Applied Mathematical Modelling 40 (2016) 5660–5671

Contents lists available at ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

Parallel machine scheduling with tool loading

Selin Özpeynirci[a,*], Burak Gökgür[b], Brahim Hnich[c]

[a] Industrial Engineering Department, İzmir University of Economics, Turkey
[b] College of Administrative Sciences and Economics, Koç University, Turkey
[c] Computer Engineering Department, İzmir University of Economics, Turkey

ARTICLE INFO

Article history:
Received 5 August 2014
Revised 19 October 2015
Accepted 6 January 2016
Available online 22 January 2016

Keywords:
Parallel machines
Scheduling
Tool assignment
Mixed integer programming
Tabu search

ABSTRACT

This paper presents a mixed integer programming approach that integrates the tool assignment and scheduling problems arising in parallel machine environments. There are a number of operations to be processed on parallel machines. Each operation requires a set of tools; however, the number of available tools are limited. Our objective is to minimize the makespan, i.e. the completion time of the final operation. We propose two different mathematical programming models for this problem. Since the problem is strongly NP-hard in general, finding the optimal solution requires extremely long computational times as the problem size increases. We therefore develop a tabu search algorithm in order to find near-optimal solutions within reasonable times.

1. Introduction

Parallel machine scheduling problems have attracted attention from both academia and industry for many years. Pinedo [1] states that parallel-machine problems are worth discussing from both a theoretical and a practical point of view. Cheng and Sin [2] and Mokotoff [3] provide literature survey of parallel machine scheduling problems.

There are several problems that may occur in a manufacturing environment, including tool loading, operation assignment, tool switching and scheduling. The *tool loading* problem is to assign the tools required to process jobs/operations on machines, *operation assignment* problem is to determine the machines that will process each operation. The *tool switching* problem, on the other hand, is to change the tools on machines, if necessary, so that the next operation in the sequence can be processed. The well-known *scheduling* problem is to determine the production sequence of jobs on machines. Crama [4] discusses combinatorial optimization problems that arise in automated manufacturing systems, and provides mathematical models and solution approaches. Blazewicz and Finke [5] provide a review of scheduling in manufacturing systems with resource management issues, such as tools, fixtures, pallets and automated guided vehicles. Yeh et al. [6], Cheng at al. [7] and Lin [8] all present mathematical programming models and heuristic algorithms for parallel machine scheduling problems.

In this paper, we consider an automated manufacturing environment with parallel machines. There are a number of operations; each must be processed on one machine and requires a set of tools to be loaded on the machine in advance. The number of tools available in the system is limited for several reasons, such as economic restrictions, inventory policies and managerial issues. Three important problems are considered simultaneously in this paper: operation assignment, tool loading and scheduling.

* Corresponding author. Tel.: +90 2324888259; fax: +90 232 279 26 26.
E-mail address: selin.ozpeynirci@ieu.edu.tr (S. Özpeynirci).

http://dx.doi.org/10.1016/j.apm.2016.01.006
S0307-904X(16)00009-3/

Several studies combine the scheduling and the resource allocation problems that occur mostly in flexible manufacturing environments. Edis et al. [9] provide a review of parallel machine scheduling problems with additional resources. Some studies approach the problem sequentially; Agnetis et al. [10] work on the joint part/tool scheduling problem in a flexible manufacturing cell consisting of two machines. Given a job sequence for two machines, the tool scheduling problem can be easily solved to optimality. Therefore, they introduce and compare several decomposition strategies. Kellerer and Strusevich [11] consider scheduling problems on parallel dedicated machines under multiple resource constraints, assuming that the machine to process each job is determined in advance. They introduce several cases and discuss computational complexity of the problems. Avci and Akturk [12] develop a sequential algorithm that first assigns tools to machines, then arranges tool magazines, and lastly, determines the operation sequence. Kumar and Sridharan [13] study the tool sharing problem in single-stage multi-machine flexible manufacturing systems. They consider three scenarios, and carry out a simulation analysis using different scheduling rules. They first sequence the parts using a dispatching rule, before loading the tools requested by the parts in sequence. Gamila and Motavalli [14] formulate loading and routing problems as a 0–1 integer programming model, using the output to generate a detailed schedule.

A small number of studies consider all problems simultaneously. Roh and Kim [15] simultaneously consider part loading, tool loading and part sequencing problems with the objective of minimizing total tardiness; hence, they include the due dates of the jobs in the problem. Also they assume that the processing times of jobs are not dependent on the machine. Rather than provide a mathematical model, they propose three heuristic approaches: (1) list scheduling approach that solves all problems simultaneously, (2) a sequential approach that first solves part loading and scheduling problems and then tool loading problem, and (3) an iterative approach. Through a simulation study, they show that an iterative approach provides the best performance. Ventura and Kim [16] consider a parallel machine scheduling problem where jobs may require additional resources. They assume unit processing time for each job. They consider two cases in which there is an arbitrary number of additional resources in the first and a single additional resource in the second. They formulate the first problem as an integer program and use Lagrangean relaxation to find bounds. Turkcan et al. [17] simultaneously study the machining conditions selection, the tool allocation, the loading and scheduling parts on non-identical parallel CNC machines, with the aim of minimizing the manufacturing cost and the total weighted tardiness. They develop a heuristic based on a genetic algorithm, and show the superiority of the simultaneous approach over the sequential approach. Aldrin Raj et al. [18] consider simultaneous scheduling of machines and tools in order to minimize makespan. They propose four heuristics: priority dispatching rules, modified non-delay schedule generation algorithm with six different priority dispatching rules, modified Giffler and Thompson algorithm and artificial immune system (AIS) algorithm. The results of computational experiments show that AIS gives the best results.

Recent studies have used a constraint programming approach for simultaneously solving loading and scheduling problems. Novas and Henning [19] propose a constraint programming model that takes into account machine loading, scheduling, parts routing, buffer scheduling, tool allocation and AGV scheduling. Zeballos [20] include tool life, number of tools in the system and tool magazine capacities in their constraint programming model. Zeballos et al. [21] consider a variety of constraints found in manufacturing environments, such as machine eligibility and upper limits on costs.

In the literature generally, operations assignment and scheduling, as well as the tool loading, are studied either separately or sequentially, due to the complexity of the problem. In this study, however, we aim to combine these problems and simultaneously optimize the whole system considering the important issues mentioned above.

