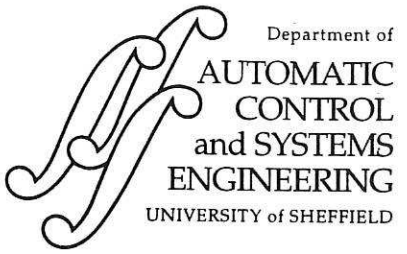



---

**Monograph:**
Dimopoulos, C. and Zalzala, A. (1998) Investigating the Use of Genetic Programming for a Classic One-Machine Scheduling Problem. Research Report. ACSE Research Report 737 . Department of Automatic Control and Systems Engineering

---




# INVESTIGATING THE USE OF GENETIC PROGRAMMING FOR A CLASSIC ONE-MACHINE SCHEDULING PROBLEM

**Christos Dimopoulos and Ali Zalzala**

*Robotics Research Group,*
*Department of Automatic Control and Systems Engineering*
*The University of Sheffield*
*Mappin Street, Sheffield S1 3JD, United Kingdom*

Research Report #737
December 1998


RRG Tel : +44 (0)114 2225250
Fax : +44 (0)114 2731729
EMail : rrg@sheffield.ac.uk
**Robotics Research Group**



# INVESTIGATING THE USE OF GENETIC PROGRAMMING FOR A CLASSIC ONE-MACHINE SCHEDULING PROBLEM

Christos Dimopoulos and Ali Zalzala
*Robotics Research Group*
*Department of Automatic Control & Systems Engineering*
*The University of Sheffield*
*Email: rrg@sheffield.ac.uk*

**Abstract**: Genetic programming has rarely been applied to manufacturing optimisation problems. In this report we investigate the potential use of Genetic Programming for the solution of the one-machine total tardiness problem. Combinations of dispatching rules are employed as an indirect way of representing permutations within a Genetic Programming framework. Hybridisation of Genetic Programming with local search techniques is also introduced, in an attempt to improve the quality of solutions. Finally, Genetic Programming is utilised for the evolution of scheduling policies in the form of dispatching rules. These rules are trained to cope with different levels of tardiness and tightness of due dates.

**Keywords:** Evolutionary computation, genetic programming, manufacturing optimisation, tardiness, scheduling.

## 1. Introduction

Manufacturing optimisation has become a major application field for evolutionary computation methods. In a recent report [1] we highlighted the surprisingly wide range of manufacturing optimisation problems covered by active evolutionary computation research.

The combinatorial nature of most manufacturing optimisation problems encourages the use of evolutionary algorithms (EAs), or any other form of meta-heuristics (Simulated Annealing [2], Tabu Search [3] etc.). While numerous applications of traditional EAs (Genetic Algorithms [4], Evolution Strategies [5], and Evolutionary Programming [6]) have been reported, manufacturing optimisation has rarely been the subject of Genetic Programming (GP) research. Some notable exceptions to this rule are Garces-Peres et al. [7] who presented an algorithm for the solution of the Facility Layout Problem (FLP) and two process identification frameworks introduced in [8] and [9].

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

In this report we investigate the potential use of traditional and modified-GP algorithms for the solution of a well-researched scheduling problem, the one-machine total tardiness problem. Potts & Van Wassenhove [10] have constructed an algorithm that is able to find optimal solutions for this problem within acceptable computational times even for very large instances. This algorithm allows the realistic evaluation of the performance of all GP-generated methods introduced in this report. However, while Potts & Van Wassenhove's algorithm is problem-dependent and has no other known applicability, all the methods described in the following sections can be used - in principle - for the solution of any other one-machine scheduling problem.

## 2. Minimising total tardiness in a single-machine environment

### 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 we define lateness of job $i$ 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 the following expression:

$$T_i=\max(0, C_i - d_i)$$

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

A number of jobs $J_1, J_2\ldots,J_n$ are to be processed in a single facility. Each job is available for processing at time zero, and is completely identified by its processing time $p_i$ and its due date $d_i$. We seek to find the processing sequence that minimises the sum of tardiness of all jobs:

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

where $C_i$ is the completion time of job $i$. If for each job there is an associated weight (penalty) $w_i$, then (1) becomes

$$\sum_{i=1}^{n} w_i\{\max(0, C_i - d_i)\} \quad (2)$$

The objective of the weighted total tardiness problem is the minimisation of expression (2). If we divide (1) or (2) by the number of jobs $n$, the objective becomes 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 $n$. In the following paragraphs we will examine various methods that have been proposed for the solution of these problems.

## 2.2 Problem complexity

The complexity of the weighted total tardiness problem was established by Lawer [11] in 1977. He proved that the associated decision problem is NP-complete by reduction from the 3-partition problem. An alternative proof was given by Lenstra et al. [12] the same year. Lawler also 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 $O(n^4P)$, where

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

and $p_i$ is the processing time of job $i$ . The complexity of the unweighted total tardiness problem remained unestablished until 1989, when Du & Leng [13] proved that the associated decision problem is NP-complete by reduction from a restricted version of the Even-Odd Partition problem. The existence of Lawler's 'pseudopolynomial' algorithm meant that the unweighted total tardiness problem is NP-hard in the ordinary sense.

## 2.3 Literature review

### *2.3.1 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 inefficient, 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 & Bound. While these algorithms are analytical, their application is restricted to relatively small-sized problems due to computational and memory requirements.

Dynamic Programming (DP), a powerful optimisation method introduced by Bellman [14], is much faster than complete enumeration (see French [15] for an illustration of computational requirements). However, it has obvious limitations in terms of memory requirements ($2^n$ values must be stored before the construction of an optimal schedule). Branch & Bound methods are quite unpredictable in their computational requirements. Their success depends heavily on the calculation of sharp lower bounds, which will help on the quick elimination of subtrees, speeding up the procedure considerably.

In recent years, especially after Potts & Van Wassenhove presented a quite efficient algorithm for the solution of very large problem instances [10], 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 we will present the research on the one-machine total tardiness problem as it has evolved during the last forty years.

### 2.3.2 *Early approaches*

The earliest investigation of the total tardiness problem is given by McNaughton [16], 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 WSPT rule produces optimal schedules when no job can be finished on time. Schild & Fredman [17] 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. The same authors [18] later proposed an algorithm for solving the associated problem with non-linear loss functions (quadratic).

Dynamic Programming was for the first time adopted for the solution of sequencing problems by Held & Karp in a much referenced paper [19]. The principle of optimality for a scheduling problem of this type states 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 are formed based on this principle. Note that this formulation describes the forward DP approach. Many researchers prefer the backward DP formulation, where the principle optimality takes a different form. Lawler [20] was the first researcher to utilise 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.

In 1968 Emmons [21] published a breakthrough paper on the one-machine total tardiness problem without associated weights. He proved a series of theorems that establish relations between jobs in an optimal sequence. Emmons also proposed an algorithm that utilises these theorems in order to reduce the size of the problem and employs branching when no further relations between jobs can be found. These theorems have formed the basis of many optimisation methods for the total tardiness problem over the years. Emmons also generalised the cases where EDD or SPT sequencing yields optimal schedules.

The same year Elmaghraby [22] presented the first Branch & Bound approach for the solution of the weighted total tardiness problem. He formulated the problem using a network model and consequently translated the minimisation function into a shortest path problem. A six-step six-step branch & bound algorithm was then described for the solution of the total tardiness problem with linear penalty functions. Elmaghraby claimed that his method could reduce substantially the computational requirements. The operation of the algorithm was illustrated with the help of an example problem, but no other results were presented.

A few years later Srinivasan [23] 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 random generated test problems showed that his method exhibited substantial gain on computational efficiency in comparison with complete enumeration and classic DP. Srinivasan also investigated that the effect of changing the parameters of the problem to the computational requirements. He reported that problems representing shops 60% tardy (tardiness factor t=0.6) were computationally hard. His observation was later confirmed by a number of different researchers.

The same year Wilkerson & Irwin [24] 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.

Shwimer [25] enhanced the branch & bound formulation of Elmaghraby by using Emmons' theorems and a number of new dominance criteria. The EDD rule was used to provide good upper bounds, but the problems considered were still relatively small ($n \leq 20$). The computational efficiency of the algorithm varied considerably with the parameters of the problem.

2.3.3 *Development of implicit enumeration algorithms during the 80's and 90's*

Rinnooy Kan et al. [26] took the brand & bound approach one step further by presenting an improved algorithm for 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 Shwimer's for problems with $15 \leq n \leq 20$, they underlined the need for sharper lower bounds, and even better dominance theorems.

Fisher [27] recognised this need and produced an algorithm that was able to find extremely sharp lower bounds for branch & bound algorithms. His method was based on a dual formulation of the problem. He introduced a sub-algorithm 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.

We already indicated that the dynamic programming approaches to the total tardiness problem had limited applicability due to the extensive memory requirements. However, in 1978 Baker & Shrage presented two dynamic programming - based methods which were able to solve large-sized problems ($n$=50) faster than branch & bound algorithms. The first method [28] was an investigation on the application of dynamic programming to sequencing problems with precedence constraints. The authors depicted that when chain-like relations between jobs exist (jobs with only one direct predecessor and only one direct successor), the number of feasible subsets that need to be enumerated is 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 can be utilised to create them a posteriori. A few months later Shrage & Baker [29] 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 & Baker showed that their method could solve 50-job test problems much faster than the algorithms of Fisher and Rinnoy Kan et al.

The same year Picard & Queyranne [30] proposed yet another branch & bound algorithm for the solution of the weighted total tardiness problem. They devised a novel method for obtaining lower bounds which is based on relaxation of the time-dependent travelling salesman problem, a special case of the Quadratic Assignment problem which serves as a model for sequencing problems. However, their method was applied only in small-sized problems, and presented no advantage over the contemporary dynamic programming approach of Shrage & Baker.

In the early 80's Potts & Van Wassenhove [10] 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 & 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 [11] was utilised to break up the problem in a number of smaller and easily tackled sub-problems. Each of these sub-problems was solved optimally by the dynamic programming approach of Shrage & Baker. The efficiency and the speed of this algorithm enabled the solution of very 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 & bound method for the solution of the latter problem [31]. 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 & Van Wassenhove's method was able to solve problems much larger than those reported in earlier branch & 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 considerably the search for an optimal schedule.

Another implicit enumeration algorithm was proposed by Sen et al.[32] who utilised Emmon's theorems and corollaries in order to establish precedence relations between jobs. They reported that their seven-step algorithm outperformed DP in terms of computational efficiency for large-sized problems ($n>50$).

### 2.3.4 *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 classic 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. [33] 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 the method with Wilkerson & Irwin's algorithm showed that API's constitute a very fast and reliable optimisation for the one-machine total tardiness problem.

Some years later, Holsenback & Russel [34] introduced their powerful Net Benefit Of Relocation (NBR) heuristic procedure. Emmons stated in one of his corollaries that an EDD

sequence is optimal if the tardiness of each job is less or equal than its processing time. The NBR heuristic is based on this observation. Starting from the last job of the EDD sequence the algorithm finds the first job which possesses 'reducible' tardiness (i.e. $T_i > p_i$). For each job preceding job $i$ in the sequence the Net Benefit of Relocation is 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) exchange its position with job $i$. The same procedure is repeated on the remaining $i$-1 jobs. NBR method produced high quality results even for very large problem instances ($n$=100). The deviations from the optimal values - obtained using Potts & Wassenhove's method - were very small, while the computational times where excellent (around one second of CPU time for problems with $n$=100).

A year later Panwalkar et al. [35] introduced the PSK heuristic, which performed impressively on a number of different cases. PSK is a simple, effective procedure that starts from a n SPT sequence and constructs a schedule by making $n$-passes one for each job. The algorithm uses two sets of jobs, the SPT set of unscheduled jobs {U}, and the set of the of jobs that have already been scheduled {S}. In each pass, a job from {U} is considered to be active, starting from the leftmost one. If the scheduling of this job is necessary since it is already tardy or on time, or because the successive job will make it tardy, then it is removed from {U}and it is placed on {S}. Otherwise the next job in {U} becomes active and the same set of comparisons is performed. The algorithm was tested on a wise range of problems producing very good results. However, Russel & Holsenback [36] questioned the validity of their results, claiming that in their own experimentation with the same problems used in [35], the NBR was generally superior to PSK. In any case, they noted that PSK is particularly suitable for a specific set of problems characterised by high values of tardiness factor and range of due dates.

Recently, Russel & Holsenback [37] 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 are extremely fast, easy to implement and complementary in nature.

