Journal of Manufacturing Engineering, September 2015, Vol. 10, Issue. 3, pp 166-170

PRODUCTION SCHEDULING PROBLEM SOLVING USING GENETIC AND GREEDY ALGORITHMS WITH SEQUENCE-DEPENDENT SET-UP TIMES

[*]Muthiah A and Ganesan K

Department of Mechanical Engineering, P.S.R.Engineering College, Sivakasi, Tamilnadu, India

ABSTRACT

In today's competitive markets, the importance of good scheduling strategies in manufacturing companies, lead to the need of developing efficient methods to solve complex scheduling problems. In scheduling attempt to fill the gap between scheduling theory and scheduling practice, with the aim to give answer to respond to market demand for more efficient method to solve complex scheduling problems. Although classical scheduling theory are one of the most studied field in Operations Research, some practical environments are often ignored in the classical models, since they improve the complexity of mathematical models. For discussion in the gap between scheduling theory and scheduling practice Main aim of this research work is to solve two production scheduling problems with sequence dependent setup times. The setup times are one of the most common complications in scheduling problems, and are usually associated with cleaning operations and changing tools and shapes in machines. The first problem considered is a single machine scheduling with release dates, sequence dependent setup times and delivery times. The performances measure is the maximum lateness. The second problem is a job shop scheduling problem with sequence dependent setup times where the objective is to minimize the make span. These two problems were addressed using genetic and greedy algorithm. There by, a highly efficient decoding procedure is proposed which strongly improves the quality of schedules. We present several priority dispatching rules for both problems, followed by a study of their performance.

Keywords: *Production Scheduling, Set-up Time, Genetic algorithm and Greedy algorithm.*

1. Introduction

Recent trends in scheduling attempt to fill the gap between scheduling theory and scheduling practice, with the aim to give answer to respond to market demand for more efficient method to solve complex scheduling problems. Although classical scheduling theory are one of the most studied field in Operations Research, some practical environments are often ignored in the classical models, since they improve the complexity of mathematical models. For a discussion in the gap between scheduling theory and scheduling practice see MacCarthy and Liu (1993).The setup times appear frequently in real scheduling problems and are one of the most frequent additional complications in scheduling. Moreover, these types of constraints are particularly relevant in production scheduling.

The setup time is defined as the time intervals between the end of job processing and beginning of next job. In this time interval no jobs can be processed in machine. The cleaning operations and changing tools and shapes are some examples of these setup times, and are frequent in manufacturing companies as commercial printing, plastics manufacturing, metal and chemical processing, paper industry, etc. The most complicated case is sequence- dependent setup times, where the setup time depends on the job previously scheduled. A typical example is the manufacturing of different colors of paint, Conway et al. (1967). In this case a cleaning operation time is needed, and is related with sequence of the colors processed. Another example is the extrusion machine for plastics films. The time spent in cleaning operations depends of film type and color. The trend in manufacturing of the production of small batches or unit products to satisfy demand and avoid inventory has made more relevant the scheduling problems with sequence-dependent setup times between all jobs, and not only between batches.

The aim of this paper is to study the performance of dispatching priority rules for the single-machine and job-shop scheduling problems with sequence dependent-dependent setup times and to indicate how to develop a good heuristic strategy to solve these problems in a practical and dynamic environment.

*Corresponding Author - E- mail: amuthiah68@gmail.com

1.1 Genetic algorithm

Genetic Algorithms are adaptive methods, which may be used to solve search and optimization problems (Beasley et al. (1993)). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection, i.e. *survival of the fittest*, first clearly stated by Charles Darwin in *The Origin of Species.* By mimicking this process, genetic algorithms are able to *evolve* solutions to real world problems, if they have been suitably encoded. Before a genetic algorithm can be run, a suitable *encoding* (or *representation*) for the problem must be devised. A *fitness function* is also required, which assigns a figure of merit to each encoded solution. During the run, parents must be *selected* for reproduction, and *recombined* to generate offspring. It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as *genes*) are joined together to form a string of values (*chromosome*). In genetic terminology, the set of parameters represented by a particular chromosome is referred to as an *individual*. The fitness of an individual depends on its chromosome and is evaluated by the fitness function. The individuals, during the reproductive phase, are selected from the population and *recombined*, producing offspring, which comprise the next generation. Parents are randomly selected from the population using a scheme, which favors fitter individuals. Having selected two parents, their chromosomes are *recombined*, typically using mechanisms of *crossover* and *mutation.* Mutation is usually applied to some individuals, to guarantee population diversity.