Firms gain advantage in today's competitive market through the efficient use of limited resources; slight improvement in manufacturing systems can result in significant benefits. Managers can use the models developed in this paper by embedding in ERP software to support their operational level decisions, such as scheduling and resource allocation, as well as tactical level decisions, such as capacity planning.

The paper is organized as follows: In the next section, the problem definition is given. In Section 3, the mathematical models are defined and computational results are reported. In Section 4, a tabu search heuristic is presented, and the results obtained are discussed. We conclude the paper in Section 5 with some future research directions.

2. Problem definition

Consider n operations to be processed on m parallel machines. There is no pre-defined precedence relation between the operations. An operation can be assigned to exactly one machine, and pre-emption is not allowed, i.e. once the machine starts processing an operation, it cannot stop until the operation is complete. There is no part movement between the machines, hence no buffers are required. Every machine can process every operation; however, the processing time of an operation on different machines may vary. We let p_{ij} be the processing time of operation i on machine j. A set of tools $l(i)$ should be loaded on the machine before processing operation i. There are t tool types and r_k tools of type k are available. Due to economic restrictions, the number of tool copies available may be smaller than the number of machines.

Extending the standard notation for scheduling problems as in Kellerer and Strusevich [11], we can denote the problem under consideration by $Rm|res\lambda\sigma\rho|Cmax$ considering tools as resources. Here "Rm" corresponds to "m unrelated machines in parallel", while "$res\ \lambda\sigma\rho$" implies that there are λ resources, the size of each resource does not exceed σ, and each job consumes no more than ρ units of a resource. In our case, there are t resources with size r_k and each operation consumes at most 1 unit of each resource.

When there are no tooling considerations, i.e. when $r_k \geq m$, our problem reduces to the parallel machine scheduling problem, which is a well-known problem in the scheduling literature. Since the parallel machine scheduling problem is NP-hard in general, then the proposed problem is also NP-hard in general [22].

The assumptions about the tool usage are given below:

- Tools are not shared between the machines, i.e. the loaded tools are not removed during the processing.
- Tools are moved between the machines whenever necessary, but tool switching times are negligible.
- Tool magazines of the machines are initially empty.
- Each tool occupies one tool slot.
- Tools never break down during processing.
- The number of tools required by an operation does not exceed the tool magazine capacity of the machines.

The problem is to schedule the operations on parallel machines with their required tools so as to minimize the makespan, i.e., the time taken by the final operation to finish its processing in the system, which is an important performance measure in the scheduling literature. The tool copies must be assigned to one or more machines, depending on demand for them by the operations scheduled on machines, taking into account timing and overlapping issues.

3. Mathematical programming models

In this section, we propose two different mathematical programming models and give some computational results.

3.1. Mixed Integer Programming Model 1

We first define the parameters and decision variables used in the mathematical model.

Indices:

i, q:	operation index, $i, q = 1, \ldots, n$
j:	machine index, $j = 1, \ldots, m$
k:	tool type index, $k = 1, \ldots, t$
h_k:	tool copy index, $h_k = 1, \ldots, r_k$

Parameters:

p_{ij}:	processing time of operation i on machine j
r_k:	number of tools of type k
$l(i)$:	set of tools required to process operation i
M:	a very large number

Decision variables:

$Cmax$:	production makespan
C_i:	production completion time of operation i
S_{ij}:	production start time of operation i on machine j
X_{ij}:	1, if operation i is processed on machine j; 0, otherwise
Z_{ih_kj}:	1 if copy h of tool k is used to process operation i on machine j; 0, otherwise
Y_{iqj}:	1, if operation i precedes operation q on machine j; 0, otherwise
A_{iqh_k}:	1, if operations i and q use tool copy h_k; 0, otherwise
b_{iq}:	1, if operation i precedes operation q when they use the same tool copy; 0, otherwise

The mathematical model, that will be referred to as MIP1 is given below:

$$(MIP1)\ Minimize\ Cmax \tag{1}$$

$$s.t.$$

$$Cmax \geq C_i, \quad \forall i \tag{2}$$

$$C_i = \sum_j (S_{ij} + p_{ij}X_{ij}), \quad \forall i \tag{3}$$

$$S_{ij} \leq M(X_{ij}), \quad \forall i, j \tag{4}$$

$$X_{ij} \leq \sum_{h_k} Z_{ih_kj}, \quad \forall i, j, \quad \forall h_k | k \in l(i) \tag{5}$$

$$\sum_j X_{ij} = 1, \quad \forall i \tag{6}$$

$$S_{ij} \geq S_{qj} + p_{qj} - M(Y_{iqj}) - M(1 - X_{qj}) - M(1 - X_{ij}), \quad \forall q, \quad \forall i \neq q, \quad \forall j \tag{7}$$

$$X_{ij} + X_{qj} \geq 2(Y_{iqj} + Y_{qij}), \quad \forall q, \quad \forall i \neq q, \quad \forall j \tag{8}$$

$$X_{ij} + X_{qj} \leq Y_{iqj} + Y_{qij} + 1, \quad \forall q, \quad \forall i \neq q, \quad \forall j \tag{9}$$

$$\sum_j Z_{ih_kj} + \sum_j Z_{qh_kj} \geq 2(A_{iqh_k}), \quad \forall q, \quad \forall i < q, \quad \forall h_k | k \in l(i) \cap l(q) \tag{10}$$

$$\sum_j Z_{ih_kj} + \sum_j Z_{qh_kj} \leq A_{iqh_k} + 1, \quad \forall q, \quad \forall i < q, \quad \forall h_k | k \in l(i) \cap l(q) \tag{11}$$

$$\sum_{h_k} A_{iqh_k} \leq M(b_{iq} + b_{qi}), \quad \forall q, \quad \forall i < q, \quad \forall h_k | k \in l(i) \cap l(q) \tag{12}$$

$$b_{iq} + b_{qi} \leq 1, \quad \forall q, \quad \forall i < q \tag{13}$$

$$\sum_j S_{ij} \geq \sum_j (S_{qj} + p_{qj} X_{qj}) - M(1 - b_{qi}), \quad \forall q, \quad \forall i \neq q \tag{14}$$

