Management and Production Engineering Review
Volume 1 • Number 4 • December 2010 • pp. 19–25

SOME HEURISTICS FOR MAKESPAN MINIMIZATION IN PERMUTATION FLOW-SHOP

Alexandre Dolgui[1], Dmitry Ofitserov[2]

[1] *Ecole des Mines de Saint-Etienne, France*
[2] *World Bank, Information and Technology Services Department, U.S.A.*

Corresponding author:
Alexandre Dolgui
Ecole des Mines de Saint-Etienne
158 cours Fauriel, 42023 Saint-Etienne Cedex, France
phone: (+33) 4 77 42 01 66
e-mail: dolgui@emse.fr

Received: 24 October 2010
Accepted: 18 November 2010

Abstract
This paper deals with the study of makespan optimization for the flow-shop production systems with or without infinite buffers between machines. Two scheduling algorithms are presented. They are based on several ideas from global optimization for continuous functions. In particular, Ψ-transform technique is employed. This is a very interesting research direction to find heuristics with solid theoretical foundations and with a possibility to estimate subsequently their error. Consequently, problems of real size and actual complexity can be dealt with a certain confidence in the quality of the results. The algorithms proposed include also the Petrov matrix and the Gilmore-Gomory methods. The efficiency of the algorithms proposed is illustrated with numerical tests on examples known in literature and on randomly generated tests.

Keywords
permutation flow-shop, makespan optimization, heuristics.

Introduction

Many optimization problems in production management and industrial logistics can be expressed as scheduling problems. Different types of scheduling problems exist, but it is well-known that these are hard to solve due to the combinatorial explosion of possible solutions.

Generally speaking, existing scheduling methods can be classified into three major categories: exact methods, ε-approximate methods and heuristics. Exact methods give optimal solutions, but they are usually developed for theoretical cases, for instance single-stage systems [1]. It has been demonstrated that for cases close to industrial reality, these problems are NP-hard. Therefore, there do not exist exact methods that find optimal solutions in a reasonable time. ε-approximate methods can find a solution which is not necessarily optimal, but they are within a distance smaller than ε from the optimal solution. The maximal error ε is fixed a priori. These methods are promising, but they are often as complex as exact methods [2]. Hence, the methods used in practice are generally heuristics. Quite often, they give good results, but it is difficult to estimate the error of these results. In addition, a lot of them have only intuitive foundations.

One very interesting research direction in this domain is then to try to find heuristics with solid theoretical foundations and with a possibility to estimate subsequently their error [3]. This would let us deal with problems of real size and actual complexity with a certain confidence in the quality of the results. From this prospective, a particular interest is placed on continuous optimization methods which have a strong mathematical basis and which could lead to interesting result for combinatorial optimization by analogy.

In this article, some results of the application of such an approach for the well known problem of the permutation flow-shop are presented. Two continuous optimization methods have been used: gradi-

ent method and Ψ-transform method. These methods and a basic heuristic are presented in Sec. 2. In Secs. 3 and 4 their applications on two different problems are considered: flow-shop systems with and without intermediate unlimited buffers. Section 3 presents the makespan minimization problem for a flow-shop with infinite intermediary buffers. Section 4 shows how the same approach can be used for a flow-shop without buffers.

Basic methods

The methods will be considered in the following order. Firstly a heuristic to optimize the permutation flow-shop will be presented. Then, a gradient method in its discrete modification which is called the sequential search method will be explained. Finally, an estimation of the optimization errors will be proposed. This estimation is based on the results obtained in the Ψ-transform method. Concluding remarks will follow.

A heuristic for makespan optimization in flow-shop scheduling

The permutation flow-shop can be described in the following way. At time t_0 arrives a set $N = \{1, 2, \ldots, n\}$ of parts on the production line which contains m machines. The duration $t_{ij} \geq 0$ of handling each part $j \in N$ by each machine $i, i = 1, \ldots, m$ are assumed to be given. Each machine i handles all parts in the same order given by the permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, $\pi_j \in N$.

