ELSEVIER

Advances in Engineering Software 32 (2001) 489–498

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

Investigating the use of genetic programming for a classic one-machine scheduling problem

C. Dimopoulos[a], A.M.S. Zalzala[b,*]

[a]*Department of Automatic Control and Systems Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, UK*
[b]*Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh EH14 4AS, UK*

Accepted 17 October 2000

Abstract

Genetic programming has rarely been applied to manufacturing optimisation problems. In this paper the potential use of genetic programming for the solution of the one-machine total tardiness problem is investigated. 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. The surprisingly wide range of manufacturing optimisation problems covered by active evolutionary computation research is highlighted in Ref. [1].

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]). Manufacturing optimisation has rarely been the subject of genetic programming (GP) research ([4,5]). 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 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 paper the potential use of GP for the solution of the one-machine total tardiness problem is investigated. The aim is to evolve a dispatching rule that challenges man-made dispatching rules in the solution of the problem. Potts and Van Wassenhove [6] 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 the GP method introduced in this paper. However, while Potts and Van Wassenhove's algorithm is problem-dependent and has no other known applicability, the method 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

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

If the lateness of job i is defined as the difference between its completion time C_i and the corresponding due date d_i, then the tardiness is calculated using the following formula:

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

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

A number of jobs $J_1, J_2, \ldots, J_n$ are to be processed in a

* Corresponding author.
E-mail addresses: cop97cd@sheffield.ac.uk (C. Dimopoulos), a.zalzala@hw.ac.uk (A.M.S. Zalzala).

0965-9978/01/$ - see front matter
PII: S0965-9978(00)00109-5

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. The aim is to find the processing sequence that minimises the sum of tardiness of all jobs:

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

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

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

The objective of the weighted total tardiness problem is the minimisation of expression (2). If Eq. (1) or (2) is divided 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. The complexity of the weighted total tardiness problem was established by Lawer [7]. He proved that the associated decision problem is NP-complete by reduction from the three-partition problem. An alternative proof was given by Lenstra et al. [8] the same year. The complexity of the unweighted total tardiness problem remained unestablished until 1989, when Du and Leng [9] proved that the associated decision problem is NP-complete by reduction from a restricted version of the Even–Odd Partition problem.

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 40 years. In the early stages researchers focused on the development of efficient implicit enumeration algorithms, mainly dynamic programming [7,10], and branch and bound [11–13]. Dynamic programming, a powerful optimisation method introduced by Bellman and Dreyfus [14], is much faster than complete enumeration. However, it has obvious limitations in terms of memory requirements. Branch and bound methods are quite unpredictable in their computational requirements. Their success depends heavily on the calculation of sharp lower bounds, which result in the quick elimination of subtrees, speeding up the procedure considerably.

In recent years, especially after Potts and Van Wassenhove [6] presented a quite efficient algorithm for the optimal solution of very large problem instances, researchers have focused on the development of fast and efficient heuristic algorithms [15–18]. While these algorithms perform much better than implicit enumeration techniques in terms of computational requirements, the optimality of their solutions is not guaranteed.

3. Brief introduction to genetic programming

Genetic Programming belongs to the family of evolutionary computation methods. During the 1980s a number of researchers investigated the use of evolutionary computation for program induction [19,20]. Koza [21] used the term 'Genetic Programming' to describe his search method that combined efficiently the concepts of evolutionary computation and automatic programming.

The concept of Darwinian strife for survival is the driving force of the GP algorithm. A potential solution of an optimisation problem is appropriately coded into a 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 individuals either survive intact to the new generation or they are genetically modified through a number of operators. In conventional EAs, solutions are usually represented by fixed-size 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). The intuition behind GP is that a solution of an optimisation problem can often be represented by a computer program [21]. The program takes a number of inputs (terminals) that are relevant to the problem considered, manipulates them through a number of functions and produces the required outputs. Genetic programs are usually illustrated as collections of function and terminal nodes in the form of a parse tree. A parse tree structure is interpreted in a depth-first, left to right way as depicted in Fig. 1.

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 also been proposed [22,23].

The functions and terminals used during the evolutionary procedure should be able to represent a solution of the problem (*sufficiency property*). In addition, any function should be able to accept any other function or terminal as its input, without bringing the system to a halt (*closure property*).

