An Algorithm of scheduling for the Job Shop Scheduling Problem

Marco Antonio Cruz-Chávez[1], Martín G. Martínez-Rangel[1,2], J. A. Hernández[1], José Crispín Zavala-Díaz[2], Ocotlán Díaz-Parra[1]

[1]*CIICAP,* [2]*FCAeI, Autonomous University of Morelos State*
Avenida Universidad 1001. Col. Chamilpa, C.P. 62210. Cuernavaca, Morelos, México
{ mcruz, mmtzr, Alfredo, crispin_zavala, ocotlandp}@uaem.mx

Abstract

This paper presents an algorithm that applies a new mechanism in order to generate scheduling which allows for evaluation of the quality of solutions that are obtained in the Job Shop Scheduling Problem (JSSP). In this research, the quality of the solution is evaluated by using the makespan as an objective function. It is demonstrated experimentally that the proposed algorithm has better efficiency and efficacy when compared to the classic form of scheduling generation used to evaluate the solution quality in the JSSP. The efficiency and efficacy obtained by the proposed algorithm make it possible to generate and evaluate a greater number of better quality solutions in less time, so a greater exploration of the solution space for the JSSP can be conducted.

1. Introduction

The Job Shop Scheduling Problem (JSSP) is one of the most well known and difficult to solve problems in the scheduling area. JSSP is probably the model most often studied and most developed of all the problems pertaining to the deterministic theory of scheduling. It serves as a reference for other techniques that try to solve problems in the same field, for example the transport problem or the knapsack problem [1]. The time required to solve the JSSP increases exponentially according to the size of the problem. According to the complexity theory [2], JSSP is classified into the NP-complete group [3]; this group of problems is considered to be the most difficult group of problems to solve in the world. For big instances, a deterministic algorithm does not exist that solves problems in this group. For this reason, metaheuristics are used to search for the global optimum of problems in this group [4] because one can generate algorithms that bound this group of problems to polynomial time. These heuristics are characterized by searches through neighborhoods in non deterministic form. For this reason, the development of more efficient and effective mechanisms that accelerate the searches is important in order to improve the search in neighborhoods.

A great number of metaheuristics have been proposed for the search for the global optimum of the JSSP in polynomial time. These algorithms include Simulated annealing [5], [6], Tabu Search [7], [8], [9], Ant Colony [10], and Genetic Algorithms [10], [11], [12], [13], among others. In order to evaluate the quality of solutions, these metaheuristics require the makespan to be found (function objective value of the problem) during each step of the heuristic algorithm. In order to do this, the scheduling algorithm is generally used [14]. Every time that the metaheuristic obtains a new solution (schedule), this algorithm is applied to the solution in order to assign a start time to each one of the operations that are part of the JSSP and obtain the value of the makespan. The makespan is defined as the completion time of the last operation O_l in the system and it is equal to the sum of the start time of O_l plus the processing time of O_l.

This paper proposes a scheduling mechanism to more quickly evaluate the quality of each solution (schedule) obtained in the process of local search by application of a neighborhood function. The neighborhood function used in this investigation is N_1[6] with the Neighborhood Generation Mechanism (NGM) proposed in [15]. This combination of N_1-NGM very simply selects only feasible solutions without needing to work through the critical path of the generated solution like N_1 normally does. This allows for the measurement of performance of the proposed scheduling algorithm with feasible solutions, leaving aside the infeasible ones.

Following the introduction, is section 2 which explains the disjunctive graph model of JSSP that is used in the proposed scheduling mechanism. In section 3, the proposed mechanism for the schedule

This work was supported by project 160 of the Fideicomiso SEP-UNAM, 2006-2007.

generation is described. In section 4 the computational study and the experimental tests are presented, and in section 5 are the conclusions.

2. The Job Shop Scheduling Problem

The Job Shop Scheduling Problem (JSSP) consists of a set *N* of jobs, a set *M* of machines and a set *O* of operations, where each one of the jobs is a subset of *O* in sequential form. Each *i* operation has a duration $\tau(O_i)$. In an assignment of jobs, a start time is defined for each operation $st(O_i)$, such that:

1. A machine cannot process more than one operation at a time.
2. The execution order of the operations in each job is respected.

