Int J Adv Manuf Technol (2000) 16:609–615

The International Journal of
Advanced
Manufacturing
Technology

The Application of Parallel Multipopulation Genetic Algorithms to Dynamic Job-Shop Scheduling

J. G. Qi, G. R. Burns and D. K. Harrison

Department of Engineering, Glasgow Caledonian University, Glasgow, UK

This paper describes the use of parallel multipopulation genetic algorithms (GAs) to meet the dynamic nature of job-shop scheduling. A modified genetic technique is adopted by using a specially formulated genetic operator to provide an efficient optimisation search. The proposed technique has been successfully implemented using the programming language MATrix LABoratory (MATLAB), providing a powerful tool for job-shop scheduling. Comparisons indicate that the proposed genetic algorithm has successfully improved upon the solution obtained from conventional approaches, particularly in coping with job-shop scheduling.

Keywords: Dispatching rules; Genetic algorithms; Integer linear programming; Job-shop scheduling; MATLAB; Multipopulation genetic algorithms

1. Introduction

Scheduling manufacturing operations is a complicated decision-making process using many input parameters. Even when decision input parameters and constraints are static and deterministic, finding effective schedules is not easy. Traditional approaches to the solution of scheduling problems depend heavily on dispatching rules and knowledge-based systems. The disadvantage with such dispatching rules is that there is no single rule that will be effective for all production conditions, and manual selection or updating is required where using rules [1,2]. Knowledge-based systems solve scheduling problems through the application of expert knowledge rather than well-defined algorithms, or mathematical formulations of traditional methodologies. This requires the existence of expert knowledge with scheduling experience. Since experience might be obtained from separate individual cases, it is therefore necessary to have a mechanism to modify the knowledge base. These conclusions are demonstrated in several applications described by Newman and Kempf [3] and Kusiak [4].

To overcome these problems and achieve the best results for production schedules, recent advances in computing technology have made it possible to implement more sophisticated algorithms for production optimisation and performance. A genetic algorithm (GA) is one method that has demonstrated significant improvement when applied to job-shop scheduling.

2. Genetic Algorithms in Manufacturing Scheduling

A genetic algorithm consists of computational procedures that mimic the adaptation that nature uses to find an optimal state. Genetic algorithms have proved to be an effective and flexible optimisation tool that can produce optimal or near-optimal solutions efficiently. Furthermore, genetic algorithms do not need to make as many assumptions regarding the form of the objective functions as many other optimisation techniques do [5]. A GA maintains a population of optimal solutions which are used as a base to improve solutions, whereas most other optimisation methods produce only a single solution at the end of calculation.

The job-shop scheduling problem has attracted many applications of genetic search algorithms. The initial research was reported by Davis [6], who proposed an approach to encode the representation of a schedule in such a way that genetic operators can be applied. A decoder would always produce a legal solution to the problem. This strategy, to encode solutions for operations and to decode them for evaluation, is very attractive and can be applied to a variety of constrained problems. He evaluated the method using an example of two orders, six machines, and three operations.

Starkwerther et al. [7] compared six sequencing operators (order crossover, partially mapped crossover, cycle crossover, enhanced edge recombination, order-based crossover and position-based crossover) for a scheduling application. The enhanced edge recombination operator was found to give the best solution, followed closely by order crossover, order-based crossover, and position-based crossover, with PMX and cycle crossovers being the worst.

Correspondence and offprint requests to: Professor D. K. Harrison, Department of Engineering, Glasgow Caledonian University, City Campus, Cowcaddens Road, Glasgow, G4 0BA. E-mail: dha2@gcal.ac.uk

Syswerda [8] also developed an evolution program for solving production scheduling problems. Three mutations were defined and used: position-based mutation, order-based mutation, and scramble mutation. All three operators performed much better than random search.

In this paper, we develop a parallel multipopulation GA which employs a migration model together with some improved currently existing crossover and mutation operators, to deal with dynamic job-shop scheduling problems. The proposed GA overcomes some weaknesses of a simple GA, for example, slowness in computation time, excess objective function evaluations, and easily trapped to local optimum.