$$\sum_j \sum_{h_k} Z_{ih_kj} \leq 1, \quad \forall i, k \in l(i) \tag{15}$$

$$Cmax \geq 0, \tag{16}$$

$$C_i \geq 0, \quad \forall i \tag{17}$$

$$S_{ij} \geq 0, \quad \forall i, j \tag{18}$$

$$X_{ij} \in \{0, 1\}, \quad \forall i, j \tag{19}$$

$$Z_{ih_kj} \in \{0, 1\}, \quad \forall i, j, k \in l(i) \tag{20}$$

$$Y_{iqj} \in \{0, 1\}, \quad \forall q, \quad \forall i \neq q, j \tag{21}$$

$$A_{iqh_k} \in \{0, 1\}, \quad \forall q, \quad \forall i < q, \quad \forall h_k | k \in l(i) \cap l(q) \tag{22}$$

$$b_{iq} \in \{0, 1\}, \quad \forall q, \quad \forall i \neq q \tag{23}$$

The objective function (1) minimizes the production makespan. Constraint (2) ensures that the completion time of any operation is less than or equal to the production makespan. Constraint (3) defines the completion time of the operations as the summation of starting time and processing time. Constraint (4) forces the start time of an item to be equal to zero for the machines to which it is not assigned. In constraint (5), an operation can be assigned to a machine only if its required tools are loaded. Constraint (6) guarantees that an operation is processed on only one machine. Constraints (7)–(9) are disjunctive constraints, which provide that a machine can process only one operation at a time. Constraint (7) ensures that if one operation precedes another on the same machine, then the succeeding operation starts after the completion of the preceding one. Constraints (8) and (9) allow an operation to precede another if and only if they are assigned to the same machine. Constraints (10)–(14) are also disjunctive constraints, which ensure that a tool copy can be used by one operation at any given time. Constraints (10) and (11) ensure that if operations i and q require the same tool copy, h_k, then A_{iqh_k} is equal to 1. Constraints (12) and (13) force a precedence relation between two operations if they use the same tool copy. Also, due to constraint (14), if there is a precedence relation between two operations resulting from the usage of the same tool copy, then the starting time of one operation is greater than or equal to the completion time of the other. Constraint (15) ensures that an operation uses only one copy of a tool. Constraints (16)–(23) are the set constraints.

3.2. Mixed Integer Programming Model 2

In this section, we present a more compact mathematical model, which will be referred to as MIP2. The same indices and parameters are used in both models. Also, $Cmax$, C_i, S_{ij}, X_{ij} and Y_{iqj} variables are used as in MIP1. Let ST be the set of operation pairs that require the same tool with one copy, and let MT be the set of operation pairs that require the same tool with more than one copy. The decision variables introduced additionally for MIP2 are given below.

Decision variables:

W_{ikh}: 1, if job i uses copy h of tool k; 0, otherwise
U_{iq}: 1, if job i precedes job q when they both require a tool with one copy; 0, otherwise
V_{iq}: 1, if job i precedes job q when they both require a tool with more than one copy; 0, otherwise

The objective function and the constraints of MIP2 are given below:

$$(MIP2)\ Minimize\ Cmax \tag{24}$$

$$s.t.$$
$$(2)\text{–}(4),\ (6)$$
$$\sum_j S_{ij} \geq \sum_j (S_{qj} + p_{qj}X_{qj}) - M(1 - U_{qi}), \quad \forall (i,q) \in ST \tag{25}$$

$$U_{iq} + U_{qi} = 1, \quad \forall (i,q) \in ST \tag{26}$$

$$\sum_j S_{ij} \geq \sum_j (S_{qj} + p_{qj}X_{qj}) - M(2 - (W_{ikh} + W_{qkh})) - M(1 - V_{qi}),$$
$$\forall (i,q) \in MT, \quad \forall k | k \in l(i) \cap l(q), \quad r_k > 1, \quad h = 1, \ldots, r_k \tag{27}$$

$$V_{iq} + V_{qi} \leq 1, \quad \forall (i,q) \in MT \tag{28}$$

$$W_{ikh} + W_{qkh} \leq 2(V_{iq} + V_{qi}), \quad \forall (i,q) \in MT, \quad \forall k | k \in l(i) \cap l(q), \quad r_k > 1, \quad h = 1, \ldots, r_k \tag{29}$$

$$S_{ij} \geq S_{qj} + p_{qj} - M(Y_{iqj}) - M(1 - X_{qj}) - M(1 - X_{ij}), \quad \forall i \neq q, j \tag{30}$$

$$X_{ij} + X_{qj} \geq 2(Y_{iqj} + Y_{qij}), \quad \forall i \neq q, j \tag{31}$$

$$X_{ij} + X_{qj} \leq Y_{iqj} + Y_{qij} + 1, \quad \forall i \neq q, j \tag{32}$$

$$\sum_{h=1}^{r_k} W_{ikh} = 1, \quad \forall i, k \in l(i) \tag{33}$$

$$W_{ih_k} \in \{0,1\}, \quad \forall i, \quad \forall k \in l(i), \quad h = 1, \ldots, r_k \tag{34}$$

$$U_{iq} \in \{0,1\}, \quad \forall i \neq q \tag{35}$$

$$V_{iq} \in \{0,1\}, \quad \forall i \neq q$$
$$(16)\text{–}(19),\ (21) \tag{36}$$

Constraints (25) and (26) ensure that two operations requiring a tool with one copy are not scheduled simultaneously. Similarly, constraints (27)–(29) guarantee that two operations using the same copy of a tool with more than one copy are not processed simultaneously. Due to constraints (30)–(32), processing times of the operations assigned to the same machine do not overlap. Lastly, constraint (33) provides the assignment of required tools to operations. Constraints (34)–(36) are the set constraints.

The main improvement made in MIP2 is the reduction of the number of decision variables and constraints. The variables Z_{ih_kj} are replaced by variables W_{ih_k}. Since we have already obtained the operation-machine assignment information from variables X_{ij}, there is no need for an additional index for machines. Furthermore, variables A_{iqh_k} are not required in the new model because the same information can be obtained from variables W_{ih_k} and W_{qh_k}, resulting in a more compact model, MIP2.

