Journal of Applied Sciences 8 (24): 4650-4655, 2008
ISSN 1812-5654

Solving the Flexible Job-Shop Scheduling Problem by a Genetic Algorithm

[1]M. Zandieh, [2]I. Mahdavi and [2]A. Bagheri
[1]Department of Industrial Management, Faculty of Management and Accounting,
Shahid Beheshti University, Tehran, Iran
[2]Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

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 Genetic Algorithm (GA) 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 GA is tested on benchmark problems and with due attention to the results of other meta-heuristics in this field, the results of GA show that our algorithm is effective and comparable to the other algorithms.

Key words: Flexible job-shop, scheduling, meta-heuristics, genetic algorithm, combinatorial optimization

INTRODUCTION

One of the most necessary subjects in the planning and managing of manufacturing environments is the scheduling of operations . To find the best schedule can be very easy or very difficult, depending on the shop environment, the process constraints and the performance indicator (Pinedo, 2002). The Job-shop Scheduling Problem (JSP) is one of the most popular scheduling models existing in practice (Cheng *et al*., 1996). It has attracted many researchers due to its wide applicability and inherent difficulty (Nowicki and Smatnicki, 1996; Jain and Meeran, 1999). The classical JSP consists in scheduling a set of jobs on a set of machines with the objective to minimize a certain performance indicator, subject to the constraint that each job has a specified processing order through all machines which are fixed and known in advance. A typical performance indicator for JSP is the makespan, i.e., the time needed to complete all the jobs. FJSP is NP-hard since it is an extension of the job-shop scheduling problem (Gao *et al*., 2008).

FJSP is an extension of the classical JSP which exist an allowable set of machines for each operation to processes on any among them. FJSP can decomposed 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.

Bruker and Schlie (1990) developed a polynomial algorithm for solving the flexible job-shop scheduling problem with two jobs. Exact methods developed, but they are not applicable for instances with more than 20 jobs and 10 machines (Pinedo, 2002). In recent years, meta-heuristics such as simulated annealing, tabu search and genetic algorithms has used to solve FJSP. They can be categorized into 2 groups: Hierarchical approach and integrated approach. The hierarchical approach attempts to solve the problem by decomposing it into a sequence of sub-problems, with reduced difficulty. A typical decomposition is assign-then-sequence, coming from the trivial observation that once the assignment is done, the resulting sequencing problem is a JSP (Pezzella *et al*., 2008). Brandimarte (1993) and Paulli (1995) followed this approach. Kacem *et al*. (2002a) proposed a genetic algorithm and expanded approach by localization to achieve capable initial assignments. Xia and Wu (2005) applied a hybrid of Particle Swarm Optimization (PSO) and Simulated Annealing (SA) as a local search algorithm to solving this problem as a hierarchical approach.

Integrated approaches are used by considering assignment and sequencing at the same time and usually obtain better results than hierarchical approach, but they are much more difficult to solve. Gao *et al*.(2008) used a hybrid of GA and Variable Neighborhood Descent (VND) for this problem as an integrated approach. Pezzella *et al*. (2008) employed several different strategies for initial population generation and new individual reproduction, specially an intelligent mutation, in a GA framework based on approach by localization for solving

Corresponding Author: M. Zandieh, Department of Industrial Management, Faculty of Management and Accounting, Shahid Beheshti University, Tehran, Iran

FJSP as an integrated approach. Fattahi *et al.* (2007) presented a SA and a Tabu Search (TS) and proposed two hierarchical approaches and four integrated approaches by hybridization of them for solving the FJSP.

A genetic algorithm as an integrated approach is presented for solving this problem. To achieve initial assignments, we do the same as (Pezzella *et al.*, 2008) that adopted the approach by localization of (Kacem *et al.*, 2002a, b) as Assignment Rule 1 and Assignment Rule 2. Then, random selection of the next job is adopted to sequence the operations to generate the initial population. Our algorithm also employs multiple different mutation and crossover operators for assigning and sequencing. We present the computational results on a number of benchmark problems and compare them with the results presented by previous authors.

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, $O_{i,1}$, $O_{i,2}$, ...,O_{in_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_{ij}, there are a set of allowable machines called $A_{ij} \subseteq A$, which O_{ij} can be performed on one of them such as $M_k \in A_{ij}$. 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).