3. A Mixed Integer Linear Programming for Job-Shop Scheduling

The determination of an optimal job-shop schedule can be formulated as an integer programming problem if a general-purpose solution approach is to be applied. The main advantage of an integer programming approach is that it can accommodate a variety of scheduling criteria without making many changes to the method [9].

This section describes one integer programming structure that will be used in relation to genetic optimisation to generate an optimal production schedule. The proposed mixed integer linear approach to solving job-shop scheduling followed the development work by Baker [9] and Foo et al. [10].

The proposed approach regards determination of an optimal job-shop schedule as a linear programming problem with integer adjustments, and then employs genetic algorithms to optimise a solution. These requirements minimise the completion times of all jobs subject to a set of precedence constraints and restrictive assumptions. The detailed mathematical formulation can be depicted as follows.

Let C_{ik} denote the completion time of job i on machine k (i.e. the completion time of the particular operation of job i that needs machine k) and t_{ijk} the processing time for operation (i,j,k). The C_{ik} are decision variables and a schedule will be essentially determined by their values. Each job-shop operation is described by a triplet (i,j,k) i.e. operation j of job i is to be executed on machine k. Assuming operation $(i,j-1,h)$ requires machine h and operation (i,j,k) requires machine k, then the inequalities representing precedence constraints in order for a set of C_{ik} to be feasible are:

$$C_{ik} - C_{ih} - t_{ijk} \geq 0, \quad 1 \leq j \leq m, \quad 1 \leq i \leq n \tag{1}$$

where m and n are the number of machines and number of jobs, respectively.

It is necessary to ensure that no two operations are processed simultaneously by the same machine. For example, if job i precedes job p on machine k (i.e. operation (i,j,k) is completed before operation (p,q,k)), this is equivalent to a constraint of:

$$C_{pk} - C_{ik} - t_{pqk} \geq 0 \tag{2}$$

On the other hand, if job p precedes job i on machine k, then it is also necessary to ensure that:

$$C_{ik} - C_{pk} - t_{ijk} \geq 0 \tag{3}$$

In order to adjust these constraints in the formulation, a variable y_{ipk} is applied to specify the operation sequence, i.e. $y_{ipk} = 1$ if job i precedes job p on machine k, and $y_{ipk} = 0$ otherwise, the above constraints become:

$$C_{ik} - C_{pk} + Ny_{jpk} - t_{ijk} \geq 0 \tag{4}$$

$$C_{pk} - C_{ik} + N(1 - y_{ipk}) - t_{pqk} \geq 0 \tag{5}$$

where constant N represents an arbitrary very large positive number.

Therefore, the entire job-shop problem formulation can be summarised as minimising the mean flow-time problem:

$$\text{Minimise} \sum_{i=1}^{n} C_{ik_i}$$

Subject to

$$C_{ik} - C_{ih} - t_{i,j,k} \geq 0 \qquad 1 \leq j \leq m,\ 1 \leq i \leq n$$

$$C_{pk} - C_{ik} + N(1 - y_{ipk}) - t_{pqk} \geq 0 \qquad i \geq 1,\ p \leq n,\ 1 \leq k \leq m$$

$$C_{ik} - C_{pk} + Ny_{ipk} - t_{ijk} \geq 0 \qquad i \geq 1,\ p \leq n,\ 1 \leq k \leq m$$

$$C_{ik} \geq 0 \qquad y_{ipk} = 0 \text{ or } 1$$

where k_i denotes the machine at which the last operation of job i is scheduled.

4. The Proposed Parallel Genetic Algorithm Approach

This paper describes a genetic algorithm with a parallel multipopulation search structure. The resultant parallel genetic algorithms use restricted selection and mating based on locality, and when executed are shown to yield better performance than single-population implementations.

4.1 Initialisation

Initialisation involves generating a set of possible solutions for the search algorithm. This initial population may be generated randomly or with some established heuristics. A suitable heuristic would reduce the number of generations required to find a solution and allow GAs to start the search in a more favourable region of the search space. This requires additional computing overhead. In our implementation, the initial population is created randomly.

