Proceedings of MIM 2002:
5th International Conference on Managing Innovations in Manufacturing (MIM)
Milwaukee, Wisconsin, USA
September 9-11, 2002

MULTIPLE DISPATCHING RULE BASED HEURISTIC FOR MULTI-OBJECTIVE SCHEDULING OF JOB SHOPS USING TABU SEARCH

Adil Baykasoğlu[1], Lale Özbakır[2], Türkay Dereli[1]
[1]University of Gaziantep, Department of Industrial Engineering, 27310, Gaziantep, TURKEY
[2]Erciyes University, Department of Industrial Engineering, Kayseri, TURKEY
{baykasoglu,dereli}@gantep.edu.tr, lozbakir@erciyes.edu.tr

ABSTRACT
In this paper, a multiple dispatching rule based meta-heuristic solution approach for Job Shop Scheduling Problems (JSSP) is presented. The proposed algorithm makes use of Giffler & Thompson's heuristic in deducting feasible schedules and Multiple Objective Tabu Search (MOTS) in generating optimal schedules. Several example problems are solved from the literature to present the effectiveness of the proposed algorithm. The results obtained from the computational study have shown that the proposed algorithm can be used as a new alternative solution technique for finding good solutions to this complex problem.

Keywords: Scheduling, Multiple Objective Tabu search, Optimization, Dispatching Rules.

1. INTRODUCTION

Production scheduling, which is a part of the planning and control of individual production units, lies at the very heart of the performance of manufacturing organizations (Stoop *et al*, 1996). Job shop scheduling problem (JSSP) is a branch of production scheduling, which is among the hardest combinatorial optimization problems (Sonmez and Baykasoglu, 1998). Not only it is hard, but even among the members of the latter class, it appears to belong to the more difficult ones (Van Laarhoven et al, 1992).

The problem definition for JSSP is as follows; there is a set $J=\{J_1,...,J_n\}$ of n jobs that must be processed on a group $M=\{1,...,m\}$ of m machines. Each job Ji, consists of a sequence of operations, O_{1j}, O_{2j},...,O_{nj}. Each operation O_{ij}, must be processed without interruption on machine $m_{ij} \in \{1,...,m\}$ with a processing time p_{ij}. The operations O_{1j}, O_{2j},..., O_{nj} must be processed one after another in the given order and each machine can process at most one operation at a time(Mastrolilli and Gambardella, 2000).

Many different approaches have been applied to JSSP, including: dispatching rules (Giffler and Thompson, 1960, Blackstone *et al*, 1982), mathematical programming (Sonmez and Baykasoglu, 1998, Pinedo, 1995), artificial intelligence techniques (Kusiak and Chen, 1988), simulation methods (Baykasoglu et al, 1998), heuristics (Kusaiak, 1990) etc. Ben-Daya (1994) provides a good classification of the techniques applied to job shop scheduling. It has been recognized that scheduling optimization using mathematical programming is very difficult, because of prohibitive computational time. It becomes more difficult to achieve an optimal result when the variety of parameters and constraints is increased (Maturana *et al*, 1997). Due to this fact, meta-heuristics (simulated annealing, genetic algorithms, tabu search etc.) and dispatching rule-based heuristics have gained significant research attention for solving JSSP problems. Many researchers from all around the world applied meta-heuristics to scheduling problems. Maturana *et al* (1997) employed genetic algorithms for solving JSSP. Cheng *et al* (1996) gave a good review of the genetic algorithms for job shop scheduling. Al-Fawzan and Al-Sultan (1996) applied tabu search to the JSSP. Van Laarhoven *et al* (1992) used simulated annealing for JSSP. Dorndorf and Pesch (1995) developed a dispatching rule-based genetic algorithm for JSSP. Many other example applications can easily be found from the literature.

However, it must be mentioned here that most of the approaches developed in the literature considered only the single objective case, although in JSSP multiple-objective solutions are more frequently required.

In parallel to the trend in the literature, in this paper, we proposed a hybrid-dispatching rule based multiple objective tabu search algorithm to solve JSSP. The proposed algorithm makes use of Giffler & Thompson's heuristic in deducting feasible schedules and Multiple Objective Tabu Search (MOTS) in generating optimal schedules. The proposed algorithm is the first attempt in applying the MOTS developed by (Baykasoglu *et al*, 1999) to JSSP. Several example problems are solved from the literature to present the effectiveness of the proposed algorithm. The results obtained from the computational study have shown that the proposed algorithm can be used as a new alternative solution technique for finding good solutions.

