Proceedings of the 2nd IMT-GT Regional Conference on Mathematics, Statistics and Applications
Universiti Sains Malaysia, Penang, June 13-15, 2006

A JOB-SHOP SCHEDULING PROBLEM (JSSP) USING GENETIC ALGORITHM (GA)

Mahanim Omar, Adam Baharum, Yahya Abu Hasan

School of Mathematical Sciences, Universiti Sains Malaysia
11800 Penang, Malaysia
Tel: (+) 04-653-3964
mahanim_omar@yahoo.com, adam@cs.usm.my, ahyahya@cs.usm.my

Abstract: *The job-shop scheduling (JSS) is a schedule planning for low volume systems with many variations in requirements. In job-shop scheduling problem (JSSP) environment, there are j jobs to be processed on m machines with a certain objective function to be minimized. JSSP with j jobs to be processed on more than two machines have been classified as a combinatorial problem. They cannot be formulated as a linear programming and no simple rules or algorithms yield to optimal solutions in a short time. In this paper we used genetic algorithm (GA) with some modifications to deal with problem of job shop scheduling. GA once proposed by John Holland is a stochastic search technique based on Darwin's principle of the survival of the strongest. In this paper, we generated an initial population randomly including the result obtain by some well known priority rules such as shortest processing time and longest processing time. From there, the population will go through the process of reproduction, crossover and mutation to create a new population for the next generation until some stopping criteria defined were reached. In this paper, we used the number of generations as a stopping criteria. In crossover and mutation, we used the critical block neighbourhood and the distance measured to help us evaluate the schedules. Result has shown that the implementation of critical block neighbourhood and the distance measure can lead us to the same result obtain by other methods.*

KEY WORDS: Job-shop scheduling problem (JSSP), genetic algorithm (GA), CB Neighbourhood, DG distance.

Introduction

Job-shop is a system that process n number of tasks on m number of machines. In this type of environment, products are made to order and in a low volume. Usually, these orders are differ in term of processing requirements, materials needed, processing time, processing sequence and setup times. Job-shop problems are widely known as a NP-hard problem. Nowadays, search algorithms based on branch and bound methods and several approximation algorithms have been developed. However, result from the branch and bound method sometimes is really unpredictable and requires a lot of time. It depends on the size of the problem. Thus, schedulers are usually satisfied with an acceptance result which is not far from optima. One of the widely used technique in industries is the local search. One of the search techniques that have been used is genetic algorithm (GA).

Genetic algorithms solve a problem using the principal of evolution. In the search process it will generate a new solution using genetic operator such as selection, crossover and mutation. In hill-climbing, the search procedure will stop once it detects no improvement in next iteration. This criterion make the hill-climbing technique tend to stop at local optima. In the other hand, genetic algorithms start its search space in a population and will maintain the number of population in iteration. It will generate a new schedule by selecting two individuals in population to apply crossover and mutation. There are many procedures that could be applied in the

selection, crossover and mutation process. Some of the procedures are not suitable for job-shop problem and some of them will make the search stop at local optima.

Our intention in this research is to find out if the idea of combining the CB neighbourhood and DG distance in crossover and mutation is suitable when dealing with job-shop scheduling problems so that the makespan value can be minimized. Result has shown that if the solution converges too quickly, it will stop at local optima. The modification has been made so that it will get a solution at least not far from optima.

Job Shop Scheduling Problem (JSSP)

Recently, researchers have been focusing on investigating machine scheduling problems in manufacturing and service environments where jobs represent activities and machines represent resources, and each machine can process one job at a time. In this paper, we will focus on the low volume system also known as job-shop. In this type of environment, products are made to order. The job-shop scheduling problem (JSSP) can be described as a set of n jobs denoted by J_j where $j = 1,2,...,n$ which have to be processed on a set of m machines denoted by M_k where $k = 1,2,...,m$. Operation of j th job on the k th machine will be denoted by O_{jk} with the processing time p_{jk}. Each job should be processed through the machines in a particular order or also known as technological constraint. Once a machine starts to process a job, no interruption is allowed. The time required for all operations to complete their processes is called makespan. In this paper, our intention is to minimize this makespan value. When we minimizing the makespan, at least one of the optimal solutions is a semi-active (no operation can started earlier without violating the technological constraints (French, 1982). For this reason, every time when makespan is optimised, a schedule can be described by the processing orders of operations on the machines (Jensen and Hansen, 1999).

