time-consuming design of the genetic programming framework, and the absence of a general-purpose genetic programming toolkit, deter engineers from employing genetic programming for the solution of manufacturing optimisation problems, especially when cheap and fast alternative techniques exist.
- The model-based formulations of manufacturing optimisation problems that were considered during the experimental phase do not always give accurate descriptions of the cases faced in manufacturing practice.

The reasons for the limited number of reported genetic programming applications in the field of manufacturing optimisation are the following:

- The genetic programming solution representation for the majority of manufacturing optimisation problems is not as straightforward as in the case of alternative evolutionary algorithms.
- Genetic programming applications require significant computational resources. The performance of the algorithms does not always justify the additional overhead induced in comparison to alternative optimisation techniques.
- Genetic programming is a relatively new technique that is still in development. The bulk of research in the field is focused on problems where a solution representation can be easily extracted from their formulation. Researchers are reluctant to experiment with problems that are not considered to be genetic programming-friendly.

All previous considerations should be viewed in the light of the following points:

- Each manufacturing optimisation problem has its own complexity and characteristics. While the cases considered in this thesis are significant, it cannot be said that genetic programming is necessarily suitable for the solution of any manufacturing optimisation problem.
- The results presented in this thesis are based on the genetic programming coding, configuration and solution representations designed by the author of this thesis. Alternative genetic programming machines that employ different solution representations will not necessarily produce the same results on the set of manufacturing optimisation problems considered in this thesis.

7.5 Future work

The research presented in this thesis provides an introductory investigation into the potential use of genetic programming for the solution of manufacturing optimisation problems. However, additional research is needed for the drawing of safer conclusions. Some promising areas for the continuation of this research are the following:

- The evolution of dispatching rules for the solution of the one-machine total tardiness problem can be extended to any other static scheduling problem. Furthermore, the proposed methodology provides the opportunity for manufacturing companies with special scheduling considerations to evolve their own case-based dispatching rules by providing the relevant inputs to the evolutionary procedure.
- A wider experimental basis is needed in order to assess the efficiency of genetic programming in solving advanced formulations of the cell-formation problem.
- The performance of the genetic programming methodology for the solution of the multiobjective process planning selection problem could not be sufficiently evaluated due to the lack of comparative results from alternative solution methodologies. It is hoped that these data will be available in the future for further comparisons to be made.
- The initialisation process of the genetic programming methodology for the generation of potential process plans favours plan selections situated near to the initialisation point (section 6.5.2.1). This bias can be reduced by introducing navigational commands that alternate between potential process plans in step-sizes larger than one. These commands were not included in the experiments of this thesis since their introduction requires a significant increase in the population size of the algorithm.
- There are several important manufacturing areas where the potential application of genetic programming has not been investigated. Dynamic scheduling, assembly lines, quality control and production planning are some optimisation areas for which genetic programming might be able to provide successful applications.

- The genetic programming algorithm employed in the experiments of this thesis followed the guidelines suggested by Koza (1992). However, alternative frameworks for the evolution of computer programs have also been introduced (section 3.3.2). The application of these frameworks on the set of manufacturing optimisation problems considered in this thesis could yield improvements in terms of the computational efficiency of the proposed methodologies.
- The use of length reduction techniques such as the Constrained Complexity Crossover (CCC) (Watson and Parmee, 1998), (Parmee and Watson, 2000) and modularization techniques such as Automatically Defined Functions (ADFs) (Koza, 1994) would help reduce the length of evolved genetic programs and simplify the task of explaining their operation. A reduction on the length of genetic programs can also be achieved through the introduction of parsimony penalties in the objective function of the algorithm.
- The availability of computational power was always a main consideration during the experimental phase of this research. The employment of parallel processing machines would allow the use of larger populations of solutions, thus increasing the search potential of the proposed methodologies.
- A number of applications presented in this thesis suggest that the hybridisation of genetic programming with alternative optimisation methods is a promising area for further research, since it combines the positive characteristics of the co-operating algorithms.

APPENDIX

SIMULATED ANNEALING PARAMETERS	
Neighbourhood structure	All general pairwise interchanges – restart from the best sequence found in the entire neighbourhood
Neighbourhood size	$\frac{1}{2}\cdot n\cdot(n-1)$
Probability of acceptance of a solution that performs worse than the best solution that has been found so far	$P_a = e^{-(a\cdot\Delta_{Tard})}$ where : $a = 10 + (ITER\cdot 4)$ $ITER =$ Number of iterations $\Delta_{Tard} =$ increase in tardiness
Termination criterion	When no better sequence has been found in an entire neighbourhood

Table 1A. Simulated Annealing implementation for the one machine total tardiness problem

NAME	ABBREVIATION	FORMULA
Mean Absolute Deviation from Optimal	MADO	$\frac{\left[\sum_{1}^{80} TD_{dis} - \sum_{1}^{80} TD_{opt}\right]}{80}$
Mean Relative Deviation from Optimal	MRDO	$\frac{\sum_{1}^{80}\left(\frac{TD_{dis} - TD_{opt}}{TD_{opt}}\right)}{80} * 100$
Maximum Relative Deviation from Optimal	MAX(RDO)	$\max_i \left\{\frac{TDi_{dis} - TDi_{opt}}{TDi_{opt}}\right\}$
Tardiness of dispatching rule or GPC	TD_{dis}	
Optimal Tardiness	TD_{opt}	

Table 2A: List of statistical terms used in Chapter 4

SETUP12	$\left[\frac{\left(\frac{p_i}{d_i} - SP - SD - (N \bullet SD)\right)}{p_i} + 2 - (2 \bullet N) - SD + d_i - p_i - \left(\frac{SD}{N} \bullet (p_i - SP)\right)\right] \bullet$ $\bullet \left[((SD - SP) \bullet (p_i + d_i)) + \left(\frac{d_i \bullet SP}{SD \bullet N}\right) + (SP \bullet d_i \bullet N \bullet (N - p_i + SP))\right]$
SETUP25	$((p_i + SD) - d_i - (N \bullet d_i) + (3 \bullet SD) - N) \bullet ((d_i \bullet N) - (N \bullet SD)) -$ $- (N \bullet p_i) + ((p_i + N) \bullet N \bullet (SP + SD) \bullet (SP - (2 \bullet p_i \bullet N) - p_i))$
SETUP50	$\left((d_i + p_i - (2 \bullet SP)) + \left(SD \bullet \frac{(p_i - SP)}{(SD + p_i)}\right)\right) \bullet (1 + (2 \bullet d_i) + (N \bullet SD)) \bullet$ $\bullet \left((3 \bullet SD) - SP - \left(N \bullet p_i \bullet (p_i^2 + p_i)\right) - (2 \bullet p_i \bullet SP) - ((N + p_i) \bullet SP)\right)$
SETUP100	$((N^2 + p_i^2) \bullet ((p_i \bullet SP) + SD) \bullet SP) - d_i + ((SP + d_i) \bullet d_i \bullet SD)$
SETVAR1	$(((SP + SD) \bullet N \bullet SD) + SD + SP) \bullet \left(\left(N + \left(\frac{d_i \bullet p_i}{(SP - N)}\right)\right) \bullet d_i\right) + \left(2 \bullet \left(\left(\frac{SP}{p_i}\right) + SP\right) \bullet N \bullet p_i^6\right)$
SETVAR2	$(N \bullet p_i \bullet ((N^3 \bullet SP) + N - p_i)) + (d_i \bullet (SD^2 - N)) - (SP \bullet N) + (SP \bullet SD) + p_i^2 - SP +$ $+ (2 \bullet p_i \bullet SD^2) - (4 \bullet SD^2) - \left(\frac{d_i}{(SP + N)}\right)$
SETVAR3	$\left(\frac{(SD \bullet d_i \bullet (SP - N)) \bullet \left(\frac{N}{d_i} + \frac{SP}{N}\right)}{N \bullet SP}\right) - \left(\left((4 \bullet SP) - (2 \bullet d_i) - \left(\frac{p_i}{d_i^2}\right) - N\right) \bullet d_i\right) +$ $((p_i \bullet SP) + (N^2 - d_i)) \bullet (p_i + (2 \bullet N))$

