Mathematical Models of Flow Shop and Job Shop Scheduling Problems

Miloš Šeda

***Abstract*—**In this paper, mathematical models for permutation flow shop scheduling and job shop scheduling problems are proposed. The first problem is based on a mixed integer programming model. As the problem is NP-complete, this model can only be used for smaller instances where an optimal solution can be computed. For large instances, another model is proposed which is suitable for solving the problem by stochastic heuristic methods. For the job shop scheduling problem, a mathematical model and its main representation schemes are presented.

***Keywords*—**Flow shop, job shop, mixed integer model, representation scheme.

I. Introduction

PRACTICAL machine scheduling problems are numerous and varied. They arise in diverse areas such as flexible manufacturing systems, production planning, computer design, logistics, communication, etc. A scheduling problem is to find sequences of jobs on given machines with the objective of minimising some function of the job completion times. In a simpler version of this problem, *flow shop scheduling* [12], all jobs pass through all machines in the same order. A more complex case is represented by a *job shop scheduling problem* where machine orderings can be different for each job. Job shop scheduling problem (abbreviated to JSSP or JSS) is one of the hardest combinatorial optimization problems. It belongs to the class of NP-hard problems, consequently there are no known algorithms guaranteed to give an optimal solution and run in polynomial time. That means, classical optimization methods (branch and bound method, dynamic programming) can be used only for small-scale tasks. Therefore, more complex tasks must be solved by heuristic methods [15], [21]. Successful heuristic methods include approaches based on *simulated annealing* [7], *tabu search* [19], [25], and *genetic algorithms* [17], [27]. A very efficient method combines a variable depth search procedure with a *shifting bottleneck* framework [1], [26]. The papers [13], [24] provide a survey and a comparison of various job shop scheduling methods. Deterministic algorithms as well as approximation and heuristic approaches (including tabu search, simulated annealing, genetic algorithms, and ejection chains) to scheduling manufacturing processes are presented and discussed in [3].

In [11], a case with uncertain processing times is studied and an approach based on fuzzy set theory is proposed. The problem has some other modifications, e.g. a "no-wait" version in which the start of job processing is delayed on the first machine so that the job need not wait for processing on subsequent machines. In this paper, we deal with another special version of the problem called a permutation flow shop scheduling problem (PFSSP) where each machine processes the jobs in the same order.

Manuscript received August 31, 2007. This work was supported in part by the Ministry of Education, Youth and Sports of the Czech Republic under research plan MSM 0021630518 "Simulation Modelling of Mechatronic Systems".

Miloš Šeda works in the Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, Technická 2896/2, CZ 616 69 Brno, Czech Republic (phone: +420-54114 3332; fax: +420-54114 2330; e-mail: seda@fme.vutbr.cz).

II. Flow Shop Scheduling

Flow shop scheduling is one of the most important problems in the area of production management [3]. It can be briefly described as follows: There are a set of m machines (processors) and a set of n jobs. Each job comprises a set of m operations which must be done on different machines. All jobs have the same processing operation order when passing through the machines. There are no precedence constraints among operations of different jobs. Operations cannot be interrupted and each machine can process only one operation at a time. The problem is to find the job sequences on the machines which minimise the makespan, i.e. the maximum of the completion times of all operations. As the objective function, mean flowtime, completion time variance [9] and total tardiness [20] can also be used. The flow shop scheduling problem is NP-complete and thus it is usually solved by approximation or heuristic methods. The use of simulated annealing is presented, e.g., in [12], [28], tabu search in [23] and genetic algorithms in [17], [27]. In [14] a deterministic heuristic is proposed that determines the order of any two jobs in the final schedule based on their order in all two-machine problems embedded in the problem.

Consider three finite sets J, M, O where
J is a set of jobs 1, … , n,
M is a set of machines 1, … , m, and
O is a set of operations 1, … , m.

Denote
J_i … the i-th job in the permutation of jobs
p_{ik} … processing time of the job $J_i \in J$ on machine k.