Genetic Algorithms

In scheduling, genetic algorithms represent schedules as individuals or a population's members. Each individual has its own fitness value which is measured by the objective function. The procedure works iteratively, and this iteration is a generation. Each generation consists of individuals who survive from the previous generations. Usually, the population size remains constant from one generation to the next generation.

Critical Block and DG distance

In job-shop scheduling problem, we also could find the critical path. Critical path is defined as the longest paths taken from the first operation processed until the last operation leaves the workplace. All operations in this path are called critical operations. Plus, the critical operations on the same machine are called critical block.

We measure the distance between two schedules S_1 and S_2 by the number of differences in the processing orders of operation on each machine (Mattfeld, 1996). By doing so, it is just like summing the disjunctive arcs whose directions are different between schedules S_1 and S_2. This distance also known as disjunctive graph (DG) distance.

Neighbourhood Search

In combinatorial problems, the most widely used technique is neighbourhood search. A solution S is represented as a point in the search space. The feasible solutions in the neighbourhood that can be reached from S with exactly one transition are defined by $N(S)$. Neighbourhood search is categorized according to the given criteria for selecting a new point from the neighbourhood (Yamada and Nakano, 1995). We defined our neighbourhood if the DG distance between

S and $y \in N(S)$ is equal to one. Another type of transition that is also well known is called adjacent swapping (AS). This transition operator exchanges a pair of consecutive operations only on critical path to form neighbourhood. According to Yamada and Nakano (1997), another transition operator that is more powerful than AS is called Critical Block neighbourhood (CB neighbourhood). CB neighbourhood permutes the order of operations in critical block by moving the operation to the beginning or end of the critical block

Data Structures and Representation

Genetic algorithms process populations of strings. We construct a population as an array of individuals where each individual contains the *phenotype*, *genotype* (artificial chromosome) and the *fitness* (objective function). In job-shop scheduling problem we noted the job order on each machine as *phenotype*, the machine schedule as *genotype* and makespan value as *fitness*.

The representation used in this paper is called the *permutation representation* (Yamada and Nakano, 1997). JSSP can be viewed as an ordering problem just like the Travelling Salesman Problem (TSP). A schedule can be represented by the set of permutations of jobs on each machine which are called *job sequence matrix*. Figure 1 shows the *job sequence matrix* for 3×3 problem. Rows will represent the machines and columns represent the order of jobs.

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

Figure1: A *job sequence matrix* for 3×3 problem.

Initial population

Most of the research papers in this area prefer to generate the initial population randomly. This technique will allow the search process in a wide space. However, this kind of technique will take a lot of time to get the optimal value. In this paper, we chose to start the initial population with a randomly generated schedules, including some of the schedules obtain by the well known priority rules such as the shortest processing time and the longest processing time.

Parents Selection

In parent selection we choose two individuals to reproduce. There are many ways of choosing *parents* to evaluate. In order to avoiding the premature convergence, we randomly choose two *individuals* from the population and called them *parent1* and *parent2*. This means that all individuals in the population have the same chances to reproduce.

Fitness Value

In this research, we focus on the static and deterministic environment. In other words, the number of jobs and their processing time are known, fixed and we assumed that there are no machines breakdown occurred. We used makespan of the operation as fitness value. Makespan is denoted as $C_{\max}$ is the time when the last operation leaves the workplace.

$$C_{\max} = \max(C_1, C_2 ....., C_n)$$

where,

$$C_j = r_j + \sum_{k=1}^{n} (W_{jk} + p_{jm(k)})$$

C_j is the completion time of job j, r_j is the release time of job j, W_{jk} is the waiting time of job j at sequence k and $p_{jm(k)}$ is the processing time needed by job j on machine m at sequence k.