Meta-heuristics (simulated annealing, tabu search, genetic algorithms etc.) have gained a considerable research interest the last decade. All of these techniques have been applied to a wide range of scheduling problems. Matsuo et al. [38] were the first researchers to employ simulated annealing for the solution of the weighted total tardiness problem. A few years later Potts & Van Wassenhove [39] 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 & Al-Fawzan [40]. 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. [33] and Holsenback & Russel.[34] in a wide range of test problems. However, while Holsenback's heuristic is extremely fast, Ben-Daya's algorithm suffers from slow convergence, a well-known disadvantage of simulated annealing.

Another dynamic programming - based approach for the solution of the weighted total tardiness problem was introduced recently by Ibaraki & Nakamura [41]. Their method is called Successive Sublimation Dynamic Programming and aims to reduce the number of subsets that need to be considered 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 classic dynamic programming. However, as the authors indicated, the method has limited applicability to problems characterised by certain levels of tardiness

Tausel & Sabuncuoglu [42] have recently presented an interesting investigation on the total tardiness problem, utilising geometric representations in order to analyse and prove Emmons' theorems. Their research further included 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 usually employed by researchers is not guaranteed.

#### 2.3.5 *General reading*

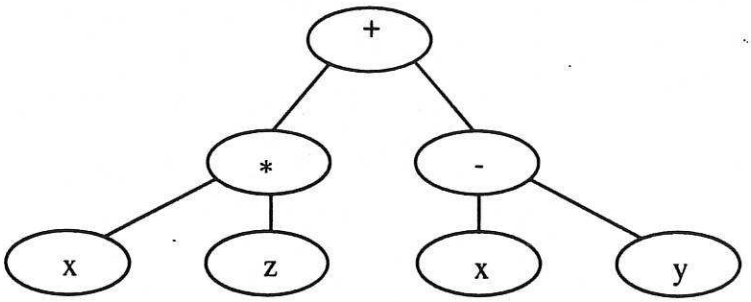
Despite the fact that this review covers a wide range of research papers, we cannot claim that forty years of academic research has been included in the previous pages. For the interested reader, we recommend the books of Baker [43], French [16] and Pinedo [44], which give an excellent introduction on scheduling problems and discuss single machine scheduling in great depth. Gupta & Kyparisis [45] present an excellent review on single machine scheduling research until mid 80's. Sen & Gupta [46] focus on scheduling problems involving due dates for the same period. Finally, Baker & Martin [47], Van Wassenhove & Gelders [48], Potts & Van Wassenhove [31] and Russel & Holsenback [34] give comparative results on the application of different optimisation methods to the one machine total tardiness problem.

## 3. Brief introduction to Genetic Programming

Genetic Programming belongs to the family of evolutionary computation methods. During the 80's a number of researchers investigated the use of evolutionary computation for program induction [49]-[51]. Koza [52], [53] introduced the most efficient of all methods and used the term 'Genetic Programming' to describe it.

The operation of GP is quite similar to those of other evolutionary algorithms. The concept of Darwinian strife for survival is the driving force of the algorithm. A potential solution of an optimisation problem is appropriately coded into a form of chromosome, and a population of these solutions is employed for the evolution of optimal or near optimal solutions through successive generations. Each new generation is created by probabilistically selecting individuals from the old generation according to their fitness. These individual either survive intact to the new generation or they are genetically modified through a number of operators. In traditional evolutionary algorithms individuals are most of the times fixed-sized strings of problem parameters. In contrast, GP evolves computer programs of variable size, i.e. representations that can be translated by a computer either as they have been evolved or with slight modifications. In that sense GP is a form of automatic programming (a method of teaching the computers how to program themselves).

Genetic programs are usually illustrated as collections of function and terminal nodes in the form of a parse tree (fig. 1).




$$= x * z + (x - y)$$

**Fig. 1:** An example of a GP parse tree and its interpretation

A tree structure is interpreted in a depth-first, left to right postfix way as depicted in the same figure. This type of representation is dominant in the GP field, since Koza adopted it in his pioneering works. However, a number of alternative program representations have been proposed in the last few years (Nordin [54], Teller & Veloso [55]).

Like in most EAs, crossover & mutation are the two major operators that are employed for the genetic modification of tree structures. The application of these operators in GP is quite simple. For the crossover operator (mostly known as subtree-crossover) two individuals are probabilistically selected from the population according to their fitness. A crossover point is selected randomly at each tree, and the subtrees defined by these points exchange their positions at the genetic programs (see Appendix: Example 1). The functions and the terminals of the system have been selected and designed in a way that ensures satisfying the closure property (closure property: any function should be able to accept any other function or terminal as its input, without bringing the system to a halt).

Mutation operates as follows: A genetic program is probabilistically selected from the population based on its fitness, and a cut-off point is chosen randomly. The subtree defined by this cut-off point is deleted, and a new subtree randomly created takes its place in the program, subject to the size constraints defined by the user (see Appendix: Example 2).

Traditionally, subtree crossover is considered to be a significant element of GP. Koza applies it to individual programs with a probability of 90%! However, in the last few years some researchers have argued that the success of the crossover operator might be problem-dependent, since studies have shown that most of crossover operations that take place in an independent GP-run produce a negative effect on the fitness of the solutions (see Banzhaf et al. [56] for an in-depth analysis on the subject of GP-crossover and its implications).

# 4. A GP-heuristic for the solution of the one-machine total tardiness problem

## 4.1. Design of the algorithm

### *4.1.1 Problem 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 report. Specially designed genetic operators (originally created for the solution of the Travelling Salesman Problem) ensure the feasibility of solutions throughout the evolutionary procedure.

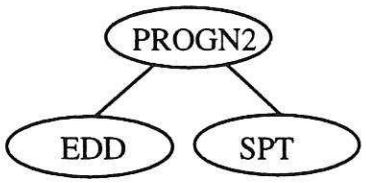
The representation of a permutation within a conventional GP framework is not straightforward, since genetic programs are structures of variable length while a permutation has a predefined length size.

In the implementation proposed in sections 4 and 5, common manufacturing dispatching rules are employed as an indirect way of representing a permutation through a genetic program. A dispatching or priority rule is a method of determining the next job to be scheduled out of a set of unscheduled jobs. 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 [57]-[59]. However, in our case the choice of dispatching rules is limited since we are addressing a static scheduling problem.

The idea of using combinations of dispatching rules for the solution of scheduling problems is not new. In a well-known scheduling textbook, Fisher & Thompson [60] proposed the probabilistic learning of scheduling rules as a method of tackling job-shop scheduling problems. Many other researchers have employed combinations of dispatching rules for the solution of similar problems [61]-[63] . The majority of these approaches are based on the idea of utilising different dispatching rules on individual machines and scheduling points in the plant. Dorndorf & Pesch [64] proposed a concept that is similar to the one proposed in this section. He employed a string of dispatching rules as a decision policy for the conflict set of jobs created by Giffler & Thompson's algorithm in job-shop problems. While this method was unsuccessful in comparison with other algorithms proposed in the same paper, it suggested that a combination of dispatching rules performed better than individual dispatching rules.

### *4.1.2. Function and terminal sets*

In the algorithm presented in this section, a sequence of jobs is constructed indirectly by an associated sequence of dispatching rules. While dispatching rules can be easily represented in a GP framework as terminal nodes, there are still several issues that need to be addressed before a GP run can take place. The first problem that needs to be solved is the connection mechanism between the dispatching rules within a genetic program. 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. Fig.2 portrays the operation of PROGN function.




Equivalent Code

schedule a job according to EDD rule

then schedule a job according to SPT rule

**Fig.2** Operation of the PROGN function

In our experimentation we employed two variations of the PROGN function, the PROGN2 and PROGN3 functions which require two and three arguments respectively. While PROGN2 function is sufficient for small sized problems PROGN3 enhances the performance of the algorithm in large-sized problems. Tests were also made with PROGN4 and PROGN5 functions without any significant change on the results. As a result, the function set of our algorithm contains the two connecting functions PROGN2 and PROGN3.

The terminal set, as expected, is comprised of a number of dispatching rules that can be considered as good building blocks for the combinations. Three rules were selected to take part in this terminal set. The first one is the Earliest Due Date rule (EDD) which sequences jobs in non-decreasing order of their due date. The second one is 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 can be completed on time, while the EDD rule schedules optimally when at most one job in the problem is tardy. More general cases for the optimality of EDD and SPT scheduling are given by Emmons [21]. Based on these theorems we can expect the SPT rule to perform better at problems with high levels of tardiness, and the EDD rule to be ideal for the inverse case. The last rule used in our terminal set was originally introduced by Montagne [65] for the solution of the weighted total tardiness problem. This rule sequences jobs in non-decreasing order of the following ratio:

$$\frac{p_i}{w_i} * \frac{1}{\left(1 - d_i \Big/ \sum_{i=1}^{n} p_i\right)}$$

where $w_i$ is the associated penalty for job $i$. By setting all weights to one, we obtain the ratio used for the unweighted version of the problem:

$$\frac{p_i}{\sum_{i=1}^{n} p_i - d_i}$$

(the summation term that is missing in the numerator of the ratio, has no effect on the operation of the rule). We can say that Montagne's rule (MON), is a problem specific dispatching rule since its design has been based on the knowledge of the problem. 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.

Two more dispatching rules were initially included in the terminal set, but their presence did not enhance the evolutionary procedure making the terminal set superfluous. These rules were the Minimum Slack Time rule (MST) which schedules jobs in non-decreasing order of their slack time ($s_i=d_i-p_i$) and the Longest Processing Time rule which schedules jobs in decreasing order of their processing times.

*4.1.3. Modifications from the basic GP-algorithm*

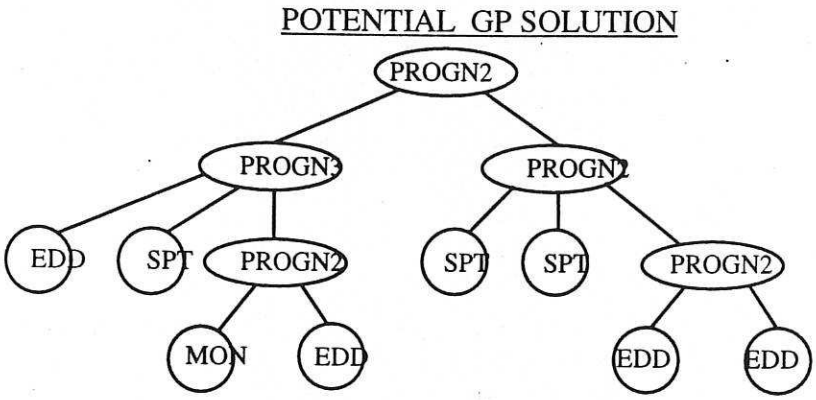
The GP-heuristic mainly has to deal with the issue of the fixed size of the problems. In traditional EAs the size of the chromosomes is fixed throughout the evolutionary procedure, since the application of operators produces offspring of the same size as the parents. In fact, fixed-length chromosomes are considered to be a major drawback on the application of EAs to other optimisation problems. Conversely, variable length size of programs is a very positive characteristic of GP. However in this case, where the total number of jobs is predefined, the variable length of programs is rather a disadvantage. In our implementation we've chosen not to constraint the evolutionary procedure by explicitly creating programs with a predefined number of terminals. Instead, the creation of programs and the application of genetic operators are the same as in any other traditional GP implementation. There are however some differences in the interpretation of the programs and the application of operators:

- During the evaluation of the programs, only the first $n$ terminals are considered to be active. After the consideration of these terminals all jobs have been scheduled, so the rest of the program is skipped.
- The application of operators is constrained within the limits of these terminals, i.e. within the *effective length* of the program (see Example 3 in Appendix for the illustration of the concept of effective length).
- If during the initialisation phase of the algorithm or during the application of operators an individual is created with number of terminals less than $n$, then it is penalised with a very high value of tardiness. Normally, the mean complexity of genetic programs after the first generation is much higher than the value of $n$ for the problem considered.

These modifications force the program to operate as a fixed-length string within a GP-framework. An example of the operation of a program of combined dispatching rules is given in Fig.4. The data of the problem can be found in Table1 (Baker [42] ).

| **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 1.** Data of the test problem




| | |
|---|---|
| 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 |

**Fig.3** Illustration of the genetic program operation on a test problem

For the terminal set used, 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. This is because even if a combination of dispatching rules does not produce a better result, the algorithm should be able to create an individual that is comprised only of terminals of the best dispatching rule. The problem described in Table 1 will help us to illustrate this concept. Table 2A in Appendix presents the results of all possible combinations of five dispatching rules on this problem. By starting from the first row of the table where the individual performance of each dispatching rule is given, we can see that the combination of two or more dispatching rules produces either an equal or better result than these rules individually or smaller combinations of them. However, when $n$ becomes big ($n \geq 25$), it becomes very difficult for the algorithm to always find the best combination of dispatching rules in the enormous search space. Results on individual test problems will later confirm this statement.