$(\forall i \in J)\ (\forall k \in M)$: v_{ik} = waiting time (idle time) on machine k before the start of the job J_i

$(\forall i \in J)\ (\forall k \in M)$: w_{ik} = waiting time (idle time) of the job J_i after finishing processing on machine k, while waiting for machine k+1 to become free

Define the following decision variables

$$\forall i, j \in J : x_{i\ j} = \begin{cases} 1, & \text{if job } j \text{ is assigned to the } i\text{th position in the permutation, i.e. } J_i = j \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The following mathematical formulation of the permutation flow shop scheduling can be derived.

$$\forall i \in J : \sum_{j=1}^{n} x_{ij} = 1 \quad (2)$$

$$\forall j \in J : \sum_{i=1}^{n} x_{ij} = 1 \quad (3)$$

$$\forall k \in M - \{m\} : w_{1k} = 0 \quad (4)$$

$$\forall k \in M - \{1\} : v_{1k} = \sum_{r=1}^{k-1} \sum_{i=1}^{n} p_{i\,r} x_{1\,i} \quad (5)$$

$$(\forall i \in J - \{n\})\ \ (\forall k \in M - \{m\}) :$$

$$v_{i+1,\,k} + \sum_{j=1}^{n} p_{jk} x_{i+1,\,j} + w_{i+1,\,k} = w_{ik} + \sum_{j=1}^{n} p_{j,\,k+1} x_{ij} + v_{i+1,\,k+1} \quad (6)$$

$$C_{\max} = \sum_{i=1}^{n} (v_{i\,m} + \sum_{j=1}^{n} p_{jm} x_{ij}) \quad (7)$$

This mixed integer programming model can be easily transformed into the form necessary for the optimisation package GAMS (General Algebraic Modelling System) [4].

The ability of computing optimal solutions was checked using standard benchmarks from OR-Library (OR = Operations Research) accessible from London Imperial College Management School [2].

All computations leading to optimum were performed in a few seconds, but for larger instances 20×10 (20×10 corresponds to 20 jobs and 10 machines), 20×15, 30×10, etc., they ended with a run time error with GAMS indicating "`insufficient space to update U-factor …`".

Therefore, for these cases, we must choose another approach. One way seems to be straightforward – to search the optimum in the space of permutations of jobs. Unfortunately, this approach is useable again only for not very high number of jobs, because its time complexity for n jobs is equal to $O(n!)$.

The mathematical formulation given by equations (2)-(7) is not suitable for searching in the space of permutations because it contains no explicitly expressed permutation. Therefore it is necessary to formulate another model. If we have processing times $p(i,j)$ for job i on machine j, and a job permutation $\pi = J_1, J_2, \ldots, J_n$, then we can calculate the completion times $C(J_i, j)$ as follows:

$$C(J_1, 1) = p(J_1, 1) \quad (8)$$

$$C(J_i, 1) = C(J_{i-1}, 1) + p(J_i, 1), \quad i = 2, \ldots, n \quad (9)$$

$$C(J_1, j) = C(J_1, j-1) + p(J_1, j), \quad j = 2, \ldots, m \quad (10)$$

$$C(J_i, j) = \max\{C(J_{i-1}, j), C(J_i, j-1)\} + p(J_i, j), \quad i = 2, \ldots, n;\ j = 2, \ldots, m \quad (11)$$

$$C_{\max}(\pi) = C(J_n, m) \quad (12)$$

There are many various methods for an approximation of the optimal solution by searching only a part of the space of feasible solutions (represented here by all permutations). For complex combinatorial problems, stochastic heuristic techniques [15], [21] are frequently used. We present an approach based on genetic algorithms. As the genetic algorithms are well known, see e.g. [16], we only concentrate on problem specific details.

As to the crossover operation, we cannot use the traditional two-point crossover, because it would lead to infeasible solutions. If we change the middle parts of the parent chromosomes P_1=(1,10,7,2,8,9,4,6,5,3) and P_2=(5,8,2,9,7,4,1,10,3,6) between the 4th and 7th position, then we would obtain offspring (1,10,7,9,7,4,1,6,5,3) and (5,8,2,2,8,9,4,10,3,6) that correspond to no permutations, because some jobs are duplicated or omitted. We used the so called crossover in a partially mapped representation where the genes in the middle part of one chromosome are ordered in its offspring by their occurrence in the second parent chromosome.