2. IMPLEMENTING MOTS FOR SOLVING FJSSP

2.1. A Review of the Classical Tabu Search Algorithm

Tabu search is a heuristic problem independent optimization method. It was first suggested by Glover (1992) and since then has become increasingly used. The basic idea of the method, described by Glover (1992), is to explore the *search space* of all feasible solutions by a sequence of moves. A move from one solution to another is the best available. However, to escape from locally optimal but not globally optimal solutions and to prevent cycling, some moves, at one particular iteration, are classified as forbidden or tabu. Tabu moves are based on the short-term and long-term history of the sequence of moves. A simple implementation, for example, might classify a move as tabu if the reverse move has been made recently or frequently. Sometimes, when it is deemed favorable, a tabu move can be overridden. Such aspiration criteria might include the case that, by forgetting that a move is tabu, leads to a solution which is the best obtained so far.

Suppose that, f is a real valued objective function on a search space S and it is required to find a $h \in S$ such that $f(h)$ has maximal value. For NP-complete problems, this requirement needs to be relaxed to finding a $h \in S$ such that $f(h)$ is close to the maximal value. This is because any known algorithm to determine the maximal solution requires time, which is exponential in the problem size. Sub-optimal problems may be solved by halting when a certain threshold for an acceptable solution has been found or when a certain number of iterations have been completed.

A characterization of the search space S for which tabu search can be applied is that there is a set of k moves $Q=\{q_1,q_2,....,q_k\}$ and the application of the moves to a feasible solution $s \in S$ leads to k usually distinct, solutions $Q(s)=\{q_1(m),q_2(m),.. ..,q_k(m)\}$. The subset $N(s) \subseteq Q(s)$ of feasible solutions is known as the neighborhood of s.

The method commences with a (generally random) solution $s_0 \in S$ and determines a sequence of solutions $s_0, s_1, s_2 ,....,s_n \in$ S. At each iteration, s_{j+1} is selected from the neighborhood $N(s_j)$. The process of selection is first to determine the tabu set $T(s_j) \in N(s_j)$ of neighbors of s_j and the aspirant set $A(s_j) \in T(s_j)$ of tabu neighbors. Then s_{j+1} is the neighbor of s_i which is either an aspirant or not tabu and for which $f(s_{j+1})$ is maximal; that is $f(s_{j+1}) \geq f(s^*) \forall\ s^* \in (N(s_j)-T(s_j)) \cup A(sj)$. The classical TS algorithm can be summarized as follows:

```
Procedure Tabu_Search
Start
k=1;
Generate initial solution s;
Repeat
       Identify N(s) ⊆ S (Neighborhood set);
       Identify T(s) ⊆ N(s) (Tabu set);
       Identify A(s) ⊆T(s) (Aspirant set);
       Choose s*∈(N(s)-T(s)) ∪A(s), for which f(s*) is maximal;
       s=s*;
       k=k+1;
Until Stop_Criterion
End
```

Note that, it is possible to avoid convergence at a local maximum, that $f(s_{j+1})<f(s_j)$. The conditions for a neighbor to be tabu or an aspirant will be problem specific. For example, a move may be tabu if it could lead to a solution, which

has already been considered in the last *m* iterations for which has been repeated many times before. A tabu move satisfies the aspiration criteria if, for example, the value of $f(s^*)$ with $s^* \in T(s_j)$ satisfies $f(s^*)>f(s_i)\ \forall i,\ 0 \le i \le j$. The main stages of TS algorithms are; *initial solution*, *generation of neighbors*, *selection, aspiration*, and *updating*. In any implementation these stages must be devised properly.

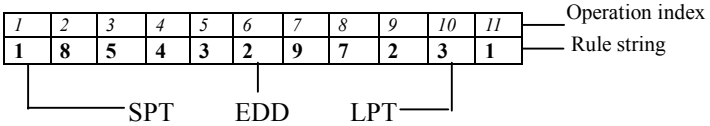
2.1 The Multiple Objective Tabu Search (MOTS)

The idea of applying tabu search to multiple-objective optimization comes from its solution structure, in working with more than one solution (neighborhood solutions) at a time. To enable the TS algorithm to work with more than one objective, *selection* and *updating* stages of classical TS are redefined. Other stages are similar to the classical TS algorithm. In contrast to classical TS algorithm, MOTS algorithm has two more lists in addition to tabu list. The first one is the *Pareto list*, which collects selected non-dominated solutions found by the algorithm. The second one is the *Candidate list*, which collects all other non-dominated solutions, which are not selected as Pareto optimal solutions in the current iteration. These solutions may become seed solutions if they maintain their non-dominated status in later iterations. The candidate list plays also an important role, it gives the opportunity to diversify the search. Detailed explanations about the MOTS can be found in Baykasoglu *et al* (1999). In the following sub-section the important stages of the MOTS for solving multiple-objective JSSP are explained.

