MODELING AND META-HEURISTIC SOLUTION FOR FLEXIBLE SHOP SCHEDULING

Hisashi Tamaki [*] **Kazutoshi Sakakibara** [*] **Hajime Murao** [*]
Shinzo Kitamura [*]

[*] *Dept. of Computer and Systems Engr., Kobe University*
Rokkodai, Nada-ku, Kobe 657-8501, Japan

Abstract: In this paper, an extended class of flexible shop scheduling problems is considered. First, the problem is translated into a mathematical programming formula, i.e., a mixed-integer programming problem. This makes it possible to apply standard packages of mixed integer programming solvers and, while lots of computational time is required in general, to obtain the optimal schedule. Then, in order to seek the schedules close to the optimal for larger-scale problems, a solution method by adopting genetic algorithms based on the formula is newly designed. Through some computational experiments, the effectiveness and the potential of the proposed approach are investigated. *Copyright © 2002 IFAC*

Keywords: Scheduling, Optimization, Mathematical Programming, Integer Programming, Search Methods, Genetic Algorithms

1. INTRODUCTION

Recently, extensive researches on scheduling have been reported from the theoretical as well as the practical view points (Morton, *et al.*, 1993; Tanaev, *et al.*, 1994; Pinedo, 1995; Blazewicz, *et al.*, 2001; Brucker, 2001). The scheduling problems are categorized into some groups with respect to machine environment (e.g., single machine problems, parallel machine problems, job shop problems, etc.), job characteristics (e.g., the type of precedence relations, with/without preemption, etc.) and optimality criteria (e.g., the maximum completion time, the maximum lateness, etc.). Here, as for the optimality criteria (i.e., objective functions) of schedules, regular ones (i.e., objective functions which is monotone with respect to completion time of all jobs) have been considered in almost all researches (Tanaev, *et al.*, 1994; Brucker, 2001). While there are many applications in which non-regular objective functions are appropriate, we consider typical criteria of the regular type in the paper.

As for the solution of practical scheduling problems in general, most of methods proposed in the past researches use some sort of dispatching rules, and researchers' efforts have been concentrated upon clarifying "which rules fit which problems." The authors have presented a new scope for the study of the practical scheduling problems by proposing a method to solve them without relying upon dispatching rules (Tamaki, *et al.*, 1995; Tamaki, *et al.*, 1999). Namely, the practical scheduling problems of the parallel machine type are transformed into the mathematical programming problem, and feasible schedules are represented by strings. This formulation makes it easy to design *meta-heuristics* (Reeves, 1993) such as simulated annealing and genetic algorithms.

This paper deals with a class of *flexible shop* scheduling problems. That is, each shop includes more than one machines of the parallel machine type in the framework of *job shop* scheduling problems. So far, several approaches have been reported to the flexible shop problems, almost all approaches adopt some kinds of heurisitic procedures (Chen, *et al.*, 1999; Yang, *et al.*, 2000), and there are few approaches based on the formal framework (Cheng, *et al.*, 2001).

In this paper, to this class of scheduling problems where several auxiliary restrictions originated from the necessity of set-up processes are additionally considered, the authors newly propose a method of modeling the problem based on a mathematical programming approach (i.e., by using mixed-integer programming formula (Namhauser, *et al.*, 1989)), and a way of designing a solution by adopting genetic algorithms (Holland, 1975; Goldberg, 1989).

In the following, the scheduling problem is revisited and is translated into a mathematical programming problem (in Section 2). A way of applying genetic algorithms based on the formula is presented (in Section 3), where a way of genetic representation of a schedule as well as a procedure of translating a genotype to a phenotype (i.e., a feasible schedule) are also introduced. Some computational experiments using several examples have been actually carried out. Results (in Section 4) indicate that the proposed approach is effective, and that our method can give satisfactory solutions to the scheduling problems within reasonable time and could possibly support the scheduling engineers.

2. SCHEDULING PROBLEM

2.1 Description of the Problem

There are m *machines* M_i $(i = 1, \ldots, m)$ and n *jobs* J_j $(j = 1, \ldots, n)$. Each job J_j includes a series of n_j *operations* O_k $(k = n_{j-1} + 1, \ldots, n_j\,;\ n_j = \sum_{\ell=1}^{j} n_\ell\,;\ n_0 = 0)$, and these n_j operations are to be processes in this order. To each operation O_k $(k = 1, \ldots, N\,;\ N = n_n = \sum_{j=1}^{n} n_j)$, the set of available machines $\mathcal{A}_k$ and the type δ_k are associated. The type of an operation represents the kind of production, and the combination of the machine and the type determines production speed.