Like in most EAs, crossover and mutation are the two major operators that are employed for the genetic modification of tree structures. The application of these operators is quite simple. For the crossover operator two individual programs 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

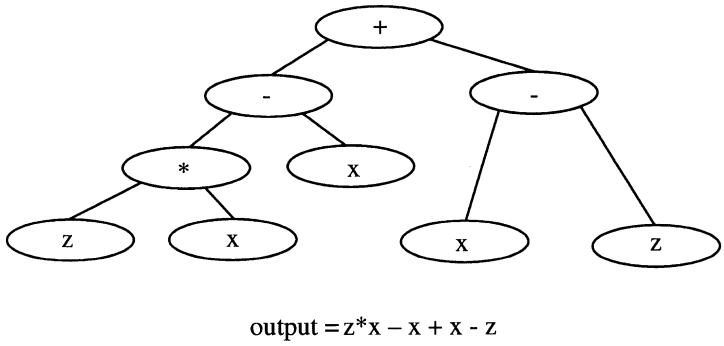

Fig. 1. An example of a GP parse tree and its interpretation.

points exchange their positions at the genetic programs. 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. Traditionally, subtree crossover is considered to be a significant element of GP. Koza applies it to individual programs with a probability of 90%. However, 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. [24] for an in-depth analysis on the subject of GP-crossover and its implications). Other recent studies have indicated that the significance of the mutation operator in GP is more important than originally thought [24].

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

4.1. Solution 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 flow-shop and one-machine scheduling problems like the one discussed in this paper. 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. Instead, in this paper, GP is employed for the evolution a new dispatching rule that will be responsible for the sequencing of jobs.

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 [25,26]. A number of dispatching rules have been associated with the solution of the one-machine total tardiness problem:

The earliest due date rule (EDD) sequences jobs in non-decreasing order of their due date. The shortest processing time rule (SPT) 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 [27]. Based on these theorems, SPT is expected to perform better on problems with high levels of tardiness, and EDD is expected to be ideal for the inverse case. The Montagne (MON) rule was originally introduced by Montagne [28] for the solution of the weighted total tardiness problem. This rule sequences jobs in non-decreasing order of the following ratio:

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

where p_i is the processing time of job i, d_i is the due date of job i and w_i is the associated penalty for job i.

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

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

(the summation term that is missing in the numerator of the ratio has no effect on the operation of the rule). It can be said that 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 sum of the processing times 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. MON performs well on different types of one-machine tardiness problems due to its unique design that takes in account both the processing times and due dates of individual jobs, as well as the sum of the processing times of all jobs. However, the possibility that there exists another formula — perhaps more complex — that utilises a priori knowledge of the problem in a more efficient way, cannot be ruled out. In this paper the possibility of evolving a dispatching rule formula through a GP-framework for the solution of the one-machine total tardiness problem is investigated. The algorithm is supplied with problem-specific information and trained on various sets of tardiness problems, aiming to evolve a dispatching rule that will perform at least as good as the dispatching rules produced by human intuition.

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

These dispatching rules, once evolved, act as independent scheduling policies for the problem considered. Their application does not require a repeat run of the GP algorithm. The intuition behind this approach is that the evolved formula of the dispatching rule would have captured enough information during its training to consider any previously unseen instance of the one-machine total tardiness problem. A successful dispatching rule should be able to produce tardiness levels that are at least as good as the ones produced by man-made dispatching rules on the entire range of validation problems.

The proposed methodology considers the general case of the one-machine total tardiness problem and attempts to extract theoretical information for the problem considered from a set of training cases. The solution methodologies described in Section 2 [15–18], require a repeat run of the algorithm for each instance of the problem considered. While, as expected, they exhibit better performance than man-made or artificial dispatching rules on individual problems, their application is limited to the one-machine total tardiness problem, since their operation is based on this specific problem. Instead, the proposed approach can be used to evolve dispatching rules for any other scheduling problem, as long as relevant information is provided for the training of the rule. In addition, the computational efficiency of dispatching rules decreases much slower with the size of the problem, in comparison with typical heuristic optimisation algorithms. The authors have proposed a combined application of GP with local search algorithms for the generation of near-optimal schedules in individual problem instances. Experimental results of this approach and comparisons with one of the leading one-machine total tardiness heuristics according to published results (M-NBR, [17]) are presented in Ref. [29].