Table 3A: Evolved dispatching rules for the one machine total tardiness problem

SETVAR4	$\left(\dfrac{\left(\dfrac{d_i^2}{SP}\right) - \left(\dfrac{N\left(SD - SP\right) \bullet p_i \bullet d_i}{SD}\right)}{SP}\right) + \left(2 \bullet d_i\right) + \left(p_i \bullet N\right)$
SETVAR5	$\left(SP^2 \bullet p_i\right) + \left(N \bullet SP\right) - SD + \left(\left(\left(SP - \dfrac{SD}{N}\right) \bullet p_i\right) + SD\right) \bullet \left(\dfrac{\left(d_i + p_i\right)}{N \bullet d_i}\right) \bullet$ $\bullet \left(\left(\left(N + p_i + d_i\right) \bullet d_i\right) + \left(p_i \bullet SP\right)\right)$

Table 3A (cont.) : Evolved dispatching rules for the one machine total tardiness problem

Name	No. of fitness cases	Problems
SET1	8	1 – 8
SET2	6	16 - 21
SET3	6	22 – 27
SET4	6	9 – 12, 14, 15
SET5	6	1 – 4, 16 – 18
SET6	7	5 – 8, 19 – 21
SET7	8	11, 12, 14, 15, 22 – 25
SET8	8	9 – 12, 24 – 27
SET9	14	1 – 12, 14, 15
SET10	14	14 – 27

Table 4A: Experimental sets for the evolution of similarity coefficients

Coefficient	**Formula**
1. Jaccard	$\frac{a}{a+b+c}$
2. Yule	$\frac{ad-bc}{ad+bc}$
3. Sorenson	$\frac{2a}{2a+b+c}$
4. Hamann	$\frac{(a+d)-(b+c)}{(a+d)+(b+c)}$
5. Rogers & Tanimoto	$\frac{a+d}{a+2(b+c)+d}$
6. Sokal & Sneath	$\frac{2(a+d)}{2(a+d)+b+c}$
7. Russel & Rao	$\frac{a}{a+b+c+d}$
8. Baroni-Urbani and Buser	$\frac{a+(ad)^{1/2}}{a+b+c+(ad)^{1/2}}$
9. Simple matching	$\frac{a+d}{a+b+c+d}$
10. Ochiai	$\frac{a}{[(a+b)(a+c)]^{1/2}}$
11. Phi	$\frac{ad-bc}{[(a+b)(a+c)(b+d)(c+d)]^{1/2}}$

Table 5A: Similarity coefficients produced by human intuition

REFERENCES

Aizpuru, J.R.Z. and Usunariz, J.A. (1995), "GA/TS: A hybrid approach for job-shop scheduling in a production system" in *Proc.of the 7th Potuguese Conf.on Artificial Intelligence - EPIA'95*, pp.153-164, Springer-Verlag, Berlin, Germany.

Aljaber, N., Baek, W., and Chen, C.-L. (1997), "A tabu search approach to the cell-formation problem", *Computers & Industrial Engineering*, vol.32, no.1, pp.169-185.

Angeline, P. (1997), "Subtree crossover: building block engine or macromutation?", in *Genetic Programming 1997: Proceedings of the Second Annual Conference,* Koza, J.R. *et al.* (eds), pp.9-17, Stanford University, CA, USA.

Askin, R.G., Cresswell, S.H., Goldberg, J.B., and Vakharia, A.J. (1991), "A hamiltonian path approach to reordering the part-machine matrix for cellular manufacturing", *Int.J. of Production Research*, vol.29, no.6, pp.1081-1100.

Awadh, B., Sepehri, N. and Hawaleshka, O. (1995), "A computer-aided process planning model based on genetic algorithms", *Computers & Operations Research*, vol.22, no.8, pp.841-856.

Aytug, H., Koehler, G.H. and Snowdon, J.L. (1994), "Genetic learning of dynamic scheduling within a simulation environment", *Computers & Operations Research*, vol.21, no.8, pp.909-925.

Baker K.R. and Schrage L.E. (1978), "Finding an optimal sequence by dynamic programming: an extension to precedence-related tasks", *Operations Research*, vol.26, no.1, pp.111-120.

Baker, J.E. (1987), "Reducing bias and inefficiency in the selection algorithm", in *Proc. of the 2nd Int. Conf. on Genetic Algorithms and their Applications*, J.J.Grefenstette (Ed.), pp.14-21, Lawrence Erlbaum, Hillsdale, NJ.

Baker, K. (1974), *Introduction to Sequencing and Scheduling*, Whiley, New York.

Bazargan-Lari, M. and Kaebernick, H. (1997) "An approach to the machine layout problem in a cellular manufacturing environment", *Production Planning & Control*, vol.8, no.1, pp.41-55.

Banerjee, P., Zhou, Y. and Montreuil, B. (1997), "Genetically assisted optimisation of cell layout and material flow path skeleton", *IIE Transactions*, vol.29, no.4, pp.277-291.

Banzhaf, W., Nordin, P., Keller, R.E. and Francone F.D. (1998), *Genetic Programming: An Introduction*, Morgan Kaufman, San Francisco, CA.

Bean, J.C. (1994), "Genetic algorithms and random-keys for sequencing and optimisation", *ORSA Journal on Computing*, vol.6, no.2, pp.154-160.

Bellman, R.E. and Dreyfus S.E. (1962), *Applied Dynamic Programming*, Princeton University Press.

Ben-Daya M. and Al-Fawzan M. (1996), "A simulated annealing approach for the one-machine mean tardiness scheduling problem", *European Journal of Operational Research*, vol.93, pp.61-67.

Beveridge, G.S. G. (1970), *Optimization : theory and practice*, New York, McGraw-Hill.

Bierwirth, C., Kopfer, H., Mattfeld, D.C. and Rixen, I. (1995), "Genetic algorithm based scheduling in a dynamic manufacturing environment", *Proc.of the 1995 IEEE Conf.on Evolutionary Computation*, Vol.1, pp.439-443, IEEE, Piscataway, NJ, USA.

Bierwirth, C., Mattfeld, D.C. and Kopfer, H. (1996), "On permutation representations for scheduling problems", in *Proc.of the 4th Int.Conf.on PPS from Nature*, Voigt, Ebeling, Rechenberg, Schwefel (Eds.), pp.310-318, Springer-Verlag, Berlin, Germany.

Billo, R.E., Bidanda, B. and Tate, D. (1996), "A genetic cluster algorithm for the machine-component grouping problem", *Journal of Intelligent Manufacturing*, vol.7, no.3, pp.229-243.

Blackstone, J.H., Philips, D.T. and Hogg, C.L. (1982), "A state of the art survey of dispatching rules for manufacturing job shop operations", *Int.J.of Production Research*, vol.20, pp.27-45.

Boctor, F.F. (1991), "A linear formulation of the machine-part cell-formation problem", *Int.J. of Production Research*, vol.29, no.2, pp.343-356.

Boe, W.J., and Cheng, C.H. (1991), "A close neighbour algorithm for designing cellular manufacturing systems", *Int.J. of Production Research*, vol.29, no.10, pp.2097-2116.

Booker, L.B., Goldberg, D.E. and Holland, J.H. (1989) "Classifier systems and genetic algorithms", in *Machine Learning: Paradigms and methods*, pp.235-282, J.G. Carbonnel (Ed.), MIT Press/Elsevier, MA.

Bowden, R. and Bullington, S.F. (1996), "Development of manufacturing control strategies using unsupervised machine learning", *IIE Transactions*, vol.28, no.4, pp.319-331.

Bowden, R.O., Hall, J.D. and Usher, J.M. (1996), "Integration of evolutionary programming and simulation to optimise a pull production system", *Computers & Industrial Engineering*, vol.31, no.1/2, pp.217-220.

Box, G.E.P. (1957), "Evolutionary operation: A method for increasing industrial productivity", *Journal of the Royal Statistics Society, C*, vol.6, no.2, pp.81-101.

Braglia, M. and Gentili, E. (1994), "An improved genetic algorithm for flowshop scheduling problems" in *Proc.of the 10th ISPE/IFAC Int.Conf. on CAD/CAM, Robotics and Factories of the Future*, pp.137-142, OCRI, Ontario, Canada.