3.3. An illustrative example

In this section, a small example is given to illustrate the problem environment and the solution structure on a Gantt chart. Consider a production environment with 10 operations to be processed on 3 parallel machines with 8 types of tools. Processing times, tool requirements and number of tool copies are given in Tables 1, 2 and 3, respectively.

The makespan is found to be 276 min. The results and the Gantt chart are given in Table 4 and Fig. 1, respectively. As can be seen in the Gantt chart, there are idle times due to the limited number of tool copies. For instance, operation 4 cannot start processing immediately after operation 8 on machine 1. Operations 1 and 4 both require tool type 4, of which there is only one copy, forcing operation 4 to wait for the completion of operation 1, so that tool type 4 is moved from machine 3 to machine 1.

Table 1
Processing times of the operations on each machine.

Machine	Operation									
	1	2	3	4	5	6	7	8	9	10
1	62	56	137	75	105	55	75	66	102	36
2	115	135	122	138	70	144	39	96	99	138
3	40	148	31	105	84	137	35	57	133	106

Table 2
Set of tools required by each operation.

Operation	1	2	3	4	5	6	7	8	9	10
Tools	4, 5, 8	1, 7	2, 6, 7, 8	4, 5	3, 6, 8	2, 5	5, 7	1, 5, 7	2, 3, 7, 8	4, 8

Table 3
Number of tool copies.

Tool	1	2	3	4	5	6	7	8
Number of copies	1	2	2	1	2	2	2	1

Table 4
Results of the illustrative example.

Operation	Machine	Starting time	Completion time	Tools (copies) used
1	3	31	71	4(1), 5(1), 8(1)
2	3	93	241	1(1), 7(2)
3	3	0	31	2(1), 6(1), 7(1), 8(1)
4	1	71	146	4(1), 5(1)
5	2	71	141	3(1), 6(1), 8(1)
6	1	146	201	2(1), 5(1)
7	3	241	276	5(2), 7(1)
8	1	0	66	1(1), 5(2), 7(2)
9	2	141	240	2(2), 3(1), 7(1), 8(1)
10	1	240	276	4(1), 8(1)

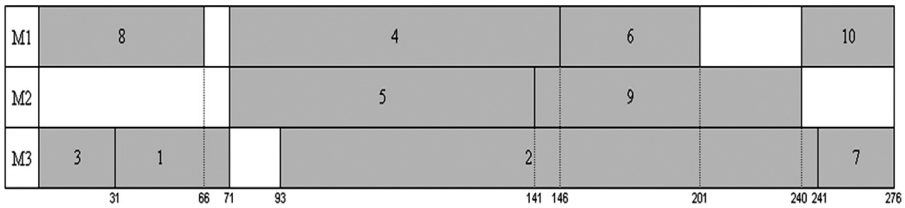

Fig. 1. Gantt chart of the illustrative example.

3.4. Comparison of MIP1 and MIP2

In this section, we compare the performances of models MIP1 and MIP2 on a set of test problems. In our experiments, we set the number of operations to 8, 10 or 15. We set the number of machines to 2 or 3, and the number of tool types to 5 or 8. The number of tools in the set $l(i)$ is generated from a discrete uniform distribution between 2 and 5. The tools in set $l(i)$ are generated randomly. The value of r_k is randomly selected from the set {1,2}. Lastly, the p_{ij} values are generated from a discrete uniform distribution in the interval [25,150]. 10 problem instances are generated under each parameter setting.

The models are solved by IBM OPL 6.1.1. We set an upper time limit of 1 h, and we terminate the execution of the algorithm if the optimal solution cannot be found within this time limit.

Tables 5 and 6 give the number of optimal solutions (Opt) found in the time limit, the minimum, average and maximum solution time (Time) of the optima in seconds, and the minimum, average and maximum percentage gap (Gap) for the instances whose optimal solution cannot be found within the time limit for models MIP1 and MIP2, respectively.

It can be seen from Tables 5 and 6 that MIP2 can find the optimal solution to the same problem instances as MIP1, and additionally two more optimal solutions. Also, compared to MIP1, the average solution time for MIP2 is on average 56.59% shorter. The improved performance of MIP2 is due to the reduced number of variables and constraints, as discussed earlier. On average, MIP2 has 52.95% fewer variables and 36.43% fewer constraints, compared to MIP1. Therefore, we can conclude

Table 5
Performance of MIP1.

⟨n, m, t⟩	Opt	Time			Gap (%)		
		Min	Average	Max	Min	Average	Max
⟨8, 2, 5⟩	10	39.39	247.64	804.13	–	–	–
⟨8, 2, 8⟩	10	8.55	61.89	135.11	–	–	–
⟨10, 2, 5⟩	4	334.98	1404.18	2320	6.64	21.39	32.73
⟨10, 2, 8⟩	9	82.84	468.54	1266.3	7.14	7.14	7.14
⟨10, 3, 5⟩	2	1308.91	1732.45	2156	19.14	29.79	36.57
⟨10, 3, 8⟩	10	101.61	1536.33	2965.92	–	–	–
⟨15, 2, 8⟩	0	–	–	–	22.27	45.08	56.64

Table 6
Performance of MIP2.

⟨n, m, t⟩	Opt	Time			Gap (%)		
		Min	Average	Max	Min	Average	Max
⟨8, 2, 5⟩	10	2.51	141.95	311	–	–	–
⟨8, 2, 8⟩	10	1.48	14.63	46.12	–	–	–
⟨10, 2, 5⟩	4	34.21	788.30	2674	18.75	34.37	40.84
⟨10, 2, 8⟩	10	4.72	183.19	822	–	–	–
⟨10, 3, 5⟩	3	209	1118	2426	8.58	28.18	46.91
⟨10, 3, 8⟩	10	7.34	120.35	333	–	–	–
⟨15, 2, 8⟩	0	–	–	–	31.60	52.34	65.87

Table 7
A small example of tool copy symmetry.

Operation	Required tools
1	1,2,4,5
2	1,2,3,4
3	1,3
4	1,3,5
5	1,2,4

that MIP2 is the more efficient mathematical programming model of the two. Nevertheless, neither model can solve any of the instances when the number of operations is 15 or higher.

3.5. Breaking tool copy symmetry in MIP2

The copies of a particular tool are indistinguishable from each other. Many alternative optimal solutions can therefore be created by renumbering the tool copies. The solution times may be improved significantly by breaking this symmetry, hence eliminating the alternative solutions.