4.2 Representation and the Objective Function

The objective of scheduling is to minimise the total processing time of all jobs, i.e. all jobs should be completed as early as possible. The suitable objective functions are max (C_{ik_i}), and the optimising process is $\min(\max(C_{ik_i}))$.

When formulating constraints for the objective function, a penalty approach suggested by Goldberg [11] is followed. This approach transforms a constrained problem into an unconstrained problem by associating a cost, or penalty, with all constraint violations. For example:

$$\text{minimise } f(\mathbf{X}) \tag{6}$$

$$\text{Subject to } n_i(\mathbf{X}) \geq 0 \quad (i = 1,2,\ldots,n) \tag{7}$$

where $\mathbf{X}$ is an m vector. A transformed unconstrained form is as follows:

$$\text{minimise } f(\mathbf{X}) + r\sum_{i=1}^{n}\theta[n_i(\mathbf{X})] \tag{8}$$

where θ is the penalty function and r is the penalty coefficient. A number of alternatives exist for the penalty function θ. In this paper, we use the transformation of $\theta[n_i(\mathbf{X})] = [n_i(\mathbf{X})]^2$.

The description of the chromosome was constructed on the basis of both job orders and waiting times [12]. The representation of the chromosome is composed of several substrings, each of which is referred to a machine as a resource. For the kth machine, the substring is represented as $\{J_{k1},J_{k2},\ldots,J_{kn},W_{k1},W_{k2},\ldots,W_{kn}\}$ where J_{ki} represents an operation of a job processed by the kth machine, and W_{ki} is within the upper limit of the waiting time of the kth machine to process a job. Thus, if there are m machines, the whole string is made up of m substrings which are then composed of $2n$ genes where n denotes the number of jobs to be processed by the machine.

This type of string description naturally satisfies the constraints of Eqs (2) and (3) or Eqs (4) and (5), but it may not satisfy the constraint Eq. (1).

To obtain a feasible solution, a penalty approach is adopted in this paper, therefore an objective function is finally constructed as:

$$ObjectiveFunction = \min[\max(C_{ik_i})] + 100 \times \sum_{i,j}(C_{ik} - C_{ih} - t_{ijk})^2 \tag{9}$$

4.3 Special Crossover and Mutation Operators

A basic operator for producing new chromosomes using a genetic algorithm is that of crossover. Like its counterpart in nature, crossover produces new individuals that have some parts of both parents' genetic attributes. The simplest form of crossover is that of single-point crossover.

The general job-shop scheduling (JSS) problem is known to be extremely difficult. The common crossover and mutation operators are shown not to be effective for the general JSS problem. A typical crossover operator called job-order is developed for the JSS problem to create a valid sequence of genes, which is the combination of the order-based crossover and normal multipoint crossover.

The order-based crossover operator applied to the first half of the substring is often used in sequencing problems because it maintains precedence constraints of operations from the parents to the offspring. It works by incorporating a string sequence from two different parents into two new strings.

The mechanism of order-based crossover is addressed by first determining two cut points at two random positions on the gene strings, then passing the portion between the two cut points to the two offspring. The operator then begins to construct the remaining lefthand and righthand sides of the first offspring by going over the second parent and eliminating the same numbers with the passed portion of the first parent, and filling up the missing left and right sides according to an order existing on the second parent. A genetic algorithm will do this operation pair-wise. Figure 1 illustrates this genetic operation process.

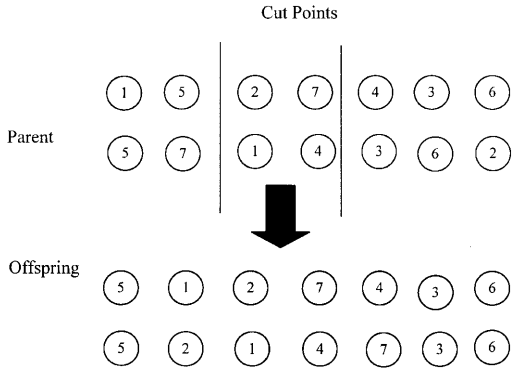

Fig. 1. Order-based crossover operator.