Braglia, M. and Sternieri, A. (1996), "A genetic algorithm for layout optimisation in a flowline cellular manufacturing system", in *Proc.of the Int.ICSC Symposia on Intelligent Industrial Automation and Soft Computing*, Anderson & Warwick (Eds.), pp.A21-A26, Int.Comp.Science Conventions, Millet, Canada.

Burbidge, J.L. (1971), "Production Flow Analysis", *Production Engineer*, vol.50, pp. 139-152.

Burbidge, J.L. (1975), *The Introduction of Group Technology*, Halste Press, John Wiley, New York, U.S.A.

Burns, R. (1993), "Direct chromosome representation and advanced genetic operators for production scheduling", in *Proc. of the 5th Int.Conf. on Genetic Algorithms*, Stephanie Forest (Ed.), pp. 352-359, Morgan Kaufman, San Mateo.

Cao, H., Xi, H., Luo, Y. and Yang, S. (1997), "GA with hierarchical evaluation: a framework for solving complex machine scheduling problems in manufacturing", in *GALESIA '97: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.326-331, IEE Conf.Publ. no.446, IEE, Stevenage, England.

Cao, Y.J. and Wu, Q.H. (1997), "Mechanical design optimisation by mixed-variable evolutionary programming", in *Proc.of the 1997 IEEE Int.Conf.on Evolutionary Computation*, pp.443-446, IEEE, Piscataway, NJ, USA.

Carrie, A.S. (1973), "Numerical taxonomy applied to group technology and plant layout", *Int.J. of Production Research*, vol.11, no.4, pp.399-416.

Cartwright, H.M. and Tuson, A.L. (1994), "Genetic algorithms and flowshop scheduling: towards the development of a real-time process control system" in *Evolutionary Computing. AISB Workshop. Selected Papers*, pp.277-290, Lecture Notes in Computer Science (865), T.C.Fogarty (Ed.), Springer-Verlag, Berlin, Germany.

Chan, H.M., and Milner, D.A. (1982), "Direct clustering algorithm for group formation in cellular manufacture", *Journal of Manufacturing Systems*, vol.1, no.1, pp.65-75.

Chandrasekharan, M.P., and Rajagopalan, R. (1986), "MODROC: An extension of rank order clustering for group technology", *Int.J. of Production Research*, vol.24, no.5, pp.1221-1233.

Chandrasekharan, M.P., and Rajagopalan, R. (1986b), "An ideal-seed non-hierarchical clustering algorithm for celllar manufacturing", *Int.J. of Production Research*, vol.24, no.2, pp.451-464.

Chandrasekharan, M.P., and Rajagopalan, R. (1987), "ZODIAC - an algorithm for concurrent formation of part families and machine-cells", *Int.J. of Production Research*, vol.25, no.6, pp.835-850.

Chandrasekharan, M.P., and Rajagopalan, R. (1989), "GROUPABILITY: an analysis of the properties of binary data matrices for group technology", *Int.J. of Production Research*, vol.27, no.6, pp.1035-1052.

Chankong, V. and Haimes, Y.Y. (1983), *Multiobjective decision making and methodology*, North Holland, New York.

Chen, C.-L., Neppali R.V. and Aljaber, N. (1996a), "Genetic algorithms applied to the continuous flow-shop problem", *Computers & Industrial Engineering*, vol.30, no.4, pp.919-929.

Chen, P., Toyota, T. and Nasu, M. (1996b), "Self-organisation method of symptom parameters for for failure diagnosis by genetic algorithms, *Proc. of the 1996 IEEE Industrial Electronics Conference - IECON*, pp.829-835, IEEE, Piscataway, NJ, USA.

Cheng, C.H., Gupta, Y.P., Lee, W.H., and Wong, K.F. (1998), "A TSP-based heuristic for forming machine groups and part families", *Int.J. of Production Research*, vol.36, no.5, pp.1325-1337.

Cheng, C.-H., Madan, M.S., and Motwani, J. (1996a), "Designing cellular manufacturing systems by a truncated tree search", *Int.J. of Production Research*, vol.34, no.2, pp.349-361.

Cheng, R., Gen, M. and Tosawa, T. (1996b), "Genetic algorithms for designing loop layout manufacturing systems", *Computers & Industrial Engineering*, vol.31, no.3/4, pp.587-591.

Cheng, R., Gen, M. and Tozawa, T. (1995), "Genetic search for facility layout design under intreflows uncertainty", *Proc.of the 1995 IEEE Conf.on Evolutionary Computation*, Vol.1, pp.400-405, IEEE, Piscataway, NJ, USA.

Cheng, R., Gen, M. and Tsujimura, Y. (1996c), "A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation", *Computers & Industrial Engineering*, vol.30, no.4, pp.983-997.

Cheng, R., Gen, M. and Tsujimura, Y. (1999), "A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies", *Computers & Industrial Engineering*, vol.36, no.2, pp.343-364.

Chiu, C. and Yih, Y. (1995), "A learning-based methodology for dynamic scheduling in distributed manufacturing systems", *Int.J.of Production Research*, vol.33, no.11, pp.3217-3232.

Cho, B.J., Hong, S.C. and Okoma, S. (1996), "Job-shop scheduling using genetic algorithms", *in Critical Technology: Proc. of the 3rd World Congress on Expert Systems*", pp.351-358, Cognizant Com.Corp., New York.

Choobineh, F.C. (1988), "A framework for the design of cellular manufacturing systems", *Int.J. of Production Research*, vol.26, no.7, pp.1161-1172.

Chu, C.-H., and Hayya, J.C. (1991), "A fuzzy-clustering approach to manufacturing cell-formation", *Int.J.of Production Research*, vol.29, no.7, pp.1475-1487.

Cohon, J.L. (1978), *Multi-Objective Programming and Planning*, Academin Press, New York.

Cohoon, J.P., Hedge, S.U., Martin, W.N. and Richards, D.S. (1991), "Distributed genetic algorithms for the floorplan design problem", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.10, no.4, pp.483-492.

Conway, D.G. and Venkataramanan, M.A. (1994), "Genetic search and the dynamic facility layout problem", *Computers & Operations Research*, vol.21, no.8, pp.955-960.

Cramer, N.L. (1985), "A representation for the adaptive generation of simple sequential programs", in *Proc. of the 1st Int. Conf. on Genetic Algorithms and their Applications*, pp.183-187, J.J.Grefenstette (Ed.), Lawrence Erlbaum, Hillsdale, NJ.

Croce, F.D., Tadei, R. and Volta, G. (1995), "A Genetic Algorithm for the job-shop problem", *Computers & Operations Research*, vol.22, no.1, pp.15-24.

Da Silveira, G. (1999), "A methodology of implementation of cellular manufacturing", *Int.J. of Production Research*, vol.37, no.2, pp.467-479.

Dagli, C.H. and Sittisathancai, S. (1995), "Genetic neuro-scheduler: a new approach for job shop scheduling, *Int.J. of Production Economics*, vol.41, no.1-3, pp.135-145.

Daida, J., Ross, S., McClain, J., Ampy, D. and Holczer, M. (1997), "Challenges with verification, repeatability, and meaningful comparisons in genetic programming", in *Genetic Programming 1997: Proceedings of the Second Annual Conference,* Koza, J.R. *et al.* (eds), pp. 64-69, Stanford University, CA, USA.

Davis, L. (1985), "Job shop scheduling with genetic algorithms", in *Proc. of the 1st Int. Conf. on Genetic Algorithms and their Applications*, pp.136-140, J.J.Grefenstette (Ed.), Lawrence Erlbaum, Hillsdale, NJ.

De Jong, K.A. (1975), "An analysis of the behaviour of a class of genetic adaptive systems" (Doctoral dissertation, University of Michigan), *Dissertation Abstracts International*, vol.36, no.10, 5140B.

De Witte, J. (1980), "The use of similarity coefficient in production flow analysis", *Int.J. of Production Research*, vol.18, no.4, pp.503-514.

Deb, K. and Goldberg, D.E. (1989), "An investigation of niches and species formation in genetic function optimisation", in *Proc. of the 2nd Int. Conf. on Genetic Algorithms and their Applications*, pp.42-50, J.D. Schaffer (Ed.), Morgan Kaufman, San Mateo.