2.2 Steps of MOTS in Solving JSSP

2.2.1 Initial solution and solution representation

In the present approach a solution is represented as a string, which is composed of a set of dispatching rules. In the string each number represents a dispatching rule for an operation starting from the first operation. In the present algorithm 10 different dispatching rules are used. The list of these rules is given in Table 1. As an example, consider a JSSP with 3 parts and these parts having in total 11 operations to be processed. In this case we will have a string with 11 numbers of entry (p_1, p_2, p_3,……., p_{11}) (i.e., $\#entries\_in\_solution\_string = \sum_{i=1}^{\#parts} \#operations_i$). The mapping for a candidate solution in the neighborhood will be as follow:



An entry p_i shows one rule of the set of previously specified dispatching rules. The entry in the *i*th position says that a conflict in the *i*th iteration of the Giffler and Thompson algorithm should be resolved using the dispatching rule p_i. In other words, an operation from the conflict set has to be selected by rule p_i; ties are broken by a random choice.

The Giffler and Thompson algorithm for the deduction a feasible schedule from a given solution string (p_1, p_2, p_3,……., p_n) works as follows:

Notation:
- PS_t = a partial schedule containing t scheduled operations.
- S_t = the set of schedulable operations at iteration t, corresponding to a given PS_t.
- σ_i = the earliest time at which operation $i \in S_t$ could be started.
- ϕ_i = the earliest time at which operation $i \in S_t$ could be completed.
- C_t = the set of conflicting operations in iteration t.

Procedure: Deduce a string for dispatching rule based encoding

Step 1: Let t =1 and begin with PS_t as the null partial schedule and S_t include all operations with no predecessors.

Step 2: Determine $\phi_t^* = \min_{i \in S_t} \{\phi_i\}$ and the machine m^* on which ϕ_i^* could be realized. If more than one such machine exists, tie is broken by a random choice.

Step 3: Form conflicting set C_t which includes all operations $i \in S_t$ with $\sigma_i \le \phi_i^*$ that requires machine m^*. Select one operation from C_t by dispatching rule p_t and add this operation PS_t as early as possible, thus creating new partial schedule PS_{t+1}. If more than one operation exists according to the dispatching rule p_t, tie is broken by a random choice.

Step 4: Update PS_{t+1} by removing the selected operation from S_t and adding the direct successor of the operation to S_t. Increment t by one.

Step 5: Return to step 2 until a complete schedule is generated.

Table 1: Dispatching Rules

1	SPT	*Shortest Process time*	6	PDR	*(Process /Remaining) Time*
2	EDD	*Earliest Due Date*	7	ERD	*Earliest Release Date*
3	LPT	*Longest Process Time*	8	MS	*Minimum Slack*
4	MWR	*Most Work Remaining Time*	9	LNS	*Largest Number of Successors*
5	LWR	*Least Work Remaining Time*	10	WSPT	*Weighted Shortest Process Time*

An initial feasible solution is generated randomly by assigning a dispatching rule to each position of the string.

2.2.2 Generation of neighborhood solutions

m number of neighborhood solutions are generated by changing the contents of *k* positions in each of the neighbor solution string. Positions in each string are selected randomly. Within the MOTS logic (Baykasoglu et al, 1999), the neighbor solutions must not be dominated by the seed string. Let us consider the string defined in section 2.2.1 as the seed string. If we set *m*=3, *k*=4 then the solution that is shown below can be obtained. In the string the selected locations are shaded. Next to the strings the performance values are given. In this case there are two objectives (e.g. makespan and total tardiness). The performance values are obtained by deducting the feasible schedules for each string using the Giffler and Thompson algorithm.

											Makespan	*T. Tardiness*
Seed String	**1**	**8**	**5**	**4**	**3**	**2**	**9**	**7**	**2**	**3**	**1**	*500*	*520*
Neighbor Strings	**1**	**7**	**5**	**4**	**3**	**1**	**9**	**7**	**2**	**2**	**6**	*420*	*600*
	1	**8**	**1**	**1**	**3**	**2**	**9**	**2**	**9**	**3**	**1**	*610*	*450*
	2	**8**	**5**	**4**	**8**	**2**	**9**	**4**	**2**	**5**	**1**	*400*	*580*

2.2.3 Selection

In MOTS algorithm, the neighbor solutions (strings), which are the Pareto-optimal strings, are determined by following the rules defined in Baykasoglu *et al* (1999). One of the Pareto-optimal-neighbor is randomly selected as the seed string. Other Pareto-optimal-neighbors (if any) are stored in *candidate list,* the selected string is put into the *Pareto list*. Solutions in the *candidate list* may later become seed solutions. Solutions in the candidate list play also a very important role in diversifying the search in later iterations Baykasoglu *et al* (1999).