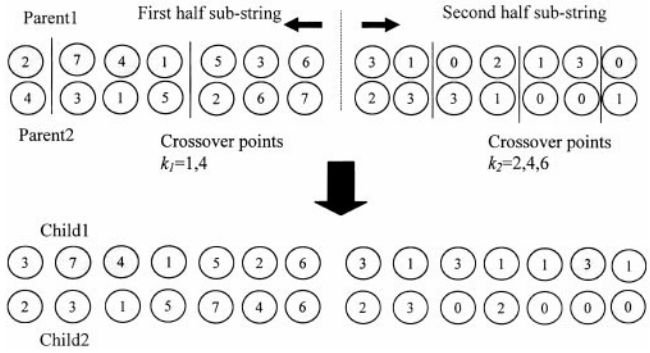

Fig. 2. An example of job-order crossover.

A multipoint crossover operator selected for the second half of the substring enables users to select the number of crossover points. By comparison with one-point crossover, the advantage of multipoint crossover as applied in this paper is that it encourages the exploration of the search space, rather than favouring the convergence to highly fit individuals early in the search, thus making the search more robust [13].

Figure 2 shows how job-order crossover performs these stepwise operations. A mutation mechanism allows for a small number of genes to alter their internal structure in order to survive. A mutation operator plays a supportive role and runs in the background. The intention is either to explore new contexts or recover loss contexts of gene structures from the reproduction operator. It minimises risks of discovering new gene structures by randomly changing gene strings.

Mutation operators applied in this study are position-based and order-based mutation. The idea behind position-based mutation is to swap two genes in the string randomly. The idea of an order-based mutation is to push one randomly selected gene in front of another randomly selected gene. These processes are illustrated in Figs 3 and 4. Each of the mutation operators can be used either in the first half of the substring,

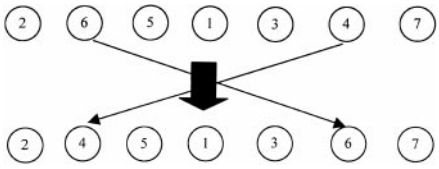

Fig. 3. Position-based mutation operator.

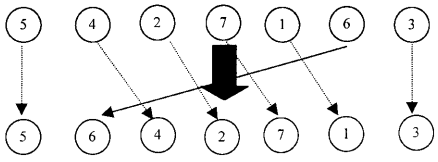

Fig. 4. Order-based mutation operator.

or in the second half of the substring, depending upon how the optimisation search is to be conducted.

4.4 Parent Selection Techniques

Parent selection is a process that allocates reproductive opportunities to individuals. In principle, individuals with higher fitness values are more likely to be selected for the mating pool and individuals with lower fitness values will receive lower or even no chance to act as parents. The biased selection enables the convergence of the search. As the process continues, the variation in fitness range will be reduced; but this sometimes leads to premature convergence. The problem could occur when a few super-fit individuals receive high reproductive trials and rapidly dominate the others in a population. If such individuals correspond to a local optimum, the GA will be trapped like hill climbing. Fitness ranking [14] is employed to overcome this problem. Individuals are sorted into their fitness values, and the number of reproductive trials are then allocated according to their rank.

4.5 Replacement

In most implementations [11], a whole population is replaced in each generation. This is referred to as a generation approach. In this study, however, a steady-state approach is adopted. In each generation only 90% of individuals are replaced. In other words, parents and offspring may coexist in a population.

4.6 Migration

The migration model divides the population in multiple subpopulations. These subpopulations evolve independently from each other for a certain number of generations (isolation time). After the isolation time a number of individuals are distributed between the subpopulations (migration). The number of exchanged individuals (migration rate), the selection method of the individuals for migration and the scheme of migration determines how much genetic diversity can occur in the subpopulations and the exchange of information between subpopulations.

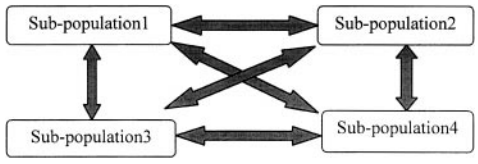

Fig. 5. Unrestricted migration topology.