Dereli, T., and Filiz, I.H. (1999), "Optimisation of process planning functions by genetic algorithms", *Computers & Industrial Engineering*, vol.36, no.2, pp.281-308.

Dimopoulos, C., and Zalzala, A.M.S. (2000), "Recent developments in evolutionary computation for manufacturing optimisation: problems, solutions and comparisons", *IEEE Transactions in Evolutionary Computation*, vol.4, no.2, pp.93-113.

Disney, S.M., Naim, M.M. and Towill, D.R. (1997), "Development of a fitness measure for an inventory and production control system", in *GALESIA '97: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.351-355, IEE Conf.Publ. no.446, IEE, Stevenage, England.

Dorndorf, U. and Pesch, E. (1995), "Evolution-based learning in a job-shop scheduling environment", *Computers & Operations Research*, vol.22, no.1, pp.177-181.

Du J. and Leung J.Y.-T. (1989), "Minimising total tardiness on one machine is NP-hard", *Mathematics of Operations Research*, vol.15, no.3, pp.483-495.

Dubois, D., and Prade, H. (1980), *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York.

Emmons H. (1968), "One machine sequencing to minimise certain function of job tardiness", *Operations Research*, vol.17, no.4, pp.701-715.

Fang, H.L., Corne, D. and Ross, P. (1996), "A Genetic Algorithm for job-shop problems with various schedule quality criteria", in *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science (1143), T.C.Fogarty (Ed.), pp.39-49, Springer-Verlag, Berlin, Germany.

Fang, J. and Xi, Y. (1997), "A rolling horizon job shop rescheduling strategy in the dynamic environment", *Int.J.of Advanced Manufacturing Technology*, vol.13, no.3, pp.227-232.

Feng, W., Burns, G.B. and Harrison, D.K. (1997), "Using genetic algorithms bounded by dynamic linear constraints for marketing/production joint decision making", in *GALESIA '97: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.339-344, IEE Conf.Publ. no.446, IEE, Stevenage, England.

Fischer, H. and Thompson, G.L. (1963), "Probabilistic learning combinations of local job-shop scheduling rules" in *Industrial Scheduling*, pp.225-251, J.F.Muth and G.L.Thompson (Eds), Prentice Hall, Englewood Cliffs, NJ.

Fisher M.L. (1976), "A dual algorithm for the one-machine scheduling problem", *Mathematical Programming*, vol.11, no.3, pp.229-251.

Fogel, L.J., Owens, A.J. and Walsch, M.J. (1966), *Artificial Intelligence through Simulated Evolution*. Wiley, New York.

Fonseca, C.M. and Fleming, P.J. (1993), "Genetic algorithms for multiobjective optimisation: Formulation, Discussion and Generalization", in *Proc. of the 5th Int. Conf. on Genetic Algorithms and their Applications*, pp.416-423, S. Forrest (Ed.), Morgan Kaufman, San Mateo.

Fonseca, C.M. and Fleming, P.J. (1995), "An overview of evolutionary algorithms in multiobjective optimisation", *Evolutionary Computation*, vol.3, no.1, pp.1-16.

Fonseca, C.M. and Fleming, P.J. (1998), "Multiobjective optimization and multiple constraint handling with evolutionary algorithms--Part I: A unified formulation" *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol.28, no.1, pp.26-37.

Friedberg, R. (1958), "A learning machine, part I", *IBM Journal on Research & Development*, vol.2, pp.2-13.

Fry T.D., Vicens L., Macleod K. and Fernadez S. (1989), "A heuristic solution procedure to minimize total tardiness", *Journal of the Operational Research Society*, vol.40, pp.293-297.

Fuchs, M. (1998), "Crossover versus mutation: An empirical and theoretical case study", in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (J.R. Koza *et al.*, eds.), pp.78-85, Morgan Kaufmann.

Fujiki, C. and Dickinson, J. (1987), "Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma", in *Proc. of the 2nd Int. Conf. on Genetic Algorithms and their Applications*, J.J.Grefenstette (Ed.), pp.236-240, Lawrence Erlbaum, Hillsdale, NJ.

Fujimoto, H., Lian-yi, C., Tanigawa, Y. and Iwahashi, K. (1995), "Application of genetic algorithm and simulation to dispatching rule-based FMS", in *Proc.of the 1995 IEEE Int.Conf.on Robotics & Automation*, pp.190-195, IEEE, Piscataway, NJ, USA.

Garavelli, A.G., Okogbaa, O.G. and Violante, N. (1996), "Global manufacturing systems: a model supported by genetic algorithms to optimise production planning", *Computers & Industrial Engineering*, vol.31, no.1/2, pp.193-196.

Garces-Perez, J., Schoenefeld, D.A. and Wainwright, R.L. (1996), "Solving facility layout problems using genetic programming", in *Genetic Programming 1996: Proc.of the 1st Annual Conference*, Koza, Goldberg, Fogel and Riolo (Eds.), pp.182-190, MIT Press.

Garey, M. and Johnson, D. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completness.* W.H.Freeman, San Francisco.

Garey, M., Johnson, D.S. and Sethi, R. (1976), "The complexity of flowshop and jobshop scheduling", *Mathematics of Operations Research*, vol.1, pp.117-129.

Gau, K.-Y. and Meller, R.D. (1993), "An iterative facility layout algorithm", *Int.J. of Production Research*, vol.37, no.16, pp.3739-3758.

Gen, M., Tsujimura, Y. and Kubota, E. (1994), "Solving job shop scheduling problems by genetic algorithms" in *Proc.of the 1994 IEEE Int.Conf. on Systems, Man & Cybernetics*, pp.1577-1582, IEEE, Piscataway, NJ, USA.

Gen, M., Ida, K. and Cheng, C. (1995), "Multirow machine layout problems in fuzzy environment using genetic algorithms", *Computers & Industrial Engineering*, vol.29, no.1-4, pp.519-523.

Giffler, B. and Thompson, G.L. (1969), "Algorithms for solving production scheduling problems", *Operations Research*, vol.8, pp.487-503.

Gilbert, R.J., Goodacre, R., Shann, B., Taylor, J., Rowland, J.J. and Kell, D.B. (1998), "Genetic Programming-Based Variable Selection for High-Dimensional Data", in *Genetic Programming 1998: Proceedings of the Third Annual Conference* (J.R. Koza *et al.*, eds.), pp.109-115, Morgan Kaufmann.

Glover, F. (1990), "Tabu Search: a tutorial", *Interfaces*, vol.20, no.3, pp.79-94.

Gohtoh, T., Ohkura, K. and Ueda, K. (1996), "An application of genetic algorithm with neutral mutations to job-shop scheduling problems", *Proc.of the Int.Conf. on Advance in Production Systems, APMS'96*, Okino, Tamura, Fujii (Eds.), pp.563-568, Kyoto Univ., Kyoto, Japan.

Goldberg, D., E. (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, Reading, MA.

Gong, D., Yamazaki, G. and Gen, M. (1996), "Evolutionary program for optimal design of material distribution system", in *Proc.of the 1996 IEEE Int.Conf.on Evolutionary Computation*, pp.139-143, IEEE, Piscataway, NJ, USA.

Gonzalez, B., Torres, M. and Moreno, J.A. (1995), "A hybrid genetic algorithm approach for the 'no-wait' flowshop scheduling problem" in *GALESIA '95: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.59-64, IEE Conf.Publ. no.414, IEE, London, England.

Gravel, M., Nsakanda, A.L., and Price, W. (1998), "Efficient solutions to the cell formation problem with multiple routings via a double-loop genetic algorithm", *European Journal of Operational Research*, vol.109, pp.286-298.

Groppeti, R. and Muscia, R. (1995), "Genetic algorithms for optimal assembly planning", in *Proc.of the 1st World Congress on Intelligent Manufacturing Processes and Systems*, pp.319-333, Univ.of Puerto Rico, Sa Juan, Puerto Rico.

Gupta, T. (1993), "Design of manufacturing cells for flexible environment considering alternative routeing", *Int.J. of Production Research*, vol.31, no.6, pp.1259-1273.

Gupta, T., and Seifoddini, H. (1990), "Production data based similarity coefficient for machine-component grouping decisions in the design of a cellular manufacturing system", *Int.J. of Production Research*, vol.28, no.7, pp.1247-1269.