Table 1: Processing times of example

	Machines		
Operation	M_1	M_2	M_3
$O_{1,1}$	6	6	∞
$O_{1,2}$	∞	5	∞
$O_{1,3}$	4	5	5
$O_{2,1}$	∞	6	∞
$O_{2,2}$	∞	5	7
$O_{2,3}$	7	9	∞
$O_{2,4}$	6	3	∞
$O_{3,1}$	5	3	3
$O_{3,2}$	4	∞	∞

THE SOLUTION ALGORITHM

In this study, a genetic 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.

Genetic algorithm: Genetic algorithms (Gas) were introduced by Holland (1975) and 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 GA approach: The proposed GA 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 $pc_* \cdot (popsize-1)$ individuals of the next population using roulette wheel selection strategy
(d) Mutation operation: Select $pm_* \cdot (popsize-1)$ chromosomes randomly and apply the mutation to make $pm_* \cdot (popsize-1)$ 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.

Genetic operators: To generate the next population, the crossover and mutation genetic operators are applied. We employ four genetic 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

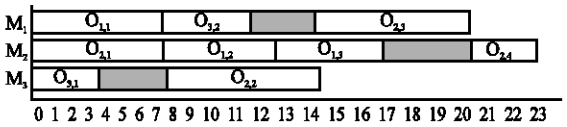

Fig. 1: Gantt chart

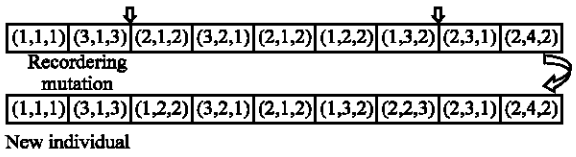

Fig. 2: Illustration of the reordering mutation

operations is preserved. Sequencing operators which only change the sequence of operations in the chromosomes and the assignment of operations is kept.

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.

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 from Fattahi *et al.* (2007) (Fdata) 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 makespans, 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:

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

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. In Table 5, the Mean Relative Error (MRE) of the best solution obtained by our GA and six different approaches by Fattahi *et al.* (2007) is presented. The relative error (RE) is defined as:

$$RE = \left(\frac{C_{best} - LB}{LB}\right) \times 100\% \quad (2)$$

which C_{best} is the best makespan obtained by the algorithm and LB is the lower bound. In the following, we present the schedule obtained by GA for problem MFJS5 of Fdata with the makespan equal to 514.

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)

Table 2: Parameter levels of GA

Problem	Popsize	ng
SFJS 1: 10	100	100
MFJS1: 5	1000	500
MFJS6: 10	1000	1000

Table 3: Comparison with the hierarchical approaches of Fattahi on Fdata

				GA			ISA		ITS	
Problem	n	m	LB	AV(C)	C_{best}	AV(CPU)*	C_{best}	Dev (%)	C_{best}	Dev(%)
SFJS1	2	2	66	66.0	66	0.01	66	0.00	66	0.00
SFJS2	2	2	107	107.0	107	0.01	107	0.00	107	0.00
SFJS3	3	2	221	221.0	221	0.01	221	0.00	221	0.00
SFJS4	3	2	355	355.0	355	0.01	355	0.00	390	+8.97
SFJS5	3	2	119	119.0	119	0.01	119	0.00	137	+13.14
SFJS6	3	3	320	320.0	320	0.02	320	0.00	320	0.00
SFJS7	3	5	397	397.0	397	0.02	397	0.00	397	0.00
SFJS8	3	4	253	253.0	253	0.02	253	0.00	253	0.00
SFJS9	3	3	210	210.0	210	0.02	215	+2.33	215	+2.33
SFJS10	4	5	516	516.0	516	0.02	516	0.00	617	+16.37
MFJS1	5	6	396	468.0	468	2.51	488	+4.10	548	+14.60
MFJS2	5	7	396	450.2	448	2.59	478	+6.28	457	+1.97
MFJS3	6	7	396	467.6	466	2.85	599	+22.20	606	+23.10
MFJS4	7	7	496	563.6	554	3.27	703	+21.19	870	+36.32
MFJS5	7	7	414	526.8	514	3.34	674	+23.74	729	+29.49
MFJS6	8	7	469	634.0	634	7.01	856	+25.93	816	+22.30
MFJS7	8	7	619	888.8	881	8.63	1066	+17.35	1048	+15.94
MFJS8	9	8	619	901.6	891	10.26	1328	+32.91	1220	+26.97
MFJS9	11	8	764	1122.6	1094	12.38	1148	+4.70	1124	+2.67
MFJS10	12	8	944	1301.8	1286	14.36	1546	+16.82	1737	+25.96