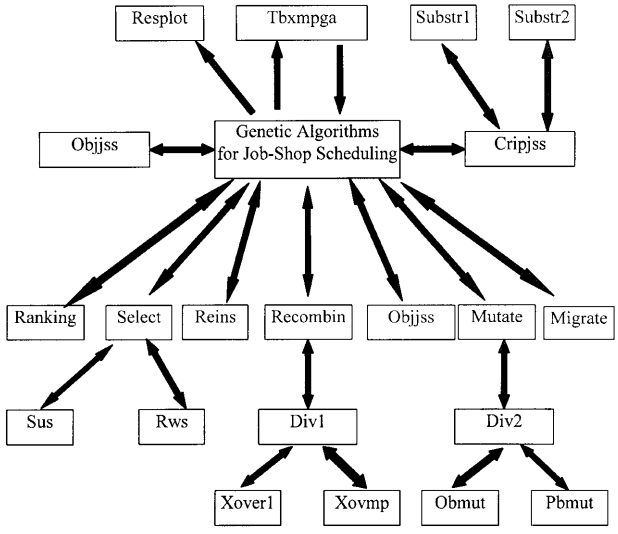

Fig. 6. A calling tree of functions for manufacturing scheduling.

The parallel multipopulation implementation of the migration model has shown not only a speedup in computation time, but it also required fewer objective function evaluations and found the global optimum more often when compared to a single population algorithm [15].

The selection of the individuals for migration in this paper is based on their fitness, and the structure of the migration of individuals takes place between all subpopulations (unrestricted migration topology), as shown in Fig. 5.

4.7 Termination

The processes of crossover, selection and replacement are repeated until a termination criterion is met. The simplest criterion is a prespecified maximum number of generations. Another criterion involves calculating the variation of individuals. If the calculated value falls below a certain threshold, the GA is terminated.

4.8 Computational Implementation Using MATLAB

Simulation programs were developed to implement this parallel multipopulation GA using MATLAB, and also based upon a genetic algorithm toolbox, developed at the University of Sheffield.

The architecture of this parallel genetic algorithm implementation for job-shop scheduling problems is described in Fig. 6. The functions are described in Table 1.

Table 1. The description of functions.

Function	Description
Tbxmpga	Implements the multipopulation genetic algorithm
Cripjss	High-level function, initialises a population using integer values
Resplot	Implements the graphical output of results of the genetic algorithm
Objjss	Supplies a test function for job-shop scheduling
Ranking	Ranks individuals according to their objective values
Select	Performs selection of individuals from a population
Recombin	Performs recombination of individuals from a population
Mutate	Performs mutation of individuals from a population
Reins	Performs insertion of offspring into the current population
Migrate	Performs migration of individuals between subpopulations in the current population
Substr1	Creates a job-order initial population
Substr2	Creates an idle times initial population
Div1	Assigns the different substrings to perform different crossover operations
Div2	Assigns the different substrings to perform different mutation operations
Xover1	Performs job-order crossover
Xovmp	Performs multipoint crossover
Obmut	Performs order-based mutation
Pbmut	Performs position-based mutation
Sus	Performs stochastic universal sampling selection
Rws	Performs roulette wheel selection

5. Simulation Study of Proposed GA Optimisation

Within a dynamic framework, a static problem is generated whenever a new job arrives in the system. At that point in time, we consider all jobs in the system and develop schedules for all the machines. We assume that the operation sequence for each part type establishes a serial precedence relationship among the operations. In addition, the machines required for each operation and the operation-processing times are deterministic and known. Pre-emption of any operation on any machine is prohibited. The optimal solution determined by a genetic algorithm provides a schedule for each machine. This schedule is implemented until the next job arrives. At that time, the process of creating and solving the static problem is repeated. The static solution is, therefore, implemented on a rolling basis [16]. Based on the above description, we assume that a workstation is configured with four machines, and eight available jobs with similar due dates (shown in Table 2) are waiting to be processed on the workstation at a given point in time. Table 2 gives a detailed example used in this simulation study.

