Single Objective Evolutionary Algorithm for Flexible Job-shop Scheduling Problem

[1]M. Nagamani, [2]Dr. E. Chandrasekaran, [3]Dr. D. Saravanan
[1]Research Scholar, Presidency College, Presidency College, University of Madras, Chennai,
[2]Associate Professor of Mathematics, Presidency College, Chennai
[3]Professor of Mathematics, Karpaga Vinayaga College of Engineering and Technology, Chennai

[1] ***corresponding author***

Abstract - A meta-heuristic approach for solving the flexible job-shop scheduling problem (FJSP) is presented in this study. This problem consists of two sub-problems, the routing problem and the sequencing problem and is among the hardest combinatorial optimization problems. We propose a Evolutionary Algorithm (EA) for the FJSP. Our algorithm uses several different rules for generating the initial population and several strategies for producing new population for next generation. Proposed EA is tested on benchmark problems and with due attention to the results of other meta-heuristics in this field, the results of EA show that our algorithm is effective and comparable to the other algorithms.

I. INTRODUCTION

Flexible Job-shop Scheduling Problem is expansion of the traditional Job-shop Scheduling Problem that an operation can be processed one or more machines. Since FJSP belongs to the category of NP-hard problems, heuristic methods had been a predominant choice for the researchers over the traditional mathematical techniques. The heuristic approaches for solving FJSP are generally classified as hierarchical and integrated approaches. In hierarchical approaches, assignment of operations to machines and the sequencing of operations on the machines are treated separately, i.e. assignment and sequencing are considered independently. In integrated approaches assignment and sequencing are not differentiated and considered together.

FJSP can decompose into two sub-problems: Assigning the operations to machines (the routing problem) and sequencing the operations on the machines (the sequencing problem) in order to minimize the performance indicators. Then, FJSP is more difficult than the classical JSP because it contains an additional problem that is assigning operations to machines.

The purpose of this problem is to look for the smallest makespan. For that purpose, it is necessary to decide optimal assignment of machines to operations and order of operations on machines. In this paper, we focus on maximum of workloads for all machines and propose new survival selection, creation method of initial solution, mutation, and escape method to Evolutionary Algorithm for this problem. The features of our method are shown below.

1. In the proposed survival selection, the combination of the machine with the possibility of the improvement of the schedule is made to be able to be left for the population.
2. In the proposed creation method of initial solution, we improve the efficiency of the search by making the initial solution that decreases maximum of workloads.
3. In the proposed mutation, We do not decrease maximum of workloads, and do not change it. As a result the search accuracy is improved.
4. In the proposed escape method, it escapes from the local solution.

II. PROBLEM DEFINITION

The FJSP can be explained as follows. It is given a set of jobs, $J = \{J_1, ..., J_n\}$, and a set of machines, $A = \{M_1, ..., M_m\}$, such that each job J_i consists of a sequence of n_i operations, $O_{i,1}, O_{i,2}, ...,Oi,n_i$. Job J_i is completed when its operations performed one after another in the given order. For each operation of job J_i, $O_{i,j}$, there are a set of allowable machines called $A_{ij} \subseteq A$, which $O_{i,j}$ can be performed on one of them such as $M_k \varepsilon A_{ij}$. The FJSP is machine-dependent because the performance of each operation on each allowable machine has a different processing time like $p_{ijk}>0$. Flexibility of problems can be categorized into partial flexibility and total flexibility. It is partial, when at least one of A_{ij} be proper subset of A ($A_{ij} \subset A$) and it is total, when we have $A_{ij} = A$ for all operations. Operations execute on machines without preemption and machines can perform at most one operation at a time. All jobs and machines are available at time 0. An example is given in Table 1. Each row refers to an operation, each column refers to a machine and cells are processing times. Notice that this example has partial

flexibility and unallowable machines for each operation denoted by ∞ in processing times table (Table 1).

III. THE SOLUTION ALGORITHM

In this study, a evolutionary algorithm for solving the flexible job-shop scheduling problem (FJSP) is presented. This algorithm uses several different rules for generating the initial population and several strategies for producing new population for next generation.