2.2.4 Updating

The initial feasible solution vector (string) is recorded as the seed string and put into the *Pareto list*. In each iteration, one of randomly selected Pareto–optimal neighbor is recorded as the current seed string. These seeds are also stored in *Pareto list*. After each addition, the *Pareto list* is checked if the last Pareto-optimal solution dominates any other solutions in Pareto list that have been added previously. If there are solution(s) in *Pareto list*, which are dominated by the last Pareto-optimal solution, then these solution(s) are eliminated from the *Pareto list*. Other Pareto optimal neighbors, which are not selected as seed solution are stored in *candidate list* with the same control of domination. The details of the updating process can also be found in Baykasoglu *et al* (1999).

2.2.4 Tabu list

The position indexes of *k* locations of the seed string are stored in the tabu list. The tabu list is circular and can hold up to *w* elements; when it is full a new pair of indexes replace the head of the list.

2.2.5 Aspiration Criteria

Any string resulting from a move that is not dominated by the solutions in candidate and Pareto lists is accepted, even if the move is tabu.

2.2.6 Termination

If a previously determined number of iterations (*n-iter*) is reached, or if the candidate list is empty the MOTS algorithm terminates.

3. EXAMPLE APPLICATIONS

The proposed algorithm is programmed in C. The algorithm is tested on a Pentium III-MMX model PC at 450 MHz (128 MB RAM). In the present study 3 different objective functions are used, namely Total tardiness, Makespan and Load balance. In this section, the execution of the program is presented by using different case study problems.

3.1. Case Study 1

The first case study problem is taken from a web site (www.ms.ic.ac.ok/jeb/pub/jobshop1.txt) where different JSSP test problems with the optimal makespan values are available. The problem data is given below.

Number of parts: 6, Number of machines: 6

Machine route for the 1st part: 3 1 2 4 6 5 Due date: 26 Machine route for the 4th part: 2 1 3 4 5 6 Due date: 35
Machine route for the 2nd part: 2 3 5 6 1 4 Due date: 47 Machine route for the 5th part: 3 2 5 6 1 4 Due date: 25
Machine route for the 3rd part: 3 4 6 1 2 5 Due date: 34 Machine route for the 6th part: 2 4 6 1 5 3 Due date: 30

Processing time data

	P1	P2	P3	P4	P5	P6
M1	3	10	9	5	3	10
M2	6	8	1	5	3	3
M3	1	5	5	5	9	1
M4	7	4	4	3	1	3
M5	6	10	7	8	5	4
M6	3	10	8	9	4	9

The optimal makespan for this problem is known and it is equal to 55. The proposed MOTS algorithm found 45 Pareto optimal solutions including the one with optimal makespan around 5 seconds of computational time. The solutions within the set of Pareto optimal solutions with the best objective function values for each objective are shown in Table 2.

Table 2: The best values for each objective function in the Pareto-optimal set

Makespan : **55**	Makespan :71	Makespan :79
Total Tardiness : 115	Total Tardiness : **75**	Total Tardiness : 148
Load Balance : 0.144	Load Balance : 0.092	Load Balance : **0.065**

3.2. Case Study 2

The second case study problem is also taken from a web site (www.ms.ic.ac.ok/jeb/pub/jobshop1.txt). The problem data is given below.

Number of parts: 10, Number of machines: 5

Machine route for the 1st part: 2 1 5 4 3 Due date: 258 Machine route for the 6th part: 2 3 5 1 4 Due date: 330
Machine route for the 2nd part: 1 4 5 3 2 Due date: 186 Machine route for the 7th part: 4 5 2 3 1 Due date: 413
Machine route for the 3rd part: 4 5 2 3 1 Due date: 232 Machine route for the 8th part: 3 1 2 4 5 Due date: 246
Machine route for the 4th part: 2 1 5 3 4 Due date: 354 Machine route for the 9th part: 4 2 5 1 3 Due date: 233
Machine route for the 5th part: 1 4 3 2 5 Due date: 237 Machine route for the 10th part: 5 4 3 2 1 Due date: 370

Processing time data

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
M1	53	21	12	55	83	92	93	60	44	96
M2	21	71	42	77	19	54	87	41	49	75
M3	34	26	31	66	64	43	87	38	98	43
M4	55	52	39	77	34	62	69	24	17	79
M5	95	16	98	79	37	79	77	83	25	77

The optimal makespan for this problem is known and it is equal to 666. The proposed MOTS algorithm found 21 Pareto optimal solutions including the one with optimal makespan around 5 seconds of computational time. The

solutions within the set of Pareto optimal solutions with the best objective function values for each objective are shown in Table 2.