It is seen that Job 4 is due in 29 hours, and it is required to be processed first on machine C for 6 h, then on machine B for 4 h, and then on machine A for 3 h, and finally on machine D for 4 h. The simulation was carried out on a P166 32MB computer, which took a few minutes to converge. The optimisation parameters are shown in Table 3 and the simulation results in Figs 7 and 8.

Table 2. A job-shop scheduling example.

Job	Sequence and processing times (h)	Due date (h)
1	A(4) C(4) B(7) D(3)	32
2	C(3) A(4) B(4) D(2)	30
3	A(3) D(4) C(6) B(3)	33
4	C(6) B(4) A(3) D(4)	29
5	D(4) B(5) C(5) A(3)	28
6	B(4) D(6) A(5) C(4)	34
7	B(2) C(4) D(5) A(5)	30
8	D(3) A(3) B(2) C(5)	36

Table 3. Genetic algorithm parameters.

Number of initial population	70
Generation gap	0.8
Crossover rate	0.7
Mutation rate	0.25
Maximum number of generations	300
Value for termination if minimum reached	1e-4
Insertion rate	0.9
Number of subpopulations	8
Migration rate	0.2
Number of generations between migration	20
Number of individuals per subpopulation	10
The length of chromosomes	64

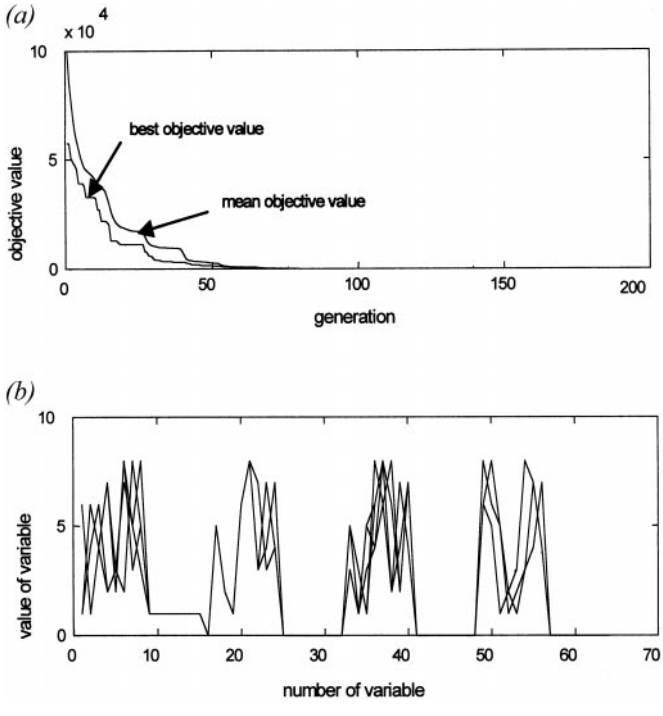

Fig. 7. (*a*) Best and mean objective value. (*b*) All individuals in generation 300.

The optimal solution found is as follows:

Job Sequence	Waiting Time
1 3 2 8 6 4 5 7	0 0 0 0 0 0 0 0
6 7 5 4 1 2 8 3	0 0 0 0 1 0 0 0
2 4 1 7 5 3 6 8	0 0 0 0 0 0 0 0
5 8 6 3 7 1 4 2	0 0 0 0 0 0 0 0

The optimal solution based on the genetic algorithm can be seen in Fig. 9.

Because there are a limited number of publications reporting the use of parallel multipopulation genetic algorithms for manufacturing scheduling, a comparative study of simulation results is difficult. In this case, results generated by traditional heuristics were chosen [17,18].

In the following, results from some studies applying traditional dispatching rules to similar scheduling requirements are considered:

EDD (earliest due date). In this case, priority is given to the job with the earliest due date.

SPT (shortest processing time). Priority is given to the job with the shortest processing time on the machine under consideration.

FCFS (first come first served). Priority is given to the processing of the job that arrived at the machine first.

LSF (least slack first). Priority is given to the processing of the job that has least slack. Slack is the difference between the due date and work remaining on the job. At time zero, the slack for job1 is $32 - (4 + 4 + 7 + 3) = 14$ h in the example.