In making a schedule, following restrictions should be taken into account.

(a) *Restriction of machines*: A kind of operations can be processed on one of the fitted machines. (This is due to the mechanical structure, size, weight, etc. of the machines and to the fitness of operations to machines.)

(b) *Set-up time*: If the types of two operations which are processed successively on any machine differs, a set-up time is needed between their processing. The set-up time consists of two parts: the time required before processing, and the time required after processing.

(c) *Lead time*: To each operation, the minimum and maximum durations before starting the processing of the following operation are specified. (This is due to the material conditions, temperature control, etc.)

To describe the scheduling problem, we need the following notations and parameters. With each machine M_i $(i = 1, \ldots, m)$,

$\mu_{i\delta}$: time per unit product on M_i when an operation of type T_δ is processed

is associated. With each job J_j $(j = 1, \ldots, n)$, the following parameters are associated:

d_j : due date,
r^{J}_j : required quantity, and
n_j : number of operations.

Finally, with each operation O_k $(k = 1, \ldots, N)$, the following parameters are associated:

δ_k : type of O_k,
r^{O}_k : required quantity (the value of r^{O}_k is determined according to that of r^{J}_j),
s^1_k : set-up time required before processing,
s^2_k : set-up time required after processing,
s^3_k : minimum lead time,
s^4_k : maximum lead time, and
$\mathcal{A}_k$: set of machines which can process O_k.

As for evaluation of schedules, various kinds of criteria may be considered. It is impossible, however, to take all of them into consideration. So, we consider here the maximum completion time $C_{\max}$ and the total tardiness T_{sum}:

$$z_1 = C_{\max} = \max_j t^{\mathrm{F}}_{n_j} = \max_k t^{\mathrm{F}}_k\,, \tag{1}$$

$$z_2 = T_{\mathrm{sum}} = \sum_j \left\{\max\left(0,\, t^{\mathrm{F}}_{n_j} - d_j\right)\right\}\,, \tag{2}$$

where

t^{S}_k : the start time of the operation O_k, and
t^{F}_k : the completion time of the operation O_k.

Then, as a scalar objective function, we use the weighted sum

$$z = w_1\, z_1 + w_2\, z_2 \;=\; w_1\, C_{\max} + w_2\, T_{\mathrm{sum}}\,, \tag{3}$$

where weights w_1 and w_2 are nonnegative.

2.2 Mathematical Programming Model

The scheduling problem defined above is categorized as flexible shop problems (or flexible shop problems, generalized shop problems (Krüger, *et al.*, 1998)), about which little have been studied (Blazewicz, *et al.*, 2001; Brucker, 2001). In this paper, by introducing the following variables x, y and u, the scheduling problem is newly transformed to a mathematical programming problem.

x_{ik} $(i = 1, \dots, m\,;\; k = 1, \dots, N)$:
: $x_{ik} = 1$ if O_k is assigned to M_i, and $x_{ik} = 0$ otherwise,

$y_{kk'}$ $(k,\, k' = 1, \dots, N)$:
: $y_{kk'} = 1$ if $\mathrm{O}_{k'}$ succeeds O_k when O_k and $\mathrm{O}_{k'}$ are assigned to the same machine, and $y_{kk'} = 0$ otherwise,

$u_{kk'}$ $(k,\, k' = 1, \dots, N)$:
: $u_{kk'} = 1$ if $x_{ik} = x_{ik'}$ (for all i), and $u_{kk'} = 0$ otherwise,

Here, the values of u are dependent on those of x and y, and are uniquely determined according to them. Moreover, the completion time t^{F} is also determined, once the assignment x and the start time t^{S} are fixed. Hence, the independent variables (i.e., the decision variables) are x, y and t^{S}.

The optimal schedule with respect to the objective function z of (3) is naturally a semi-active schedule. (A schedule is called "semi-active" if there does not exist any operation which could be started earlier without altering the processing order or violating the restrictions.) If one restricts the class of schedules to semi-active schedules, the duplet $(x,\, y)$ uniquely determines a schedule. On the other hand, a schedule evidently determines the duplet $(x,\, y)$ uniquely by definition. Thus, the duplet $(x,\, y)$ and a semi-active schedule correspond one-to-one. So, z of (3) becomes a function of x and y, and the following *mathematical programming* problem can be obtained.