It is assumed, that even if $t_{ij} = 0$, part j will be handled by the machine i but for a negligible time. The permutation π^0, which leads to the minimal time of handling all parts, needs to be found.

Minimizing makespan is equal to minimizing completion time $C_{\max}(\pi)$: the overall time which is needed to handle all parts. Most of the heuristics used to optimize this flow-shop scheduling problem are based on the method developed by S.M. Johnson [4] which gives an exact solution for two machines. An analysis of the influence of Johnson's ideas can be found in [5]. Please see also [6–9] and [10]. Many others results have also been stated, but perhaps less known, by the Soviet scientific school [11].

In this paper, our work is based on some results mostly unknown in the wider scientific community.

Petrov V. [12, 13] has proposed to use the Johnson method for $m > 2$ machines by considering two "dummy" machines:

$$P_j^I := \sum_{i=1}^{s} t_{ij} \quad \text{and} \quad P_j^{II} := \sum_{i=s+1}^{m} t_{ij}, \qquad (1)$$

where $s = m/2$, P_j^I is the processing time for the part j on the first machine, P_j^{II} is the processing time of the part j on the second machine. Lvov [14] split the sets of machines into two subsets, not only once but $m - 1$ times. This with its numerous modifications was known in the former USSR as the Petrov matrix method. In the western countries, a similar approach is better known as the CDS method [15].

In this article, the modification of the Petrov matrix method, presented in Algorithm 1, is used. It was shown that this is the best known extension of the Petrov method [16].

Algorithm 1:

i) for each $s = 1, 2, \ldots, m-1$, build "dummy" machines and compute $P_j^I(s)$ et $P_j^{II}(s)$ according to formulae (1);
ii) compute $\lambda_j(s) = P_j^{II}(s) - P_j^I(s)$;
iii) generate the set $J_1 = \{ j \mid \lambda_j(s) \geq 0,\ j \in N\}$;
iv) generate the set $J_2 = N \backslash J_1$;
v) find the permutations $\pi^I(s)$ by first ordering the elements of J_1 in the $P_j^I(s)$ increasing order, then the elements of J_2 in the $P_j^{II}(s)$ decreasing order;
vi) find a permutation $\pi^{II}(s)$ by ordering the elements of the set N in the $\lambda_j(s)$ decreasing order, $j \in N$;
vii) find the best permutation π^* according the formula:

$$\pi^* := \underset{s=1,2,\ldots,m-1}{\arg\min} \left\{ C_{\max}\left(\pi^I(s)\right),\ C_{\max}\left(\pi^{II}(s)\right) \right\}.$$

In Algorithm 1, at step *v)* if some elements have the same value for $P_j^I(s)$ or $P_j^{II}(s)$, then the order will be defined according to the decreasing order of $\lambda_j(s)$. At step *vi)*, if some elements have the same value $\lambda_j(s)$ and if $\lambda_j(s) \geq 0$, the elements are arranged in increasing order of the $P_j^I(s)$, otherwise $(\lambda_j(s) < 0)$ in decreasing order of the $P_j^{II}(s)$.

Finally, if m is an odd number, then for $s = \lceil m/2 \rceil$ the formulae (1) will be replaced in Algorithm 1 by the next formulae:

$$P_j^I := \sum_{i=1}^{s} t_{ij} \quad \text{and} \quad P_j^{II} := \sum_{i=s}^{m} t_{ij}, \qquad (2)$$

where $\lceil k \rceil$ means the integer value greater than or equal to k.

Sequential search method

The gradient method – see, for example, [17] – proved its efficiency for the resolution of the problems of the nonlinear programming. The idea to use it in discrete optimization belongs to Feigin [18]. In the

discrete modification of the gradient method, the decision variables are modified one by one. Firstly, the variable which highly modifies the objective function is incremented until the objective function remains unchanged, the other variables staying fixed. Then, the variable which has just been modified is set at its last value and the incrementing process starts for another variable (which modifies the objective function the most at this point), and so on. This method is called the *sequential search method*.