*4.1.4 Genetic operators*

Crossover and mutation operators work in the way already described in section 3, with the additional constraint that the cut-off points must be within the effective length of programs. In addition we utilised a point-mutation operator, which within the context of this problem, operates as a traditional EA mutation operator: A terminal point of a tree is selected randomly, and is 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 combination was a mixture of tree-mutation and point-mutation with probabilities of 0.8 and 0.2 respectively. Configurations involving the crossover operator performed slightly worse than the proposed setting. In addition, because of the effective length constraint the application of subtree crossover in small-sized problems ($n<50$) lead the system to the core very quickly due to the crossover maximum depth constraint. The performance of crossover was also worse for large values of $n$. However, this is a modified version of GP and there is nothing to suggest a generalisation of this statement. We can summarise our GP-implementation in the form of a Koza-tableau:

| Parameters | Values |
|---|---|
| Objective: | Evolve a combination of disp. 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 |

**Table 2.** Koza Tableau

*4.1.5 Why use GP?*

It can be argued that since individual programs are interpreted as fixed-size strings, there is no point in employing a GP-framework for the solution of the problem. It is obvious that a similar EA can be built for the same purpose. However, the aim of our research is to investigate ways in which GP can be used for the solution of sequencing problems. The interpretation of a genetic program as a structure similar to a string chromosome is one of them. In addition, the algorithm proposed in this section does not operate in exactly the same way as a traditional EA for sequencing problems, since the application of the tree-mutation operator is not the same with the application of various crossover operators used in these algorithms (OX, PMX, edge recombination etc.). Finally, the performance of the GP-heuristic is tested against the leading methods for the problem considered, thus the quality of the method can be assessed irrespectively of the existence of a similar EA. In any case, it is a matter of some importance that a GP-framework is able to handle permutation problems without major modifications in the actual operation of the algorithm.

### 4.2 Experimental phase

The experimental basis that we employed in this report is the same that has been used for many years by most researchers investigating the one-machine total tardiness problem. A number of problems were generated randomly for particular settings of $n$. The processing times for each job are drawn out of the uniform distribution [1..100]. Some researchers restrict the possible values to the space [1..10], but it has been argued that the former case introduces more difficult problems. Due dates are drawn out of a uniform distribution which is 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, we expect 20% of jobs 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 3 describes the combinations of R and T that were used in our 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 3.** Configuration settings for the test problems

Problems with T=1.0 were not included in our experimental set-up since the levels of tardiness in this case are unacceptable in practice. Four other combinations of T and R were not included because EDD scheduling almost always finds the optimal value in these cases. The set of combinations presented on Table 3 was tested on four different settings of $n$ (12,25,50,100). In total, our GP heuristic was tested on 320 problems, a fairly representative set of benchmark problems. The following methods were also applied in the same set of problems:

- The optimal values were generated using Potts & Van Wassenhove's algorithm, which has already been described in Section 2.
- As a comparison heuristic we employed the modified version of Holsenback & Russel's NBR heuristic, which has shown outstanding performance in published results [30].
- A Simulated Annealing algorithm was additionally designed and tested on the same benchmark problems. The implementation of the SA algorithm was based on two previous successful approaches of Potts & Van Wassenhove [39] and Ben-Daya & Al-Fawzan [40]. Table 1A of the Appendix describes the characteristics of the SA approach that was used in this section. The search strategy of the algorithm is systematic in the form of adjacent pairwise interchanges, but temperature is decreased according to the schedule proposed by Ben-Daya & Al-Fawzan.
- The individual dispatching rules EDD, SPT and MON.

Three versions of the GP-heuristic were tested during the experimental phase.:

- The simple version, which is the algorithm as it has already been described (called GPC from now on)
- A local search method which utilises the best individual evolved by GPC as the starting sequence of an adjacent pairwise procedure (GLS)
- The Simulated Annealing algorithm described previously which utilises the best individual evolved from a 5-generation GPC run as its starting sequence.

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 [66] - [67].
- 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. A combination evolved by GPC should in principle provide a better starting point than individual dispatching rules for a local search procedure.

### 4.3 Results

*4.3.1 GPC vs. other dispatching rules*

Our main hypothesis for the introduction of combinations of dispatching rules was that they would perform better than individual dispatching rules. Tables 5-8 support this hypothesis by highlighting the superiority of GPC in a variety of cases. Table 4 explains in detail the statistical terms used in the tables (all algorithms tested on a Pentium-based system, 133MHz)

| 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 4.** List of statistical terms

| | 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 5.** GPC vs. EDD,SPT & 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 6.** GPC vs. EDD,SPT & 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 7.** GPC vs. EDD,SPT & 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 8.** GPC vs. EDD,SPT & MON ($n$=100)

In comparison with GPC the difference of EDD in terms of MADO starts from 1314% for $n$=12, falls to 671% for $n$=25, and further decreases to 431% and 339% for $n$=50 and $n$=100 respectively. SPT performs poorly as well, especially in problems with small levels of tardiness, as it is expected from the theory. 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 are 284% higher than GPC for $n$=12, 212% higher for $n$=25 and 175% higher for $n$=50. The difference in penalties drops significantly for $n$=100 (only 35% higher), showing that as the problem increases and the algorithm struggles to find a suboptimal solution in the enormous search space ($3^{100}$) the MON rule becomes dominant in the combination of dispatching rules of the GPC solutions. In terms of optimal values GPC has a significant advantage only in very small problem instances. When $n \geq 25$, GPC fails to find more optimal solutions than individual dispatching rules. EDD outperforms 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 9 illustrates that when GPC is compared with the cumulative performance of the three individual dispatching rules in terms of 'non-dominated' solutions, it outperforms 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 9.** GPC vs. dispatching rules in terms of non-dominated solutions

While it is obvious that an evolved combination of dispatching rules will generally produce a better schedule than the same rules alone, a significant computational overhead is introduced by GPC at the same time. The evolution of a combination rule requires from 39.5 seconds of CPU time (worst case) for $n$=12, to 235.63 seconds for $n$=100 (worst case). Taking in account that GPC schedules are generally sub-optimal for moderate and large problem instances, using GPC as a stand-alone optimisation procedure will not bring significant improvements.

*4.3.2 M-NBR vs. GLS, SA & GSA*

Since M-NBR is a well-tested and well-documented optimisation method for the one-machine total tardiness problem, it will serve as basis for the evaluation of the remaining methods introduced in this section. In addition, M-NBR is the only deterministic of the methods discussed here, i.e. its outcome is always the same on a given problem. GLS, SA and GSA are stochastic optimisers, thus independent runs on the same problem might yield different results.

| | OPTIMAL SOLUTIONS | MADO (UNITS) | MRDO (%) | MAX(RDO) (%) |
|---|---|---|---|---|
| M-NBR | 75 | 0.8 | 0.12705 | 5.1136363 |
| SA | 68 | 2.325 | 0.735869 | 15.286624 |
| GLS | 63 | 2.425 | 0.246131 | 5.9431524 |
| GSA | 64 | 1.7125 | 0.509119 | 19.277108 |

**Table 10.** GPC vs. SA, GLS & GSA ($n$=12)

| | OPTIMAL SOLUTIONS | MADO (UNITS) | MRDO (%) | MAX(RDO) (%) |
|---|---|---|---|---|
| M-NBR | 61 | 9.75 | 0.352129 | 8.3456790 |
| SA | 60 | 3.1875 | 0.401984 | 9.7142857 |
| GLS | 25 | 22.375 | 1.201713 | 11.059907 |
| GSA | 57 | 5.1375 | 0.988265 | 29.761904 |

**Table 11.** GPC vs. SA, GLS & GSA ($n$=25)

| | OPTIMAL SOLUTIONS | MADO (UNITS) | MRDO (%) | MAX(RDO) (%) |
|---|---|---|---|---|
| M-NBR | 32 | 65.675 | 1.012977 | 9.2341356 |
| SA | 45 | 21.675 | 0.765335 | 25 |
| GLS | 9 | 78.575 | 1.682691 | 23.728813 |
| GSA | 53 | 9.3125 | 0.286414 | 4.2517006 |

**Table 12.** GPC vs. SA, GLS & GSA ($n$=50)

| | OPTIMAL SOLUTIONS | MADO (UNITS) | MRDO (%) | MAX(RDO) (%) |
|---|---|---|---|---|
| M-NBR | 13 | 494.8625 | 2.438429 | 19.846647 |
| SA | 36 | 86.525 | 0.298231 | 4.3756145 |
| GLS | 8 | 454.8125 | 3.543847 | 78.971962 |
| GSA | 42 | 67.8 | 0.571563 | 23.809523 |

**Table 13.** GPC vs. SA, GLS & GSA ($n$=100)

GLS performs significantly better than its parent method GPC (Tables 10-13). However, GLS cannot reach the level of M-NBR performance in small and moderate problem sizes. GLS imposes 203% higher penalties for $n$=12, 129% for $n$=25 and 19% for $n$=50. However, the performance of GLS seems to drop slower than M-NBR as the problem-size increases. For $n$=100 M-NBR has a slightly worse MADO than GLS. In addition while t-tests for $n$=12 and $n$=25 suggest 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 results 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 14 shows that M-NBR consistently finds better solutions than GLS over the whole range of problems. In addition, M-NBR produces a considerably larger number of optimal solutions especially for $n$=25 and $n$=50. The consideration of these data leads us to the conclusion that M-NBR is a better optimisation procedure than GLS for the one machine total tardiness problem. If computational efficiency is also considered, then the result will significantly favour M-NBR since it requires at worse ($n$=100) a few milliseconds of CPU time. GLS has similar computational requirements with GPC (the local search procedure is extremely fast).

| | 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 14.** M-NBR vs. GLS terms of non-dominated solutions

The superiority of M-NBR over GLS is not replicated with the other two stochastic optimisers. In the smallest problem size considered ($n$=12), M-NBR has the advantage since SA and GSA produced 190% and 114% higher penalties respectively. As the problem size increases, the performance of M-NBR drops significantly faster than SA and GSA. For $n$=25 M-NBR imposes 205% higher penalties than SA and 89% higher penalties than GSA on average. Tables 10-13 illustrate the increasing difference in performance for increasing values of $n$. In addition, t-test results on the null hypothesis for the means suggest a significant difference for $n$>50 (Table 14).

| | (M-NBR) - SA | | (M-NBR) - GSA | |
|---|---|---|---|---|
| $N$ | $t$ | $p(T \le t)$ | $t$ | $p(T \le t)$ |
| 12 | 1.95 | 0.027 | 1.26 | 0.105 |
| 25 | 1.89 | 0.03 | 1.25 | 0.106 |
| 50 | 4.21 | $3.27*10^{-5}$ | 5.03 | $1.48*10^{-6}$ |
| 100 | 6.35 | $6.18*10^{-9}$ | 6.34 | $6.44*10^{-9}$ |

**Table 15.** t-test results

We can therefore conclude with high confidence that the performance of SA and GSA is significantly different for large values of $n$. In addition, the data in tables 16 & 17 highlight the superiority of SA & GSA in moderate and large problem instances in terms of non-dominated solutions.

| | Number of times SA was better than M-NBR | Number of times SA was worse than M-NBR | Number of times SA was equal to M-NBR |
|---|---|---|---|
| $n$=12 | 3 | 7 | 70 |
| $n$=25 | 12 | 13 | 55 |
| $n$=50 | 35 | 10 | 35 |
| $n$=100 | 62 | 5 | 13 |

**Table 16.** M-NBR vs. SA 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 17.** M-NBR vs. GSA terms of non-dominated solutions

*4.3.2 SA vs.GSA*

From the results presented in the previous paragraphs, it can be concluded that SA and GSA had the best overall performance on the set of benchmark problems that we created. While their superiority from the other methods is obvious, a comparison between them does not bring any safe conclusions. Differences in terms of MADO are relatively small. SA seems to perform slightly better in small problem instances, while GSA has an advantage for large values of n. Pairwise t-tests suggest that we should not easily reject the testing hypothesis that the mean of the differences is zero (Table 18).

| | 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 18.** t-test results

| | 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 19.** SA vs. GSA in terms of non-dominated solution

The consideration of the pairwise performance of SA and GSA in terms of non-dominated solutions (Table 19) shows that the number of equal solutions remain close to 50% of the total number of problems even for $n$=100. This figure suggests that there is a high correlation on the performance of SA and GSA. In other words there is not enough evidence to suggest

that the introduction of a five-generation evolved GPC seed significantly improve the performance of SA.