The time at which the execution of all the operations of the JSSP is finished is known as the makespan. Usually one of the objectives of JSSP searches is to find a schedule that minimizes the makespan.

2.1 The Disjunctive Graph Model

Figure 1 show the disjunctive graph model for a JSSP of 3x3 (three machines and three jobs), this model is introduced by Roy and Sussmann [17]. The general problem of JSSP is defined for the graph $G = (V,A,E, \tau)$ where:

$$V = O \cup \{I, F\}$$

$$A = \{[I, O_{1j}], [O_{ij}, O_{(i+1)j}], [O_{mj}, T] | \forall i, j: O_{ij} \in O\}$$

$$E = \left\{\{O_{ij}, O_{ij}\} \mid \forall i, j, i, j, j \neq j: O_{ij}, O_{ij} \in O \wedge M_k(O_{ij}) = M_k(O_{ij})\right\}$$

$$\mu: V \rightarrow IN$$

The vertexes *V* represent the operations. There are two vertexes (fictitious operations) that do not have a processing time, these are the *I* (source) and the *F* (finish). The weight of a vertex $\tau(O_{ij})$ is the processing time $p(O_{ij})$, $\tau(O_{ij}) = p(O_{ij})$, $\tau(I) = \tau(F) = 0$. The first operation of each job connects toward the *I* (source) node, while the last operation of each job connects toward the *F* (finish) node. The precedence constraint between a pair of operations O_{ij}, $O_{(i+1)j}$ carried out in the same job, is represented by an conjunctive arc $[O_{ij}, O_{(i+1)j}] \in A$.

In same way, the resource capacity constraint between a pair of operations O_{ij}, $O_{ij} \in O$, $M(O_{ij}) = M(O_{ij})$ is represented by a disjunctive arc $\{O_{ij}, O_{ij}\} \in E$ where the two forms of directing this arc correspond to the two possible orientations of $\{O_{ij}, O_{ij}\}$.

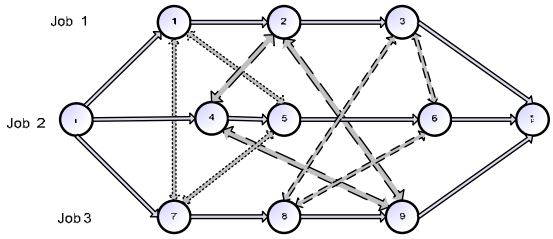

Fig. 1. Disjunctive graph representation of the JSSP 3x3.

An orientation of *E* is a function $\delta: E \rightarrow OxO$ such that $\delta(\{O_{ij}, O_{ij}\}) \in \{\langle O_{ij}, O_{ij}\rangle\}, \{\langle O_{ij}, O_{ij}\rangle\}$ for each $\{O_{ij}, O_{ij}\} \in E$. If the orientation δ of the set *E* results in a digraph $D = G' = (V, A, E, \tau, \delta(E))$ then a schedule is obtained that represents a solution of the JSSP, which could be feasible or infeasible. Figure 2 presents an example of a feasible solution for the JSSP in Figure 1.

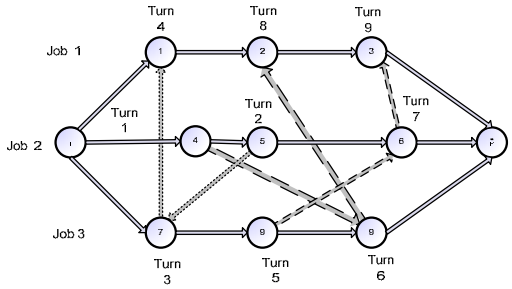

Fig. 2. A feasible solution for the JSSP 3x3 in fig. 1.

3. Scheduling Procedure

In order to apply the proposed mechanism in the construction of a scheduling, it is necessary to begin with a schedule (a solution) that has defined its scheduling. The method applied to generate a schedule with its scheduling is the one used in [14].