* Av(CPU) is the average computing time in seconds

Table 4: Comparison with the integrated approaches of Fattahi on Fdata

			HSA/SA		HSA/TS		HTS/TS		HTS/SA	
Problem	LB	GA	C_{best}	Dev (%)	C_{best}	Dev (%)	C_{best}	Dev (%)	C_{best}	Dev (%)
SFJS1	66	66	66	0.00	66	0.000	66	0.000	66	0.000
SFJS2	107	107	107	0.00	107	0.000	107	0.000	107	0.000
SFJS3	221	221	221	0.00	221	0.000	221	0.000	221	0.000
SFJS4	355	355	355	0.00	355	0.000	355	0.000	355	0.000
SFJS5	119	119	119	0.00	119	0.000	119	0.000	119	0.000
SFJS6	320	320	320	0.00	320	0.000	320	0.000	320	0.000
SFJS7	397	397	397	0.00	397	0.000	397	0.000	397	0.000
SFJS8	253	253	253	0.00	253	0.000	253	0.000	256	+1.170
SFJS9	210	210	210	0.00	210	0.000	210	0.000	210	0.000
SFJS10	516	516	516	0.00	516	0.000	516	0.000	516	0.000
MFJS1	396	468	479	+2.30	491	+4.680	469	+0.210	469	+0.210
MFJS2	396	448	495	+9.49	482	+7.050	482	+7.050	468	+4.270
MFJS3	396	466	553	+15.73	538	+13.380	533	+12.570	538	+13.380
MFJS4	496	554	656	+15.55	650	+14.770	634	+12.620	618	+10.360
MFJS5	414	514	650	+20.92	662	+22.360	625	+17.760	625	+17.760
MFJS6	469	634	762	+16.80	785	+19.240	717	+11.580	730	+13.150
MFJS7	619	881	1020	+13.63	1081	+18.500	964	+8.610	947	+6.970
MFJS8	619	891	1030	+13.50	1122	+20.590	970	+8.140	922	+3.360
MFJS9	764	1094	1180	+7.29	1243	+11.990	1105	+1.000	1105	+1.000
MFJS10	944	1286	1538	+16.38	1615	+20.370	1404	+8.400	1384	+7.080

Table 5: Mean relative error on Fdata

	GA	HSA/SA	HSA/TS	HTS/TS	HTS/SA	ISA	ITS
MRE	14.29	24.29	26.48	20.61	19.9	29.27	33.84

M_3: ($O_{1,1}$:0-100)($O_{7,1}$: 100-245)($O_{7,2}$: 245-369)
M_4: ($O_{6,1}$: 0-154)($O_{6,2}$: 154-304)($O_{1,3}$: 318-468)
M_5: ($O_{4,2}$: 65-238)($O_{3,3}$: 238-338)($O_{7,3}$: 369-514)
M_6: ($O_{4,3}$: 238-374)($O_{2,3}$: 384-479)
M_7: ($O_{3,2}$: 87-232)($O_{5,2}$: 232-318)($O_{6,3}$: 318-498)

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. We have some recommendation on future works:

- Studying in this field with additional constrains, such as maintenance requirements or breakdowns can be interesting
- Developing the meta-heuristics to optimize the other objectives in this field, such as earliness and tardiness will be remarkable

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.

Fattahi, P., M. Saidi Mehrabad and F. Jolai, 2007. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J. Intel. Manuf., 18: 331-342.

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.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edn. Addison-Wesley Longman Publishing Co. Inc., MA, USA., ISBN: 0201157675 pp: 372.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. 1st Edn., University of Michigan Press, Michigan. ISBN: 0262580969 .

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

Kacem, I., S. Hammadi and P. Borne, 2002a. Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems. IEEE. T. Syst. Man. Cy. C., 32: 1-13.

Kacem, I., S. Hammadi and P. Borne, 2002b. Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. Math. Comput. Simulat., 60: 245-276.

Lee, K.M., T. Yamakawa and K.M. Lee, 1998. A genetic algorithm for general machine scheduling problems. Int. J. Knowledge-Based Elect., 2: 60-66.

Nowicki, E. and C. Smutnicki, 1996. fast taboo search algorithm for the job shop problem. Manage. Sci., 42: 797-813.

Paulli, J., 1995. A hierarchical approach for the FMS scheduling problem. Eur. J. Operat. Res., 86: 32-42.

Pezzella, F., G. Morganti and G. Ciaschetti, 2008. A genetic algorithm for the flexible job-shop scheduling problem. Comput. Operat. Res., 35: 3202-3212.

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

Xia, W. and Z. Wu, 2005. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problem. Comput. Ind. Eng., 48: 409-425.