Few studies in the literature discuss symmetry breaking issues in mathematical programming. Jans [23] considers the lot sizing problems on parallel identical machines, proposing mathematical programming formulations and constraints in order to break the symmetry caused by parallel identical machines. Different approaches for symmetry breaking are possible in constraint programming and mathematical programming: reformulation (Degraeve et al. [24]), symmetry-breaking constraints (Sherali and Smith [25], Sherali et al. [26]) and dynamic symmetry breaking (Margot [27,28]).

We impose symmetry-breaking constraints to MIP2 for the tools with multiple copies. Consider the small example given in Table 7. Suppose tool type 4 has only one copy, while others have 2.

Tool type 5 is required by operations 1 and 4. The solution in which operation 1 uses copy 1 and operation 4 uses copy 2 ($W_{151} = 1$ and $W_{452} = 1$) is equivalent to the solution where operation 1 uses copy 2 and operation 4 uses copy 1 ($W_{152} = 1$ and $W_{451} = 1$). In order to force the model to consider one of these, the following constraint can be written without affecting the optimal solution:

$2^2W_{151} + 2^1W_{451} \leq 2^2W_{152} + 2^1W_{452}$

Similarly, for tool 2, the following constraint can be written:

$2^3W_{121} + 2^2W_{221} + 2^1W_{521} \leq 2^3W_{122} + 2^2W_{222} + 2^1W_{522}$

Let N_k be the number of operations that require tool k and z_{fk} be the fth operation that requires tool k for $f = 1, \ldots, N_k$. For example, $z_{13} = 2, z_{23} = 3$ and $z_{33} = 4$. We can generalize the symmetry breaking constraints as below:

$$\sum_{f=1}^{N_k} 2^{(N_k+1)-f} W_{z_{fk}kg} \leq \sum_{f=1}^{N_k} 2^{(N_k+1)-f} W_{z_{fk}k(g+1)}, \quad \forall g = 1, \ldots, r_k - 1, \quad \forall k | r_k > 1 \tag{37}$$

Table 8
Performance of MIP2 with symmetry breaking.

⟨n, m, t⟩	Opt	Time			Gap (%)		
		Min	Average	Max	Min	Average	Max
⟨8, 2, 5⟩	10	5.70	9.22	15.13	–	–	–
⟨8, 2, 8⟩	10	4.64	5.89	9.67	–	–	–
⟨10, 2, 5⟩	10	11.62	328.15	1442.02	–	–	–
⟨10, 2, 8⟩	10	8.81	78.35	623.20	–	–	–
⟨10, 3, 5⟩	10	9.75	239.88	619.89	–	–	–
⟨10, 3, 8⟩	10	5.53	16.963	45.33	–	–	–
⟨15, 2, 8⟩	2	1053.80	2021.78	2989.76	16.92	30.75	45.87

3.6. The effect of breaking tool copy symmetry in MIP2

The results obtained by adding the symmetry-breaking constraints to MIP2 are summarized in Table 8. Table 8 gives the number of optimal solutions (Opt) found in the time limit; the minimum, average and maximum solution time (Time) of the optima in seconds; and the minimum, average and maximum percentage gap (Gap) for the instances whose optimal solution cannot be found in the time limit for MIP2 with symmetry breaking constraints. Compared with Table 6, the number of optimal solutions increases, and the solution time decreases significantly with the decreasing number of alternative solutions, due to the symmetry caused by the tool copies.

Although the performance of the model is improved by symmetry-breaking constraints, it is still impossible to solve the problem instances with more than 15 operations. There is clearly a need for an efficient algorithm that runs in reasonable time and returns near-optimal solutions. In the next section, the tabu search algorithm developed for this purpose is presented.

4. Tabu search algorithm

Tabu search is a metaheuristic introduced by Glover [29] to find solutions to combinatorial optimization problems. The tabu search algorithm starts with an initial feasible solution, and at each iteration, moves to another solution in the defined neighborhood of the current one. Generally the best solution in the neighborhood is selected for the move; however this solution is not required to be an improving one. It is forbidden to repeat a move for a specified number of iterations; this number is known as the tabu tenure, and forbidden moves are called tabu. The aim is to prevent repeated visits to the same solution, i.e. limiting the search to a certain area of the solution space. A tabu move can only be made when it is an improvement on the best solution found so far; this condition is called aspiration criterion.

The Tabu search method employs two strategies: intensification and diversification. The intensification strategy searches for moves in the neighborhood of the current solution. After making a specified number of moves in the starting neighborhood, the algorithm jumps to a solution in a different neighborhood via the diversification strategy. The frequency memory retains the moves made during the intensification phase, and helps to find an initial solution in a different part of the search space. The algorithm terminates after the diversification strategy has been employed a specified number of times.

Detailed information about tabu search can also be found in Glover and Laguna [30]. The tabu search algorithm proposed in this study is defined below.

4.1. Initial solution

The initial solution is found by a greedy heuristic that selects the operations according to a priority rule, similar to the one used by Roh and Kim [15], and assigns to the first available machine with the required tools. Tools that are required to process the next operation but unavailable at a particular machine must be taken from another machine, if available, or otherwise, from the tool crib. If there is no available tool copy on other machines or the tool crib, then the transfer must be delayed until the process of the operation using the required tool is finished. If there is no empty tool slot on the tool magazine of the machine, then the tools used for processing previous operations but not necessary for the current one are removed to allow the loading of the new tools. The stepwise description of the algorithm that finds an initial feasible solution is given below.

Let S_0 be the set of operations that have not yet been assigned to a machine, and S_1 be the set of operations that have been assigned to a machine.

Step 0: Set $S_0 = \{1,2,\ldots,n\}$ and $S_1=\{\}$. List the operation-machine pairs in non-decreasing order of processing times. Select the first operation-machine pair in the list and assign the operation to the machine. Set the starting time equal to zero (all tool copies are available at the beginning). Then, assign the second operation to the machine from the list. Set the starting time equal to zero if all tools required by the operation have available copies. Otherwise, insert idle time until at least one copy of each required tool is free. Continue until one operation is assigned to each machine. Find the completion times of the assigned operations on corresponding machines. Update sets S_0 and S_1.