This randomly selects an operation that belongs to the set which consists of operations that do not have an predecessor (prior operation in the same job) or that in their defect, are operations that have a predecessor but that this has been assigned, identifying its final processing time. If the selected operation fulfills this rule (precedence constraint), it is assigned an execution

time (turn) and the final moment that the processing machine has for the operation programmed is verified (resources capacity constraint). The start time of the operation is then determined evaluating the two restrictions (precedence and resource capacity) considering that of greater value.

Next, an example is presented that defines the symbolic representation of the first S_o solution (schedule) for the JSSP 3x3 presented in Figure 1. The symbolic representation presented in Table 1 is used for the proposed scheduling method. This first S_o solution is a Gantt Chart shown in Figure 3. One can observe in the figure the start (scheduling) and the end of the execution of each operation with regard to job, these data are included in Table 1, columns ten and eleven. Following the order of the schedule in Figure 3, the operations are assigned in the following way: 4, 5, 7, 1, 8, 9, 6, 2 and 3 (by turn, in ascending order) like the sample in Table 1, columns one and two.

Máquina 0.- 20000000333100
Máquina 1.- 00000033200011
Máquina 2.- 02223311100000

Fig. 3. Allocation of the operations in machines in S_0

The first starting S_o solution used by the proposed scheduling mechanism appears in Table 3, which has all the necessary data in order to proceed in the generation of a new solution (schedule) and the construction of its scheduling for the proposed mechanism.

Table 1 shows in column six that the operations that could be exchanged are those that are assigned in turns 3, 4, 7 8 (operations 7, 1, 6, and 2 respectively, see Table 1, column 2). Each a has an adjacent operation shown in column seven.

Table 1. Data for the first solution (S_0)

T	O	J	M	D	In	Ao	Pr	EP	B	E
1	4	2	0	1	0	0	0	0	1	1
2	5	2	2	3	0	0	4	1	2	4
3	7	3	2	2	1	5	0	0	5	6
4	1	1	2	3	1	7	0	0	7	9
5	8	3	1	2	0	0	7	6	7	8
6	9	3	0	3	0	0	8	8	9	11
7	6	2	1	1	1	8	5	4	9	9
8	2	1	0	1	1	9	1	9	12	12
9	3	1	1	2	0	0	2	12	13	14

3.1 Scheduling Generation Mechanism

When a solution (schedule) S_0 of the JSSP is applied using the neighborhood function N1-NGM proposed in [15] for local searches, it is possible to generate new feasible solutions upon exchanging (permuting) only one pair of adjacent operations that do not have slack-time between them. This neighborhood function carries out a perturbation of S_0 to obtain a new solution S_1. In order to evaluate the makespan of S_1 it is necessary to obtain the scheduling of S_1. The proposed mechanism of scheduling construction consists of beginning the scheduling starting from the position where the solution S_0 is perturbed, respecting the partial scheduling that is completed before the place where the perturbation was done.

The next example is of the proposed scheduling mechanism of the solution S_1 (Table 2) obtained by exchanging a pair of adjacent operations without slack-times in the schedule S_0 presented in Table 1.

In order to make the exchange in the solution S_0, a turn is randomly selected from the list of restricted candidates (pairs of operations that could be exchanged). The candidates to be exchanged are designated in column six by means of a flag, 1 if it is interchangeable and zero if it is not.

It can be observed that for the values in column six, the pairs of candidate operations are (7, 5), (1, 7), (6, 8) and (2, 9). These can be seen in the rows highlighted in gray, columns two and seven of Table 1. In this case, turn 7 is randomly selected (operation 6) to be exchanged with the operation that is in turn 5 (operation 8). This change is carried out in Figure 4, which shows the corresponding solution S_1:

Machine 0.- 20000000333100
Machine 1.- 00002033000011
Machine 2.- 02223311100000

Fig. 4. New configuration after the interchange of operation 6 by 8 in machine 1. Solution S_1.