Gupta, Y., Gupta, M., Kumar, A. and Sundaram, C. (1996), "A genetic algorithm-based approach to cell-composition and layout design problems", *Int.J.of Production Research*, vol.34, no.2 pp.447-482.

Guzman, C.-R. and Kramer, M.A., (1994), "Remote diagnosis and monitoring of complex industrial systems using a genetic algorithm approach, in *Proc.of the IEEE Int.Symposium on Industrial Electronics*, pp.363-376, IEEE, Piscataway, NJ, USA.

Hamada, K., Baba, T., Sato, K. and Yufu, M. (1995), "Hybridising a genetic algorithm with rule-based reasoning for production planning", *IEEE Expert*, vol.10, no.5, pp.60-67.

Harhalakis, G.R., Nagi, R., and Proth, J.-M. (1990), "An efficient heuristic in manufacturing cell formation for group technology applications", *Int.J. of Production Research*, vol.28, no.1, pp.185-198.

Held M. and Karp R.M. (1962), "A dynamic programming approach to sequencing problems", *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, vol.10, no.1, pp.196-210.

Herrmann, J.W. and Lee, C.Y. (1995), "Solving a class scheduling problem with genetic algorithms", *ORSA Journal on Computing*, vol.7, no.4, pp.443-452.

Herrmann, J.W., Lee, C.Y. and Hinchman, J. (1995), "Global job-shop scheduling with a genetic algorithm", *Production & Operations Management*, vol.4, no.1, pp.30-45.

Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, MI.

Holsenback J.E. and Russel R.M. (1992), "A heuristic algorithm for sequencing on one machine to minimize total tardiness", *Journal of the Operational Research Society*, vol.43, pp.53-62.

Horn, J. and Nafpliotis N. (1993), "The Niched Pareto Genetic Algorithm", *IlliGAL Report no.93005*, Illinois Genetic Algorithms Laboratory, University of Illinois.

Horvath, M., Markus, A. and Vancza, C. (1996), "Process planning with genetic algorithms on results of knowledge-based reasoning", *Int.J.of Computer Integrated Manufacturing*, vol.9, no.2, pp.145-166.

Ibaraki T. and Nakamura Y. (1994), "A dynamic programming method for single machine scheduling", *European Journal of Operational Research*, vol.76, pp.72-81.

Iri, M. (1968), "On the synthesis of loop and cutset matrices and related problems", *RAAG Memoirs*, vol.4, A-XII, pp.376-410.

Jaccard, P., 1908, "Nouvelles recherches sur la distribution florale", *Bull. Soc. Vaud. Sci. Nat.*, 44, 223-270.

Jain, A.K. and Elmaraghy, H.A. (1997), "Production scheduling/rescheduling in flexible manufacturing systems", *Int.J.of Production Research*, vol.35, no.1, pp.281-309.

Jensen, A.P. and Barnes, J.W. (1980), *Network Flow Programming*, Wiley, New York.

Jinxing, X. (1997), "An application of genetic algorithms for general dynamic lotsizing problems", in *GALESIA '97: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.82-87, IEE Conf.Publ. no.446, IEE, Stevenage, England.

Joines, J.A., Culbreth, C.T. and King, R.E. (1996), "Manufacturing cell design: an integer programming model employing genetic algorithms", *IIE Transactions*, vol.28,no.1, pp.69-85.

Jones, A., Rabelo, L. and Yih, Y. (1995), "A hybrid approach for real-time sequencing and scheduling", *Int.J.of Computer Integrated Manufacturing*, vol.8, no.2, pp.145-154.

Kado, K., Ross, P. and Corne, D. (1995), "A study of genetic algorithms for facility layout problems", in *Proc.of the 6th Int.Conf. on Genetic Algorithms*, pp.499-505, Morgan Kaufman.

Kamhawi, H.N., Leclair, S.R. and Philip, C.L (1996), " Feature sequencing in the rapid design system using a genetic algorithm", *Journal of Intelligent Manufacturing*, vol.7, no.1, pp.55-67.

Kao, Y., and Moon, Y.B. (1998), "Feature-based memory association for group technology", *Int.J. of Production Research*, vol.36, no.6, pp.1653-1677.

Karabati, S. and Kouvelis, P. (1997), "Flow line scheduling problem with controllable processing times", *IIE Transactions*, vol.29, no.1, pp.1-14.

Kazerooni, M., Luong, L.H.S., Abhary, K., Chan, F.T.S. and Pun, F. (1996), "An integrated method for cell layout problem using genetic algorithms", in *Proc.of the 12th Int.Conf.on CAD/CAM, Robotics and Factories of the Future*, pp.752-762, Middlesex Univ.Press, London, UK.

Kebbe, I., Yokoi, H., Suzuki, K. and Kakazu, Y. (1996), "Vibrational-Potentional method for large scale scheduling problems", *Proc.of the Int.Conf. on Advance in Production Systems, APMS'96*, Okino, Tamura, Fujii (Eds.), pp.585-590, Kyoto Univ., Kyoto, Japan.

Kernighan, B.W., and Lin, S. (1970), "An efficient heuristic procedure for partitioning graphs", *Bell Systems Technology Journal*, vol.49, pp.291-307.

Kim, G.H. and Lee, C.S.G. (1995), "Genetic reinforcement learning approach to the machine scheduling problem", in *Proc.of the 1995 IEEE Int.Conf.on Robotics & Automation*, pp.196-201, IEEE, Piscataway, NJ, USA.

Kim, G.H. and Lee, C.S.G. (1996), "Genetic reinforcement learning for scheduling heterogeneous machines", in *Proc.of the 1996 IEEE Int.Conf.on Robotics & Automation*, pp.2798-2803, IEEE, Piscataway, NJ, USA.

Kim, H., Koichi, N. and Gen, M. (1994), "A method for maintenance scheduling using GA combined with SA", *Computers & Industrial Engineering*, vol.27, no.1-4, pp.477-480.

Kim, J.-U. and Kim, Y.-D. (1996), "Simulated annealing and genetic algorithms for scheduling problems with multi-level product structure", *Computers & Operations Research*, vol.23, no.9, pp.857-868.

Kim, Y.K., Hyun, C.J. and Kim, Y. (1996), " Sequencing in mixed-model assembly lines: a genetic algorithm approach", *Computers & Operations Research*, vol.23, no.12, pp.1131-1145.

King, J.R. (1980), "Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm", *Int.J. of Production Research*, vol.18, no.2, pp.213-232.

King, J.R. and Nakornchai, V. (1982), "Machine-component group formation in group technology: review and extension", *Int.J. of Production Research*, vol.20, no.2, pp.117-133.

Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P. (1985), "Optimisation by Simulated Annealing", *Science*, vol.220, pp.671-679.

Kobayashi, S., Ono, I. and Yamamura, M. (1995), "An efficient genetic algorithm for job-shop scheduling problems", in *Proc. of the 6th Int.Conf.on Genetic Algorithms and their Applications*, L.Eshelman (Ed.), pp.506-511, Morgan Kaufman Publishers, San Francisco, California.

Koza J.R. (1994), *Genetic Programming II: Automatic Discovery of reusable programs*, MIT Press, Cambridge.

Koza, J.R. (1992), *Genetic Programming: On the programming of Computers by Means of Natural Selection*, MIT Press, Cambridge.

Kumar, C.S., and Chandrasekharan, M.P. (1990), "Grouping efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology", *Int.J. of Production Research*, vol.28, no.2, pp.603-612.

Kumar, K.R., and Vannelli, A. (1987), "Strategic subcontrcting for efficient disaggregated manufacturing", *Int.J. of Production Research*, vol.25, no.12, pp.1715-1728.

Kumar, K.R., Kusiak, A., and Vannelli, A. (1986), "Grouping of parts and components in flexible manufacturing systems", *European Journal of Operational research*, vol.24, pp.387-397.

Kumar, N.S.H. and Srinivasan, G. (1996), "A genetic algorithm for job-shop scheduling - a case study, *Computers in Industry*, vol.31, no.2, pp.155-160.