Step 1: Find the machine that becomes idle first, i.e. the machine with the minimum completion time. Let the machine be j. Calculate the priority values for all operations in set S_0 on machine j, using the following equation:

$$\pi_{ij} = p_{ij} \times a_{ij} \times b_{ij}^2 \tag{38}$$

where

π_{ij}: priority value of operation i on machine j
a_{ij}: the number of tools needed additionally to assign operation i to machine j that we have available copy for
b_{ij}: the number of tools needed additionally to assign operation i to machine j that we do not have available copy for

By calculating the priorities of the operations as above, we give higher importance to the operations that have shorter processing times and that demand fewer additional tools. If an operation requires an unavailable tool, we need to insert idle time until the tool becomes free. Hence, we take the square of b_{ij} to increase the π_{ij} values of these operations.

Select the operation with minimum priority value. Let the operation be i.

Step 2: Remove the tools from machine j that are not elements of $l(i)$.

Step 3: Load the tools that are elements of $l(i)$ to machine j and that are not already loaded on machine j. If a required tool is not free, then delay the starting time of operation i on machine j until the tool becomes free, i.e. the operation that uses the tool is completed. Update sets S_0 and S_1. If $S_0 = \{\}$, then stop. Otherwise, go to Step 1.

4.2. Solution representation and neighborhood structure

The solution is represented by the sequence of operations that will be processed on each machine. The sample representation is below:

Machine 1: Operation 1 - Operation 5 - Operation 7 - Operation 8
Machine 2: Operation 2 - Operation 9 - Operation 10
Machine 3: Operation 3 - Operation 4 - Operation 6

The neighborhood of a solution is generated in two ways:

1. Swap two operations without considering whether or not they are assigned to different machines.
2. Remove an operation from its position and insert it into another location on either the same or a different machine.

4.3. Tabu moves

When an operation is assigned to a position on a machine, removing that operation from its place is considered tabu for a certain number of iterations, called tabu tenure. An excessively high tabu tenure may prevent further possible moves after several iterations; but if too low, the search will be confined to the same region.

If a move results in a solution that is better than the best solution obtained so far, then the tabu status is removed, i.e. the move is made regardless.

4.4. Intensification and diversification phases

In the intensification phase of the algorithm, i.e. the inner loop, we search the neighborhood of the current solution and move to the best feasible neighbor solution that is not tabu. This allows us to exploit the search space around the current solution. The inner loop is terminated when a pre-specified number of iterations is reached.

In the diversification phase of the algorithm, a search is made towards unvisited regions. In this phase, the processing times of the operations are updated considering frequency-based memory, and the algorithm starts with a different initial solution. Therefore, the exploration of the search space is provided.

Frequency based memory records the number of times that operation i is assigned to machine j by using an array $f[i][j]$. When operation i is assigned to machine j, we increment $f[i][j]$ by one.

When we terminate the inner loop, we update the processing time of operation i on machine j by adding the frequency of operation-machine pair to the processing time, i.e. $p_{ij} = p_{ij} + f[i][j]$. Then, a new initial solution is found with the updated p_{ij} values, and the inner loop is restarted. By this method, the operations will be assigned to different machines, and different regions in the solution space will be searched. The outer loop is repeated for a pre-specified number of iterations.

4.5. Stepwise description of tabu search algorithm

In this section, the stepwise description of application for our tabu search algorithm is provided. The following notation is used in the description.

N^i: the iteration number of the inner loop
N^o: the iteration number of the outer loop
N^i_{max}: the maximum number of iterations in the inner loop
N^o_{max}: the maximum number of iterations in the outer loop
Z^c: the makespan value of the current solution

Table 9
Performance of tabusearch.

⟨n, m, t⟩	Opt	Time			Deviation		
		Min	Average	Max	Min	Average	Max
⟨8, 2, 5⟩	9	67.75	107.06	141.61	0	0.31	3.07
⟨8, 2, 8⟩	7	91	112.51	135.40	0	3.19	15.35
⟨10, 2, 5⟩	10	157.76	208.91	266.04	0	0	0
⟨10, 2, 8⟩	3	58	191.47	2229	0	3.47	9.95
⟨10, 3, 5⟩	10	229.71	291.09	342.63	0	0	0
⟨10, 3, 8⟩	6	166.02	229.99	302.07	0	2.26	11.61
⟨15, 2, 8⟩	0	566.58	635.58	721.91	1.47	5.74	12.89

Z^b: the makespan value of the best neighbor of the current solution
Z^*: the makespan value of the incumbent solution
tt: tabu tenure
ts^c: tabu status of the current solution
ts^b: tabu status of the best neighbor of the current solution

Step 0: Set $Z^c = \infty$, $Z^* = \infty$ and $N^o = 0$.

Step 1: Set $N^i = 1$ and $N^o = N^o + 1$. Set $f[i][j] = 0, \forall i, j$. Find an initial solution using the greedy heuristic described in Section 4.1 and consider this solution as the current solution. Suppose the makespan value of initial solution is Z^i. Set $Z^c = Z^i$ and $ts^c = tt$. Set $Z^* = Z^i$ if $Z^i < Z^*$. Set $f[i][j] = f[i][j] + 1$ if operation i is assigned to machine j in the current solution.

Step 2: Find all neighbors of the current solution.

Step 3: Select the neighbor with the minimum makespan value (call it the best neighbor).

a. If $ts^b = 0$, go to Step 4.
b. If $ts^b > 0$,

   i. If $Z^b < Z^*$, go to Step 4.
   ii. If $Z^b > Z^*$, remove the best solution from the list of neighbors of current solution and go to Step 3.

Step 4: Consider the best neighbor as the current solution. Set $Z^c = Z^b$, $ts^c = tt$ and reduce the tabu status of all solutions by one if it is currently positive. Set $Z^* = Z^b$ if $Z^b < Z^*$. Set $f[i][j] = f[i][j] + 1$ if operation i is assigned to machine j in the current solution. Set $N^i = N^i + 1$.

a. If $N^i \leq N^i_{max}$, go to Step 2.
b. If $N^i > N^i_{max}$,

   i. If $N^o < N^o_{max}$, set $p_{ij} = p_{ij} + f[i][j], \forall i, j$ and go to Step 1.
   ii. If $N^o = N^o_{max}$, stop.