Problem MP: Minimize

$$z = w_1 z_1 + w_2 z_2 \tag{4}$$

subject to

$$\sum_{i=1}^{m} x_{ik} = 1 \qquad (k = 1, \dots, N) \tag{5}$$

$$x_{ik} = 0 \qquad (i \notin \mathcal{M}_k\,;\; k = 1, \dots, N) \tag{6}$$

$$x_{ik} \in \{\,0,\, 1\,\} \qquad (i = 1, \dots, m\,;\; k = 1, \dots, N) \tag{7}$$

$$u_{kk'} \ge x_{ik} + x_{ik'} - 1 \quad (i = 1, \dots, m\,;\; k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{8}$$

$$u_{kk'} \le x_{ik} - x_{ik'} + 1 \quad (i = 1, \dots, m\,;\; k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{9}$$

$$u_{kk'} \in \{\,0,\, 1\,\} \qquad (k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{10}$$

$$y_{kk'} \le u_{kk'} \qquad (k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{11}$$

$$y_{kk'} + y_{k'k} = u_{kk'} \qquad (k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{12}$$

$$y_{kk'} \in \{\,0,\, 1\,\} \qquad (k,\, k' = 1, \dots, N\,;\; k \ne k') \tag{13}$$

$$t_k^{\mathrm{S}} \ge 0 \qquad (k = n_{j-1} + 1\,;\; j = 1, \dots, n) \tag{14}$$

$$t_k^{\mathrm{S}} \ge t_{k-1}^{\mathrm{F}} + s_{k-1}^{3} \;(k = n_{j-1} + 2, \dots, n_j\,;\; j = 1, \dots, n) \tag{15}$$

$$t_k^{\mathrm{S}} \le t_{k-1}^{\mathrm{F}} + s_{k-1}^{4} \;(k = n_{j-1} + 2, \dots, n_j\,;\; j = 1, \dots, n) \tag{16}$$

$$t_k^{\mathrm{S}} \ge t_\ell^{\mathrm{F}} + \left(s_\ell^2 + s_k^1\right) \Delta_{\ell k} - M\,(1 - y_{\ell k}) \quad (k,\, \ell = 1, \dots, N\,;\; k \ne \ell) \tag{17}$$

$$t_k^{\mathrm{F}} = t_k^{\mathrm{S}} + \sum_{i=1}^{m} \left(r_k^{\mathrm{O}}\, \mu_{i\delta_k}\, x_{ik}\right) \quad (k = 1, \dots, N) \tag{18}$$

$$z_1 \ge t_{n_j}^{\mathrm{F}} \qquad (j = 1, \dots, n) \tag{19}$$

$$z_2 \ge \sum_{j=1}^{n} \max\left(\,0\,,\; t_{n_j}^{\mathrm{F}} - d_j\,\right) \tag{20}$$

Here, the parameter $\Delta_{\ell k}$ in the equation (17) is a constant defined by

$$\Delta_{\ell k} = \begin{cases} 1\,, & \text{if } \delta_k \ne \delta_\ell\,, \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{21}$$

M is also a constant with sufficiently large positive value.

In the above formulation (Problem MP), the equations (5) through (7) fix the machines to which operations are assigned. The equations (11) through (13) determine the (possible) processing order of the operations, and the equations (8) through (10) represent the "flag" whether each pair of operations should be assigned to the same machine or not. The start time of operations are given by the equations (14) through (17), and the completion time of operations are given by the equation (18). The equations (19), (20) and (4) give the value of the objective function z of (3).

There appears a nonlinear operator "max" in the right-hand side of the equation (20). This inequality can be equivalently rewritten by using the set of linear inequalities (e.g., $a \ge \max\{b,\, c\}$ is equivalent to $a \ge b$ and $a \ge c$). Therefore, the problem MP is a *mixed-integer programming* problem.

3. META-HEURISTIC SOLUTION

Through the MP formula in Section 2.2, the scheduling problem described in Section 2.1 can be solved by applying mathematical programming techniques (e.g., branch-and-bound procedures). This approach, however, need much computational cost in general. And besides, in this paper, the authors consider an approach in which genetic algorithms are used for obtaining better, possibly near-optimal, schedules.