Crossover

Yamada and Nakano (1997) in most of their papers have introduced plenty of techniques that could be use in solving the job-shop problem. One of them is by making use of CB neighbourhood and DG distance. The idea of this technique is to evaluate a point x by the distance $d(x, p_2)$. Lets denote *parent1* and *parent2* as p_1 and p_2. First, set $x = p_1$. Then, we generated the CB neighbourhood for x, $N(x)$. For each member, y_i, in $N(x)$ we calculated the distance between the members and p_2 to produce $D(y_i, p_2)$. Then, we sort $D(y_i, p_2)$ in ascending order. Starting from the first index in $D_{sort}(y_i, p_2)$, we accepted y_i with probability one if the fitness value is less than the current fitness value $V(y_i) \le V(x)$. Otherwise, we accepted it with probability 0.5. Starting from p_1, we modified x step by step approaching p_2. After some iteration, we will find that x will gradually loses p_1's characteristics and started to inherit p_2's characteristics although in a different ratios. We choose the *child* depending on the less DG distance between the *child* and both its parents. The algorithm for this procedure is shown in Algorithm 1.

Algorithm 1: Crossover

1. Let p_1 and p_2 be the parent solution.
2. Set $x = p_1 = q$.
3. Find CB Neighbourhood for x, $N(x)$.
4. Do
   a. For each member $y_i \in N(x)$, calculate the distance between y_i and p_2, $D(y_i, p_2)$.
   b. Sort the distance value in ascending order, $D_{sort}(y_i, p_2)$.
   c. Starting from $i = 1$, do
      i. Calculate the fitness value for y_i, $V(y_i)$.
      ii. If $V(y_i) \le V(x)$ accept y_i with probability one, and with probability 0.5 otherwise.
      iii. If y_i is not accepted, increase i by one.

   Repeat i-iii until y_i is accepted.
   d. Set $x = y_i$
   e. If $V(x) \le V(q)$ then set $q = x$.

   Repeat 3-4 until number of iterations.
5. q is the child.

Mutation

Instead of using some random probability, we apply mutation if the DG distance between *parent1* and *parent2* are less than some predefined value. It is also defined based on the same idea as crossover. However, we choose the *child* which has the largest distance from the neighbourhood. The algorithm for mutation is shown in Algorithm 2.

Algorithm 2: Mutation

1 Set $x = p_1$.

2 Find Neighbourhood for x, $N(x)$

3 Do a. For each member $y_i \in N(x)$, calculate the distance between y_i and p_1, $D(y_i, p_1)$.

b. Sort the distance value in descending order, $D_{sort}(y_i, p_1)$.

c. Starting from $i = 1$, do

i. Calculate the fitness value for y_i, $V(y_i)$.

ii. If $V(y_i) \le V(x)$ accept y_i with probability one, and with probability 0.5 otherwise.

iii. If y_i is not accepted, increase i by one.

Repeat i-iii until y_i is accepted.

d. Set $x = y_i$.

e. If $V(x) \le V(q)$ then set $q = x$.

Repeat 2-3 until number of iteration.

q is the child.

Acceptance Criterion

The final and the most important step in the GA procedure is to choose the individual to be replaced by *child*. It is widely known that we always choose the fittest individual to reproduce in the next iteration. In Yamada and Nakano (1997), they did not consider or choose the child with the same fitness value with other population members. However, by not even considering that child, we may lose the good individual without considering its abilities to be evaluated. So, in this paper, after considering the worst individual in the population, we also consider if the child has the same fitness value with the member of population. Instead of dropping that child, we replaced the old one with the child assuming that we have given chance for the old individual to reproduce. Noted that we could not take both of the individuals to avoid having problem later, we might have problem falling in the local optima. The algorithm for the whole procedure is shown in Algorithm 3.