The flow chart of tabu search algorithm can bee seen in Fig. 2.

4.6. Experimental results

The tabu search heuristic is tested on problem instances defined in Section 3.3. The tabu tenure is set to be $\lceil \sqrt{n} \rceil$; the numbers of iterations for the inner and outer loops are set as 100 and 30, respectively. These parameters are determined after preliminary experimentation. Therefore, a total of 3000 iterations are performed for each problem instance. The Tabu search is coded in the Microsoft Visual Studio 2008 C language version.

In Table 9 the results are shown for the tabu search heuristic: the number of optimal solutions found (Opt); the minimum, average and maximum solution time in seconds (Time); and the minimum, average and maximum deviations from the optimal solution (Deviation). For the problems that can be solved to optimality, the percent deviations from the optimal solution are reported. For the problems whose optimal solutions are unknown, percent deviations from the upper bound found by CPLEX within 1 h are given.

Table 9 shows that the tabu search algorithm obtains near-optimal results. The solution quality does not have a significant pattern with respect to the problem size; however, the solution time increases with the number of operations, due to the high number of neighbor solutions. For larger problems, the solution time may increase further. Nevertheless, we can predict that tabu search will still find good quality solutions in reasonable times due to the robustness of the solution quality.

We employ t-test to the distance from optimum solution to study the robustness of the proposed tabu search algorithm. To do so, the following hypothesis is designed:

$$H_0 : \mu \leq 0.01 \tag{39}$$

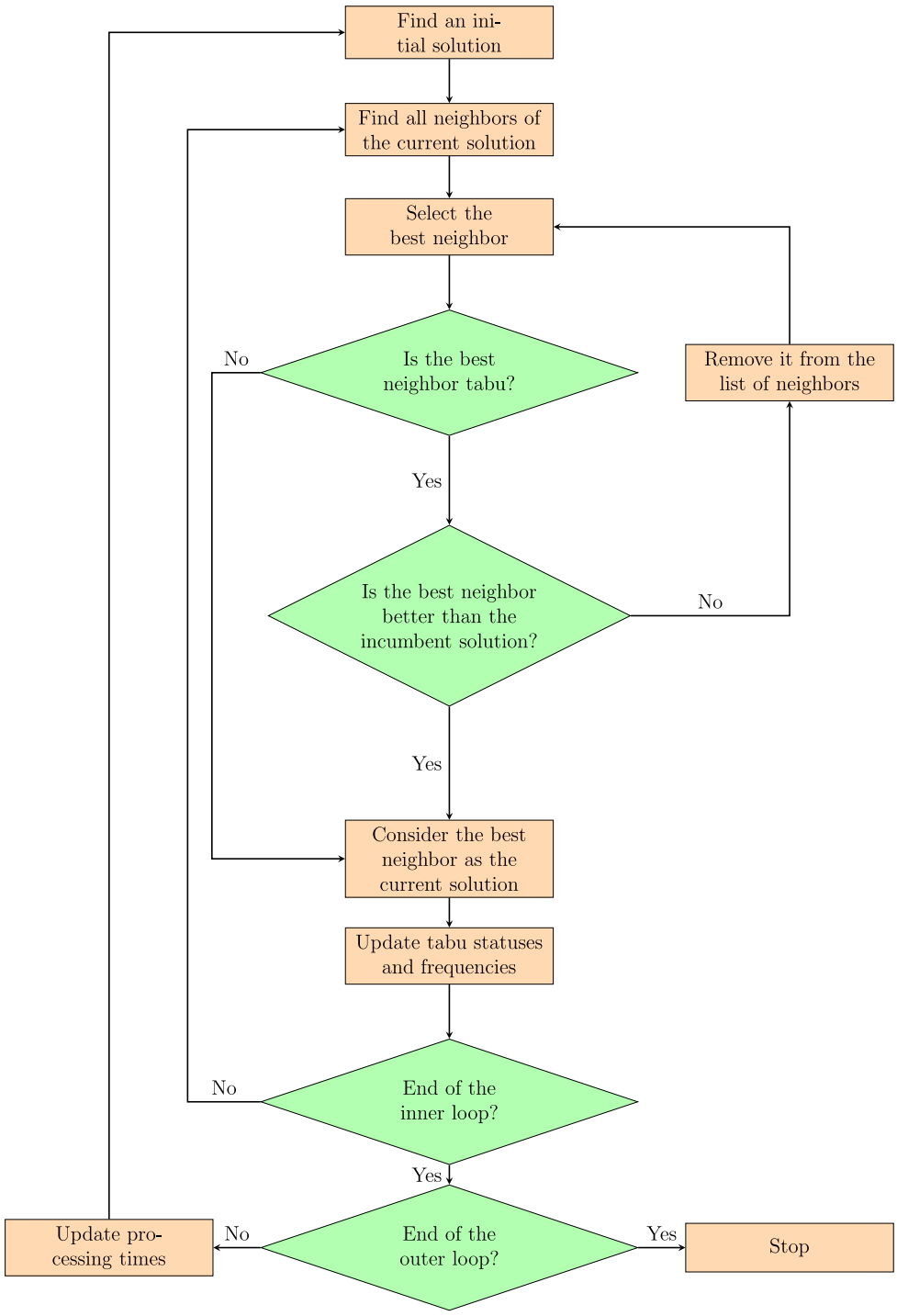

Fig. 2. Tabu search algorithm.

$$H_1 : \mu \geq 0.01 \tag{40}$$

with a $1-\alpha = 0.95$ confidence. There are 62 problem instances, of which we know the optimal solution. The value of t statistic is obtained by the following equation:

$$t = \frac{\overline{d} - 0.01}{S/\sqrt{n}} \tag{41}$$

where $\bar{d}$ and S stand for sample mean and standard deviation, respectively. These parameters are calculated as $\bar{d} = 0.017226$ and $S = 0.037846$ and hence the value of test statistic as $t = 1.5034$. Since $t_{0.05,\,61} = 1.67 > 1.5034$, the null hypothesis cannot be rejected, and we can claim that the expected distance of the solutions obtained by the proposed tabu search from globally optimum ones is less than 1%.