2. Problem Description

The problem proposed is a manufacturing system consisting of 6 machines M1, M2, …, M6 for which several suppositions are made. The machines cannot substitute each other and the processing units are non-preemptive. In this ideal system no machine breakdown occurs and the passing times of jobs between machines are neglected. In addition, job-specific setup times of machines are not considered. Generally, a production plan consists of *n* jobs, and each job consists of *mi* jobs, each of them having to be processed by a single machine. The production schedule represents an order of the tasks and the starting times for each task considering the technological machine order of jobs. The completion time of a task can be obtained by adding the processing time to its starting time. The input data for this problem is represented by the technological machine order of the task for each job and the corresponding processing time on the machines. The problem's constraints are represented by the operation sequence that is different for each product and by the fact that the machine can operate only one product at the time. The objective is to determine a schedule for the tasks on the machines in minimum time. The genetic algorithm parameters are: initial population size, maximum number of generations, maximum number of individuals, and crossover probability and mutation probability. The main objective of this algorithm is to identify the best plan, i.e. to perform all necessary tasks in order to minimize the makespan.

The genetic representation for each candidate solution is:
(G1, G2, G3, G4, G5, G6, G7, G8, G9, G10, G11, G12, G13, G14, G15),

Where G*i* is a structure that encodes the series number, the type and the starting times for each product series (there are 5 series for each type of products).

For example Gi=(si, pi, [t1, t2, t3, t4, t5]), represents a structure that refers to the si series of the product type noted with pi,, which accesses the first machine form the operation sequence at the time t1, the second machine form the operation sequence at the time t2, the third machine form the operation sequence at the time t3, and so on. An individual's performance will be evaluated by his fitness function value, which needs to be minimized. At the beginning, the fitness function is initialized with 0 (because the execution time at the beginning of the execution is equal to 0). The objective function minimizes the finishing time of tasks *n-1* (the last task), and therefore minimizes the makespan. The function value is updated according to both the ending time on the last machine, of the last product and several restrictions on total waiting times at machines and job sequence, by applying a set of specific rules.

The genetic operators selected for this implementation are:

- roulette selection;
- one-point crossover;
- rotation mutation.

By using the roulette selection, the future parents are chosen by simulating the launching of a roulette needle on the field of fitness values for the current population. The one-point crossover copies the string from the beginning of the chromosome to the crossover point from one of the parents and the rest is copied from the other parent. The construction of the new individual is made considering the restrictions imposed by the problem. The rotation mutation performs a small chromosome modification by selecting a bloc of genes with random length and by inversing the gene order. This modification is made considering the restrictions imposed by the problem. The genetic algorithm stops when either the upper limit of

generation or the maximum number of individuals has been reached. The individuals with the lowest value of the fitness function represent the solution returned by this algorithm. This individual represents the production scheduling with minimum makespan.

3. Experimental Results

To test the genetic algorithm results, we propose the production simulation of 15 series of products (5 for each of the three products types). The machines sequences for each product and the processing time on each machine for each product type are presented in Table 1. The tasks sequence and the necessary time for each job are different for each of the three types:

The task sequence for the first type of products (called Product 1) is represented by (M1, M2, M3, M5, M6), where Mi represents machine number i;

The task sequence necessary to obtain the second type of products (called Product 2) is represented by (M1, M3, M4, M5, M6);

The task sequence necessary to obtain the third type of products (called Product 3) is represented by (M1, M4, M5, M6);

Table 1. Machine sequences and processing time

Machine No	parameter	Product 1	Product 2	Product 3
Machine 1	Task Number	1	1	1
	ExecutionTime	30	10	20
Machine 2	Task Number	2	-	-
	ExecutionTime	10	-	-
Machine 3	Task Number	3	2	-
	Execution Time	10	10	-
Machine 4	Task Number	-	3	2
	ExecutionTime	-	10	40
Machine 5	Task Number	4	4	3
	ExecutionTime	30	30	20
Machine 6	Task Number	5	5	4
	ExecutionTime	10	10	10

Three sets of experimental values for the specific parameters of the genetic algorithm (initial population size, maximum population size, maximum generation number, crossover probability, and mutation probability).

Twenty tests have been executed for each set of input values, and the final results are synthesized in Table 3. Best performances average represents the mean of lowest values of the fitness function obtained in the 60 sets of results. Worst performances average at last generation represents the average of highest values of the fitness function. The average performance averages is calculated considering the values of the medium performances obtained during the experimental tests.

Table 2. Experimental values set

Values Set/ Parameters	1	2	3
Number of tests	20	20	20
Initial population size	10	15	20
Maximum population size	20	30	50
Maximum generation number	10	15	20
Crossover probability	0.5	0.3	0.6
Mutation probability	0.07	0.02	0.15

Table 3. Final results

Final results	
Best performance average at last generation	455
Worst performance average at last generation	703
Average performance	579
Best performance	470
Best performance solutions number	1
Best performance solution	(0, 10, 3, 14, 13, 6, 5, 9, 8, 1, 12, 2, 4, 7, 11)

After completing all the experimental tests, the lowest value obtained for the fitness function is 470 and only one solution has managed to reach this value. This solution represents a scheduling plan that manages to respect both the planned production and the 480 minutes limit imposed by the problem.