Algorithm 3: Genetic Algorithm

1. Initialize population: Randomly generated a set of 10 schedules including the schedules obtained by some priority rules.
2. Randomly select two schedules, named them as p_1 and p_2. Calculate DG distance between p_1 and p_2.
3. If DG distance is smaller from some predefined value, apply Algorithm 2 to p_1. Generate child. Then go to step 5.
4. If DG distance is large, we apply Algorithm 1 to p_1 and p_2. Generate child.
5. Apply neighbourhood search to child to find the fittest child in the neighbourhood. Noted it as child'.
6. If the makespan for the child' is less than the worst and not equal to any member of population, replace the worst individual with child'. If there is a member having the same makespan value, replace the member with the child'.
7. Repeat 2-6 until some termination condition are satisfied.

Results and Discussions

Consider a 5 jobs and 5 machines problem with the operation sequence and the processing time for each operation have been determined in Table1 and Table2.

We run the programme for five times using the population size = 10, number of iterations for mutation is 100 and crossover is 200. The algorithm was terminated after 200 generations. From the result in Table3, it can be shown that the combination of critical block, DG distance and genetic algorithm could provide a result as good as other methods. From Table 3, we could see that the last job processed is job 5 on machine 4. So, our makespan value for this problem is 34. The result also gives us the job sequence for each machine to process, the starting time and the finish time for each operation. For example, on machine 1, we start to process job 3 at time 0 and finished at 7. Then we process job 1, followed by job 4, job 5 and job 2.

Table1: Operation Sequence

	Sequence1	Sequence2	Sequence3	Sequence4	Sequence5
Job1	Machine3	Machine1	Machine2	Machine4	Machine5
Job2	Machine2	Machine3	Machine5	Machine1	Machine4
Job3	Machine1	Machine5	Machine4	Machine3	Machine2
Job4	Machine4	Machine3	Machine2	Mahine1	Machine5
Job5	Machine5	Machine3	Machine1	Machine2	Machine4

Table2: Processing Time for each Operation

	Machine1	Machine2	Machine3	Machine4	Machine5
Job1	8	4	2	6	7
Job2	3	6	5	2	4
Job3	7	3	9	4	8
Job4	4	5	5	4	3
Job5	3	6	7	4	5

Table3: Result

	Processing Time	Start	Finish
Machine1			**25**
Job3	7	0	7
Job1	8	7	15
Job4	4	15	19
Job5	3	21	24
Job2	3	24	27
Machine2			**24**
Job2	6	0	6
Job4	5	9	14
Job1	4	15	19
Job5	6	24	30
Job3	3	30	33
Machine3			**28**
Job1	2	0	2
Job4	5	4	9
Job2	5	9	14
Job5	7	14	21
Job3	9	21	30

Continued Table 3.

Machine4			**20**
Job4	4	0	4
Job3	4	15	19
Job1	6	19	25
Job2	2	27	29
Job5	4	30	34
Machine5			**27**
Job5	5	0	5
Job3	8	7	15
Job2	4	15	19
Job4	3	19	22
Job1	7	25	32

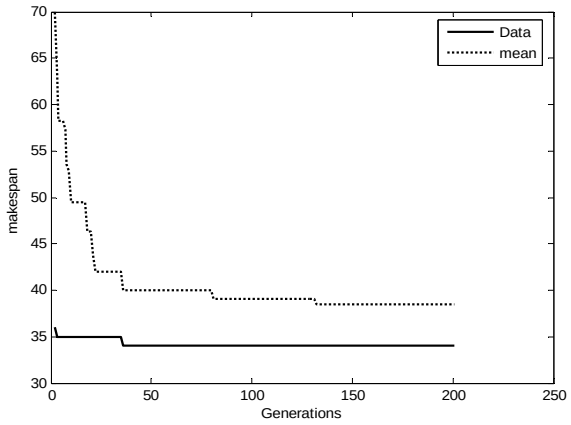

Fig . 1.1 . Result obtained using the combination of randomly generated schedules and priority rule schedules as initial population.