For mutation we considered three operators:

- *exchange mutation* (it exchanges two randomly selected positions in a permutation),
- *shift mutation* (it removes a value at one position and puts it at another position), and
- *mutation* inspired by well-known *Lin-2-Opt change operator* usually used for solving the travelling salesman problem [10]. Here first two elements are added to the permutation (into positions 0 and $|n|$+1) and then the same values are assigned to them to simulate a cyclic tour. Two 'edges' (pairs of neighbour elements in permutation) are randomly chosen ((p_1, p_2) and (q_1, q_2) say), the inner elements p_2, q_1 are swapped and the elements between p_2 and q_1 are reversed.

In the literature, slight modifications of these shift operations can be found, e.g. *1stSwap*, *FullSwap*, *DoubleCut*, *DoublePointShift* and *RightDoublePoint* [8], [15], [16].

Best results were achieved with the shift mutation. In Fig. 1, a skeleton of proposed genetic algorithm is shown. Operators that have not been yet discussed are clear from their denotations in the algorithms. Number of individuals *Npop* in the population was set to 50 and the number of iterations to $10 \times n^2$.

$P(0) := \{P_1, P_2, \ldots, P_{Npop}\}$;
$P_{\min} :=$ Permutation_minimizing_makespan $\in P(0)$;
$t := 0$;
while $t <$ number_of_iterations **do**
begin repeat BinaryTournamentSelection($P(t)$, *parent*1, *parent*2);

```
    offspring : = ModifiedTwoPointCrossover(
                                parent1, parent2);
    offspring : = ShiftMutation(offspring)
      until not (offspring in population P(t));
      P_max : = Permutation_maximizing_makespan ;
                                { the worst permutation }
      P(t +1) : = P(t) − {P_max} ∪ {offspring};
          { SteadyStateReplacement (P_max , offspring) }
      if  Makespan(offspring) < Makespan(P_min)
          then P_min : = offspring;
      { update the best permutation }
      t : = t +1
end;
```

Fig. 1 Genetic algorithm skeleton for permutation flow shop scheduling problem

III. Flow Shop Scheduling with Fuzzy Processing Times

Let us suppose now that processing times of the jobs on machines are not deterministic, but they are given by fuzzy numbers [22].

A *fuzzy number*} A is a fuzzy set represented by 4-tuple (a_1,a_2,a_3,a_4) and a piecewise continuous membership function with the following properties [18]:

- $a_1 \le a_2 \le a_3 \le a_4$
- $\mu_{A(x)} = 0$ for $x \le a_1$, $x \ge a_4$
- $\mu_{A(x)} = 1$ for $a_2 \le x \le a_3$
- μ_A is increasing on $[a_1, a_2]$ and decreasing on $[a_3, a_4]$.

The fuzzy set defined by the membership function is an example of fuzzy number. In this paragraph we consider trapezoidal fuzzy numbers.

In the next text we will denote fuzzy numbers by the type F in the upper index. The completion times of the jobs are calculated as follows:

$$C^F(J_1, 1) = p^F(J_1,1) \tag{13}$$

$$(J_i, 1) = C^F(J_{i-1}, 1) \oplus p^F(J_i, 1), \quad i = 2, \dots, n \tag{14}$$

$$C^F(J_1, j) = C^F(J_1, j-1) \oplus p^F(J_1, j), \quad j = 2, \dots, m \tag{15}$$

$$C^F(J_i, j) = \max\{C^F(J_{i-1}, j), C^F(J_i, j-1)\} \oplus p^F(J_i, j), \quad i = 2, \dots, n;\ j = 2, \dots, m \tag{16}$$

$$C^F_{\max}(\pi) = C^F(J_n, m) \tag{17}$$

where $\oplus$ and max are operations over fuzzy numbers.

The addition of fuzzy numbers can be derived using the extension principle and it is determined as follows [6]:

$$X^F \oplus Y^F = (x_1, x_2, x_3, x_4) \oplus (y_1, y_2, y_3, y_4) = (x_1+y_1, x_2+y_2, x_3+y_3, x_4+y_4) \tag{18}$$

When the maximum operation would be derived in the same way, then its results may not be trapezoidal fuzzy numbers. Therefore we approximate this operation as follows [6].

$$\max(X^F,Y^F)=(\max(x_1, y_1), \max(x_2, y_2), \max(x_3, y_3), \max(x_4, y_4)) \tag{19}$$

To find a job permutation π which minimises the fuzzy makespan, we must compare fuzzy numbers in some way, which is a difficult problem. An *ordering relation* $\le$ can be defined e.g. as follows:

$$X^F \le Y^F \Leftrightarrow (x_1 \le y_1) \wedge (x_2 \le y_2) \wedge (x_3 \le y_3) \wedge (x_4 \le y_4) \tag{20}$$

However, this relation is not a complete ordering relation, as fuzzy numbers X^F, Y^F satisfying

$$(\exists\, i,j \in \{1,2,3,4\}): (x_i < y_i) \wedge (x_j > y_j) \tag{21}$$

are not comparable by $\le$.