4.2. Design of the algorithm

The main parameters that need to be defined in the design of the GP algorithm are the following:

Function set. The function set comprises of the four main mathematical operations: addition, subtraction, multiplication and division ($+$, $-$, $\times$, %). The '%' symbol corresponds to the protected division function that returns the value of '1' when the value of the denominator is equal to '0'.

Terminal set. The terminal set of the algorithm includes the parameters from which the MON rule is formed (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). The set is completed by two additional parameters, 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 about the suitability of the additional terminals for the evolution of an optimal

Table 1
Training sets for the evolution of dispatching rules

Name	n	Fitness cases (problems) per set-up
SETUP12	12	20
SETUP25	25	20
SETUP50	50	20
SETUP100	100	20
SETVAR1	$5\times(n=12)+5\times(n=25)+5\times(n=50)+5\times(n=100)+$	20
SETVAR2	$5\times(n=12)+5\times(n=25)+5\times(n=50)+5\times(n=100)+$	20
SETVAR3	$5\times(n=12)+5\times(n=25)+5\times(n=50)+5\times(n=100)+$	20
SETVAR4	$5\times(n=12)+5\times(n=25)+5\times(n=50)+5\times(n=100)+$	20
SETVAR5	$5\times(n=12)+5\times(n=25)+5\times(n=50)+5\times(n=100)+$	20

formula. In any case, the GP algorithm should be at least able to converge to the formula of MON rule, since all its elements are included in the function and terminal sets.

Objective function. The objective of the algorithm is the minimisation of the sum of tardiness over the entire set of test problems that are used as fitness cases. Tardiness is measured by scheduling jobs in non-decreasing order of their priority value, as this is calculated based on the formula of the evolved dispatching rule.

Fitness cases. A set of 20 tardiness problems was used for the training of the dispatching rules in each individual GP run. The test problems were generated using the same method that has been used by the majority of researchers investigating the one-machine total tardiness problem. The processing times for each job were 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 were drawn out of a uniform distribution that is defined as follows:

$$[SP(1 - T - (R/2)), \mathrm{SP}(1 - T + (R/2))]$$

where SP is equal to the sum of processing times of all jobs and T is the tardiness factor, it defines the percentage of jobs that are expected to be tardy on average. If, for example, $T = 0.2$, 20% of jobs are expected to be tardy on average. R is the range of due dates, it defines the tightness of due dates around the sum of the processing times of all jobs. Generally speaking, problems with tight due dates are more difficult to solve.

The algorithm was trained on nine different sets of test problems (Table 1). In the first four of them the value of n in the training set was fixed ($n = 12$, 25, 50 and 100, respectively). The remaining sets comprise 20 problems, five for each value of n.

Tables A1–A6 in Appendix A illustrate the configuration of the training sets in detail. 20 runs were conducted in total for each training set. Results are reported in Section 5. A

Table 2
Koza tableau for the proposed methodology

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:	0.5
Tree mutation probability:	0.5
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 3
Tardiness results for all evolved dispatching rules and comparison with dispatching rules produced by human intuition