For the permutation flow-shop problem, the sequential search method cannot be applied directly. In the permutations, it is not possible, as in a vector, to change one component without changing another. Only some exchanges are possible. The modifications of the sequential search method, presented in Algorithm 2, are then used.

Algorithm 2:

i) $k = 1$, study the possible exchanges between the component π_k and the components π_d, $d = k+1, k+2, \ldots, n$, by computing each time the value $C_{\max}(\pi|\pi_k \leftrightarrow \pi_d)$;
ii) repeat step *i*) for $k = 2, \ldots, n-1$ starting for each k from the initial permutation π;
iii) find the value for k and d for which the gain is maximum (i.e. $C_{\max}(\pi|\pi_k \leftrightarrow \pi_d)$ is minimum);
iv) form a new permutation π by exchanging π_k and π_d;
v) repeat steps *i*)–*v*) while $C_{\max}$ decreases;

where $\pi_k \leftrightarrow \pi_d$ means that in the permutation π the k-th and d-th components have been exchanged. At each iteration, this algorithm compares $(n^2 - n)/2$ permutations where each component is in all the possible positions. For instance, for $n = 3$ and $\pi = (2, 1, 3)$, the studied permutations are:

(3, 1, 2)

(1, 2, 3)

(2, 3, 1).

Error estimation based on the discrete Ψ-transform method

The initial idea of the Ψ-transform method has been proposed by Chinchinadze [19] for continuous functions. The idea is to substitute an objective function $F(X)$ with several arguments and various local optimums by a continuous function $\Psi(\xi)$ with one argument and with only one optimum.

Let us define a set:

$$E(\xi) = \{X \,|\, F(X) \leq \xi,\, X \in D\} \qquad X = (x_1, x_2, \ldots, x_n) \tag{3}$$

with the characterization function $\Theta(X, \xi)$,

$$\Theta(X,\xi) = \begin{cases} 1, \text{ if } X \text{ belongs to } E(\xi), \\ 0, \text{ otherwise,} \end{cases} \tag{4}$$

then, the function $\Psi(\xi)$ is defined as a normalized weighted value of the set $E(\xi)$ on the domain D:

$$\Psi(\xi) = \int_D \Theta(X, \xi)\, dX \tag{5}$$

$\Psi(\xi)$ is a continuous strictly increasing function which is equal to zero at the following points:

$$\xi \leq \xi^*, \qquad \xi^* = \min_{X \in D} F(X). \tag{6}$$

For combinatorial problems, the function $\Psi(\xi)$ can be statistically estimated [20, 21] with L random simulations of the vector X. In this case, in order to estimate the optimization error, a result concerning the confidence interval for ξ^* can be used. Indeed, Boender et al. [22] have shown that, if there exists positive constants r and χ such that:

$$\lim_{\xi -> \xi^*} \frac{\Psi(\xi)}{(\xi - \xi^*)^{1/r}} = \chi \tag{7}$$

then, asymptotically ($L \to \infty$)

$$\text{Prob}\left(\xi^* \in \left[\breve{F},\, F_1\right]\right) = p, \tag{8}$$

$$\breve{F} = F_1 - \frac{F_2 - F_1}{p^{1/r} - 1}, \tag{9}$$

where F_1, F_2 are respectively the smallest and the nearest to the smallest values of objective function $F(X)$; p is the confidence probability.

Ofitserov [16] has shown that for combinatorial problems and for $r = 1$:

$$\lim_{\xi -> \xi^*} \frac{\Psi(\xi)}{(\xi - \xi^*)} = \frac{1}{|D|} \times |E(\xi^*)|. \tag{10}$$

Then, the value ξ^* of the global minimum of the objective function $F(\pi) = C_{\max}(\pi)$ is estimated as follows:

$$\xi^* = F_1 + \left(\frac{1}{d} - 1\right) \frac{F_2 - F_1}{(1/p - 1)} \tag{11}$$

then, the proposed error of the solution will be:

$$\widehat{\varepsilon} = \frac{F_1 - \xi^*}{\xi^*}, \tag{12}$$

where d is a parameter which gives us the ability to vary value ξ^* within the confidence interval, $1 \leq d \leq n$.