Table 2. Data for the solution S_1 (Result of interchange of operation 6 and operation 8)

T	O	J	M	D	In	Ao	Pr	EP	B	E
1	4	2	0	1	0	0	0	0	1	1
2	5	2	2	3	0	0	4	1	2	4
3	7	3	2	2	1	5	0	0	5	6
4	1	1	2	3	1	7	0	0	7	9
5	6	2	1	1	0	0	5	4	5	5
6	8	3	1	2	0	0	7	6	7	8
7	9	3	0	3	0	0	8	8	9	11
8	2	1	0	1	1	9	1	9	12	12
9	3	1	1	2	0	0	2	12	13	14

Table 2 shows the solution S_1; in column eleven the process of re-scheduling affects only turns 5, 6, 7, 8 and 9, leaving intact turns 1, 2, 3 and 4. In Table 2, column eleven shows that the quality of the solution obtained by function of the makespan is 14.

As is shown in Table 2, it is not necessary obtain the scheduling of all the operations, rather one can start at the turn that was perturbed (50% of re-scheduling). In this case, operation 6 is assigned before operation 8 (column 2), causing a landslide (a posterior turn) of operations 9, 2 and 3 (that were programmed for after the operations that were exchanged). Operations 4, 5, 7 and 1 remain in the same turns (column 2), occupying turns 1, 2, 3 and 4 respectively (column 1). The algorithm for re-scheduling the turns 5 to the 9 is the one proposed in [14].

In order to obtain the following solution S_2, it is necessary to start over with a new permutation in the previous solution S_1. A turn is randomly selected from the new restricted candidates, the candidates are selected according to the column six. The candidates are the pairs of operations (7, 5), (1, 7) and (2, 9). The procedure of re-scheduling repeats and this way the quality of each solution (makespan) is evaluated in a more efficient manner when re-scheduling the new solutions in a partial form.

Table 3. Partial Scheduling Algorithm (S-Partial)

```
problem = JSSP Instance;
S0 = Initial_solution(Problem);
for(i = 0, j = 1; j <= NT; i++, j++){
        list = restrict_list(Si);
        position = turn_perturbation(Si);
        Sj = perturbation(Si, position, list);
        Sj = scheduling_MS(Sj,position);
}
```

Table 3 presents the scheduling algorithm that applies the proposed partial scheduling mechanism. The algorithm is begun by choosing an instance of the JSSP problem. Next an initial solution S_0 (schedule) is obtained including the scheduling. *NT* is the total number of solutions to generate, in order to obtain the scheduling and the makespan for each solution. With the function *restrict_list*(S_i), the list of pairs of operations that can be perturbed in the solution S_i is obtained. The function *turn_perturbation*(S_i), is chosen randomly the T turn that belongs to the operation that will be perturbed in the solution S_i. The function *perturbation*(S_i, *position, list)*, generates a new S_j solution upon perturbing the pair of operations of the T turn in the solution S_i. The function *scheduling_MS*(S_j,*position)* obtains the scheduling of the new S_j solution by re-scheduling. It starts re-scheduling from the T turn which is defined for the variable *position*. The quality of the solution is also obtained by evaluating the makespan.

Nomenclature of tables.

T = Turn assigned to the operation, ascending order.
Or = Operation assigned in the T turn.
J = Job to which belong the assigned operation.
M = Machine that processes the operation with T turn.
D = Duration of the operation in the machine.
In = If the operation is interchangeable with any another, this is marked with 1, otherwise with 0.
Ao = Adjacent operation with which the operation in turn could be interchanged.
Pr = Operation that precedes to the operation assigned in the T turn.
EP = Completion Time of the Pr operation.
B = Start time of the operation of the T turn (scheduling).
And = End time of the operation of the T turn.

4. Experimental Results

The proposed scheduling mechanism was implemented in an algorithm in language C in a PC with 2 GHz, and 1 GB in RAM. In order to prove the efficiency of the proposed scheduling method, a set of benchmarks was used [16] of small, medium and large sizes of JSSP (Mt06, MT10, LA40 YN1). For the generation of feasible solutions, the neighborhood structure N1-NGM was used [15]. The initial solution was obtained by means of the procedure described in section 3.

In order to compare the efficiency and the efficacy of the proposed scheduling mechanism, two strategies were used for the generation of scheduling for each one of the instances of test. The first strategy is called S-Total which uses the classical procedure of scheduling proposed in [14] to evaluate the quality of a solution. This procedure requires obtaining the scheduling for all the operations in the JSSP. The S-Total strategy is used in many of the algorithms presented in the JSSP literature. The second strategy, called S-Partial (proposed mechanism described in 3.1), obtains re-scheduling for only a part of the operations involved in the problem instance in order to evaluate the quality of the solution.