	Disp-rule SETUP12	Disp-rule SETUP25	Disp-rule SETUP50	Disp-rule SETUP100	Disp-rule SETVAR1	Disp-rule SETVAR2	Disp-rule SETVAR3	Disp-rule SETVAR4	Disp-rule SETVAR5	EDD	SPT	MON
SETUP12	*13 189*	20 094	16 357	17 073	17 068	13 858	14 812	13 795	15 676	17 598	17 044	14 916
SETUP25	68 440	*54 380*	58 734	68 101	69 135	56 292	57 228	57 101	57 915	77 523	69 698	60 657
SETUP50	263 231	242 878	*200 935*	227 332	245 993	202 102	203 149	213 328	206 380	286 042	264 225	226 565
SETUP100	1 047 471	1 128 759	868 513	*790 374*	779 475	808 824	806 609	828 507	852 978	115 8371	1 093 840	886 600
SETVAR1	596 043	613 494	489 485	452 603	*440 238*	457 345	452 723	453 549	490 575	643 558	501 370	472 004
SETVAR2	295 745	317 238	248 306	250 549	262 970	*238 507*	239 999	253 411	245 431	336 645	355 541	263 629
SETVAR3	478 636	493 293	399 584	384 416	380 885	380 406	*377 472*	383 781	396 611	522 807	449 199	420 317
SETVAR4	487 201	495 474	392 716	377 739	377 760	377 886	378 137	*377 781*	397 891	526 316	427 636	393 561
SETVAR5	219 631	235 816	188 116	184 426	190 791	180 388	182 291	193 274	*183 361*	248 575	266 219	204 599
Total	3 469 599	3 601 451	2 862 796	2 752 713	2 764 315	2 715 608	2 712 420	2 774 527	2 846 818	3 817 435	3 444 772	2 942 848

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

Fig. 2. Dispatching rule evolved from set-up SETVAR3.

number of additional parameters need to be defined for a valid run of the GP algorithm. The values of these parameters are included in the Koza table of Table 2. Note that the term *maximum depth for crossover* indicates the maximum allowable depth for the offspring resulting from the crossover operation. If an offspring's depth exceeds this value, the operation is repeated with the same parents, until an offspring that does not violate the constraint is produced. The corresponding constraint for the mutation operation indicates that any subtree randomly generated for this purpose should not exceed the specified depth value. Both these constraints aim to slow the uncontrollable growth of genetic programs, a phenomenon known as *bloat* in genetic programming terminology. The ramped half-and-half method is an initialisation method for genetic programs when parse tree solution representation is employed by the designer of the algorithm. A discussion on GP initialisation methods can be found in Refs. [21,24].

5. Results

The GP framework evolved nine different dispatching rules out of each individual training set. Individual and cumulative performance for each of these rules is illustrated in Table 3. The outlined cells in the table indicate the performance of the corresponding dispatching rule on the set of test problems that were used for its training. The rest of the cells in the same column illustrate the performance of the dispatching rule on the previously unseen test problems.

From Table 3 it can be concluded that in most cases the algorithm was able to evolve dispatching rules that had better overall performance than MON rule and much better performance than the EDD and SPT rules. Most evolved rules performed quite well in a very large set of previously unseen problems (160 in total). Based on this observation it is thought that these rules did not just fit the data of the fitness cases but they extracted information that was relevant to the solution of the problem considered. However, the formulas of these dispatching rules were not as straightforward as the formula of the MON rule. Table B1 in Appendix B presents the mathematical formulas of the nine rules evolved.

In order to compare the performance of a GP-evolved rule with all the traditional dispatching rules used in this report, the rule evolved from the experimental set-up SETVAR3 (Fig. 2) was chosen. This rule produced the best overall performance in terms of the total tardiness produced in all training and validation problems. Note that the expression in Figs. 2 and 3 have been cleared from *introns* (segments of code that have no effect on the outcome of the problem) and have also been simplified wherever that was possible.

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 acts 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 sum of the processing times, the second term produces

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

Fig. 3. Dispatching rule evolved from set-up SETVAR2.

Table 4
Comparative performance of SETVAR3 for all test problems

	OPT	EDD	SPT	MON	SETVAR3
Total tardiness (units)	2 430 198	3 817 435	3 444 772	2 942 848	2 712 420
MADO (units)		7706.872	5636.522	2848.056	1567.9
Optimal solutions		39	0	4	25

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.

Table 3 illustrates that SETVAR3 was not only able to perform well on the set of training problems, but it was also able to perform better than man-made dispatching rules on the considerable number of validation problems. In that sense, it can be concluded that the generalisation of the dispatching rule is satisfactory.

In Table 4 the performance of the dispatching rule SETVAR3 is compared with those of the classic EDD, SPT and MON rules in all test problems (both training and validation sets).

The improvement in overall performance by using SETVAR3 was significant. MON imposed 81% higher penalties in terms of MADO (mean absolute deviation from optimal), while the *t*-test on the penalties rejected the null hypothesis with a very high probability ($t = 5.62$, $p < 3.49 \times 10^{-8}$). In addition, SETVAR3 consistently outperformed all the other rules in terms of non-dominated solutions. In all cases, at least 77% of the solutions produced by SETVAR3 were better or equal than those produced by the alternative man-made dispatching rules (Table 5).