Flow-shop with infinite buffers

In this section, it is shown how the methods explained in the previous section can be used for permutation flow-shops with infinite buffers between machines.

Problem statement

Presence or absence of the infinite buffers is essential for the process of handling parts. For instance, if buffers are not present in the system, part π_j can not leave machine i, $1 \leq i \leq m-1$, until part π_{j-1} has left machine $i+1$; therefore, machine i will be blocked and will not handle any other part.

In the system with infinite buffers, the length of schedule $F(\pi) = C_{\max}(\pi)$ of handling n parts by the m sequential machines (were π is a permutation of the serial number of the parts) can be found as a solution of recurrent statement:

$$T_{i,\pi_j} = t_{i,\pi_j} + \max\left\{T_{i-1,\pi_j}, T_{i,\pi_{j-1}}\right\}, \qquad (13)$$

$$F(\pi) = C_{\max}(\pi) = T_{m,\pi_n}, \qquad (14)$$

where $T_{0,j} = 0$, $T_{i,\pi_0} = 0$, $i = 1, 2, \ldots, m$, $j \in N$.

Optimization algorithm

An approximate algorithm for constructing optimal schedules to minimize the completion time was developed on the basis of the Petrov matrix method, the method of sequential search and the discrete Ψ-transform method. The method suggests splitting the set of machines into two subsets $m-1$ times. The best schedule is selected. Then, this permutation is used as input for the method of sequential search.

Algorithm 3:

i) apply Algorithm 1 (heuristic based on the Petrov matrix method);
ii) apply Algorithm 2 (a modification of the sequential search method);
iii) find $\widehat{\varepsilon}$ according statement (12).

Let us notice that in order to do the step *iii)* of this algorithm, it is necessary to calculate the value F_2 at the steps *i)* and *ii)*.

Efficiency of the algorithm

The complexity of Algorithm 3 is the complexity to compute the parameters P_j^I, P_j^{II}, λ_j, $j = 1, 2, \ldots, n$ plus that of the sequential search method. A computation of those parameters can be done in $(m+1)n + n\log n$ operations, applied $m-1$ times. Thus, the complexity of Algorithm 3 does not exceed $O(m^2\, n + nm\log n)$ operations. For each iteration of Algorithm 3, $(n^2-n)/2$ permutations are performed. The computation of statement (13) can be done in $O(m\ n)$ operations. The sequential search complexity is $O(m\ n^3)$. The global complexity of Algorithm 3 is finally less than $O(m^2 n + mn^3)$. These calculi are done under the hypothesis that the number of calls at the sequential search is constant. Experiments show that this number is proportional to n.

This algorithm was used to solve some test examples. The parameter d from statement (11) was taken to equal to 1, that means, for all test examples, the values $\delta = (\xi^* - F_0)/F_0$ 100% and $\varepsilon = (F^* - F_0)/F_0$ 100% are equal (where F^* is the value of the objective function $F(\pi)$ obtained by the appropriate algorithm and F_0 is the optimal value of the objective function). Ten test examples in which the number of machines m changes from 2 to 10 with fixed $n = 10$ have been solved. In these examples, duration t_{ij} of processing part i at the machine j were generated as random values in [1..9]. In addition, we solved the test example which was considered in the paper [23] with $m = 12$ and $n = 12$. Results have been placed in Table 1.

Table 1
Relative errors compared with the optimal schedule in the case of flow-shop with buffers.