LWR (least work remaining). Priority is given to the job with the least amount of total processing remaining to be done.

Each of the five rules can either be used individually or combined to yield five different schedules, which have been compared with the schedule created by the genetic algorithm approach and results are shown in Fig. 10.

It can be seen that the proposed genetic algorithm demonstrated higher reliability in handling dynamic scheduling requirements, and as a result produced improved solutions that outperformed the dispatching rules in their performance criteria.

6. Concluding Remarks

A genetic algorithm approach has been proposed to address the dynamic nature of manufacturing scheduling. Simulation and comparison results demonstrated that:

Genetic optimisation has been successfully implemented, providing a powerful tool for production application.

The model is capable of accommodating a variety of scheduling criteria without making many changes.

Comparisons with the schedules created by the dispatching rules indicate that the proposed genetic algorithm has successfully improved the solution.

The proposed model can deal with the problem expansion and jobs with a different number of operations. In addition, it has demonstrated a speedup in computation time and fewer objective function evaluations compared to a simple GA.

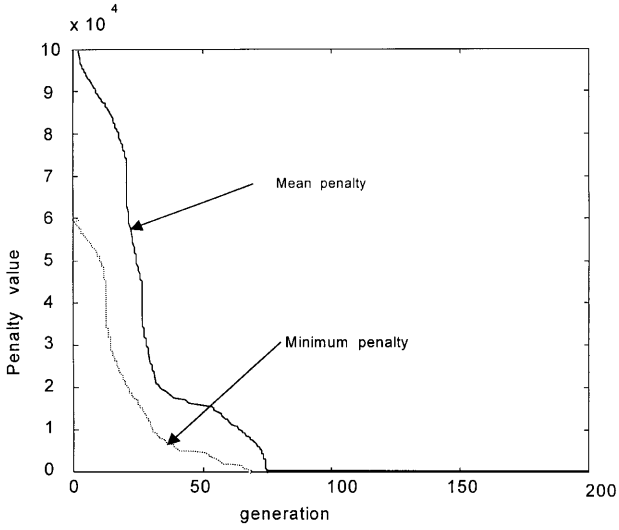

Fig. 8. The minimum and mean penalty.

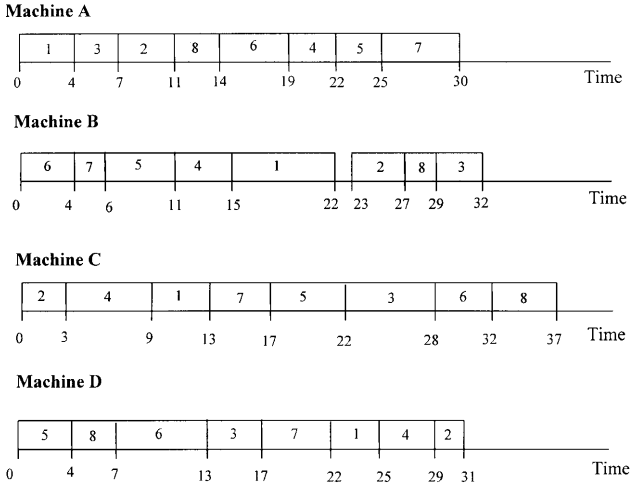

Fig. 9. The scheduling result using the genetic algorithm approach.

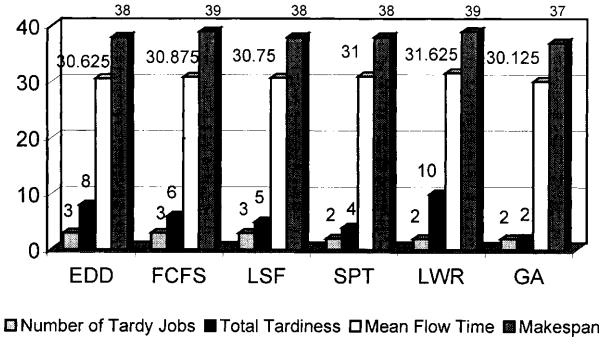

Fig. 10. A comparison of the scheduling rules and the genetic algorithm approach.