***Evolutionary algorithm*:** Different main schools of evolutionary algorithms have evolved during the last 40 years. Genetic algorithms, mainly developed in the USA by J. H. Holland, evolutionary strategies, developed in Germany by I. Rechenberg and H.-P. Schwefel and evolutionary programming. Each of these constitutes a different approach, however, they are inspired by the same principles of natural evolution. These algorithms have been applied in a number of fields, e.g., mathematics, engineering, biology, and social science (Goldberg, 1989). GAs are intelligent stochastic optimization techniques based on the mechanism of natural selection and genetics. GAs start with an initial set of solutions, called population. Each solution in the population is called a chromosome (or individual), which represents a point in the search space. A chromosome consists of some genes. GAs work iteratively, each single iteration is called a generation. At each generation, the fitness of each chromosome is evaluated, which is decided by the fitness function, and the chromosome is stochastically selected for the next generation based on its fitness. New chromosomes, called offspring (or children), are produced by two genetic operators, crossover and mutation. The offspring are supposed to inherit the excellent genes from their parents, so that the average quality of solutions is better than that in the previous generations. This evolution process is repeated until some termination criteria are met.

Proposed EA approach: The proposed EA steps are as follows:

Step 1: Initialization:

(a) Parameters setting: set the number of initial population (popsize), number of generation(ng), percent of Assignment Rule 1(pa1), percent of Assignment Rule 2 (pa2), probability of each crossover (pc), and probability of each mutation (pm)

(b) Initial population generation

(b1) Initial assignments

(i) Generate $pa1_*$ popsize initial assignments by Assignment Rule 1

(ii) Generate $pa2_*$ popsize initial assignments by Assignment Rule 2

(b2) Sequence the initial assignments randomly

Step 2: Objective function evaluation: Evaluate the makespan of each chromosome

Step 3: Producing next population:

(a) Sort the chromosomes based on makespan value

(b) Reproduction: Copy only the best chromosome to the next generation

(c) Crossover operation: Apply crossover to generate pc_* (popsize-1) individuals of the next population using roulette wheel selection strategy

(d) Mutation operation: Select pm_* (popsize-1) chromosomes randomly and apply the mutation to make pm_* (popsize-1) new individuals of the next population

Step 4: Termination test: Check the stopping criteria, if the stopping criterion is met return the best chromosome; else go to step 2

Initial population generation: In order to generate the initial population, first, we generate initial assignments, then make the initial population by randomly sequence (randomly select a job) of them. To generate the initial assignments we apply two ways proposed by Pezzella *et al*. (2008) as Assignment Rule 1 and Assignment Rule 2. They followed the approach by localization of (Kacem *et al*., 2002a, b) and modified it. Assignment Rule 1 foresees to start from the operation that corresponds to the global minimum in the processing times data. Assignment Rule 2 foresees to permute randomly the jobs and the machines before to apply the approach by localization.

Chromosome representation: In order to perform our algorithm, we use a string of triples (i, j, k), proposed by Kacem *et al*. (2002a), one for each operation, in which

- i is the job that operation is belong to;
- j is the progressive number of that operation within job i
- k is the machine assigned to that operation

The length of the string equals to the total number of operations. For example, for the problem explained in Table 1, a feasible solution can be represented like this: (1, 1, 1), (3, 1, 3), (2, 1, 2), (3, 2, 1), (2, 2, 3), (1, 2, 2), (1,

3, 2), (2, 3, 1), (2, 4, 2). The Gantt chart of this solution is shown in Fig. 1.

Evolutionary operators: To generate the next population, the crossover and mutation genetic operators are applied. We employ four evolutionary operators that can be divided to assignment operators and sequencing operators. Assignment operators which only change the assignment nature of the chromosomes and the sequencing of operations is preserved. Sequencing operators which only change the sequence of operations in the chromosomes and the assignment of operations is kept.

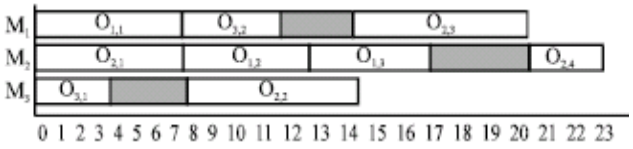

Fig. 1: Gantt chart

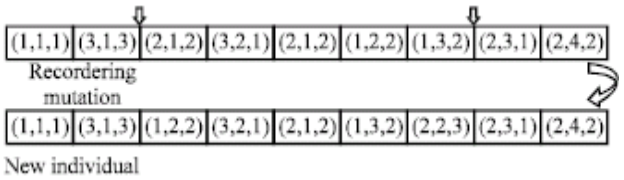

Fig. 2: Illustration of the reordering mutation

Mutation operators: In this study, we employ two mutation operators namely, assignment mutation and reordering mutation. Assignment mutation only exchanges the assignment of a single operation in a single parent. In reordering mutation, we select two positions and reorder the operations between randomly, with observing the precedence constraints, as shown in Fig. 2.