As expected, EDD performed 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 were evenly distributed in terms of T and R, EDD scheduling produced the worse performance over the available dispatching rules.

It will be interesting to take a closer look at the dispatching rule evolved using the experimental set-up SETVAR2 (Fig. 3), that performed almost identical to the previous rule. While there were terms with similar operation between the two rules (one favouring EDD scheduling and one favouring SPT scheduling), there were no other easily observed 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.

Table 5
Performance of SETVAR3 of non-dominated solutions (all test problems)

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

The GP algorithm had difficulties in converging to the same solution in each training set. The rules described in Table 3 were the ones that produced the best overall performance for each training set, and occurred only once in 20 independent runs. One of the reasons for that phenomenon was the difficulty imposed on the algorithm by the training sets. The variety in problem parameters within the same training set created a considerable number of local-optima. The use of small-sized populations of genetic programs due to the limited computational resources also made the convergence of the algorithm to a single solution more difficult. However, the rules evolved in all other runs were not significantly worse in terms of total tardiness.

6. Conclusions

In this paper the potential use of genetic programming for the solution of the one-machine total tardiness problem was investigated. This problem has been the subject of academic research for almost four decades. To the best of the authors' 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. A traditional GP-framework was employed as a basis for evolving a formula of a dispatching rule that will act as a general scheduling policy for the solution of the one machine total tardiness problem. Nine dispatching rules were evolved during the experimental phase of the algorithm. A number of these rules generalised quite well, i.e. they were able to produce tardiness level that were at least as good as the ones produced by man-made dispatching rules not only on the training problems, but on a considerable number of validation problems as well.

7. Recommendations for future work

There is a variety of ways in which this research can be extended. The previous method 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, and the parameters are available a priori. GP as well as other evolutionary algorithms are extremely parameter sensitive, especially when difficult experimental training sets like the ones used in this paper 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-reutilisation technique like automatic defined functions [30].

Acknowledgements

The authors would like to help the reviewers for their helpful comments. The first author would like to thank Greek State Fund (I.K.Y.) for its support.

Appendix A

Tables A1–A6 illustrate the configuration of the training sets in detail.

Table A1
Configuration settings for SETUP12, SETUP25, SETUP50, SETUP100 ($n = 12, 25, 50, 100$)

R	T	R	T
0.2	0.2	0.6	0.6
0.2	0.4	0.6	0.8
0.2	0.6	0.8	0.2
0.2	0.8	0.8	0.4
0.4	0.2	0.8	0.6
0.4	0.4	0.8	0.8
0.4	0.6	1.0	0.2
0.4	0.8	1.0	0.4
0.6	0.2	1.0	0.6
0.6	0.4	1.0	0.8

Table A2
Configuration settings for SETVAR1

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 A3
Configuration settings for SETVAR2

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 A4
Configuration settings for SETVAR3

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 A5
Configuration settings for SETVAR4

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 A6
Configuration settings for SETVAR5

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

Appendix B

Table B1 presents the mathematical formulas of the nine rules evolved.

Table B1
Evolved dispatching rules for each set of training problems

SETUP12 $$\left[\frac{\left(\frac{p_i}{d_i} - \mathrm{SP} - \mathrm{SD} - (n \times \mathrm{SD})\right)}{p_i} + 2 - (2 \times n) - \mathrm{SD} + d_i - p_i - \left(\frac{\mathrm{SD}}{n} \times (p_i - \mathrm{SP})\right)\right] \times \left[((\mathrm{SD} - \mathrm{SP}) \times (p_i + d_i)) + \left(\frac{d_i \times \mathrm{SP}}{\mathrm{SD} \times n}\right) + (\mathrm{SP} \times d_i \times n \times (n - p_i + \mathrm{SP}))\right]$$

SETUP25 $$((p_i + \mathrm{SD}) - d_i - (n \times d_i) + (3 \times \mathrm{SD}) - n) \times ((d_i \times n) - (n \times \mathrm{SD})) - (n \times p_i) + ((p_i + n) \times n \times (\mathrm{SP} + \mathrm{SD}) \times (\mathrm{SP} - (2 \times p_i \times n) - p_i))$$