# 5. Evolving scheduling policies using Genetic Programming

## 5.1 Introduction

In the previous section we employed a heuristic GP algorithm in an attempt to find near-optimal solutions for one-machine total tardiness problems. A combination of dispatching rules was employed as an indirect way of creating a permutation of jobs. An independent combination of rules was created for each problem considered. The procedure was relatively slow, while it failed to find quality solutions unless it was hybridised with other heuristics. In this section we broaden the scope of the algorithm by investigating the possibility of employing combinations of dispatching rules not as problem-specific optimisers but rather as general scheduling policies aiming to minimise the total tardiness of a series of problems just like individual dispatching rules operate.

## 5.2 Problem representation

The GP-algorithm employed in this application is essentially the same as the one described in the previous section. The major difference is the existence of multiple fitness cases in the form of a series of one-machine total tardiness problems with diverse parameter settings. The aim is to simultaneously minimise total tardiness over the whole range of the fitness cases. In other words we will investigate the potential existence of a combination of dispatching rules which could be employed as a scheduling policy and could be used instead of the repeated application of an individual dispatching rule. In the case of single problems, combinations of dispatching rules consistently outperform individual rules as we saw in the previous section. However, different combinations were evolved for different problems, so we can conclude that there was no sign of a generalisation of a combination on a top level. However, there is a possibility that a combination of dispatching rules might be able to generalise on a lower level of fitness and produce better overall results than individual dispatching rules. This possibility will be investigated in the following paragraphs.

## 5.3 Experimental set-up

The application of this section was tested on 20 different experimental set-ups. Details of the experimental basis are given in tables 20 & 21.

| *n* | SET-UPS | FITNESS CASES (PROBLEMS) PER SET-UP |
|---|---|---|
| 12 | 5 | 20 |
| 25 | 5 | 20 |
| 50 | 5 | 20 |
| 100 | 5 | 20 |

**Table 20.** Experimental set-ups

| | T=0.2 | T=0.4 | T=0.6 | T=0.8 |
|---|---|---|---|---|
| **R=0.2** | 1 problem | 1 problem | 1 problem | 1 problem |
| **R=0.4** | 1 problem | 1 problem | 1 problem | 1 problem |
| **R=0.6** | 1 problem | 1 problem | 1 problem | 1 problem |
| **R=0.8** | 1 problem | 1 problem | 1 problem | 1 problem |
| **R=1.0** | 1 problem | 1 problem | 1 problem | 1 problem |

**Table 21.** Fitness cases for individual set-ups

Note that due to the structure of this particular algorithm, GP cannot simultaneously evolve scheduling policies for problems of variable size, since the value of the effective length cannot change from problem to problem for a single program, otherwise no conclusion can be reached about the policy evolved. As it can be seen from table 20 each set-up incorporates problem over the entire range of T and R. This design was based on the assumption that a scheduling policy should be able to handle with the minimal possible cost any problem faced in a plant.

### 5.4 Results

Given a particular setting of n, ten runs of the GP framework were conducted on each of the five experimental set-ups. The best individual combination from these runs was then applied as a scheduling policy to the previously unseen set of problems taken from the rest of the set-ups (4×20=80 problems). In total, each scheduling policy was tested on 100 problems, 20 of them being the training set. The performance of individual dispatching rules on the same set of problems was also recorded. Tables 22-25 present detailed results both in terms of set-up specific as well as cumulative performance of the scheduling policies. The performance of the best-evolved policies on their training set-ups can be found on the diagonal of the matrices.

| | BEST POLICY EVOLVED BY SET-UP A | BEST POLICY EVOLVED BY SET-UP B | BEST POLICY EVOLVED BY SET-UP C | BEST POLICY EVOLVED BY SET-UP D | BEST POLICY EVOLVED BY SET-UP E | EDD | SPT | MON |
|---|---|---|---|---|---|---|---|---|
| **SETUP A** | 14617 | 14940 | 14968 | 15336 | 14880 | 18145 | 17137 | 14932 |
| **SETUP B** | 17935 | 17275 | 18105 | 19633 | 17417 | 22753 | 20536 | 17385 |
| **SETUP C** | 18043 | 18228 | 17838 | 19112 | 18089 | 21635 | 19421 | 18105 |
| **SETUP D** | 15428 | 15284 | 15801 | 14946 | 15304 | 16809 | 16685 | 15369 |
| **SETUP E** | 16826 | 16703 | 16859 | 17507 | 16556 | 18862 | 19474 | 16641 |
| **TOTAL** | 82849 | 82430 | 83571 | 86534 | 82246 | 98204 | 93253 | 82432 |

**Table 22.** Tardiness results for $n$=12

| | BEST POLICY EVOLVED BY SET-UP A | BEST POLICY EVOLVED BY SET-UP B | BEST POLICY EVOLVED BY SET-UP C | BEST POLICY EVOLVED BY SET-UP D | BEST POLICY EVOLVED BY SET-UP E | EDD | SPT | MON |
|---|---|---|---|---|---|---|---|---|
| SETUP A | 59481 | 60695 | 60657 | 61009 | 61467 | 77253 | 69698 | 60657 |
| SETUP B | 68054 | 65170 | 68331 | 65561 | 67586 | 88854 | 76709 | 65394 |
| SETUP C | 68507 | 69119 | 67768 | 69375 | 68860 | 85703 | 79834 | 69662 |
| SETUP D | 59627 | 57468 | 60055 | 57294 | 60745 | 77910 | 68893 | 57715 |
| SETUP E | 64538 | 63922 | 63424 | 63768 | 62527 | 75665 | 72286 | 63357 |
| TOTAL | 320207 | 316374 | 320235 | 317007 | 321185 | 405385 | 367420 | 316785 |

**Table 23.** Tardiness results for $n$=25

| | BEST POLICY EVOLVED BY SET-UP A | BEST POLICY EVOLVED BY SET-UP B | BEST POLICY EVOLVED BY SET-UP C | BEST POLICY EVOLVED BY SET-UP D | BEST POLICY EVOLVED BY SET-UP E | EDD | SPT | MON |
|---|---|---|---|---|---|---|---|---|
| SETUP A | 220882 | 222690 | 222651 | 222387 | 222446 | 286042 | 264225 | 226585 |
| SETUP B | 226795 | 225060 | 226281 | 227230 | 228698 | 290203 | 264877 | 226475 |
| SETUP C | 219993 | 219540 | 216918 | 220469 | 220797 | 280275 | 261468 | 221061 |
| SETUP D | 235313 | 234811 | 234308 | 231827 | 234113 | 303219 | 294412 | 237238 |
| SETUP E | 225370 | 225211 | 224611 | 225229 | 222058 | 298459 | 282257 | 226997 |
| TOTAL | 1128353 | 1127312 | 1124769 | 1127142 | 1128112 | 1458198 | 1367239 | 1138356 |

**Table 24.** Tardiness results for $n$=50

| | BEST POLICY EVOLVED BY SET-UP A | BEST POLICY EVOLVED BY SET-UP B | BEST POLICY EVOLVED BY SET-UP C | BEST POLICY EVOLVED BY SET-UP D | BEST POLICY EVOLVED BY SET-UP E | EDD | SPT | MON |
|---|---|---|---|---|---|---|---|---|
| SETUP A | 889161 | 889905 | 893794 | 887929 | 891996 | 1158371 | 1093840 | 886600 |
| SETUP B | 920573 | 913516 | 919948 | 918594 | 920437 | 1259142 | 1088402 | 923794 |
| SETUP C | 887440 | 885937 | 882899 | 888151 | 888705 | 1174196 | 1050774 | 889988 |
| SETUP D | 890247 | 888301 | 886158 | 880183 | 889014 | 1134456 | 1060882 | 886146 |
| SETUP E | 863899 | 860739 | 864885 | 860602 | 860336 | 1162836 | 1050688 | 870589 |
| TOTAL | 4451320 | 4438398 | 4447684 | 4435459 | 4450488 | 5889001 | 5344586 | 4457117 |

**Table 25.** Tardiness results for $n$=100

Starting with $n$=12, the scheduling policy with the best overall performance (B) produces similar tardiness levels with MON. Table 3A in Appendix illustrates the reason for this similarity. Most of the spots in this 12-step policy are occupied by the MON rule except for the first and the $10^{th}$ decision points (note that the last decision point is the $11^{th}$ since there is only one job left to be scheduled independently of the dispatching rule that occupies the $12^{th}$ decision point.). Similar scheduling policies were created by the rest of the set-ups, which performed slightly worse overall. Our best policy generally found better solutions than MON (Table 26), but the differences in terms of MADO and optimal values were insignificant

(Table 27). The pairwise t-test confirms that there is not much to separate between these two policies ($t=1.61$, $p<0.05$).

| | Number of times Best Policy was better | Number of times Best Policy was worse | Number of times Best Policy was equal |
|---|---|---|---|
| **EDD** | 60 | 37 | 3 |
| **SPT** | 73 | 25 | 2 |
| **MON** | 31 | 8 | 61 |

**Table 26.** Performance of Best Policy in terms of non-dominated solutions ($n$=12)

| | OPT | EDD | SPT | MON | BEST POLICY |
|---|---|---|---|---|---|
| TOTAL TARDINES (UNITS) | 69398 | 97896 | 93208 | 82432 | 82246 |
| MADO (UNITS) | | 284.98 | 238.1 | 130.34 | 128.48 |
| OPTIMAL SOLUTIONS | | 28 | 1 | 5 | 4 |

**Table 27.** Comparative performance of best policy for $n$=12

As expected, the performance of the EDD and SPT rules is overall much worse, since these rules are ideal for particular settings of T and R. The suitability of EDD for problems with low levels of tardiness is highlighted by the increased number of optimal solutions recorded for this rule. However, this is not a significant statistic for the problem considered, since we are mainly interested in minimising the cumulative tardiness of all problems. A high number of optimal values do not necessarily minimise total tardiness as it can be seen from the results in the tables.

The experimentation with the training sets for $n$=25 produced similar results. The main body of the best policy evolved is again constructed by the MON rule, with SPT and EDD rules occupying some places in the beginning and the end of the policy (Table 4A in Appendix). The performance of the best policy evolved and MON is quite similar, as depicted by the t-test on the penalties ($t=0.58$, $p<0.28$). However, it is interesting to note that while the cumulative tardiness produced by both policies is almost identical (Table 29), their performance on individual problems seems to differentiate, since the actual number of problems where both rules had the same performance drops to 17 (Table 28). In any case, the backbone of the best scheduling policy evolved for $n$=25 is the MON dispatching rule.

| | Number of times Best Policy was better | Number of times Best Policy was worse | Number of times Best Policy was equal |
|---|---|---|---|
| **EDD** | 59 | 38 | 3 |
| **SPT** | 88 | 11 | 1 |
| **MON** | 47 | 36 | 17 |

**Table 28.** Performance of Best Policy in terms of non-dominated solutions ($n$=25)

| | OPT | EDD | SPT | MON | BEST POLICY |
|---|---|---|---|---|---|
| TOTAL TARDINES (UNITS) | 262567 | 405655 | 367420 | 316803 | 316314 |
| MADO (UNITS) | | 1430.88 | 1048.53 | 542.36 | 537.47 |
| OPTIMAL SOLUTIONS | | 20 | 0 | 4 | 4 |

**Table 29.** Comparative performance of Best Policy for $n$=25

MON continues to hold the majority of positions in the best policies evolved for $n$=50 and $n$=100 (Tables 5A & 6A in Appendix). However, as $n$ becomes greater the search space expands rapidly ($3^n$), thus the outcome of the evolutionary procedure should not be considered as reliable as for the smaller cases. There is a high probability that a better scheduling policy exists somewhere else in the search space where the algorithm could not reach. The t-tests on the penalties suggest that there is a similarity between the best policy and the MON rule ($t=1.51$, $p<0.06$, for $n$=50 and $t=0.86$,$p<0.19$ for $n$=100), but the number of different individual solutions increases as $n$ becomes larger (Tables 30 & 32). This is perhaps an indication that the presence of EDD and SPT rules in some parts of the best policies fine-tune MON schedules to fit particular problems, forcing them at the same time to perform worse on some other problems. These two effects cancel each other in terms of cumulative tardiness (Tables 31 & 33), making the distinction between MON and the best policies evolved almost impossible.