Figure 5 shows that S-Partial requires less time to generate the same number of solutions (65,500) than S-Total requires for small, medium and large problems. As the size of the problem increases, S-Partial is increasingly efficient. For instances of large problems,

with around 400 operations, S-Partial is on average 58% more efficient than S-Total. In order to generate 65,500 solutions in the large instance YN1, on average, S-Partial takes 1,250 seconds while S-Total takes 3,010 seconds.

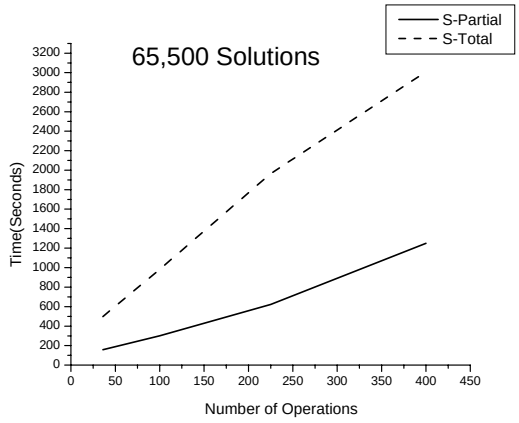

Fig. 5. Efficiency of the S-Partial vs. S-Total

In order to measure the efficacy of the proposed mechanism, Table 4 shows that for 65,500 generated solutions, S-Partial generates better quality of solutions than S-Total in almost all the instances of the problem. The exception is the small instance MT06 where the quality is equal. In the average value it can be observed that S-Partial is better in two instances, worse in MT10 and equal in LA40.

Table 4. solution Quality

Strategies	Problem	Mean	Min	Max
S-Total	MT06	94.26	55	158
S-Partial	MT06	93.73	55	159
S-Total	MT10	1782.5	1324	2405
S-Partial	MT10	1797.42	1291	2466
S-Total	La40	1894.39	1354	2772
S-Partial	La40	1894.64	1228	2786
S-Total	YN1	1308.44	991	1731
S-Partial	YN1	1280.98	978	1805

S-Partial allows for dispersed solutions in the solution space of the JSSP instance to be found to a greater degree because the range of S-Partial is greater, this is shown in Figure 6 with an average of 65,500 solutions evaluated for different instances of different sizes. This behavior favors the Genetic Algorithms and Simulated Annealing because the solutions generated by the proposed mechanism embrace a grater solution space of the problem and this permit a better exploration of the solution space of the JSSP.

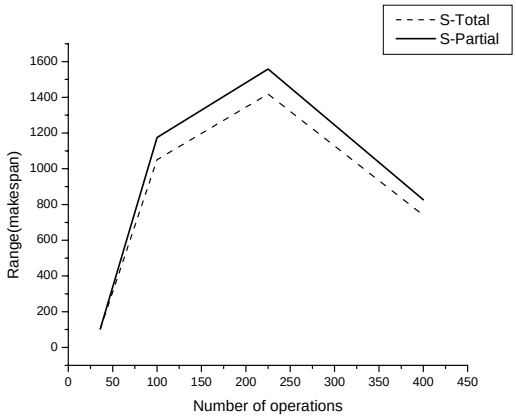

Fig. 6. Range of solutions of S-Partial vs. S-Total

The proposed S-partial algorithm, is of polynomial complexity and according to the value of T turn that is evaluated, the polynomial order could be: In the worst case $O(3n^2)$, in the better case $O(n)$ and the average case $O(0.5n^2)$. According to the experimental results presented in Figure 5, where S-Total with complexity $O(n^2)$ is compared with S-Partial, one could observe that for small problems, the complexity of S-Partial is similar to S-Total and when the size of the problem increases, one could conclude that the complexity of S-Partial tends to be minor that S-Total, this is, it move between the interval of $O(n)$ to $O(0.5n^2)$.