It is evident that, for noncomparable fuzzy numbers X^F, Y^F, this fuzzy max operation results in a fuzzy number different from both of them. For example, for X^F=(4,9,12,16) and Y^F=(6,8,13,15), we get from equation (19) a fuzzy max (6,9,13,16) which differs from X^F and Y^F.

If $C_{\max}(\pi_1)$ and $C_{\max}(\pi_2)$ are not in the $\le$ relation then we say that the solutions π_1, π_2 of our scheduling problem are *non-dominated*.

It is straightforward that studied problem of the minimisation of $C^F_{\max}(\pi)$ can be replaced by the four-criterial problem as follows:

$$\text{Minimise } C_{\max,1}(\pi) \text{ and } C_{\max,2}(\pi) \text{ and } C_{\max,3}(\pi) \text{ and } C_{\max,4}(\pi) \tag{22}$$

where objective functions $C_{\max,i}(\pi)$ are deterministic.

There are various techniques of the multi-criterial optimization. It is possible to get a compromise solution of this problem on the basis of its transformation into a single-criterial problem where an objective function can be designed as a weighted sum of criteria:

$$C_{\max}(\pi) = \sum_{i=1}^{4} w_i\, C_{\max,i}(\pi) \tag{23}$$

IV. Job Shop Scheduling

The classical JSS problem can be described as follows [24]: There are a set of m machines and a set of n jobs. Each job consists of a sequence of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can process at most one operation at a time. We assume that any successive operations of the same job are processed on different machines. A schedule is an assignment of the operations to time intervals on the machines.

The problem is to find a schedule which optimises a given objective. Assume that three finite sets J, M, O are given where J is a set of jobs 1, … , n, M is a set of machines 1, … , m, and O is a set of operations 1, … , N.

Consider the following denotations: J_i = the job to which operation i belongs, M_i = the machine on which operation i is to be processed, t_i = the start time for operation i, p_i = the processing time for operation i, $C_{\max}$ = the makespan.

On O, a binary relation $\rightarrow$ is defined that represents *precedence constraints* between operations of the same job.

If $i \rightarrow j$, then $J_i = J_j$ and there is no $k \in \{i,j\}$ satisfying $i \rightarrow k$ or $k \rightarrow j$. (Operation i is the predecessor of operation j). Thus, if $i \rightarrow j$, then $M_i \ne M_j$ by the JSSP specifications.

The problem of optimal job shop scheduling is to find a starting t_i time for each operation $i \in O$ such that

$$\max_{i \in O} \left(t_i + p_i \right) \tag{24}$$

is minimised subject to:

$$\forall i \in O : \; t_i \geq 0 \tag{25}$$

$$\forall i, j \in O, \; i \rightarrow j : \; t_j \geq t_i + p_i \tag{26}$$

$$\forall i, j \in O, \; i \neq j, \; M_i = M_j : \; (t_j \geq t_i + p_i) \vee (t_j \geq t_i + p_j) \tag{27}$$

The conditions (3) express *precedence constraints* which represent technological link-up of operations within the same task. The conditions (4) express *machine capacity constraints*, i.e. each machine can process at most one operation at a time.

The described equations cannot be directly used for determining a schedule. We need to eliminate symbols of binary relation $\rightarrow$ and disjunction $\vee$ and try to get a formulation of *integer programming*.

The binary relation can be eliminated easily so that O will be decomposed into subsets of operations that correspond to tasks. Then we will assign to operations in each task numbers creating a sequence of consecutive integers by the operation order.

Denote n_j = the number of operations in job j, and N_j = the total number of operations of the first j jobs.

Evidently:

$$N_0 = 0, \; N_j = \sum_{k=1}^{j} n_k, \; N = \sum_{k=1}^{n} n_k . \tag{28}$$

Using the denotation for total number of operations of the first j–1 jobs, we assign to n_j operations of task j numbers $N_{j-1} + 1, \ldots, N_{j-1} + n_j$ where $N_{j-1} + n_j = N_j$.