| | Number of times Best Policy was better | Number of times Best Policy was worse | Number of times Best Policy was equal |
|---|---|---|---|
| **EDD** | 65 | 31 | 4 |
| **SPT** | 84 | 16 | 0 |
| **MON** | 62 | 27 | 11 |

**Table 30.** Performance of Best Policy in terms of non-dominated solutions ($n$=50)

| | OPT | EDD | SPT | MON | BEST POLICY |
|---|---|---|---|---|---|
| TOTAL TARDINES (UNITS) | 921120 | 1458198 | 1366789 | 1138336 | 1124699 |
| MADO (UNITS) | | 5370.78 | 4456.69 | 2172.16 | 2035.79 |
| OPTIMAL SOLUTIONS | | 24 | 0 | 4 | 4 |

**Table 31.** Comparative performance of Best Policy for $n$=50

| | Number of times Best Policy was better | Number of times Best Policy was worse | Number of times Best Policy was equal |
|---|---|---|---|
| **EDD** | 65 | 30 | 5 |
| **SPT** | 89 | 11 | 0 |
| **MON** | 56 | 37 | 7 |

**Table 32.** Performance of Best Policy in terms of non-dominated solutions ($n$=100)

| | OPT | EDD | SPT | MON | BEST POLICY |
|---|---|---|---|---|---|
| TOTAL TARDINES (UNITS) | 3631628 | 5890023 | 5344588 | 4456135 | 4437271 |
| MADO (UNITS) | | 22583.95 | 17129.6 | 8245.07 | 8056.43 |
| OPTIMAL SOLUTION S | | 25 | 0 | 5 | 5 |

**Table 33.** Comparative performance of Best Policy for $n$=100

The conclusion made from the above results is that for the case of a plant where the values of T and R vary significantly for the scheduling problems considered, it is sensible to employ the MON rule as a scheduling policy when the objective is the minimisation of total tardiness. While it may be beneficial for some problems to utilise EDD and SPT rules as parts of the scheduling policies, the GP algorithm was not able to identify a pattern for this contribution among the four settings of $n$. It can be argued that the actual size of the problem is a factor that affects the scheduling procedure. We will examine this argument more closely in section 6 of this report. The significant variability in the parameters of the problems used as fitness cases, results in an increased number of local-optima. Given a particular setting of $n$ and a particular set-up, the algorithm was able to converge to the same solution in the majority of ten runs only for small values of $n$. Thus, as we stated earlier, there is an increased probability that there are better combinations of dispatching rules for larger values of $n$ than those GP was able to find.

The experimentation discussed in this section suffered from the significant drawback that different scheduling policies needed to be evolved for different settings of $n$. In the following section we will discuss a more straightforward GP approach which aims to evolve complete dispatching rules and employ them as scheduling policies for one-machine total tardiness problems of variable size and characteristics.

# 6. Evolving dispatching rules using Genetic Programming

## 6.1 Introduction

Results presented in the previous sections suggested that the dispatching rule introduced by Montagne (MON) produces a good overall performance in a variety of one-machine scheduling problems, due to its unique design which takes in account both the processing times 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 3) and create a ratio that defines the scheduling priority of each job. Montagne constructed this formula based on his understanding of the problem. However, we cannot rule out the possibility that there exists another formula – perhaps more complex – which utilises a priori knowledge of the problem in order to create better scheduling ratios. In this section we investigate the possibility of evolving a dispatching rule formula through a GP-framework for the solution of the one-machine total tardiness scheduling problem.

## 6.2 Design of the algorithm

The GP algorithm discussed in this section takes the traditional Koza's form, since we don't have to deal any more with unfeasibility problems and the concept of effective length. GP has been used for regression since the early stages of the field [51]. The main parameters that

need to be defined in the design of the algorithm are the mathematical functions that will form the function set and the terminals that will be handled by these functions:

Function Set: The function set is comprised of the four major mathematical operations addition, subtraction, multiplication and division (+, -, *, %). The '%' symbol corresponds to the protected division mathematical operation that returns the value of '1' when the value of the denominator is '0'.

Terminal Set: Three basic parameters employed by MON rule are included in the terminal set ($p_i$: processing time of job *i*, $d_i$: due date of job *i*, SP: sum of the processing time of all jobs in the problem or 'makespan' as it is widely known). The set is completed by two additional terminals, SD, which corresponds to the sum of the due dates of all jobs in the problem, and N, which corresponds to the total number of jobs in the problem. There is no a priori knowledge that would guarantee the suitability of the additional terminals for the evolution of an optimal formula. However, in the worst possible case, the GP algorithm should be able to converge to the formula of MON rule, since all its elements are included in the function and terminal sets.

The algorithm schedules jobs in non - decreasing order of the ratios produced by the formula of the evolved dispatching rule. Table 33 is the Koza tableau for the algorithm described in this section.

| 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 |
| Tree crossover probability: | .5 |
| Tree mutation 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 |

**Table 33.** Koza Tableau

Note that the algorithm employs the combination of subtree crossover and mutation that performed better in our experiments. The population size is again kept at a low level in comparison with what is traditionally proposed in GP-literature. Experiments with increased population size (up to 500 individuals) were not able to produce fitter individuals in terms of cumulative tardiness. In general, choosing the values of the parameters for a GP-run is not an easy task. Many researchers have reported quite different settings for different problems. It can be argued that while a stable mathematical basis for the operation of evolutionary

computation techniques has not been discovered, the whole procedure will continue to be more or less problem specific.

### 6.3 Experimental basis

The algorithm presented in this section was tested on nine different experimental set-ups. The first four of them were taken from the experimental basis of the previous section (SET-UP A for $n$=12, $n$=25, $n$=50 and $n$=100). These training sets allow us to evaluate the performance of the evolved dispatching rule when the value of $n$ in the training set is fixed. Five additional set-ups were used in the experimentation phase of the algorithm (Table 34). Each of these sets is comprised of twenty different problems, five for each value of n. Values of T and R were chosen randomly for each problem. Tables 7A-11A in Appendix present these variable set-ups in detail. In total 20 runs were conducted on each experimental set-up and results are reported in the following paragraph.

| 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 34.** Experimental set-ups

### 6.4 Results

The GP framework evolved nine different dispatching rules out of the nine set-ups. Individual and cumulative performance for each of these rules is illustrated in Table 35.

| | 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 | 2948488 |

**Table 35.** Tardiness results for all set ups

From this table we can conclude that in most cases the algorithm was able to evolve a dispatching rule that had a better overall performance than MON. Rules evolved from small fixed-size training sets were not able to generalise at all, while the rest of the training sets produced rules which performed quite well in a very large set of previously unseen problems (160 in total). Based on this observation we can safely say that these rules do not just fit the data of the fitness cases but they contain information that is relevant to the problem considered. However, the formulas of these dispatching rules are not as straightforward as the formula of the MON rule. Table 12A in Appendix presents the mathematical formulas of the nine rules evolved. These equations have been cleared from introns (segments of code which have no effect on the outcome of the problem) and have also been simplified wherever that was possible.

There is no certain pattern emerging from these rules in a form of a mathematical equation that relates the parameters of the problem in an optimal way. The GP algorithm had difficulties in converging to the same solution in each run of an individual experimental set-up. All the rules described in Table H in the Appendix occurred only once in twenty independent runs. As we discussed earlier this is due to the difficulty imposed on the algorithm by the training sets. The variety in problem parameters within the same experimental set-up creates a considerable number of local-optima. The change of a simple sign in the formula of an evolved rule produces a significantly different result on the tardiness of each problem. However, the rules evolved all other runs were not significantly worse in terms of total tardiness. In order to compare the performance of a GP-evolved rule with all the traditional dispatching rules used in this report, we've chosen the rule that was evolved by the experimental set-up SETVAR3, which produced the best overall performance:

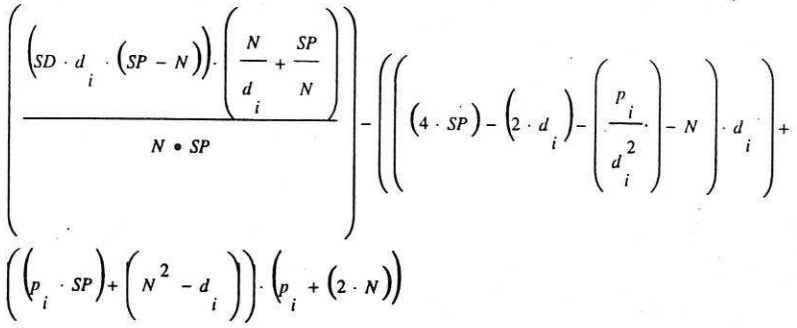


**Fig.4** Dispatching rule evolved from set-up SETVAR3

This particular rule is constructed from three main terms. The first and the third term operate more or less in favour of EDD and SPT scheduling respectively. The second term seems to act as a control segment that shifts the operation of the rule towards EDD or SPT scheduling according to the values of the parameters of the problem. For example, when the due date of a job is small in comparison with the makespan, the second term produces a significant negative result, which decreases the value of the ratio and therefore assigns urgency to the job. In the inverse case the value of the term becomes less significant thus the two big positive terms control the ratio. It is also obvious that the significance of both due date and processing time of a job is a complex function of N, SP and SD. It seems that the algorithm is able to construct these relations successfully since the generalisation of the rule is quite good. However, we cannot trace exactly the same functions in other evolved dispatching rules, thus it is possible that the algorithm is rather efficiently using all the numerical resources to interrelate p*i* and d*i* in an optimal way.

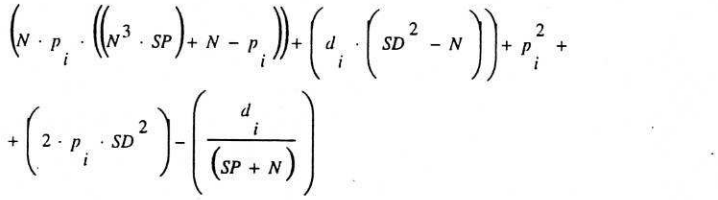


**Fig.5** Dispatching rule evolved from set-up SETVAR2

The dispatching rule evolved using the experimental set-up SETVAR2 performs almost identical to the previous rule (note that the constant terms were kept in the formula as they were evolved by GP. These terms can be ignored, since they have no effect on the scheduling of jobs). However, while there are terms with similar operation between the two rules (one favouring EDD scheduling and one favouring SPT scheduling), there are no other easily observable similarities. In the case of SETVAR2 it is interesting to note the control nature of the SD value in the second term. When the sum of due dates is large in comparison with the sum of the processing times, it is quite likely that that the scheduling problem is not too tardy, thus EDD scheduling is favoured. In the inverse case, the first term becomes more significant, and SPT scheduling is favoured. In any case, the relations produced by these dispatching rules are far more complicated than our simple analysis. This is not a surprise since the evolved rules were trained to cope with scheduling problems of quite diverse nature.

In table 36 we compare the performance of the dispatching rule SETVAR3 with those of the classic EDD, SPT and MON rules in all the experimental set-ups.

| | 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 36.** Comparative performance of SETVAR1 for all experimental set-ups

In contrast with what we saw in section 5, the improvement in overall performance is significant. MON imposes 81% higher penalties in terms of MADO, while the t-test on the penalties rejects the null hypothesis with a very high probability ($t=5.62$, $p<3.49\times10^{-8}$). In addition, SETVAR3 consistently outperforms all the other rules in terms of non-dominated solutions. In all cases, at least 77% of the solutions produced by SETVAR1 were better or equal than those produced by the other rules (Table 37).

| | Number of times SETVAR1 was better | Number of times SETVAR1 was worse | Number of times SETVAR1 was equal |
|---|---|---|---|
| **EDD** | 115 | 40 | 25 |
| **SPT** | 164 | 8 | 8 |
| **MON** | 147 | 30 | 3 |

**Table 37.** Performance of SETVAR1 of non-dominated solutions (all set-ups)

EDD still performs well in the set of problems identified by small levels of tardiness and not too tight due dates. However, when the scheduling problems in the plant are evenly distributed in terms of T and R, EDD scheduling produces the worse performance over the available dispatching rules.