5. Conclusion

In this study, we consider the scheduling of operations on a group of parallel machines together with their required tools with the objective of minimizing the makespan. In the literature, with a small number of exceptions, the operation scheduling and the tool assignment problems have been studied separately. This paper differs in that the two problems are considered simultaneously. The presented models may be used for any type of resource required during the processing of operations, such as tools, workers for manual operations, fixtures, etc.

We provide two mathematical models for the problem. The second model improves on the first by reducing the number of variables and constraints. We also aim to reduce the number of possible solutions, and thus, the tree size, by breaking tool copy symmetry to further improve the second model. A number of problem instances were generated and solved using the mathematical models. The results show that, as expected, large sized problems cannot be solved to optimality within a one-hour time limit due to the difficulty of the problem considered. In future studies, in order to arrive at a solution within a more acceptable time limit, approaches such as constraint programming or Lagrangean relaxation will be appropriate.

We also propose a tabu search algorithm that gives near-optimal results within reasonable time limits. The statistical analysis shows that we cannot reject the hypothesis that the average deviation of the results of tabu search from the optimal solution does not exceed 1%. Unfortunately, it is impossible to compare the results of tabu search algorithm with other approaches since there exists no work in the literature on the particular problem studied in this paper.

Switching times, which are considered negligible in this study, may also be given consideration in future studies.

Acknowledgment

This work is supported by The Scientific and Technological Research Council of Turkey, grant no: 110M492.

References

[1] M. Pinedo, Scheduling: Theory, Algorithms and Systems, Prentice-Hall, New Jersey, 2008.
[2] T.C.E. Cheng, C.C.S. Sin, A state-of-the-art review of parallel-machine scheduling research, Eur. J. Oper. Res. 47 (3) (1990) 271–292.
[3] E. Mokotoff, Parallel machine scheduling problems: a survey, Asia-Pac. J. Oper. Res. 18 (2) (2001) 193–242.
[4] Y. Crama, Combinatorial optimization models for production scheduling in automated manufacturing systems, Eur. J. Oper. Res. 99 (1997) 136–153.
[5] J. Blazewicz, G. Finke, Scheduling with resource management in manufacturing systems, Eur. J. Oper. Res. 76 (1994) 1–14.
[6] W. Yeh, M. Chuang, W. Lee, Uniform parallel machine scheduling with resource consumption constraint, Appl. Math. Model. 39 (8) (2015) 2131–2138.
[7] B. Cheng, S. Yang, X. Hu, B. Chen, Minimizing makespan and total completion time for parallel batch processing machines with non-identical job sizes, Appl. Math. Model. 36 (2012) 3161–3167.
[8] Y. Lin, Fast LP models and algorithms for identical jobs on uniform parallel machines, Appl. Math. Model. 37 (2013) 3436–3448.
[9] E. Edis, C. Oguz, I. Ozkarahan, Parallel machine scheduling with additional resources: Notation, classification, models and solution methods, Eur. J. Oper. Res. 230 (2013) 449–463.
[10] A. Agnetis, A. Alfieri, P. Brandimarte, P. Prinsecchi, Joint job/tool scheduling in a flexible manufacturing cell with no on-board tool magazine, Comput. Integr. Manuf. Syst. 10 (1997) 61–68.
[11] H. Kellerer, V.A. Strusevich, Scheduling problems for parallel dedicated machines under multiple resource constraints, Disc. Appl. Math. 133 (2004) 45–68.
[12] S. Avci, M. Akturk, Tool magazine arrangement and operations sequencing on CNC machines, Comput. Oper. Res. 23 (1996) 1069–1081.
[13] N.S. Kumar, R. Sridharan, Simulation modeling and analysis of tool sharing and part scheduling decisions in single-stage multimachine flexible manufacturing systems, Robot. Comput. Integr. Manuf. 23 (2007) 361–370.
[14] M.A. Gamila, S. Motavalli, A modeling technique for loading and scheduling problems in FMS, Robotic. Comput. Integr. Manuf. 19 (2003) 45–54.
[15] H.-K. Roh, Y.-D. Kim, Due-date based loading and scheduling methods for a flexible manufacturing system with an automatic tool transporter, Int. J. Prod. Res. 35 (1997) 2989–3003.
[16] J.A. Ventura, D. Kim, Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints, Comput. Oper. Res. 30 (2003) 1945–1958.
[17] A. Turkcan, S. Akturk, R.H. Storer, Due date and cost-based FMS loading, scheduling and tool management, Int. J. Prod. Res. 45 (2007) 1183–1213.
[18] J. Aldrin Raj, D. Ravindran, M. Saravanan, T.H. Prabaharan, Simultaneous scheduling of machines and tools in multimachine flexible manufacturing systems using artificial immune system algorithm, Int. J. Comput. Int. Manuf. 27 (5) (2014) 401–414.
[19] J.M. Novas, G.P. Henning, Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming, Expert Syst. Appl. 41 (2014) 2286–2299.
[20] L.J. Zeballos, A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems, Robotic. Comput. Integr. Manuf. 26 (2010) 725–743.
[21] L.J. Zeballos, O.D. Quiroga, G.P. Henning, A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations, Eng. Appl. Artif. Intell. 23 (2010) 229–248.
[22] M. Garey, D. Johnson, Computer and Intractability. A Guide to the Theory of NP-Completeness, Bell Laboratories, 1979.
[23] R. Jans, Solving lot sizing problems on parallel identical machines using symmetry-breaking constraints, INFORMS J. Comput. 21 (2009) 123–136.
[24] Z. Degraeve, W. Gochet, R. Jans, Alternative formulations for a layout problem in the fashion industry, Eur. J. Oper. Res. 143 (2002) 80–93.
[25] H. Sherali, J. Smith, Improving discrete model representations via symmetry considerations, Manag. Sci. 47 (2001) 1396–1407.
[26] H. Sherali, B.M.P. Fraticelli, R. Meller, Enhanced model formulation for optimal facility layout, Oper. Res. 51 (2003) 629–644.
[27] F. Margot, Exploiting orbits in symmetric ILP, Math. Program. Ser. B 98 (2002) 3–21.
[28] F. Margot, Pruning by isomorphism in branch-and-cut, Math. Program. Ser. B 94 (2002) 71–90.
[29] F. Glover, Future paths for integer programming and linkage to artificial intelligence, Comput. Oper. Res. 13 (1986) 533–549.
[30] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.