Now we can express equation (26) as follows:

$$\left(\forall j \in J \right) \left(N_{j-1} + 1 \leq i \leq N_j - 1 \right) : \; t_{i+1} \geq t_i + p_i \tag{29}$$

The makespan is then determined as the maximum of the completion times of the last operations in jobs. Hence, we get:

$$\forall j \in J : C_{\max} \geq t_{N_j} + p_{N_j} \tag{30}$$

Let us define capacity constraints using binary variables $x_{ij} \in \{0,1\}$ as follows:

$$\forall i, j \in O, \; i \neq j, \; M_i = M_j :$$

$$x_{ij} = \begin{cases} 1, \; t_j \geq t_i + p_i, & \text{operation } i \text{ precedes operation } j \\ 0, \; t_i \geq t_j + p_j, & \text{operation } j \text{ precedes operation } i \end{cases} \tag{31}$$

If T is an upper bound of the makespan, then, using x_{ij}, we can replace equation (27) by pairs of inequalities (32) as follows:

$$\forall i, j \in O, \; i \neq j, \; M_i = M_j : \begin{cases} t_j \geq t_i + p_i x_{ij} - T\left(1 - x_{ij}\right) \\ t_i \geq t_j + p_j \left(1 - x_{ij}\right) - T x_{ij} \end{cases} \tag{32}$$

Hence, the job shop scheduling problem with makespan objective can be formulated as follows:

Minimise

$$C_{\max} \tag{33}$$

subject to

$$\forall i \in O : \; t_i \geq 0 \tag{34}$$

$$\left(\forall j \in J \right) \left(N_{j-1} + 1 \leq i \leq N_j - 1 \right) : \; t_{i+1} \geq t_i + p_i \tag{35}$$

$$\forall j \in J : C_{\max} \geq t_{N_j} + p_{N_j} \tag{36}$$

$$\forall i, j \in O, \; i \neq j, \; M_i = M_j : \begin{cases} x_{ij} \in \{0,1\} \\ t_j \geq t_i + p_i x_{ij} - T\left(1 - x_{ij}\right) \\ t_i \geq t_j + p_j \left(1 - x_{ij}\right) - T x_{ij} \end{cases} \tag{37}$$

An important feature of heuristic methods is problem representation. In [5], a review of frequently used representations is presented. Here, we briefly describe three representations (one based on jobs, one on operations, and one on disjunctive graphs).

The *job-based representation* consists of a list of n jobs and a schedule is constructed according to the sequence of jobs. For a given sequence of jobs, all operations of the first job in the list are scheduled first, and then all operations of the second job in the list are considered. The first operation of the job under treatment is allocated in the best available processing time to the machine the operation requires, and then the second operation, and so on until all operations of the job are scheduled. The process is repeated with each of the jobs in the list considered in the appropriate sequence. Any permutation of jobs corresponds to a feasible schedule.

The *operation-based representation* encodes a schedule as a sequence of operations and each element of this sequence stands for one operation. All operations for a job are named by the same symbol in the sequence and they are interpreted according to the order of occurrence in the given sequence. For an n-job and m-machine problem, a sequence contains $n \times m$ elements.

Each job appears in the chromosome exactly m times and each repeating refers to a unique operation which is context-dependent rather than indicating a concrete operation of a job. It is straightforward that any permutation of elements in a sequence always yields a feasible schedule.

The *disjunctive graph-based representation*: A disjunctive graph is defined as follows:

$$G = \left(V, C \cup D \right) \tag{38}$$

where

V is a set of vertices representing operations. This set contains also two special vertices numbered 0 and N+1 representing the fictitious start and end operations, respectively. The processing time of each operation is denoted as the weight of the vertex. The two fictitious operations 0 and N+1 have operation times of zero.

C is a set of directed *conjunctive* edges. These edges represent pairs of consecutive operations of the same job, as well as edges from the start vertex 0 to the first operation of each job and edges from the last operations of each job to the end vertex N+1.

D is a set of undirected *disjunctive* edges representing pairs of operations to be processed by the same machine.