In order to apply meta-heuristics such as genetic algorithms and simulated annealing methods, it

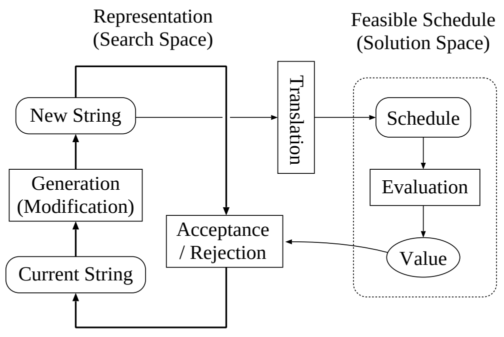

Fig. 1. Outline of the application of meta-heuristics. When a genetic algorithm is adopted as a meta-heuristic procedure, the "generation (modification)" and the "acceptance/rejection" parts are implemented by using "crossover & mutation" and "selection/reproduction" operations, respectively.

is convenient (or required) to represent a schedule symbolically (e.g., by using a *string*). In Fig. 1, the outline of the application of the meta-heuristics is shown.

The simple way to represent a *schedule* S by a string is to use a binary *string* B (i.e., to use the linear array obtained by lining up $\{x_{ik}\}$ and $\{y_{kk'}\}$ in the lexicographical order). This method seems not to include any problem when we only looks at the process of determining B from S. However, if we look at the reverse process, we immediately face with the problem that almost all strings correspond to infeasible schedules. This problem causes serious inefficiencies in the application of meta-heuristics that the search can be proceeded by one step only after an extremely large repetition of producing new strings.

As mentioned in Section 2.2, one can restrict a search within the set of semi-active schedules. Hence, a method to produce a feasible schedule (i.e., a semi-active schedule) from an arbitrarily given string of a sufficient length can be considered, and this procedure is implemented in the "translation" part in Fig. 1.

In this approach, the mapping from strings to schedules evoked by the method is not one-to-one, but many-to-one, as shown in Fig. 2. Therefore, there is a possibility that it may cause another sort of demerit because the same schedule can be scanned repeatedly. Paying attention to this point, this paper introduce a way of representing a schedule and translating *any* string to a feasible schedule as well. In the following, a solution method, in which a genetic algorithm (Goldberg, 1989; Michalewicz, 1996) is adopted as a meta-heuristic method, is designed.

3.1 Representation of an Individual

An *individual* (*genotype*) of genetic algorithms is represented as a combination of two sub-strings

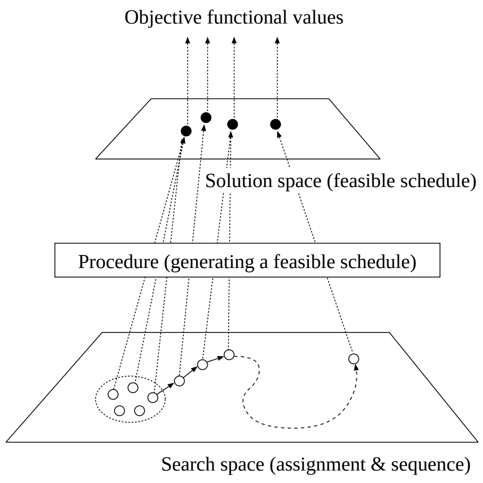
Objective functional values
Solution space (feasible schedule)
Procedure (generating a feasible schedule)
Search space (assignment & sequence)

Fig. 2. Search space and solution space. The mapping from strings in the search spaces to schedules in the solution space is not one-to-one, but one-to-many.

(α and β) with the length of the number of operations, and each sub-string is formed as follows:

α_k : the machine to which O_k is assigned,
β_k : the priority of O_k,

where O_k is assigned to M_{α_k} by referring to α_k (to fix the values of x of the MP formula in Section 2.2, and then, on each machine, the operations are sequenced in the non-decreasing order of β_k (to fix the values of y). The ranges of α and β are

$$\alpha_k \in \{1,\, 2, \cdots,\, m\}\,, \tag{22}$$

$$\beta_k \in \{0,\, 1, \cdots,\, B\}\,, \tag{23}$$

where B is a positive constant.

3.2 Generating a Schedule

According to the property of the problem described in the beginning of this section, a procedure to generate a schedule (*phenotype*) according to the assignment (determined by referring to α) and the sequences (determined by β) is designed, where the start time of each operation is determined in such a way that any operation should be processed as early as possible.