Kusiak, A. (1987), "The generalised group technology concept", *Int.J. of Production Research*, vol.25, no.4, pp.561-569

Kusiak, A., and Chow, W.S. (1987), "Efficient solving of the group technology problem", *Journal of Manufacturing Systems*, vol.6, no.2, pp.117-124.

Lam, S.S., Tang K.W.C. and Cai, X. (1996), "Genetic algorithm with pigeon-hole coding scheme for solving sequencing problems", *Applied Artificial Intelligence*, vol.10, no.3, pp.239-256.

Langdon, W.B. and Poli, R. (1997), "An analysis of the MAX problem in genetic programming", in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Koza, J.R. *et al.* (eds), pp.222-230, Stanford University, CA, USA.

Lawer E.L. (1977), "A 'pseudopolynomial' algorithm for sequencing jobs to minimise total tardiness", *Annals of Discrete Mathematics*, vol.1, pp.331-342.

Lawler E.L. (1964), "On scheduling problems with deferral costs", *Management Science*, vol.11, no.2, pp.280-288.

Lawrence, S., (1984), *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, GSIA, Carnegie Mellon University.

Lee, C.Y. and Choi, J.Y. (1995), "A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights", *Computers & Operations Research*, vol.22, no.8, pp.857-869.

Lee, C.-Y., Piramuthu, S. and Tsai, Y.-K. (1997a), "Job-shop scheduling with a genetic algorithm and machine learning", *Int.J.of Production Research*, vol.35, no.4, pp.1171-1191.

Lee, H., and Garcia-Diaz, A. (1993), "A network flow approach to solve clustering problems" *Int.J. of Production Research*, vol.31, no.3, pp.603-612.

Lee, I., Sikora, R. and Shaw, M.J. (1997b), "A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes", in *IEEE Transactions on Systems, Man & Cybernetics - Part B: Cybernetics*, vol.27, no.1, pp.36-54, IEEE, Piscataway, NJ, USA.

Lee, J.G., Vogt, W.G., and Mickle, M.H. (1979), "Optimal decomposition of large-scale networks", *IEEE Transactions on Supply and Maintenance Command*, SMC-9, pp.369.

Leu, Y.-Y., Matheson, L.A. and Rees, L.P. (1996), " Sequencing mixed-model assembly lines with genetic algorithms", *Computers & Industrial Engineering*, vol.30, no.4, pp.1027-1036.

Lin, L.T., Dessouky, M.M., Kumar, K.R. and Ng, M.S. (1996), "A heuristic-based procedure for the weighted production – cell-formation problem", *IIE Transactions*, vol.28, pp.579-589.

Lu, C.G., Morton, D., Wang, Z., Myler, P. and Wu, M.H. (1995), "A genetic algorithm solution of inspection path planning system for multiple tasks inspection on co-ordinate measuring machine (CMM), in *GALESIA '95: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.436-441, IEE Conf.Publ. no.414, IEE, London, England.

Luke, S. and Spector, L. (1997), "A comparison of crossover and mutation in genetic programming", in *Genetic Programming 1997: Proceedings of the Second Annual Conference,* Koza, J.R. *et al.* (eds), pp.240-248, Stanford University, CA, USA.

MacQueen, J.B. (1967), "Some methods for classification and analysis of multivariate observations", in *Proc.of the 5th Symposium on Mathematical Statistics and Probability*", vol.1, pp.281, Univ.of California, Berkeley.

Mak, K.L. and Wong, Y.S. (1995), "Design of integrated production production-inventory-distibution systems using genetic algorithms", in *GALESIA '95: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.454-460, IEE Conf.Publ. no.414, IEE, London, England.

Matsuo H., Suh C.J. and Sullivan R.S. (1989), "A controlled search simulated annealing method for the single machine weighted tardiness problem", *Annals of Operations Research*, vol.21, pp.85-108.

Mavridou, T.D. and Pardalos, P.M. (1997), "Simulated annealing and genetic algorithms for the facility layout problem: a survey", *Computational Optimisation and its Applications*, vol.7, no.1, pp.111-126.

McAuley, J. (1972), "Machine grouping for efficient production", *Production Engineer*, vol.51, no.2, pp.53-57.

McIlhaga, M. (1997), "Solving generic scheduling problems with a distributed genetic algorithm", " in *Evolutionary Computing, AISB Workshop*, pp.199-212, " in *Evolutionary Computing, AISB Workshop*, Lecture Notes in Computer Science (1305), D.Corne & L.Shapiro (Eds.), pp.199-212, Springer Verlag, Berlin, Germany.

McIlhagga, M., Husbands, P. and Ives, R. (1996), "A comparison of optimisation techniques for integrated manufacturing planning and scheduling" in *Proc.of the 4th Int.Conf.on PPS from Nature*, Voigt, Ebeling, Rechenberg, Schwefel (Eds.), pp.604-613, Springer-Verlag, Berlin, Germany.

McKay, B.M., Willis, M.J., Hiden, H.G., Montague, G.A. and Barton, G.W. (1996), "Identification of industrial processes using genetic programming", in *Proc. of the Conf. on Identification in Engineering Systems*, Friswell and Mottershead (Eds.), pp.510-519, Univ.of Wales, Swansea, UK.

McNaughton R (1959), "Scheduling with deadlines and loss functions", *Management Science*, vol.6, no.1, pp.1-12.

Michalewicz, Z. (1992), *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York.

Mitrovanov, S.P. (1966), *The Scientific Principles of Group Technology*, National Lending Library Translation, Boston Spa, Yorkshire, U.K.

Montagne, G.R. (1969), "Sequencing with time delay costs", *Industrial Engineering Research Bulletin*, 5, Arizona State University, 1969.

Murata, T., Ishibuchi, H. and Tanaka, H. (1996a), "Genetic algorithms for flowshop scheduling problems", *Computers & Industrial Engineering*, vol.30, no.4, pp.1061-1071.

Murata, T., Ishibuchi, H. and Tanaka, H. (1996b), "Multi-objective genetic algorithm and its application to flow-shop scheduling", *Computers & Industrial Engineering*, vol.30, no.4, pp.957-968.

Nair, J.G., nad Narendran, T.T. (1998), "CASE: a clustering algorithm for cell-formation with sequence data", *Int.J. of Production Research*, vol.36, no.1, pp.157-179.

Nair, J.G., nad Narendran, T.T. (1999), "ACCORD: a bicriteria algorithm for cell-formation using ordinal and ratio level data", *Int.J. of Production Research*, vol.37, no.3, pp.539-556.

Ng, S.M. (1993), "Worst-case analysis of an algorithm for cellular manufacturing", *European Journal of Operational Research*, vol.69, pp.384-398.

Niemeyer, G. and Shiroma, P. (1996), "Production scheduling with genetic algorithms and simulation", in *Proc.of the 4th Int.Conf.on PPS from Nature*, Voigt, Ebeling, Rechenberg, Schwefel (Eds.), pp.930-939, Springer-Verlag, Berlin, Germany.

Nordin, P. (1994), "A compiling genetic programming system that directly manipulates machine code", in *Advances in Genetic Programming*, Kinnear Jr., K.E. (Ed.), pp.311-331, MIT Press, Cambridge.

Norman, B.A. and Bean, G.C. (1997), "Operation sequencing and tool assignment for multiple spindle CNC machines", in *Proc.of the 1997 IEEE Conf.on Evolutionary Computation*, pp.425-429, IEEE, Piscataway, NJ, USA.

Norman, B.A. and Smith, A.E. (1997), "Random-keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems", in *Proc.of the 1997 IEEE Int.Conf.on Evolutionary Computation*, pp.407-411, IEEE, Piscataway, NJ, USA.

O'Reilly, U-M. and Oppacher, F. (1995), "The troubling aspects of a building block hypothesis for genetic programming", in *Foundations of Genetic Algorithms 3*, Whitley, L.D. and Vose M.D., (eds), pp.73-88, Morgan Kaufman, San Francisco, CA.

Oei, K.O, Goldberg, D.E. and Chang, S.-J. (1991), "Tournament selection, niching, and the preservation of diversity", *IlliGAL Report no.91011*, Illinois Genetic Algorithms Laboratory, University of Illinois.

Ono, I., Yamamura, M. and Kobayashi, S. (1996), "A genetic algorithm for job-shop scheduling problems using job-based order crossover", in *Proc.of the 1996 IEEE Int.Conf. on Evolutionary Computation*, pp.2798-2803, IEEE, Piscataway, NJ, USA.