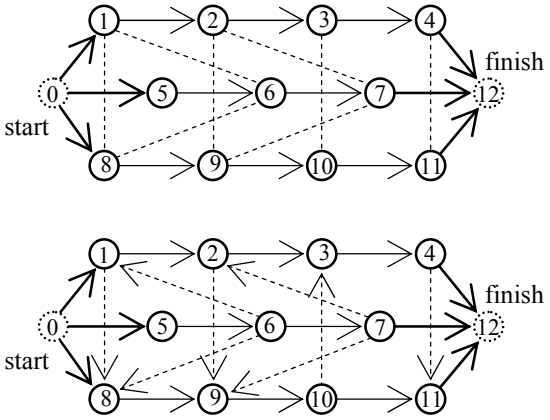

Fig. 2 Disjunctive graph for the JSSP instance and a feasible schedule represented as an acyclic directed graph

To determine a schedule we must define an ordering of all operations processed on the same machine. It can be done by turning all undirected (disjunctive) edges into directed ones. A set S of all directed edges selected from disjunctive edges is called a *complete selection*. A complete selection S defines a feasible schedule if and only if the resulting directed graph is acyclic which guarantees there are no precedence conflicts between operations, see Fig. 2. Obviously, the time required to complete all jobs (makespan) is given by the length of the longest weighted path from the start vertex to the end vertex in a directed graph $G(S)=(V,C\cup S)$, where S is an acyclic complete selection. This path is called the *critical path* and is composed of a sequence of *critical operations*.

Using Critical Path Method (CPM), we easily get the earliest possible start times of operations and the corresponding schedule by the Gantt chart.

The key operator of the tabu search and simulating annealing methods is one used to construct a neighbourhood of the current solution in which these algorithms search foe a solution to be used in the next iteration. In the literature, many sophisticated strategies can be found. For lack of space, we only mention the neighbourhood search strategy of Nowicki and Smutnicki [19], [25]. It is based on modifications of critical blocks that create a critical path evaluated by the CPM. These blocks are given by maximal sequences of consecutive critical operations on the same machine.

For a single (arbitrarily selected) critical path u and critical blocks $B_1, \dots, B_r$ defined for u, it swaps the first (and the last) two operations in blocks $B_2, \dots, B_{r-1}$. In the first block B_1 it swaps only the last two operations, and, via symmetry in the last block B_r, it swaps only the first two operations. These swaps define the set of moves from the processing order p_i. This set of moves is not empty only if the number of blocks is greater than one ($r>1$) and if there exists at least one block with the number of elements greater than one. The neighbourhood of p_i is then defined as all the processing orders obtained by applying moves from p_i. This strategy implemented within the framework of a tabu search led to the best known results for benchmarks from the OR-Library.

V. Conclusion

In this paper, we presented mathematical models of manufacturing processes and their representation schemes. Based on mixed integer programming formulations, they could be used for computation in such optimization tools as GAMS and LINDO. Unfortunately, because of NP-completeness of the models, they can only get an optimal solution for small FSSP/JSSP instances. Therefore, other representation schemes, more suitable for computations by approximation or heuristic methods, must be searched. As frameworks of these methods are known enough, we focused on the key operators such as proposed by Nowicki and Smutnicki to disjunctive graph-based representation of JSSP.

Further investigation will include fuzzy versions of these problems where two cases of uncertainties can be obtained - uncertain due dates and uncertain processing times.

VI. References

[1] E. Balas and A. Vazacopoulos, "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling," *Management Science*, vol. 44, pp. 262-275, 1998

[2] J.E. Beasley, "OR-Library," Report, The Management School Imperial College, London, http://mscmga.ms.ic.ac.uk/pub/flowshop1.txt.

[3] J. Blazewicz, K.H. Ecker, G. Schmidt and J. Weglarz, *Scheduling Computer and Manufacturing Processes*. Berlin: Springer-Verlag, 1996.

[4] A. Brooke, D. Kendrick and A. Meeraus, *GAMS Release 2.25. A User's Guide*. Massachussets: The Scientific Press, Boyd & Fraser Publishing Company, 1992.

[5] R. Cheng, M. Gen and Y. Tsujimura, "A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms - {I}. Representation," *Computers & Industrial Engineering*, vol. 30, pp. 983-997, 1996.

[6] Dubois and H. Prade, *Theorie des possibilites. Applications a la representation des connaissances en informatique*. Paris: MASSON, 1988.