## 7. Conclusions

In this report we investigated the potential use of Genetic Programming for the solution of the one-machine total tardiness problem, which has been the subject of academic research for almost four decades. To our knowledge, no previous effort has been made to solve static scheduling problems in a GP-framework, in contrast with other evolutionary computation techniques that have been extensively used for this scope. It is difficult to evolve a permutation representation without producing infeasible solutions when subtree crossover and mutation are utilised. Three different ways of approaching the problem were tested during the experimental phase. 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 with considering only the part of the program that had significance on the solution of the problem. This configuration was employed as a heuristic procedure for the solution of individual problems. While it outperformed dispatching rules, near optimal solutions were only reached when other heuristic procedures like local search and Simulated Annealing 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 the possible existence of a combination of dispatching rules that produces good overall performance was investigated. The same GP-framework was utilised but the algorithm was trained on experimental set-ups

comprised of a number of problems with significant differences in their parameters. Four combinations of dispatching rules were evolved as scheduling policies, each one for a particular value of $n$. Their performance was largely dominated by the presence of the MON rule, which is a problem-specific dispatching rule. However, there was an indication that for large-sized problems an evolved combination might produce a better overall performance than the application of the MON rule by itself. Finally, a traditional GP-framework was employed as a basis for evolving a formula of a dispatching rule for the one machine total tardiness problem. Nine dispatching rules were evolved during the experimental phase of the algorithm. A number of these rules showed an outstanding level of generalisation while clearly outperforming MON rule in all aspects. We can summarise the conclusions of our investigation in the form of the following points:

- The evolution of a direct permutation representation in a GP-framework is not straightforward, however, an indirect representation can be sometimes constructed based on the problem considered
- Combinations of dispatching rules perform better or equal than individual dispatching rules in the majority of cases for the one-machine total tardiness problem
- The hybridisation of GP or evolutionary algorithms, with other local search techniques yields significant improvements in the results
- While it is possible to employ GP as a heuristic procedure, the algorithm is quite slow in comparison with other heuristic methods for the one-machine total tardiness problem.
- It is sensible to employ the MON dispatching rule as a scheduling policy in a plant where tardiness problems of variable characteristics are encountered. Applying combinations of dispatching rules as scheduling policies may be beneficial in some cases, but overall the improvement is not significant
- Evolving scheduling policies as 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 us with significant insights to the nature of the problem.

## 8. Recommendations for future work

There is a variety of ways in which this research can be extended. All the previous methods can be tested for possible generalisations on the weighted version of the one-machine total tardiness problem, as well as on any other sequencing problem where dispatching rules can be applied. Additionally, dispatching rules can be evolved in the way introduced in section 6 for any scheduling problem where the parameters are available a priori. In the case of the job-shop scheduling problem the algorithm proposed in section 4 can be employed for the evolution of a combination of dispatching rules in the way Dorndorf & Pesch [64] illustrated in their paper.

GP as well as other evolutionary algorithms are extremely parameter sensitive, especially when difficult experimental set-ups like the ones used in sections 5 and 6 are employed. A meta-level GA or any other form of optimally controlling the parameters of the run could significantly improve the performance of the algorithm.

Finally, the existence of interesting similarities within the formulas of dispatching rules can be investigated by using any form of code-reutilization technique like ADFs.

## REFERENCES


[1] Dimopoulos, C. and Zalzala, A.M.S., "Evolutionary computation for manufacturing optimisation: Recent developments", Research Report no.716, Dept.of Automatic Control & Systems Engineering, University of Sheffield,1998.

[2] Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P., "Optimisation by Simulated Annealing", *Science*, **220**, pp.671-679, 1985.

[3] Glover, F., "Tabu Search: a tutorial", *Interfaces*, vol.20, no.3, pp. 74-94, 1990.

[4] Holland, J.H, *Adaptation in Natural and Artificial Systems*.Ann Arbor, Univ.of Michigan Press, MI, 1975.

[5] Rechenberg, I., Evolutionsstrategie: *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, Germany, 1973.

[6] Fogel, L.J., Owens, A.J. and Walsch, M.J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1996.

[7] Garces-Perez, J., Schoenefeld, D.A. and Wainwright, R.L., "Solving facility layout problems using genetic programming", in *Genetic Programming 1996: Proc.of the 1st Annual Conference*, Koza,Goldberg,Fogel & Riolo (Eds.), pp.182-190, MIT Press, 1996.

[8] Polheim, H. and Marenback, P., "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, 1996.

[9] McKay, B.M., Willis, M.J, Hiden, H.G., Montague, G.A. and Barton, G.W., "Identification of industrial processes using genetic programming", in *Proc.of the Conf.on Identification in Engineering Systems*, Friswell & Mottershead (Eds.), pp.510-519, Univ.of Wales, Swansea, UK. 1996.

[10] Potts, C.N. and Van Wassenhove, L.N., "A decomposition algorithm for the single machine total tardiness problem", *Operations Research Letters*, **1**(5), 177-181,1982.

[11] Lawer, E.L., "A'pseudopolynomial' algorithm for sequencing jobs to minimise total tardiness", *Annals of Discrete Mathematics*, **1**, 331-342, 1977.

[12] Lenstra, J.K., Rinnooy Kan, A.H.G and Lageweg B.J., "Complexity of machine scheduling problems", *Annals of Discrete Mathematics*, **1**,343-362, 1977.

[13] Du, J. and Leung, J.Y.-T., "Minimising total tardiness on one machine is NP-hard", *Mathematics of Operations Research*, **15**(3), 483-495, 1989.

[14] Bellman, R.E. and Dreyfus, S.E., *Applied Dynamic Programming*, Princeton University Press, 1962.

[15] French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood Limited, 1982.

[16] McNaughton, R., "Scheduling with deadlines and loss functions", *Management Science*, **6**(1), 1-12, 1959.

[17] Schild, A. and Fredman, I.J., "On scheduling tasks with associated linear loss functions", *Management Science*, **7**(3), 280-285 1961.

[18] Schild, A. and Fredman, I.J., "Scheduling tasks with deadlines and non-linear loss functions", *Management Science*, **9**(1), 73-81, 1962.

[19] Held, M. and Karp, R.M., "A dynamic programming approach to sequencing problems", *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, **10**(1), 196-210, 1962.

[20] Lawler, E.L., "On scheduling problems with deferral costs", *Management Science*, **11**(2), 280-288, 1964.

[21] Emmons, H., "One machine sequencing to minimise certain function of job tardiness", *Operations Research*, **17**(4), 701-715, 1968.

[22] Elmaghraby, S.E., "The one machine sequencing problem with delay costs", *Journal of Industrial Engineering*, **19**(2), 105-108, 1968.

[23] Srinivasan, V., A hybrid algorithm for the one machine sequencing problem to minimize total tardiness, *Naval Research Logistics Quarterly*, **18**(3), 317-327, 1971.

[24] Wilkerson, L.J. and Irwin J.D., "An improved algorithm for scheduling for independent tasks", *AIIE Transactions*, **3**(3), 239-245, 1971.

[25] Shwimer, J., "On the N-job, one-machine, sequence-independent scheduling problem with tardiness penalties: a branch & bound approach", *Management Science*, **18**(6), 301-313, 1972.

[26] Rinnooy Kan, A.H.G, Lageweg, B.J. and Lenstra, J.K., "Minimising total costs in one-machine scheduling", *Operations Research*, **23**(5), 908-927, 1975.

[27] Fisher, M.L., "A dual algorithm for the one-machine scheduling problem", *Mathematical Programming*, **11**(3), 229-251, 1976.

[28] Baker, K.R. and Schrage, L.E., "Finding an optimal sequence by dynamic programming: an extension to precedence-related tasks", *Operations Research*, **26**(1), 111-120, 1978.

[29] Schrage, L.E. and Baker, K.R., "Dynamic programming solution of sequencing problems with precedence constraints", *Operations Research*, **26**(3), 444-449, 1978.

[30] Picard, J and Queyranne, M., "The time-dependent travelling salesman problem and its application to the tardiness problem in one-machine scheduling", *Operations Research*, **26**(1), 86-110, 1978.

[31] Potts, C.N. and Van Wassenhove, L.N., "A branch & bound algorithm for the total weighted tardiness problem", *Operations Research*, **33**, 363-377, 1985.

[32] Sen, T., Austin, L.M. and Ghandforoush, P., "An algorithm for the single machine sequencing problem to minimize total tardiness", *AIIE Transactions*, **15**, 363-366, 1983.

[33] Fry, T.D., Vicens, L., Macleod, K. and Fernadez, S., "A heuristic solution procedure to minimize total tardiness", *Journal of the Operational Research Society*, **40**, 293-297, 1989.

[34] Holsenback, J.E. and Russel, R.M., "A heuristic algorithm for sequencing on one machine to minimize total tardiness", *Journal of the Operational Research Society*, **43**, 53-62, 1992.

[35] Panwalkar, S.S., Smith, M.L. and Koulamas, C.P., "A heuristic for the single machine tardiness problem", *European Journal of Operational Reserarch*, **70**, 304-310, 1993.

[36] Russel, R.M. and Holsenback, J.E., "Evaluation of leading heuristics for the single machine tardiness problem", *European Journal of Operational Reserarch*, **96**, 304-310, 1996.

[37] Holsenback, J.E. and Russel, R.M., "Evaluation of greedy, myopic and less-greedy heuristics for the single-machine, total tardiness problem", *Journal of the Operational Research Society*, **48**, 640-646, 1997.

[38] Matsuo, H., Suh, C.J. and Sullivan, R.S., "A controlled search simulated annealing method for the single machine weighted tardiness problem", *Annals of Operations Research*, **21**, 85-108, 1989.

[39] Potts, C.N. and Van Wassenhove, L.N., "Single machine tardiness sequencing heuristics", *IIE Transactions* **23**(4) 346-354, 1991.

[40] Ben-Daya, M. and Al-Fawzan, M., "A simulated annealing approach for the one-machine mean tardiness scheduling problem", *European Journal of Operational Research*, **93**, 61-67, 1996.

[41] Ibaraki, T. and Nakamura, Y., "A dynamic programming method for single machine scheduling", *European Journal of Operational Research*, **76**, 72-81, 1994.

[42] Tansel, B.C. and Sabuncuoglu, I., "New insights on the single machine total tardiness problem", *Journal of the Operational Research Society*, **48**, 82-89, 1997.

[43] Baker, K.R., *Introduction to Sequencing and Scheduling*, John Wiley & Sons, 1974.

[44] Pinedo, M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, 1995.

[45] Gupta, S.K. and Kyparisis, J., "Single machine scheduling research", *OMEGA, International Journal of Management Science*, **15**(3), 207-227, 1987.

[46] Sen, T. and Gupta, K.S., "A state-of-art survey of static scheduling research involving due dates", *OMEGA, International Journal of Management Science*, **12**(1), 63-76, 1984.

[47] Baker, K.R. and Martin, J.B., "An experimental comparison of solution *algorithms for the single machine tardiness problems", Naval* Research Logistics Quarterly, **21**(1), 187-199, 1974.

[48] Van Wassenhove, L. and Gelders, L.F., "Four solution techniques for a general one machine scheduling problem", *European Journal of Operational Research*, **2**(4), 281-290, 1978.

[49] Cramer, N.L., "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, 1985.

[50] Fujiki, C. and Dickinson, J., "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, 1987.

[51] Hicklin, J.F., "Application of the genetic algorithm to automatic program generation", Master's Thesis, Dept. of Computer Science, University of Idaho, 1986.

[52] Koza, J.R, *Genetic Programming: On the programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[53] Koza, J.R, *Genetic Programming II: Automatic Discovery of reusable programs*, MIT Press, Cambridge, 1994.

[54] Nordin, P., "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, 1994.

[55] Teller A, and Veloso M., " PADO: A new learning architecture for object recognition, in *Visual Learnin*g, Ikeuchi & Veloso (Eds.), pp.81-116, Oxford University Press, Oxford, UK, 1996.

[56] Banzhaf, W., Nordin, P., Keller, R.E. and Francone F.D., *Genetic Programming: An Introduction*, Morgan Kaufman, San Francisco, CA, 1998.

[57] Haupt, R., "A survey of priority rule-based scheduling", *OR Spektrum*, **11**(1), pp.3-16, 1989.

[58] Blackstone, J.H, Philips, D.T and Hogg, C.L., "A state of the art survey of dispatching rules for manufacturing job shop operations", *Int.J.of Production Research*, **20**, pp.27-45, 1982.

[59] Caskey, K. and Storch, R.L., "Heterogeneous dispatching rules in job and flow shops", *Production Planning & Control*, **7**(4), pp.351-361, 1996.

[60] Fischer, H. and Thompson, G.L., "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, 1963.

[61] Herrmann, J.W., Lee, C.Y. and Hinchman, J., "Global job-shop scheduling with a genetic algorithm", *Production & Operations Management*, **4**(1), pp.30-45, 1995.

[62] Fujimoto, H., Lian-yi, C., Tanigawa, Y. and Iwahashi, K., "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, 1995.