Here, in the procedure, the constraint (16) in the MP formula is discarded, so that the procedure should be kept simple and fast. Then, in order to compensate this issue, in evaluating the schedule, the degree of infeasibleness with respect to this constraint is added to the original objective value as a penalty. That is, the constraint (16) is relaxed, and the following augmented objective function:

$$z' = w_1\, z_1 + w_2\, z_2 + w_3\, z_3 \tag{24}$$

is considered, where

$$z_3 = \sum_{j=1}^{n} \sum_{k=n_{j-1}+2}^{n_j} \max\left\{0,\, t_k^{\mathrm{S}} - \left(t_{k-1}^{\mathrm{F}} + s_{k-1}^{4}\right)\right\}$$

(25)

and the weight w_3 is also non-negative.

3.3 Calculation of a Fitness

As introduced in the following, since the tournament selection is adopted in our genetic algorithm approach, the augmented objective value z' of (24) is directly used as the *fitness* value f.

3.4 Genetic Operators

Three kinds of genetic operators are implemented.

(a) *Crossover* : Two individuals are paired randomly in a population, the crossing sites (the number of crossing sites n_c is prescribed) are selected randomly, and with a prescribed probability p_c some genes are exchanged between the paired individuals.

(b) *Mutation* : One locus is selected randomly, and then the gene is changed to another gene with a prescribed probability p_m.

(c) *Selection* : The *tournament* selection method (Michalewicz, 1996) is adopted. A prefixed number (the tournament size n_s) of individuals are randomly selected, and the best one from this set of individuals survives to the next generation. This process is repeated n_p (population size) number of times.

4. COMPUTATIONAL EXAMPLES

Three kinds of examples :

(a) E_1 : 12 machines, 6 jobs and 18 operations,

(b) E_2 : 6 machines, 6 jobs and 18 operations,

(c) E_3 : 12 machines, 9 jobs and 45 operations,

have been prepared and solved by using the genetic algorithm (GA) introduced in Section 3, where the setting of the parameters are shown in Table 1 and Table 2. To each example and each setting, 100 trials have been executed by changing the initial conditions, and then the best schedule is selected for each trial.

In Table 3, the objective function values of the obtained schedules are summarized, where the minimum, the maximum, the average and the standard deviation values of the best schedules are shown. In the table, the results by using the commercial package of mathematical programming solvers (Nuopt 4.0) are also shown. Furthermore, the average computational time (i.e., the CPU time when using a Pentium III 933MHz computer) required for each method is summarized in Table 4.

From these tables, the followings can be observed :

Table 1 Setting of Parameters

Weights w	$(0, 0.5, 0.5)$, $(0.5, 0, 0.5)$, $(0.25, 0.25, 0.5)$
Constant M	289 (E_1), 649 (E_2), 901 (E_3)

Table 2 Setting of GA Parameters

Population size n_p	200
Number of generations n_g	100
Crossover cites n_c	1
Crossover rate p_c	0.6
Mutation rate p_m	0.01 / individual
Tournament size n_s	2
Maximum priority B	25

(a) To all examples E_1, E_2 and E_3, rather good schedules are obtained sufficiently fast by applying our genetic algorithm.

(b) However, the stableness of finding the best schedule is rather poor. This issue may be overcome by setting n_p (population size) and/or n_g (number of generations) larger.

(c) By changing the weight coefficients w of the (augmented) objective function, the obtained schedules are preferably varied.

As a result of above observations, the proposed approach is effective for finding good (possibly, cloth to the optimal) schedules within preferable computational time, and has a potential of applicability to larger-size practical examples.

5. CONCLUSION

In this paper, a class of flexible shop scheduling problems is studied, and a method of forming a mathematical programming model is proposed. Then, in order to seek the schedules close to the optimal one within preferable computational time, a genetic algorithm approach based on the formula is designed. From computational experiments, we assured the usefulness of our proposal.

Furthermore, the following issues have been left to be investigated :

(a) To examine the effectiveness of our method especially in comparison with heuristic methods using dispatching rules,

(b) To investigate the applicability of our approach to larger-size practical examples,

(c) To explore possibility and also stableness of finding better schedules (e.g., by applying or combining other meta-heuristics such as simulated annealing and tabu search).

REFERENCES

Blazewicz, J., K. Ecker, E. Pesch, G. Schmidt and J. Weglarz (2001). *Scheduling Computer and Manufacturing Processes*, 2nd Ed. Springer-Verlag, Heidelberg.