The solution with the minimum fitness function returned is: (0, 10, 3, 14, 13, 6, 5, 9, 8, 1, 12, 2, 4, 7, 11) and represents the technological order of production series. For each series a set of detailed specifications (the starting times on each machine, the total execution time and the completion times on each resource) is stored in a text file. The solution returned by the algorithm can be decoded as: first series of products that enters the system is Series 0 (that corresponds to the Product 1), after that follows the Series 10 (that corresponds to Product 3), the Series 3 (that corresponds to Product 2), and so on. The list of values that correspond to the moments of time in which each series accesses the machines is presented in a text file that contains the detailed solution.

3.1 Greedy algorithms

In an optimization problem, we are given an input and asked to compute a structure, subject to various constraints, in a manner that either minimizes cost or maximizes profit. Such problems arise in many applications of science and engineering. Given an optimization problem, we are often faced with the question of whether the problem can be solved efficiently

(as opposed to abrute-force enumeration of all possible solutions), and if so, what approach should be used to compute the optimal solution? In many optimization algorithms a series of selections need to be made. Today we will consider a simple design technique for optimization problems, called greedy algorithms. Intuitively, a greedy algorithm is one that builds up a solution for some problem by "myopically" selecting the best alternative with each step. When applicable, this method typically leads to very simple and efficient algorithms. The greedy approach works for a number of optimization problems, including some of the most fundamental optimization problems in computer science (minimum spanning trees, for example). Thus, this is an important algorithm design technique to understand. Even when greedy algorithms do not produce the optimal solution, they often provide fast heuristics (nonoptimal solution strategies) and are often used in finding good approximations. In this lecture we will discuss some examples of simple scheduling problems that have efficient greedy.

3.2 Problem description

The Simple flow Shop has x processing centers which are P1, P2, ……, Pm where processing center Pj have pj parallel processors; there are n jobs, and each job must be processed by any processor of processing centers P1, P2, ……, Pm in order and the processing time of job in the processing center is vector Jj, where Jj =(j1, j2,……, jm). On processing center can only process one job at the same time. One processing center consists of one or many parallel processors, all of which can work at the same time, processing different jobs. All the processes of each job must be done in order, and a process shouldn't be started until its pervious process is finished and one process can be done by any parallel processor of the corresponding processing center.

The following requirements must be met when we adopt Greedy method to solve flow shop scheduling:

- Every processor must record a last completion time.
- Every job must also record its last completion time in its pervious process to ensure the beginning time for process must equals to or later than completion time of the previous process.

3.3 Greedy solving strategies

- Let the current processing center be the first processing center, the current processor be the first processor of the processing center and the current process of the job be the first process.
- Choose the job which is finished earliest currently and place it into the current processor and then let the current processor be the next processor of the current processing center; Repeat the processes until the placement of all the jobs are finished, and then start the next process.
- Let the process be the next process. Return to b) and execute again until all the processes are finished. The algorithm is finished.

3.3 Greedy algorithm steps for this problem

Step 1: Initialization of maximum number of job.

Step 2: Schedule the jobs according to their completion times.

Step 3: Initialization the number of processor in the job.

Step 4: After scheduling all the jobs, initiate the position of job by use of number of parallel processor and number of processor.

Step 5: Position of job will find by processing time of job. This will be continued until reached the maximum number of processor.

Suppose there are n jobs, each of which has m processes, that is, there are m processing centers, each of which has z1, z2,……, zx processors and the processing time for each job at every processing center is t1, t2, ……, tn and then the time complexity of Greedy Algorithm is

$$x * (c + s + n) \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Where c is a constant, which is mainly determined by P, the number of parallel processors of all the processing centers; s is the time complexity of the scheduling algorithm. If the scheduling algorithm adopt quick sorting, the expected time complexity is O(n*log2 n) and the equation (1) transformed into :

$$x * (c + s + n) = O(x*n) \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Therefore, the time complexity for Greedy Algorithm is O(m*n).

In the common productions, the number of processing centers is not more than 20, each of which consists of 10 processors or less and the number of jobs is commonly about 20-30. In this way, x=20, n=30, P = 10+ 10+ ……+10=200, the scheduling time is about 172, and therefore, the time of Greedy Algorithm is about 10*(1+172+30) = 2030.

There are big difference between the approximate solutions acquired by Greedy Algorithm and the optimal solutions. But it can be adopted to small scale production because of its low time complexity.