SETUP50 $$\left((d_i + p_i - (2 \times \mathrm{SP})) + \left(\mathrm{SD} \times \frac{(p_i - \mathrm{SP})}{(\mathrm{SD} + p_i)}\right)\right) \times (1 + (2 \times d_i) + (n \times \mathrm{SD})) \times ((3 \times \mathrm{SD}) - \mathrm{SP} - (n \times p_i \times (p_i^2 + p_i)) - (2 \times p_i \times \mathrm{SP}) - ((n + p_i) \times \mathrm{SP}))$$

SETUP100 $$((n^2 + p_i^2) \times ((p_i \times \mathrm{SP}) + \mathrm{SD}) \times \mathrm{SP}) - d_i + ((\mathrm{SP} + d_i) \times d_i \times \mathrm{SD})$$

SETVAR1 $$(((\mathrm{SP} + \mathrm{SD}) \times n \times \mathrm{SD}) + \mathrm{SD} + \mathrm{SP}) \times \left(\left(n + \left(\frac{d_i \times p_i}{(\mathrm{SP} - n)}\right)\right) \times d_i\right) + \left(2 \times \left(\left(\frac{\mathrm{SP}}{p_i}\right) + \mathrm{SP}\right) \times n \times p_i^6\right)$$

SETVAR2 $$(n \times p_i \times ((n^3 \times \mathrm{SP}) + n - p_i)) + (d_i \times (\mathrm{SD}^2 - n)) - (\mathrm{SP} \times n) + (\mathrm{SP} \times \mathrm{SD}) + p_i^2 - \mathrm{SP} + (2 \times p_i \times \mathrm{SD}^2) - (4 \times \mathrm{SD}^2) - \left(\frac{d_i}{(\mathrm{SP} + n)}\right)$$

SETVAR3 $$\left(\frac{(\mathrm{SD} \times d_i \times (\mathrm{SP} - n)) \times \left(\frac{n}{d_i} + \frac{\mathrm{SP}}{n}\right)}{n \times \mathrm{SP}}\right) - \left(\left((4 \times \mathrm{SP}) - (2 \times d_i) - \left(\frac{p_i}{d_i^2}\right) - n\right) \times d_i\right) + ((p_i \times \mathrm{SP}) + (n^2 - d_i)) \times (p_i + (2 \times n))$$

SETVAR4 $$\left(\frac{\left(\frac{d_i^2}{\mathrm{SP}}\right) - \left(\frac{n(\mathrm{SD} - \mathrm{SP}) \times p_i \times d_i}{\mathrm{SD}}\right)}{\mathrm{SP}}\right) + (2 \times d_i) + (p_i \times n)$$

SETVAR5 $$(\mathrm{SP}^2 \times p_i) + (n \times \mathrm{SP}) - \mathrm{SD} + \left(\left(\left(\mathrm{SP} - \frac{\mathrm{SD}}{n}\right) \times p_i\right) + \mathrm{SD}\right) \times \left(\frac{(d_i + p_i)}{n \times d_i}\right) \times (((n + p_i + d_i) \times d_i) + (p_i \times \mathrm{SP}))$$

References

[1] Dimopoulos C, Zalzala AMS. Recent developments in evolutionary computation for manufacturing optimisation: problems, solutions and comparisons. IEEE Trans Evol Comput 2000;4(2):93–113.

[2] Kirkpatrick S, Gelatt Jr, C D, Vecchi MP. Optimisation by simulated annealing. Science 1985;220:671–9.

[3] Glover F. Tabu search: a tutorial. Interfaces 1990;20(3):74–94.

[4] Garces-Perez J, Schoenefeld DA, Wainwright RL. Solving facility layout problems using genetic programming. In: Koza, Goldberg, Fogel, Riolo, editors. Genetic Programming 1996: Proceedings of the 1st Annual Conference. Cambridge, MA: MIT Press, 1996. p. 182–90.