Table 3 Results of the Computational Experiments

(a) E_1

Method	Weights (w_1, w_2, w_3)	Objective Value			
		Minimum *	Maximum *	Average	Std. Dev.
GA	(0.0, 0.5, 0.5)	4.00 (67, 8, 0)	11.00 (70, 22, 0)	6.91	1.62
	(0.5, 0.0, 0.5)	31.50 (63, 15, 0)	36.00 (72, 21, 0)	33.66	1.03
	(0.25, 0.25, 0.5)	18.75 (67, 8, 0)	25.25 (72, 29, 0)	21.01	.20
Nuopt	(0.0, 0.5, −)	4.00 (63, 8, −)			
	(0.5, 0.0, −)	31.50 (63, 12, −)			
	(0.25, 0.25, −)	17.75 (63, 8, −)			

(b) E_2

Method	Weights (w_1, w_2, w_3)	Objective Value			
		Minimum *	Maximum *	Average	Std. Dev.
GA	(0.0, 0.5, 0.5)	22.00 (78, 44, 0)	32.50 (87, 65, 0)	27.39	2.16
	(0.5, 0.0, 0.5)	37.50 (75, 63, 0)	41.50 (82, 94, 1)	38.37	0.88
	(0.25, 0.25, 0.5)	30.50 (78, 44, 0)	36.75 (84, 63, 0)	34.08	1.45
Nuopt	(0.25, 0.25, −)	29.50 (74, 44, −)			

(c) E_3

Method	Weights (w_1, w_2, w_3)	Objective Value			
		Minimum *	Maximum *	Average	Std. Dev.
GA	(0, 0.5, 0.5)	14.00 (197, 24, 4)	56.00 (192, 108, 4)	32.25	9.42
	(0.5, 0, 0.5)	85.50 (171, 63, 0)	104.50 (209, 231, 0)	93.87	4.21
	(0.25, 0.25, 0.5)	52.50 (174, 36, 0)	82.25 (214, 115, 0)	67.77	6.75

∗ In the parensis, three kinds of sub-objective functional values (z_1, z_2, z_3) are also shown.

Table 4 Computational Time

Example	Method	CPU Time [sec]
E_1	GA (average)	3.99
	Nuopt (w = (0.0, 0.5, −))	69.77
	Nuopt (w = (0.5, 0.0, −))	128.48
	Nuopt (w = (0.25, 0.25, −))	86.46
E_2	GA (average)	9.76
	Nuopt (w = (0.25, 0.25, −))	36277.05
E_3	GA (average)	115.62

∗ A Pentium III (933MHz) computer has been used for all trials.

Brucker, P. (2001). *Scheduling Algorithms*, 3rd Ed. Springer-Verlag, Heidelberg.

Chen, H., J. Ihlow and C. Lehmann (1999). A Genetic Algorithm for Flexible Job-Shop Scheduling. *Proc.of Int. Conf. on Robotics and Automation.* 1120–1125.

Cheng, J., Y. Karuno and H. Kise (2001). A Shifting Bottleneck Approach for a Parallel-Machine Flowshop Scheduling Problem. *J. of the Operations Research Society of Japan.* **44** (2), 140–156.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor.

Krüger, K., Y. Sotskov and F. Werner (1998). Heuristics for Generalized Shop Scheduling Problem Based on Decomposition. *Int. J. on Production Research.* **36** (11), 3013–3033.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Ed. Springer-Verlag, Heidelberg.

Morton, T. and D. Pentico (1993). *Heuristic Scheduling Systems.* John Wiley & Sons, New York.

Nemhauser, G., A. H. G. Rinnooy Kan and M. Todd (1989). *Optimization.* North-Holland, Amsterdam.

Pinedo, M. (1995). *Scheduling : Theory, Algorithms, and Systems.* Prentice-Hall, New Jersey.

Reeves, C. R. (1993). *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Pub., Oxford.

Tamaki, H., K. Taguchi and M. Araki (1995). Application of Meta-Heuristics to Scheduling Problems in Plastics Forming Plant. *Proc. of 7th IFAC/IFORS/IMACS Symp. on Large Scale Systems.* 409–414.

Tamaki, H. and E. Nishino (1999). A Genetic Algorithm Approach to Multi-Objective Scheduling Problems with Regular and Non-Regular Objective Functions. *Proc. of 8th IFAC/IFORS/IMACS/IFIP Symp. on Large Scale Systems.* 289–294.

Tanaev, V., V. Gordon and Y. Shafransky (1994). *Scheduling Theory : Single-Stage Systems* and *Scheduling Theory : Multi-Stage Systems.* Kluwer Academic, Dordrecht.

Yang, Y., S. Kreipl and M. Pinedo (2000). Heuristics for Minimizing Total Weighted Tardiness in Flexible Flow Shops. *J. of Scheduling.* **3** (2), 89-108.