4. Conclusion

In this paper, we presented the simple genetic and greedy algorithms that are capable of finding good solutions for production scheduling problem. The genetic algorithms are not well-suited for fine-tuning of solutions. This algorithm would be an upgrade to a hybrid algorithm. Quality of final solution depends on the initial population and pure chance. So that search procedure has to be repeated several times for a specific problem. Greedy Algorithm also adopted to simulate approximate solution. There is difference between the approximate salutation and theoretical solution. But it can be adopted for small-scale production due to its extremely low time complexity.

References

1. *Adams J Balas E and Zawack D (1988), "The Shifting Bottleneck Procedure for Job Shop Scheduling", Management Science, Vol. 34(3), 391-401.*
2. *Barman S (1997), "Simple priority rule combinations: an approach to improve both flow time and tardiness", International Journal of Production Research, Vol.35 (10), 2857-2870.*
3. *Beasley J E (1990), OR-Library: "Distributing Test Problems by Electronic Mail", Journal of the Operational Research Society, Vol. 41(11), 1069-1072.*
4. *Bratley P Fox B L and Schrage L E (1983), "A guide to Simulation", New York: Springer- Verlag.*
5. *Bruno J and Downey P (1978), "Complexity of Task Sequencing With Deadlines", Setup Times and Changeover Costs. SIAM Journal of Computing, Vol.7, 393-404.*
6. *Carlier J (1982), "The one-machine sequencing problem", European Journal of Operational Research,Vol. 11, 42-47.*
7. *Conway R W Maxwell W L and Miller L W (1967), "Theory of Scheduling", eading, Massachussets: Addison-Wesley.*
8. *Fisher H and Thompson G L (1963), "Probabilistic learning combinations of local job-shop scheduling rules", Industrial Scheduling. (Englewood Cliffs, New Jersey: Prentice Hall: 225-251).*
9. *Garey M R and Johnson D S (1979), "Computers and Intractability: A Guide to the Theory of NP-ompleteness", San Francisco: Freeman.*
10. *Grabowski J E Nowicki E and Zdrzalka S (1986), "A block approach for single machine scheduling with release dates and due dates", European Journal of Operational Research, Vol. 26, 278-285.*
11. *Gupta J N D (1988), "Single facility scheduling with multiple job classe",. European journal Operations Research, Vol.8, 42-45.*
12. *Hoogeveen J A Lenstra J K and van de Velde S L (1997), "Sequencing and scheduling: an annotated bibliography, Memorandum COSOR 97-02", Eindhoven University of Technoloy, Eindhoven, The Nederlands.*
13. *Ríos R and Bard J (1997), "A branch-and-bound algorithm for flowshop scheduling with setup times", Technical Report ORP97-02, University of Texas, Austin.*
14. *Schrage L (1971), "Obtaining optimal solutions to resource constrained network scheduling problem", unpublished manuscript.*
15. *Storer R H Wu S D and Vaccari R (1992), "New search spaces for sequencing instances with application to job-shop scheduling", Management Science, Vol. 38, 1495-1509.*
16. *Taillard E (1993), "Benchmarks for basic scheduling problems", European Journal of Operational Research,Vol. 64, 278-285.*
17. *Williams D and Wirth A (1996), "A New Heuristic for a Single Machine Scheduling Problem with Setup Times", Journal of the Operations Research Society, Vol. 47, 175-180.*
18. *Florentina Alina Chircu (2010), Using Genetic Algorithms for Production Scheduling, Petroleum-Gas University of Ploiesti, Informatics Department, Ploieşti, Romania.*
19. *Ashwani Dhingra, Pankaj Chandna, Hybrid Genetic Algorithm for Multicriteria Scheduling with Sequence Dependent Set up Time, Haryana,India.*
20. *João António Noivoa and Helena Ramalhinho-Lourençob, Solving two production scheduling problems with sequence-dependent set-up times, Departamento de Electrónica Industrial, Universidade do Minho, Campus de Azurém, 4800Guimarães, Portugal.*
21. *Man K F Tang K S and Kwong S (1996), Member IEEE, Genetic Algorithms: Concepts and Applications.*
22. *Falkenauer E and Bouffouix S, "A Genetic Algorithm for Job Shop", CRIF - Research Center for Belgian Metalworking Industry, Brussels, Belgium.*
23. *Ruben Ruiz Thomas St Äutzley, "An Iterated Greedy Algorithm for the Flowshop Problem with Sequence Dependent Setup Times", Valencia, Spain.*
24. *Avi Dechter and Rina Dechter (1989), "On the Greedy solution of Ordering Problems", CA.*
25. *Naderi B and Rubén Ruiz, "The distributed permutation flowshop scheduling problem"*