[5] McKay BM, Willis MJ, Hiden HG, Montague GA, Barton GW. Identification of industrial processes using genetic programming In: Friswell, Mottershead, editors. Proceedings of the Conference on Identification in Engineering Systems, University of Wales, Swansea, UK, 1996. p. 510–9.

[6] Potts CN, Van Wassenhove LN. A decomposition algorithm for the single machine total tardiness problem. Oper Res Lett 1982;1(5):177–81.

[7] Lawer EL. A pseudopolynomial algorithm for sequencing jobs to minimise total tardiness. Ann Discrete Math 1997;1:331–42.

[8] Lenstra JK, Rinnooy Kan AHG, Lageweg BJ. Complexity of machine scheduling problems. Ann Discrete Math 1997;1:343–62.

[9] Du J, Leung JY-T. Minimising total tardiness on one machine is NP-hard. Math Oper Res 1989;15(3):483–95.

[10] Srinivasan V. A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. Naval Res Logistics Q 1971;18(3):317–27.
[11] Elmaghraby SE. The one machine sequencing problem with delay costs. J Industrial Engng 1968;19(2):105–8.
[12] Baker KR, Schrage LE. Finding an optimal sequence by dynamic programming: an extension to precedence-related tasks. Oper Res 1978;26(1):111–20.
[13] Schrage LE, Baker KR. Dynamic programming solution of sequencing problems with precedence constraints. Oper Res 1978;26(3):444–9.
[14] Bellman RE, Dreyfus SE. Applied dynamic programming. Princeton, NJ: Princeton University Press, 1962.
[15] Fry TD, Vicens L, Macleod K, Fernadez S. A heuristic solution procedure to minimize total tardiness. J Oper Res Soc 1989;40:293–7.
[16] Holsenback JE, Russel RM. A heuristic algorithm for sequencing on one machine to minimize total tardiness. J Oper Res Soc 1992;43:53–62.
[17] Holsenback JE, Russel RM. Evaluation of greedy, myopic and less-greedy heuristics for the single-machine, total tardiness problem. J Oper Res Soc 1997;48:640–6.
[18] Panwalkar SS, Smith ML, Koulamas. A heuristic for the single machine tardiness problem. Eur J Oper Res 1993;70:304–10.
[19] Cramer NL. A representation for the adaptive generation of simple sequential programs. In: Grefenstette JJ, editor. Proceedings of the 1st International Conference on Genetic Algorithms and their Applications. Hillsdale, NJ: Lawrence Erlbaum, 1998. p. 183–7.
[20] Fujiki C, Dickinson J. Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. In: Grefenstette JJ, editor. Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications. Hillsdale, NJ: Lawrence Erlbaum, 1987. p. 236–40.
[21] Koza JR. Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA: MIT Press, 1992.
[22] Nordin P. A compiling genetic programming system that directly manipulates machine code. In: Kinnear Jr. KE, editor. Advances in genetic programming, Cambridge, MA: MIT Press, 1994. p. 311–31.
[23] Teller A, Veloso M. PADO: a new learning architecture for object recognition. In: Ikeuchi, Veloso, editors. Visual learning. Oxford: Oxford University Press, 1996. p. 81–116.
[24] Banzhaf W, Nordin P, Keller RE, Francone FD. Genetic programming: an introduction. San Francisco, CA: Morgan Kaufman, 1998.
[25] Blackstone JH, Philips DT, Hogg CL. A state of the art survey of dispatching rules for manufacturing job shop operations. Int J Prod Res 1982;20:27–45.
[26] Haupt R. A survey of priority rule-based scheduling. OR Spektrum 1989;11(1):3–16.
[27] Emmons H. One machine sequencing to minimise certain function of job tardiness. Oper Res 1968;17(4):701–15.
[28] Montagne GR. Sequencing with time delay costs. Industrial Engineering Research Bulletin, Arizona State University, 1969; 5.
[29] Dimopoulos C, Zalzala AMS. A genetic programming heuristic for the one-machine total tardiness problem, Proceedings of the Congress on Evolutionary Computation (CEC'99), Washington, DC, vol. 3. New York: IEEE Press, 1999. p. 2207–14.
[30] Koza JR. Genetic programming II: automatic discovery of reusable programs. Cambridge, MA: MIT Press, 1994.