[7] A. El-Bouri, N. Azizi and S. Zolfaghari, "A Comparative Study of a New Heuristic Based on Adaptive Memory Programming and Simulated Annealing: The Case of Job Shop Scheduling," *European Journal of Operational Research*, 2007, 17 pp., in press.

[8] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Addison-Wesley, 1989.

[9] K. Gowrishankar, C. Rajendran and G. Srinivasan, "Flow Shop Scheduling Algorithms for Minimizing the Completion Time Variance and the Sum of Squares of Completion Time Deviations from a Common Due Date," *European Journal of Operational Research*, vol. 132, pp. 643-665, 2001.

[10] G. Gutin and A.P. Punnen (eds.), *The Traveling Salesman Problem and Its Variations*. Dordrecht: Kluwer Academic Publishers, 2002.

[11] H. Ishibuchi, N. Yamamoto, T. Murata and H. Tanaka, "Genetic Algorithms and Neighborhood Search Algorithms for Fuzzy Flowshop Scheduling Problems," *Fuzzy Sets and Systems*, vol. 67, pp. 81-100, 1994.

[12] H. Ishibuchi, S. Misaki and H. Tanaka, "Modified Simulated Annealing Algorithms for Flow Shop Sequencing Problem," *European Journal of Operational Research*, vol. 81, pp. 388-398, 1995.

[13] A. Jain and S. Meeran, "Deterministic Job-Shop Scheduling: Past, Present and Future," *European Journal of Operational Research*, vol. 113, pp. 390-434, 1999.

[14] C. Koulamas, "A New Constructive Heuristic for the Flowshop Scheduling Problem" *European Journal of Operational Research*, vol. 105, pp. 66-71, 1998.

[15] Z. Michalewicz and D.B. Fogel, *How to Solve It: Modern Heuristics*. Berlin: Springer-Verlag, 2000.

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag, 1996.

[17] T. Murata, H. Ishibuchi and H. Tanaka, "Genetic Algorithms for Flowshop Scheduling Problems," *Computers & Industrial Engineering*, vol. 30, No. 4, pp. 1061-1071, 1996.
[18] V. Novák, *Fuzzy Sets and their Applications*, Bristol: Adam Hilger, 1989.
[19] E. Nowicki and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Problem," *Management Science*, vol. 42, pp. 797-813, 1996.
[20] J.C.-H. Pan, J.-S. Chen and C.-M. Chao, "Minimizing Tardiness in a Two-Machine Flow-Shop," *Computers & Operations Research*, vol. 29, pp. 869-885, 2002.
[21] C.R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. Oxford: Blackwell Scientific Publications, 1993.
[22] M. Šeda, J. Dvořák and P. Majer, "Scheduling with Fuzzy Processing Times in a Flow Shop," in *Proc. of the 7th European Congress on Fuzzy and Intelligent Techniques & Soft Computing EUFIT '99*, Aachen, 1999, 6 pp.
[23] U.A. Turki, C. Fedjki and A. Andijani, "Tabu Search for a Class of Single-Machine Scheduling Problems," *Computers & Operations Research*, vol. 28, pp. 1223-1230, 2001.
[24] R. Vaessens, E. Aarts and J. Lenstra, "Job Shop Scheduling by Local Search," *INFORMS Journal on Computing*, vol. 8, pp. 302-317, 1996.
[25] J.P. Watson, A. Howe and L. Whitley, "Deconstructing Nowicki and Smutnicki's i-TSAB Tabu Search Algorithm for the Job-Shop Scheduling Problem," *Computers & Operations Research*, vol. 33, pp. 2623-2644, 2006.
[26] H. Wenqi and Y. Aihua, "An Improved Shifting Bottleneck Procedure for the Job Shop Scheduling Problem," *Computers & Operations Research*, vol. 31, pp. 2093-2110, 2004.
[27] T. Yamada and C.R. Reeves, "Permutation Flowshop Scheduling by Genetic Local Search," in *Proc. of the International Conference Genetic Algorithms in Engineering Systems: Innovations and Applications GALESIA 97*, Glasgow, 1997, pp. 232-238.
[28] S.H. Zegordi, K. Itoh and T. Enkawa, "Minimizing Makespan for Flow Shop Sequencing by Combining Simulated Annealing with Sequencing Knowledge," *European Journal of Operational Research*, vol. 85, pp. 515-531, 1995.