Panwalkar S.S., Smith M.L. and Koulamas C.P. (1993), "A heuristic for the single machine tardiness problem", *European Journal of Operational Reserarch*, vol.70, pp.304-310.

Park, L.-J. and Park, C.H. (1995a), "Genetic algorithms for job-shop scheduling problems based on two representational schemes", *Electronics Letters*, vol.31, no.3, pp.205-207.

Park, L.-J. and Park, C.H. (1995b), "Application of genetic algorithms to job-shop scheduling problems with active-schedule constructive crossover", in *Proc.of the 1995 IEEE Int.Conf. on Systems, Man & Cybernetics*, pp.530-535, IEEE, Piscataway, NJ, USA.

Parmee, I. (1998), "Exploring the design potential of evolutionary/adaptive search and other computational intelligence technologies", in *3rd Int. Conf. of Adaptive Computing in design and manufacture*, pp. 27-44, Springer-Verlag, Plymouth, U.K..

Parmee, I. and Watson, A.H. (2000), "An investigation of the utilization of genetic programming techniques for response curve modelling", in *Statistics for Engine Optimization*, Edwards, S., P., Grove, D., M. and Wynn, H., P. (eds.), pp.125-143, Professional Engineering Publishing, Bury St Edmunds, UK.

Patro, S. and Kolarik, W.J. (1997), "Neural networks and evolutionary computation for real-time quality control of complex processes", *Proc. of the 1997 IEEE Annual Reliability and Maintainability Symposium*, pp.327-332, IEEE, Piscataway, NJ, USA.

Pinedo, M. (1995), *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs.

Pohlheim, H. and Marenback, P. (1996), "Generation of structured process models using genetic programming", in *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science (1143), T.C.Fogarty (Ed.), pp.102-109, Springer-Verlag, Berlin, Germany.

Potts C.N. and Van Wassenhove L.N. (1982), "A decomposition algorithm for the single machine total tardiness problem", *Operations Research Letters*, vol.1, no.5., pp.177-181.

Potts C.N. and Van Wassenhove L.N. (1985), "A branch & bound algorithm for the total weighted tardiness problem", *Operations Research*, vol.33, pp.363-377.

Potts C.N. and Van Wassenhove L.N. (1991), "Single machine tardiness sequencing heuristics", *IIE Transactions*, vol.23, no.4, pp.346-354.

Rajagopalan, R., and Batra, J.L. (1975), "Design of cellular production systems: a graph-theoretic approach", *Int.J. of Production Research*, vol.13, no.6, pp.567-579.

Rasheed, K., Hirsch, H. and Gelsey, A. (1997), " A genetic algorithm for continuous design search space", *Artificial Intelligence in Engineering*, vol.11, pp.295-305.

Rechenberg, I. (1973), Evolutionsstrategie: *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, Germany.

Reeves, C.R. (1995), "A genetic algorithm for flowshop scheduling", *Computers & Operations Research*, vol.22, no.1, pp.5-13.

Reeves, C.R., Dai, P. and Burham, K.J. (1996), "A hybrid genetic algorithm for system identification", in *Proc.of the Int.ICSC Symposia on Intelligent Industrial Automation and Soft Computing*, Anderson & Warwick (Eds.), pp.B278-B283, Int.Comp.Science Conventions, Millet, Canada.

Rinnooy Kan A.H.G, Lageweg B.J. and Lenstra J.K. (1975), "Minimising total costs in one-machine scheduling", *Operations Research*, vol.23, no.5, pp.908-927.

Ross, P. and Tuson, A. (1997), "Directing the search of evolutionary and neighbourhood-search optimisers for the flowshop sequencing problem with idle-time heuristic" in *Evolutionary Computing, AISB Workshop*, Lecture Notes in Computer Science (1305), D.Corne & L.Shapiro (Eds.), pp.213-225, Springer Verlag, Berlin, Germany.

Rubinovitz, J. and Levitin, G. (1995), "Genetic algorithm for assembly line balancing", *Int.J.of Production Economics*, vol.41, no.1-3, pp.343-354.

Russel R.M. and Holsenback J.E. (1996), "Evaluation of leading heuristics for the single machine tardiness problem", *European Journal of Operational Research*, vol.96, pp.304-310.

Russel R.M. and Holsenback J.E. (1997), "Evaluation of greedy, myopic and less-greedy heuristics for the single-machine, total tardiness problem", *Journal of the Operational Research Society*, vol.48, pp.640-646.

Sannomiya, N. and Iima, H. (1996), "Application of a genetic algorithm to scheduling problems in manufacturing processes", *Proc.of the 1996 IEEE Int.Conf.on Evolutionary Computation*, pp.523-528, IEEE, Piscataway, NJ, USA.

Sarker, B.R. (1996), "The resemblance coeficients in group technology: a survey and comparative study of relational metrics", *Computers & Industrial Engineering*, vol.30, no.1, pp.103-116.

Sarker, B.R., and Mondal, S. (1999), "Grouping efficiency measures in cellular manufacturing: a survey and critical review", *Int.J. of Production Research*, vol.37, no.2, pp.385-314.

Schaffer, J.D. (1985), "Multiple objective optimization with vector evaluated genetic algorithms", in *Proc. of the 1st Int. Conf. on Genetic Algorithms and their Applications*, pp.93-100, J.J.Grefenstette (Ed.), Lawrence Erlbaum, Hillsdale, NJ.

Schild A. and Fredman I.J. (1961), "On scheduling tasks with associated linear loss functions", *Management Science*, vol.7, no.3, pp. 280-285.

Schrage L.E. and Baker K.R. (1978), "Dynamic programming solution of sequencing problems with precedence constraints", *Operations Research*, vol.26, no.3, pp.444-449.

Sebaaly, M.F. and Fujimoto, H. (1996), "A genetic planner for assembly automation", in *Proc.of the 1996 IEEE Int.Conf.on Evolutionary Computation*, pp.401-406, IEEE, Piscataway, NJ, USA.

Seifoddini, H (1989), "Single linkage vs. average linkage clustering in machine cells formation application", *Computers & Industrial Engineering*, vol.16, pp.419-426.

Seifoddini, H., and Djassemi, M. (1995), "Merits of the production volume based similarity coefficient in machine cell-formation", *Journal of Manufacturing Systems*, vol.14, no.1, pp.35-44.

Selim, M.H., Askin, R.G., and Vakharia, A.J. (1998), "Cell-formation in group technology: review, evaluation and directions for future research", *Computers & Industrial Engineering*, vol.34, no.1, pp.3-20.

Sen T., Austin L.M. and Ghandforoush P. (1983), "An algorithm for the single machine sequencing problem to minimize total tardiness", *AIIE Transactions*, vol.15, pp.363-366.

Shaw, K.J. and Flemming, P.J. (1997), "Use of rules and preferences for schedule builders in genetic algorithms for production scheduling" in *Evolutionary Computing, AISB Workshop*, Lecture Notes in Computer Science (1305), Corne, D. and Shapiro, L. (Eds.), pp.237-250, Springer Verlag, Berlin, Germany.

Shi, G. (1997), "A genetic algorithm applied to a classic job-shop scheduling problem", *Int.J. of Systems Science*, vol.28, no.1, pp.25-32.

Shtub, A. (1989), "Modelling group technology cell-formation as a generalised assignment problem", *Int.J. of Production Research*, vol.27, no.5, pp.775-782.

Sikora, R. (1996), "A genetic algorithm for integrating lot-sizing and sequencing in scheduling a capacitated flow-line", *Computers & Industrial Engineering*, vol.30, no.4, pp.969-981.

Singh, N. (1993), "Cellular manufacturing systems: an invited review", *European Journal of Operational Research*, vol.69, pp.284-291.

Singh, N. (1996), *Systems Approach to Computer-Integrated Design and Manufacturing*, Wiley, Chchester, New York.

Sneath, P.H.A, (1957), "The application of computers to taxonomy", *Journal of General Microbiology*, vol.17, pp.201-206.

Sniedovich, M. (1988), "A multi-objective routing problem revisited", *Engineering Optimization*, vol.13, pp.99-108.