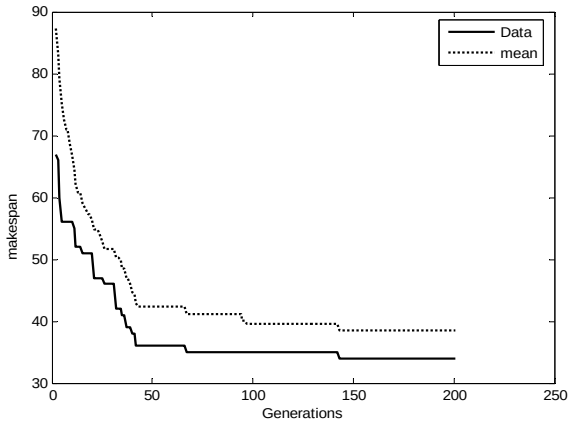

Fig . 1.2 . Result obtained using the randomly generated schedules as initial population.

We applied both types of initial population to the data. First we used the combination of schedules we generated using the priority rules and the randomly generated schedules as the initial population. From the five runs, we find the optimum before the generation exceeded 100.
As for Figure 1.2, we found the optimum value at generation 139. From the five runs, we could conclude that if we used the randomly generated schedules as the initial population, we will only find the optimum value at generation larger than 100. However, both results gave the same makespan value which is 34.

Conclusion

The study on GA and job shop scheduling problem provides a rich experience for the constrained combinatorial optimization problems. Application of genetic algorithm gives a good result most of the time. Although GA takes plenty of time to provide a good result, it provides a flexible framework for evolutionary computation and it can handle varieties of objective function and constraint.
For further research, the technique in this paper would be applied to a larger size problem to see how it performed.

References

Ab. Rahman Ahmad and Faisal. RM *Job Shop Scheduling using GA*. National Conference Management Science and Operations Research 2003. (2003).

C. Bierwirth , D. C. Mattfeld and H. Kopfer. *On Permutation Representations for Scheduling Problems.* Parallel Problem Solving from Nature IV, Springer-Verlag, pp . 310-318 (1996).

G. Mintsuo and C. Runwei.. *GA and Engineering Design*, John Willey & Sons Inc, New York USA.(2002).

J. Adams, E. Balas and D. Zawack *The Shifting Bottleneck Procedure for Job Shop Scheduling*. Management Science vol.34. (1988).

J.F. Gonçalves, , Jorge José de Magalhães Mendes and M. C. Resende. *A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem.*.AT&T Labs Research Technical Report TD-5EAL6J, September 2002. (2002).

M. Pinedo and X. Chao. *Operation Scheduling with Applications in Manufacturing and Services.* McGraw-Hill International Editions. (1999).

M. T. Jensen and T. K. Hansen. *Robust Solutions to Job Shop Problems.* Proceedings of the 1999 Congress on Evolutionary Computation, pages 1138-1144. (1999).

S. French. *Sequencing and Scheduling : An Introduction to the Mathematics of the Job Shop.* John Willey & Sons Inc, New York USA.(1982).

T. Yamada and R. Nakano. *Genetic Algorithms for Job-shop Scheduling Problems*. Proceedings of Modern Heuristic for Decision Support. Pp. 67-81, UNICOM Seminar, 18-19 March 1997,London. (1997)

T. Yamada and R. Nakano. *Scheduling by Genetic Local Search with Multi-Step Crossover.* The Fourth International Conference on Parallel Problem Solving from Nature, Berlin, Germany. (1996).

T. Yamada and R. Nakano. *A Genetic Algorithm with Multi-Step Crossover for Job- Shop Scheduling Problems.* International Conference on Genetic Algorithms in Engineering Systems: Innovations and Application (GALESIA ’95). (1995)