[63] Fang, H., Ross, P. and Corne, D., "A promising hybrid GA/heuristic approach for open-shop scheduling problems", in *ECAI'94: 11th European Conf. on Artificial Intelligence. Proceedings*, A.G.Cohn (Ed.),pp.590-594, Chicester, UK, 1994.

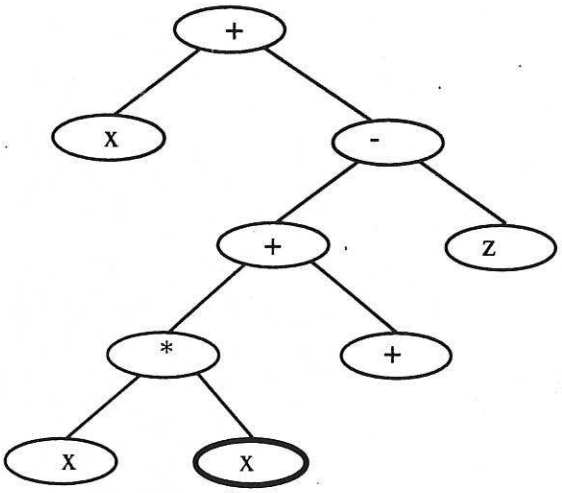
[64] Dorndorf, U. and Pesch, E., "Evolution-based learning in a job-shop scheduling environment", *Computers & Operations Research*, **22**(1), pp.177-181, 1995.

[65] Montagne, G.R., "Sequencing with time delay costs", *Industrial Engineering Research Bulletin*, **5**, Arizona State University, 1969.

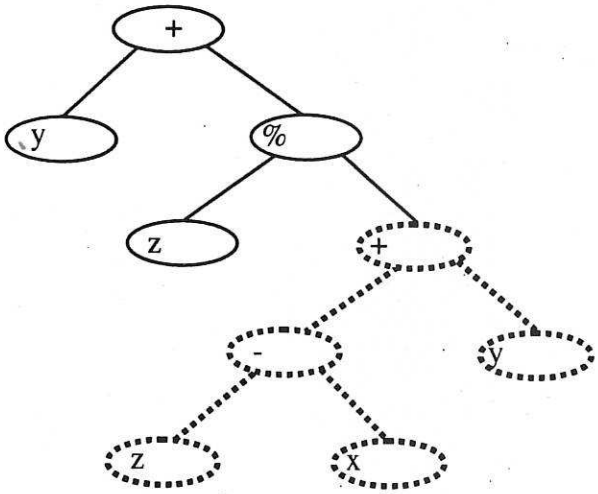
# APPENDIX

## Example 1: Crossover

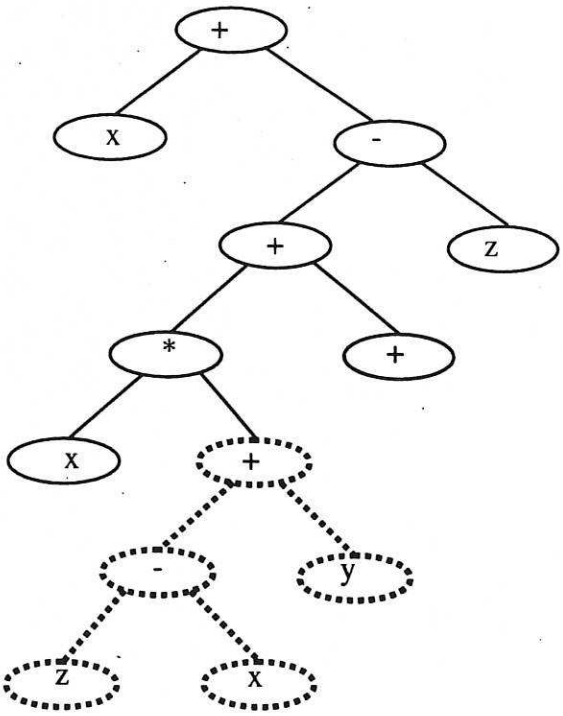
PARENT1



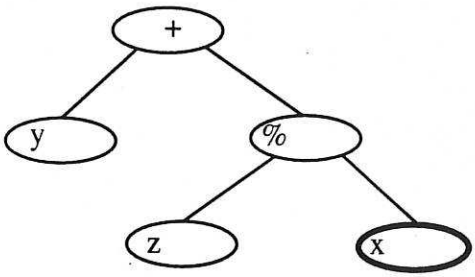

PARENT2




CHILD1

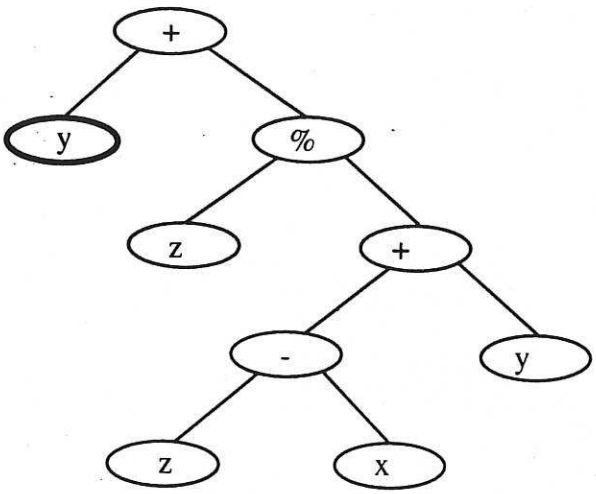



CHILD2

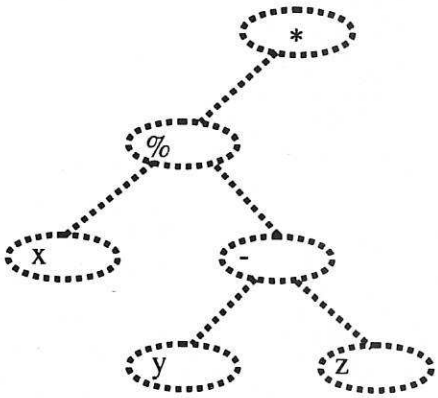


## Example 2: Mutation

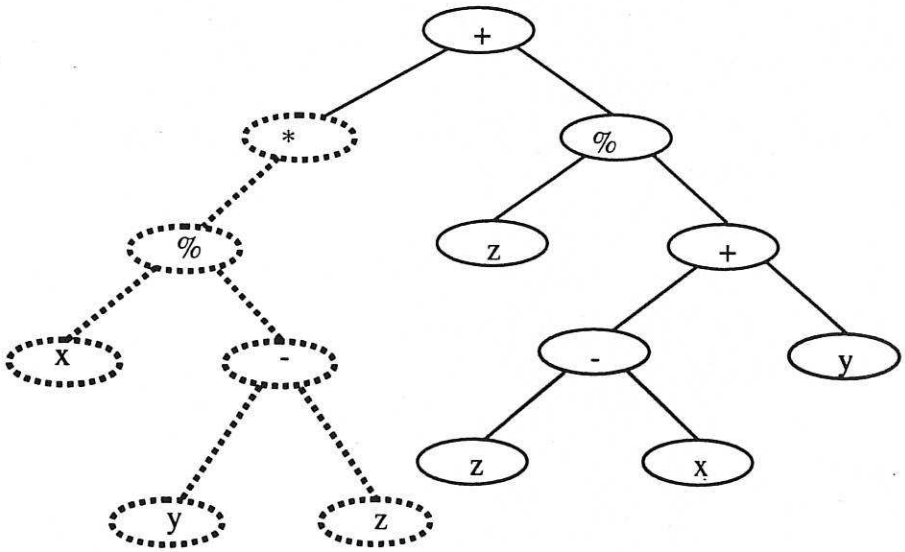
PARENT



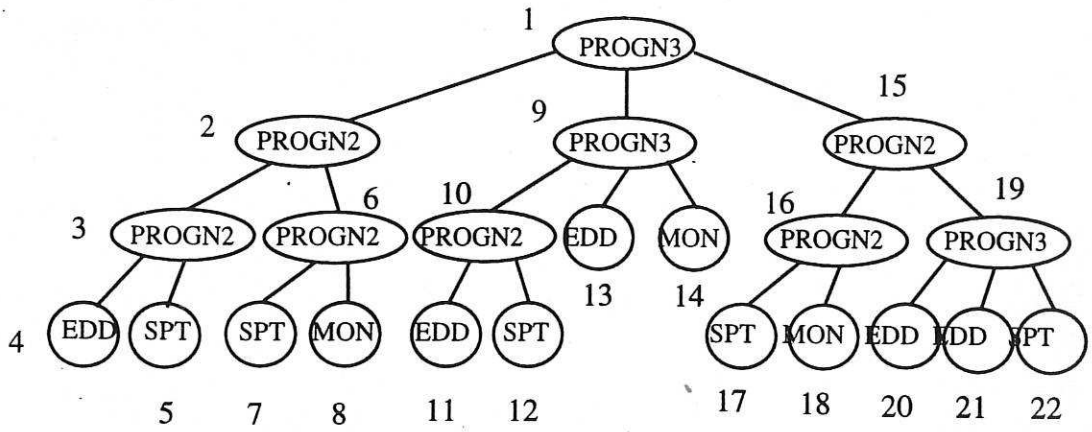

RANDOMLY CREATED SUBTREE




CHILD



## Example 3: Effective Length




The length of the genetic program in the previous diagram is 22. However, if the total number of jobs in the problem considered is 8, then the effective length of the program is 14, since it's within that length that the first $n$ terminal nodes are present. Following the same logic, if $n$=12 the effective length of the program is 21. In both cases chunks of code that follow the effective part are skipped during the evaluation phase of the algorithm.

| SIMULATED ANNEALING PARAMETERS | |
|---|---|
| Neighbourhood structure | All adjacent pairwise interchanges – restart from the best sequence found in the entire neighbourhood |
| Neighbourhood size | $\frac{1}{2} \cdot n \cdot (n-1)$ |
| Probabilty of acceptance | $P_a = e^{-(a \cdot \Delta_{Tard})}$ where : $a = 10 + (ITER \cdot 4)$ $ITER$ = Number of iterations |
| Termination criterion | When no better sequence has been found in an entire neighbourhood |

**Table 1A.** Simulated Annealing implementation

| | EDD | SPT | MST | LPT | MON |
|---|---|---|---|---|---|
| VALUE | **859** | **1296** | **917** | **1249** | **813** |

| | EDD-MON | SPT-MON | EDD-MST | SPT-MST | MON-LPT |
|---|---|---|---|---|---|
| BEST VALUE | **790** | **805** | **859** | **849** | **813** |
| TIMES FOUND | **20** | **20** | **20** | **20** | **20** |

| | MST-LPT | EDD-LPT | LPT-SPT | MON-MST | SPT-EDD |
|---|---|---|---|---|---|
| BEST VALUE | **917** | **859** | **858** | **813** | **840** |
| TIMES FOUND | **20** | **20** | **20** | **20** | **20** |

| | EDD-LPT-MON | LPT-MON-MST | EDD-MST-MON | EDD-LPT-MST | SPT-EDD-LPT |
|---|---|---|---|---|---|
| BEST VALUE | **790** | **813** | **790** | **859** | **840** |
| TIMES FOUND | **20** | **20** | **20** | **20** | **20** |

| | SPT-EDD-MON | SPT-EDD-MST | SPT-LPT-MON | SPT-MON-MST | SPT-LPT-MST |
|---|---|---|---|---|---|
| BEST VALUE | **782** | **840** | **805** | **805** | **849** |
| TIMES FOUND | **20** | **20** | **20** | **20** | **17** |

| | EDD-LPT-MST-MON | SPT-EDD-LPT-MST | SPT-EDD-LPT-MON | SPT-EDD-MST-MON | SPT-LPT-MST-MON |
|---|---|---|---|---|---|
| BEST VALUE | **790** | **840** | **782** | **782** | **805** |
| TIMES FOUND | **20** | **17** | **17** | **20** | **15** |

| | SPT-EDD-LPT-MST-MON |
|---|---|
| BEST VALUE | **782** |
| TIMES FOUND | **15** |

**OPTIMAL VALUE = 755**

**Table 2A.** Performance of the GP-heuristic system using different combinations of dispatching rules (20 runs)

| DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED |
|---|---|---|---|
| 1 | SPT | 7 | MON |
| 2 | MON | 8 | MON |
| 3 | MON | 9 | MON |
| 4 | MON | 10 | MON |
| 5 | MON | 11 | SPT |
| 6 | MON | 12 | EDD |