Sofianopoulou, S. (1997), "Application of simulated annealing to a linear model for the formulation of machine cells in group technology", *Int.J. of Production Research*, vol.35, no.2, pp.501-511.

Sofianopoulou, S. (1999), "Manufacturing cells design with alternative process plans and/or replicate machines", *Int.J. of Production Research*, vol.37, no.3, pp.707-720.

Srinivas, N. and Deb, K. (1994), "Multiobjective optimisation using nondominated sorting in genetic algorithms", *Evolutionary Computation*, vol.2, no.3, pp.221-248.

Srinivasan V. (1971), "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness", *Naval Research Logistics Quarterly*, vol.18, no.3, pp. 317-327.

Srinivasan, G, and Narendran, T.T. (1991), "GRAFICS – a nonhierarchical clustering algorithm for group technology", *Int.J. of Production Research*, vol.29, no.3, pp.463-478.

Srinivasan, G, and Narendran, T.T., and Mahaderan, B. (1990), "An assignment model for the part-families problem in group technology", *Int.J. of Production Research*, vol.28, no.1, pp.145-152.

Srinivasan, G. (1994), "A clustering algorithm for machine cell-formation in group technology using minimum spanning trees", *Int.J. of Production Research*, vol.32, no.9, pp.2149-2158.

Stanfel, L.E. (1985), "Machine clustering for economic production", *Engineering Costs & Production Economics*, vol.9, pp.73-81.

Stockton, D.J. and Quinn, L. (1995), "Aggregate production planning using genetic algorithms", *Proc. of the Inst.of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol.209, no.B3, pp.201-209.

Su, C.-T. and Hsu, C.-M. (1996), "A two-phased genetic algorithm for the cell formation problem", *Int.J.of Industrial Engineering*, vol.3, no.2, pp.114-125.

Suresh, G., Vivod, V.V. and Sahu, S. (1995), "A genetic algorithm for facility layout", *Int.J.of Production Research*, vol.33, no.12, pp.3411-3423.

Suresh, G., Vivod, V.V. and Sahu, S. (1996), "A genetic algorithm for assembly line balancing", *Production Planning & Control*, vol.7, no.1, pp.38-46.

Swaragi, Y., Nakayama, H. and Tanino, T. (1985), *Theory of Multi-Objective Optimisation*, Academic Press, New York.

Tailard, E. (1993), "Benchmarks for basic scheduling problems", *European Journal of Operations Research*, vol.64, pp.278-285.

Takatori, N., Minagawa, M. and Kakazu, Y. (1994), "A GA-based approach to a process palnning problem with geometric constraints", in *ANNIE'94: Artificial Neural Networks in Engineering. Proceedings*, pp.369-374, ASME Press.

Tam, K.Y. (1992), "Genetic algorithms, function optimisation and facility layout design", *European Journal of Operational Research*, vol.63, no.2, pp.322-346.

Tansel B.C. and Sabuncuoglu I. (1997), New insights on the single machine total tardiness problem, *Journal of the Operational Research Society*, vol.48, pp.82-89.

Tate, D.M. and Smith, A.E. (1995), "Unequal-area facility layout by genetic search", *IIE Transactions*, vol.7, no.4, pp.465-472.

Teller, A. and Veloso, M. (1996), "PADO: A new learning architecture for object recognition" in *Symbolic Visual Learning*, Ikeuchi, Katsushi and Veloso Manuela (eds), Oxford University Press.

Tsujimura, Y., Gen, M. and Kubota, E. (1995), "Solving fuzzy assembly-line balancing problems with genetic algorithms", *Computers & Industrial Engineering*, vol.29, no.1-4, pp.543-547.

Usher, J.M. and Bowden, R. (1996), "The application of genetic algorithms to operation sequencing for use in computer-aided process planning", *Computers & Industrial Engineering*, vol.30, no.4, pp.999-1013.

Vakharia, A.J., and Chang, Y.-L. (1997), "Cell-formation in group technology: a combinatorial search approach", *Int.J. of Production Research*, vol.35, no.7, pp.2025-2043.

Vakharia, A.J., and Wemmerlov, U. (1990), "Designing a cellular manufacturing system: a material flow approach based on operation sequences", *IIE Transactions*, vol.22, no.1, pp.84-97.

Vannelli, A., and Kumar, K.R. (1986), "A method for finding minimal bottleneck cells for grouping part-machine families", *Int.J. of Production Research*, vol.24, no.2, pp.387-400.

Venugopal, V. and Narendran, T.T. (1992), "A genetic algorithm approach to the machine-component grouping problem with multiple objectives", *Computers & Industrial Engineering*, vol.22, no.4, pp.269-480.

Viswanadham, N., Sharma, S.M. and Taneja, M. (1996), "Inspection allocation in manufacturing systems using stochastic search techniques", *IEEE Transactions on Systems, Man, & Cybernetics - Part A: Systems and Humans*, vol.26, no.2, pp.222-230.

Watson, A.H. and Parmee, I. (1996), "Systems Identification using Genetic Programming", in *Proceedings of the 2nd International Conference on Adaptive Computing in Engineering Design and Control (ACEDC '96)*, Parmee, I.C. (ed.), pp.248-255, PEDC, University of Plymouth, UK.

Watson, A.H. and Parmee, I. (1998), "Improving Engineering Design Models using An Alternative Genetic Programming Approach", in *3rd Int. Conf. of Adaptive Computing in design and manufacture*, pp. 193-206, Springer-Verlag, Plymouth, U.K..

Wang, D. and Fang, S.-C. (1997), "A genetics-based approach for aggregated production planning in a fuzzy environment", *IEEE Transactions on Systems, Man, Cybernetics - Part A: Systems and Humans*, vol.27, no.5, pp.636-645.

Wei, J.C., and Gaither, N. (1990), "An optimal model for cell-formation decisions", Decision Sciences, vol.21, no.2, pp.416-433.

Wilkerson L.J. and Irwin J.D. (1971), "An improved algorithm for scheduling for independent tasks", *AIIE Transactions*, vol.3, no.3, pp.239-245.

Winston, W. L. (1995), *Introduction to mathematical programming : applications and algorithms*, Belmont, California, Wadsworth Pub.Co.

Yamada, T. and Nakano, R. (1992), "A genetic algorithm applicable to large scale job shop problems", in *Proc.of the 2nd Int.Conf.on PPS from Nature*, Männer & Manderick (Eds.), pp.281-290, Elsevier Science Publishers, North Holland.

Yamada, T. and Nakano, R. (1995), "A genetic algorithm with multi-step crossover for job-shop scheduling problems", in *GALESIA '95: Genetic Algorithms in Engineering Systems: Innovations and Applications. Conference Proceedings*, pp.146-151, IEE Conf.Publ. no.414, IEE, London, England.

Yip-Hoi, D. and Dutta, P. (1996), "A genetic algorithm application for sequencing operations in process planning for parallel machining", *IIE Transactions*, vol.28, no.1, pp.55-68.

Zhang, F., Zhang, Y.F. and Nee, A.Y.C (1998), "Using genetic algorithms in process planning for job shop machining", *IEEE Transactions on Evolutionary Computation*, vol.1, no.4, pp.278-289, IEEE, Piscataway, NJ, USA.

Zhao, L., Tsujimura, Y. and Gen, M. (1996), "Genetic algorithm for robot selection and workstation assignment problem", *Computers & Industrial Engineering*, vol.31, no.3/4, pp.599-602.

Zhou, G. and Gen, M. (1997), "Evolutionary computation of multi-criteria production process planning problem", in *Proc.of the 1997 IEEE Conf.on Evolutionary Computation*, pp.419-424, IEEE, Piscataway, NJ, USA.

Zhu, Z., Heady, R.B., and Reiners, S. (1995), "An efficient zero-one formulation of the cell-formation problem", *Computers & Industrial Engineering*, vol.28, no.4, pp.911-916.

Zitzler, E. and Thiele, L. (1999), "Multiobjective evolutionary algorithms: a comparative case study and the Strength Pareto approach", *IEEE Transactions on Evolutionary Computation*, vol.3, no.4, pp.257-271.

Zurada, J., M. (1992), *Introduction to Artificial Neural Systems*, West Publishing Company, St.Paul.