5. Conclusions

The results observed in this research, demonstrate that the proposed scheduling mechanism is much more efficient than the classical mechanism of scheduling frequently used in metaheuristics for the JSSP. The efficacy that the proposed mechanism presents is also superior in almost all the evaluated cases, and only in inferior, but still competitive, in a few cases when compared with the classical scheduling mechanism.

According to the obtained results, when measuring the range for an average of 65,500 solutions in each problem size, one could conclude that the solutions obtained by S-Partial have a greater dispersion in the solution space for the problem instance. This can favor the behavior of the metaheuristics for JSSP, because the search can embrace a greater exploration of the solution space.

6. References

[1] A. Jain S. Meeran.: A State of the Art Review of JOB-SHOP Scheduling Techniques. Technical Report. Department of Applied Physics, Electronic and Mechanical Engineering University of Dundee, Dundee, Scotland, UK, DD1 4HN,1998.

[2] C H. Papadimitriou, K. Steigliths.: Combinatorial Optimization. Algorithms and Complexity. Dover Publications, Inc. 1998.

[3] M.R. Garey, D.S. Johnson and R. Sethi, The complexity of Flow shop and Job shop Scheduling. Mathematics of Operations Research, Vol. I, No 2, USA, 117-129, May, 1976.

[4] D.Applegate, W. Cook, "A computational study of the job shop scheduling problem", ORSA Journal on Computing, 3, 149-156, 1991.

[5] S. Kirkpatrick, S. D. Gelatt Jr., and M. P. Vecchi, Optimization by simulated annealing. Science, 220(4598), 13 May, 671-680, 1983.

[6] V. Laarhoven PJM, EHL Aarts, and JK Lenstra. Job shop scheduling by simulated annealing. Operations Research, 40, pp.113-125, 1992.

[7] M.D Amico M, M. Turbian. Applying tahu search to the job shop scheduling problem. Annual Operations Research, 40, pp. 231-252, 1993.

[8] K. Morikawa, T. Furuhashi, Y. Uchikawa. Single Populated Genetic Algorithm and its Application to Job-shop Scheduling. Proc. Of Industrial Electronics, Control, Instrumentation, and Automation on Power Electronics and Motion Control, pp. 1014-1019, 1992.

[9] E. Nowicki, C. Smutnicki. A Fast Taboo Search Algorithm: for the Job Shop Problem. Managemenr Science, vol. 42, pp. 797-813, 1996.

[10] Cheng-Fa Tsai, Chun-Wei Tsai, and -Chin-Chang Tseng. New and efficient antibased heuristic method for solving the traveling salesman problem; Expert Systems (accepted, will appear in vol. 20, no. 4, 2003)

[11] T.Cheng-Fa, T.Chun-Wei , and T.Ching-Chang. ACOMAC: An Efficient Method for Solving raveling Salesman Problem. 2002 IEEE Intemational Joint Conference on Neural Network(IJCNN 2002), pp. 1540-1545, Honolulu, Hawaii, USA

[12] T.Cheng-Fa , T.Chun-Wei, and C.Chi-Ping: A Multiple-Searching Approach to Genetic Algorithms for Solving Large .Traveling Salesman Problem. 6th Intem. Conf on Computer Science and Informatics, pp. 362-366, Durham; NC, USA.

[13] J. Adams, E. Balas, D. Zawack. The 'shifting bottleneck procedures for job shop scheduling. Management Science, vol. 34, pp. 391-401, 1988.

[14] P. J. Zalzala, and Flemming. Zalzala, A.M.S. (Ali M.S.), ed., Genetic algorithms in engineering systems /Edited by A.M.S. Institution of Electrical Engineers, London, 1997.

[15] M. A. Cruz-Chávez, J. Frausto-Solís, J. R. Cora-Mora, Experimental Analysis of a Neighborhood Generation Mechanism Applied to Scheduling Problems, Proceedings of CERMA2006, IEEE-Computer Society, ISBN 0-7695-2569-7, pp 226-229, 26-29 September, México, 2006.

[16] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society, Vol. 41. No. 11, 1069-1072, 1990. Last update 2003.

[17] Roy and Sussman, Les problemes d'ordonnancement avec contraintes disjonctives, Note D.S. no 9 bis, SEMA, Paris, France, December 1964.