**Table 3A.** Best scheduling policy evolved for $n$=12

| DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED |
|---|---|---|---|
| 1 | MON | 14 | MON |
| 2 | SPT | 15 | MON |
| 3 | MON | 16 | MON |
| 4 | MON | 17 | MON |
| 5 | MON | 18 | MON |
| 6 | MON | 19 | MON |
| 7 | EDD | 20 | MON |
| 8 | MON | 21 | MON |
| 9 | MON | 22 | SPT |
| 10 | MON | 23 | EDD |
| 11 | MON | 24 | MON |
| 12 | MON | 25 | MON |
| 13 | MON | | |

**Table 4A.** Best scheduling policy evolved for $n$=25

| DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED |
|---|---|---|---|---|---|---|---|
| 1 | EDD | 14 | EDD | 27 | MON | 40 | MON |
| 2 | SPT | 15 | MON | 28 | MON | 41 | MON |
| 3 | SPT | 16 | EDD | 29 | MON | 42 | MON |
| 4 | EDD | 17 | EDD | 30 | MON | 43 | MON |
| 5 | MON | 18 | MON | 31 | EDD | 44 | MON |
| 6 | MON | 19 | MON | 32 | EDD | 45 | EDD |
| 7 | EDD | 20 | MON | 33 | MON | 46 | MON |
| 8 | EDD | 21 | MON | 34 | MON | 47 | MON |
| 9 | SPT | 22 | MON | 35 | MON | 48 | MON |
| 10 | MON | 23 | EDD | 36 | MON | 49 | MON |
| 11 | MON | 24 | MON | 37 | MON | 50 | MON |
| 12 | MON | 25 | MON | 38 | MON | | |
| 13 | MON | 26 | MON | 39 | MON | | |

**Table 5A.** Best scheduling policy evolved for $n$=50

| DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED | DECISION POINT NO. | RULE EMPLOYED |
|---|---|---|---|---|---|---|---|
| 1 | EDD | 26 | EDD | 51 | MON | 76 | MON |
| 2 | MON | 27 | MON | 52 | EDD | 77 | MON |
| 3 | SPT | 28 | MON | 53 | MON | 78 | MON |
| 4 | MON | 29 | MON | 54 | MON | 79 | MON |
| 5 | MON | 30 | MON | 55 | EDD | 80 | MON |
| 6 | EDD | 31 | MON | 56 | MON | 81 | SPT |
| 7 | MON | 32 | EDD | 57 | MON | 82 | EDD |
| 8 | SPT | 33 | MON | 58 | MON | 83 | EDD |
| 9 | SPT | 34 | MON | 59 | MON | 84 | MON |
| 10 | MON | 35 | MON | 60 | MON | 85 | MON |
| 11 | MON | 36 | MON | 61 | MON | 86 | MON |
| 12 | MON | 37 | MON | 62 | MON | 87 | MON |
| 13 | EDD | 38 | MON | 63 | EDD | 88 | SPT |
| 14 | SPT | 39 | MON | 64 | MON | 89 | EDD |
| 15 | MON | 40 | MON | 65 | MON | 90 | EDD |
| 16 | MON | 41 | MON | 66 | MON | 91 | MON |
| 17 | EDD | 42 | SPT | 67 | MON | 92 | MON |
| 18 | MON | 43 | MON | 68 | EDD | 93 | MON |
| 19 | EDD | 44 | MON | 69 | MON | 94 | SPT |
| 20 | EDD | 45 | MON | 70 | MON | 95 | MON |
| 21 | EDD | 46 | MON | 71 | EDD | 96 | MON |
| 22 | MON | 47 | MON | 72 | MON | 97 | MON |
| 23 | MON | 48 | MON | 73 | MON | 98 | EDD |
| 24 | MON | 49 | MON | 74 | MON | 99 | EDD |
| 25 | MON | 50 | MON | 75 | MON | 100 | SPT |

**Table 6A.** Best scheduling policy evolved for $n$=100

| R | T | $n$ | R | T | $n$ |
|---|---|---|---|---|---|
| 0.2 | 0.2 | 12 | 0.4 | 0.4 | 50 |
| 0.2 | 0.4 | 12 | 0.4 | 0.6 | 50 |
| 0.4 | 0.8 | 12 | 0.6 | 0.8 | 50 |
| 0.8 | 0.6 | 12 | 0.8 | 0.6 | 50 |
| 0.8 | 0.8 | 12 | 1.0 | 0.2 | 50 |
| 0.2 | 0.2 | 25 | 0.4 | 0.6 | 100 |
| 0.2 | 0.6 | 25 | 0.4 | 0.8 | 100 |
| 0.6 | 0.8 | 25 | 0.6 | 0.2 | 100 |
| 0.8 | 0.8 | 25 | 0.6 | 0.8 | 100 |
| 1.0 | 0.2 | 25 | 0.8 | 0.8 | 100 |

**Table 7A.** Configuration settings for the SETVAR1

| R | T | *n* | R | T | *n* |
|---|---|---|---|---|---|
| 0.2 | 0.4 | 12 | 0.2 | 0.2 | 50 |
| 0.4 | 0.8 | 12 | 0.4 | 0.2 | 50 |
| 0.6 | 0.6 | 12 | 0.8 | 0.4 | 50 |
| 0.8 | 0.2 | 12 | 0.8 | 0.6 | 50 |
| 1.0 | 0.4 | 12 | 1.0 | 0.4 | 50 |
| 0.2 | 0.8 | 25 | 0.2 | 0.4 | 100 |
| 0.4 | 0.6 | 25 | 0.2 | 0.8 | 100 |
| 0.4 | 0.8 | 25 | 0.4 | 0.6 | 100 |
| 0.6 | 0.2 | 25 | 0.8 | 0.4 | 100 |
| 0.8 | 0.4 | 25 | 1.0 | 0.6 | 100 |

**Table 8A.** Configuration settings for the SETVAR2

| R | T | *n* | R | T | *n* |
|---|---|---|---|---|---|
| 0.4 | 0.4 | 12 | 0.2 | 0.8 | 50 |
| 0.4 | 0.6 | 12 | 0.4 | 0.4 | 50 |
| 0.4 | 0.8 | 12 | 0.6 | 0.6 | 50 |
| 0.6 | 0.2 | 12 | 0.6 | 0.8 | 50 |
| 0.8 | 0.2 | 12 | 0.8 | 0.4 | 50 |
| 0.2 | 0.2 | 25 | 0.2 | 0.4 | 100 |
| 0.4 | 0.6 | 25 | 0.4 | 0.6 | 100 |
| 0.8 | 0.2 | 25 | 0.4 | 0.8 | 100 |
| 1.0 | 0.4 | 25 | 0.8 | 0.8 | 100 |
| 1.0 | 0.6 | 25 | 1.0 | 0.2 | 100 |

**Table 9A.** Configuration settings for the SETVAR3

| R | T | *n* | R | T | *n* |
|---|---|---|---|---|---|
| 0.2 | 0.2 | 12 | 0.2 | 0.2 | 50 |
| 0.4 | 0.4 | 12 | 0.2 | 0.4 | 50 |
| 0.6 | 0.8 | 12 | 0.2 | 0.6 | 50 |
| 0.8 | 0.2 | 12 | 0.4 | 0.8 | 50 |
| 0.8 | 0.8 | 12 | 0.6 | 0.2 | 50 |
| 0.2 | 0.6 | 25 | 0.2 | 0.2 | 100 |
| 0.2 | 0.8 | 25 | 0.2 | 0.8 | 100 |
| 0.6 | 0.6 | 25 | 0.4 | 0.8 | 100 |
| 0.8 | 0.4 | 25 | 0.6 | 0.4 | 100 |

| | | | | | |
|---|---|---|---|---|---|
| 1.0 | 0.4 | 25 | 0.8 | 0.8 | 100 |

**Table 10A.** Configuration settings for the SETVAR4

| R | T | *n* | R | T | *n* |
|---|---|---|---|---|---|
| 0.2 | 0.2 | 12 | 0.2 | 0.2 | 50 |
| 0.4 | 0.4 | 12 | 0.4 | 0.8 | 50 |
| 0.6 | 0.8 | 12 | 0.6 | 0.2 | 50 |
| 0.8 | 0.2 | 12 | 0.8 | 0.6 | 50 |
| 0.8 | 0.8 | 12 | 1.0 | 0.2 | 50 |
| 0.2 | 0.2 | 25 | 0.2 | 0.4 | 100 |
| 0.4 | 0.4 | 25 | 0.4 | 0.4 | 100 |
| 0.4 | 0.8 | 25 | 0.4 | 0.6 | 100 |
| 0.6 | 0.4 | 25 | 0.6 | 0.2 | 100 |
| 1.0 | 0.2 | 25 | 1.0 | 0.6 | 100 |

**Table 11A.** Configuration settings for the SETVAR4

| | |
|---|---|
| **SETUP 12** | $\left[\frac{\left(\frac{p_i}{d_i} - SP - SD - (N \bullet SD)\right)}{p_i} + 2 - (2 \bullet N) - SD + d_i - p_i - \left(\frac{SD}{N} \bullet (p_i - SP)\right)\right] \bullet \left[((SD - SP) \bullet (p_i + d_i)) + \left(\frac{d_i \bullet SP}{SD \bullet N}\right) + (SP \bullet d_i \bullet N \bullet (N - p_i + SP))\right]$ |
| **SETUP 25** | $((p_i + SD) - d_i - (N \bullet d_i) + (3 \bullet SD) - N) \bullet ((d_i \bullet N) - (N \bullet SD)) - (N \bullet p_i) + ((p_i + N) \bullet N \bullet (SP + SD) \bullet (SP - (2 \bullet p_i \bullet N) - p_i))$ |
| **SETUP 50** | $\left((d_i + p_i - (2 \bullet SP)) + \left(SD \bullet \frac{(p_i - SP)}{(SD + p_i)}\right)\right) \bullet (1 + (2 \bullet d_i) + (N \bullet SD)) \bullet \left((3 \bullet SD) - SP - \left(N \bullet p_i \bullet \left(p_i^2 + p_i\right)\right) - (2 \bullet p_i \bullet SP) - ((N + p_i) \bullet SP)\right)$ |
| **SETUP 100** | $((N^2 + p_i^2) \bullet ((p_i \bullet SP) + SD) \bullet SP) - d_i + ((SP + d_i) \bullet d_i \bullet SD)$ |

| | |
|---|---|
| **SETV AR1** | $(((SP + SD) \bullet N \bullet SD) + SD + SP) \bullet \left( \left( N + \left( \frac{d_i \bullet p_i}{(SP - N)} \right) \right) \bullet d_i \right) + \left( 2 \bullet \left( \left( \frac{SP}{p_i} \right) + SP \right) \bullet N \bullet p_i^6 \right)$ |
| **SETV AR2** | $\left( N \bullet p_i \bullet \left( \left( N^3 \bullet SP \right) + N - p_i \right) \right) + \left( d_i \bullet (SD2 - N) \right) - (SP \bullet N) + (SP \bullet SD) + p_i^2 - SP +$ $+ \left( 2 \bullet p_i \bullet SD^2 \right) - \left( 4 \bullet SD^2 \right) - \left( \frac{d_i}{(SP + N)} \right)$ |
| **SETV AR3** | $\left( \frac{\left( SD \bullet d_i \bullet (SP - N) \right) \bullet \left( \frac{N}{d_i} + \frac{SP}{N} \right)}{N \bullet SP} \right) - \left( \left( \left( (4 \bullet SP) - \left( 2 \bullet d_i \right) - \left( \frac{p_i}{d_i^2} \right) - N \right) \bullet d_i \right) + \left( \left( p_i \bullet SP \right) + \left( N^2 - d_i \right) \right) \bullet \left( p_i + (2 \bullet N) \right) \right)$ |
| **SETV AR4** | $\left( \frac{\left( \frac{d_i^2}{SP} \right) - \left( \frac{N(SD - SP) \bullet p_i \bullet d_i}{SD} \right)}{SP} \right) + \left( 2 \bullet d_i \right) + \left( p_i \bullet N \right)$ |
| **SETV AR5** | $\left( SP^2 \bullet p_i \right) + (N \bullet SP) - SD + \left( \left( \left( SP - \frac{SD}{N} \right) \bullet p_i \right) + SD \right) \bullet \left( \frac{\left( d_i + p_i \right)}{N \bullet d_i} \right) \bullet \left( \left( \left( N + p_i + d_i \right) \bullet d_i \right) + \left( p_i \bullet SP \right) \right)$ |

**Table 12A.** Evolved dispatching rules for each set-up