Acknowledgements

This research was supported by both the Engineering Research Center and the University Library at Glasgow Caledonian University.

References

1. J. H. Blackstone Jr, D. T. Phillips and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operation", International Journal of Production Research, 20(1), pp. 27–45, 1982.
2. R. Haupt, "A survey of priority rule-based scheduling", OR Spektrum, 11, pp. 3–16, 1989.
3. P. A. Newman and K. G. Kempf, "Opportunistic scheduling for robotic machine tending", Proceedings of IEEE Second Conference on Artificial Intelligence Applications: The Engineering of Knowledge-Based Systems, Miami Beach, FL, pp. 168–175, 11–13 December 1985.
4. A. Kusiak, "Designing expert systems for scheduling of automated manufacturing", Industrial Engineering, 19(7), pp. 42–46, 1987.
5. B. Y. Lee, S. Piramuthu and Y. K. Tsai, "Job shop scheduling with a genetic algorithm and machine learning", International Journal of Production Research, 35(4), pp. 1171–1191, April 1997.
6. L. Davis, "Job shop scheduling with genetic algorithms", Proceedings of the First International Conference on Genetic Algorithms, Pittsburgh, PA, Lawrence Erlbaum, pp. 202–207, 1985.
7. T. Starkwerther, S. McDaniel, K. Mathias, C. Whitley and D. Whitley, "A comparison of genetic sequencing operators", Proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, Morgan Kaufmann, pp. 69–76, 1991.
8. G. Syswerda, "Schedule optimization using genetic algorithms", in L. Davis (ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, pp. 332–349, 1991.
9. K. R. Baker, Introduction to Sequencing and Scheduling, John Wiley, 1974.
10. S. Y. Foo, Y. Takefuji and H. Szu, "Scaling properties of neural networks for job shop scheduling", Neurocomputing, 8(1), pp. 79–91, May 1995.
11. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
12. R. W. Cheng, M. S. Gen and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms-I. Representation", Computers and Industrial Engineering, 30(4), pp. 983–997, September 1996.
13. W. M. Spears and K. A. De Jong, "On the virtues of parameterised uniform crossover", Proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, Morgan Kaufmann, pp. 230–236, 1991.
14. D. Whitley, "The Genitor algorithm and selection pressure: why rank-based allocations of reproductive trials is best", Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann, pp. 116–121, 1989.
15. H. Pohlheim, GEATbx: Genetic and Evolutionary Algorithm Toolbox for Use with MATLAB, version 1.91, Technical University Ilmenau, Germany, July 1997.
16. N. Raman, F. B. Talbot and R. V. Rachamadugu, "Due date based scheduling in a general flexible manufacturing system", Journal of Operations Management, 8(2), pp. 115–132, April 1989.
17. L. Rabelo, Y. Yih, A. Jones and J. S. Tsai, "Intelligent scheduling for flexible manufacturing systems", Proceedings of IEEE International Conference on Robotics and Automations, vol. 3, Atlanta, GA, USA, pp. 810–815, 2–6 May 1993.
18. N. S. Hemant Kumar and G. Srinivasan, "A genetic algorithm for job shop scheduling – a case study", Computers in Industry, 31, pp. 155–160, 1996.

Nomenclature

C_{ik}	completion time of jobs i on machine k
t_{ijk}	processing time for operation (i,j,k)
C_{ih}	completion time of i on machine h
C_{pk}	completion time of p on machine k
t_{pqk}	processing time for operation (p,q,k)
y_{ipk}	operation sequence
N	arbitrary very large positive number
k_i	machine at which the last operation of job i is scheduled
C_{ik_i}	completion time of last operation of job i on machine k
$n_i(\mathbf{X})$	constraints
$\mathbf{X}$	an m vector
$f(\mathbf{X})$	objective function
$\theta(\mathbf{X})$	penalty function
r	penalty coefficient
J_{ki}	operation of a job processed by the kth machine
W_{ki}	upper limit of waiting time of the kth machine to process a job
m	number of machines
n	number of jobs