Crossover operators: We also apply two crossover operators, precedence preserving order-based crossover (POX) operator of Lee *et al*. (1998) and assignment crossover. In POX generates two children starting from two parents. First, POX selects an operation from the first parent, copies in the first child all the operations of the job which the selected operations belong to, then complete this new individual with the remaining operations, in the same order as they appear in the second parent. The symmetric process is repeated for the second parent and the second child. POX preserves the sequencing constraints (Pezzella *et al.*, 2008). Assignment crossover exchanges the assignment of the subset of operations between the two parents and generate new individual.

IV. COMPUTATIONAL RESULTS

We coded proposed GA by C++ language and implemented it on a 2 GHz Pentium IV processor, with 256 MB RAM. To evaluate the performance of our algorithm, we tested it on the data set consist of 20 problems and compare our results with recent results obtained by them.

Computational experience proves that roulette wheel selection strategy is suitable for performing crossover, randomly selection is suitable for performing mutation and the following values are more effective for our algorithm parameters:

- Percent of assignment rule 1 (pa1): 20%
- Percent of assignment rule 2 (pa2): 80%
- Probability of assignment mutation: 10%
- Probability of reordering mutation: 10%
- Probability of assignment crossover: 40%
- Probability of POX crossover: 40%

Values of some parameters are different for each problem (Small Flexible Job-Shop (SFJS) and Medium Flexible Job-Shop (MFJS)) which are presented in Table 2.

In Table 3 we show the best obtained result by our GA after five runs and compare them with the other results by integrated approaches by Fattahi *et al*. (2007) on Fdata. They proposed two integrated approaches: ISA (integrated approach with simulated annealing heuristic) algorithm and ITS (integrated approach with tabu search heuristic). The first column refers to problem name. The second and third columns refer to the number of jobs and the number of machines, respectively. The fourth column shows the lower bound of instances. The fifth, 6th and 7th columns present the average of makespan, the best makespan and the average of CPU times over five runs of our GA, respectively.

Other columns are the results of other algorithms we compare to. Also, relative deviation of them with respect to our algorithm is presented in Table 3. The relative deviation is defined as:

Table 2: Parameter levels of GA

Problem	Popsize
SFJS 1: 10	100
MFJS1: 5	1000
MFJS6: 10	1000

$$Dev = \left(\frac{C_f - C_{best}}{C_f} \right) \times 100\% \quad (1)$$

which C_{best} is the makespan obtained by GA and C_f is the makespan of the algorithm we compare to.

Table 4 compares the results of our GA with the results by hierarchical approaches by Fattahi *et al*. (2007) on Fdata. They presented four hierarchical approaches HSA/SA, HSA/TS, HTS/TS and HTS/SA algorithms.

M_1: ($O_{3,1}$: 0-87)($O_{2,1}$: 87-301)

M_2: ($O_{4,1}$: 0-65)($O_{5,1}$:65-188)($O_{1,2}$:188-318)($O_{2,}$: 318-384) ($O_{5,3}$: 384-484)

M3: (O1,1:0-100)(O7,1: 100-245)(O7,2: 245-369)

M4: (O6,1: 0-154)(O6,2: 154-304)(O1,3: 318-468)

M5: (O4,2: 65-238)(O3,3: 238-338)(O7,3: 369-514)

M6: (O4,3: 238-374)(O2,3: 384-479)

M7: (O3,2: 87-232)(O5,2: 232-318)(O6,3: 318-498)

V. CONCLUSION

In this research, a genetic algorithm investigated for solving flexible job-shop scheduling problem. In this study, the proposed approach is explained in detail by solving benchmark problems and programmed in C++ language. As shown in Table 3 and 4, our GA is effective to overcome mentioned problems. Obtained results are comparable properly with the best results of other authors.

VI. REFERENCES

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by taboo search. Ann. Operat. Res., 41: 157-183

Cheng, R., M. Gen and Y. Tsujimura, 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms, Part I: Representation. Comput. Ind. Eng., 30: 983-997

Gao, J., L. Sun and M. Gen, 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Operat. Res., 35: 2892-2907

Jain, A.S. and S. Meeran, 1999. Deterministic job-shop scheduling: Past, present and future. Eur. J. Operat. Res., 113: 390-434

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edn., Addison-Wesley Publishing Company, New York, USA., ISBN: 0201157675, pp: 36-90

Pinedo, M., 2002. Scheduling: Theory, Algorithms and Systems. 1st Edn., Prentice Hall, New Jersey, ISBN 0-13-281387