Table 2: The best values for each objective function in the Pareto-optimal set

Makespan : **666**	Makespan :857	Makespan :808
Total Tardiness : 2977	Total Tardiness : **2507**	Total Tardiness : 3298
Load Balance : 0.067	Load Balance : 0.111	Load Balance : **0.028**

The extensive computational study is still under progress. However the preliminary results are encouraging. In many cases MOTS algorithm is able to find good solutions.

4. CONCLUSION

In this research, a dispatching rule based multiple objective tabu search algorithm is developed to find Pareto-optimal solutions in JSSP. The proposed algorithm is tested with several test problems. The results have shown that the proposed algorithm is able to find competitive solutions. In order to test the behavior of the proposed algorithm an extensive computational study is under progress. The preliminary results have shown that the proposed algorithm can be considered an alternative algorithm for multiple objective job shop scheduling problems.

REFERENCES

1. Al-Fawzan, M. A., and Al-Sultan, K. S., 1996, "A tabu search algorithm for minimizing the makespan in a job shop scheduling", Proceedings of the 5th Industrial Engineering Research Conference, Minneapolis, MN, USA, pp. 115-119.
2. Baykasoglu, A., Owen, S. and Gindy, N., 1999, "A taboo search based approach to find the Pareto optimal set in multiple objective optimisation", J. of Eng. Opt., 31, 731-748.
3. Baykasoglu, A., Saad, S. M., and Gindy, N., 1998, "A loading approach for cellular manufacturing systems", FAIM'1998: 8th International Conference on Flexible Automation and Intelligent Manufacturing, July 1-3, Portland, Oregon, USA, pp.215-226.
4. Ben-Daya, M.,1994, "Solution methodologies for scheduling problems in flexible manufacturing systems", Int. Journal of Manufacturing Systems Design, 1(4), 315-328.
5. Blackstone, J. H., Phillips, D. T., and Hogg, G. L., 1982, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations", International Journal of Production Research, 20(1), 27-45.
6. Cheng, R., Gen, M., Tsujimura, Y., 1996, "A tutorial survey of job-shop scheduling problems using genetic algorithms-1. Representation", Computers ind Engng, 30(4), 983-997.
7. Dorndorf, U., Pesch, E., 1995, "Evolution based learning in a job shop scheduling environment", Complex Systems, 22, 25-40.
8. Giffler, B., and Thompson, G., 1960, "Algorithms for solving production scheduling problems", Operations Research, 8, 487-503.
9. Glover, F., 1990, "Tabu search: a tutorial", Interfaces, 20, 74-94.
10. Kusiak, A., 1990, Intelligent Manufacturing Systems, Prentice Hall.
11. Kusiak, A., and Chen, M., 1988, "Expert systems for planning and scheduling manufacturing systems", European Journal of Operational Research, 34, 113-130.
12. Mastrolilli, M., Gambardella L.M., 2000, "Effective neighbourhood functions for the flexible job shop problem", Journal of Scheduling, 3, 3-20.
13. Maturana, F., Gu, P., Naumann, A., and Norrie, D. H., 1997, "Object oriented job-shop scheduling using genetic algorithms", Computers in Industry, 32, 281-294.
14. Pinedo, M., 1995, Scheduling: Theory, Algorithms, and Systems, New Jersey: Prentice-Hall.
15. Sönmez, A. I., Baykasoglu, A., 1998, "A new dynamic programming formulation of (n*m) flowshop sequencing problems with due dates", International Journal of Production Research, 36(8), 2269-2283.
16. Sönmez, A. I., Baykasoğlu, A., 1998, "A new dynamic programming formulation of (n*m) flowshop sequencing problems with due dates", *International Journal of Production Research*, 36(8), 2269-2283.
17. Stoop, P. P. M., and Wiers, V. C. S., 1996, "The complexity of scheduling in practice", *Int. J. of Operations & Production Management*, 16(10), 37-53.
18. Van Laarhoven, P. J. M., Aarts, E. H. L., and Lenstra, J. K., 1992, "Job shop scheduling by simulated annealing", *Operations Research*, 40, 113-125.