Examples	Error of the matrix method, %	Error of the proposed algorithm, %
2	0	0
3	0	0
4	10.3	0
5	11	1.4
6	5.2	0
7	8.5	0
8	7.2	1.2
9	10	0
10	9.2	2
12	8.6	2.7
ε, %	7.0	0.73

Here $\varepsilon = (1/10)(\sum_{m=2}^{10} \varepsilon_m + \varepsilon_{12})$; ε_m is the relative error of the solution with respect to the optimal value of the function (13)–(14) obtained by using a branch and bound method with a given m; ε_{12} is the error compared to the solution given by Artomonov [23].

In the work of Petrov [12], it was shown that the average error of the matrix method is no more than 10%. In the publication of Lvov [14], the average error has been reduced to 5%. Results of numerical experiments shows that using the sequential search method in the last stage of the proposed algorithm reduces the average error to 1% (see Table 1).

Flow-shop without buffers

The same approach used in Sec. 3 is employed here; but, this time, for flow-shops without buffers between the machines.

Problem statement

In flow-shops without buffers, for a given permutation π of the parts, the schedule length $C_{\max}(\pi)$ of processing all n parts by m sequential machines, can be found as a solution of the following recurrent statement:

$$T_{i,\pi_j} = t_{i,\pi_j} + \\ + \max\left\{T_{i-1,\pi_j}, T_{i+1,\pi_{j-1}} - t_{i+1,\pi_{j-1}}\right\}, \quad (15)$$

$$F(\pi) = C_{\max}(\pi) = T_{m,\pi_n}, \quad (16)$$

where $i = 1, 2, \ldots, m$, $j \in N$, $T_{0,\pi_j} = 0$ and

$$T_{i,\pi_0} - t_{i,\pi_0} = 0, \quad (17)$$

$$T_{m+1,\pi_{j-1}} - t_{m+1,\pi_{j-1}} = T_{m,\pi_{j-1}}. \quad (18)$$

Optimisation algorithm

An approximate algorithm (see Algorithm 4) for constructing optimal schedule for such a flow-shop is developed on the same basis as in Sec. 3, but instead of the Petrov matrix method, the Gilmore-Gomory algorithm is used [24].

Algorithm 4:

i) for each $s = 1, 2, \ldots, m-1$, build two "dummy" machines and compute P_j^I et P_j^{II} according to the formulae (1)–(2);
ii) apply the Gimore-Gomory algorithm for each result obtained at step *i*);
iii) choose the best permutation $\pi^* := \arg\min\left\{F(\pi^{(s)})\right\}$;
iv) apply Algorithm 2 with the initial permutation π^*;
v) find $\widehat{\varepsilon}$ according to statement (12).

The use of the Gilmore-Gomory algorithm is justified by the fact that for $m = 2$ the absence of waiting time is equivalent to the lack of buffers [25].

Efficiency of the algorithm

The Gilmore-Gomory algorithm gives an optimal schedule, for two machines without a buffer in between, in $O(n^2)$ operations. In the work of Papadimitriou and Kanellakis [26] the complexity of this algorithm was reduced to $O(n \log n)$ by the use of a more efficient algorithm to search the minimal cost spanning tree inside the Gilmore-Gomory algorithm.

Algorithm 4 complexity is defined by both the complexity of the objective function (15)–(16) computation and the complexity of the Gilmore-Gomory and sequential search procedures. The computation of function (15)–(16) can be done in the less time than $O(m\,n^2)$. Then, the sequential search procedure needs only $O(m^2n^3)$ operations. If it is admitted that Gilmore-Gomory procedure complexity is $O(n \log n)$, then Algorithm 4 complexity is not more than $O(m^2n^3)$.

But this algorithm can become of an exponential complexity because, for $m = 2$, the Gilmore-Gomory method can provide several optimal solutions. As these solutions are no longer optimal for $m > 2$, it will be necessary to try all of them to find the best one. In the worst case, it will need $n!$ calculations.

This algorithm was used to solve some sets of test examples. For these examples, the number of parts is equal to 4, 5, 6 while $m = 6$. Processing times were the same as in the previous cases (see Subsec. 3.3). For these examples, we obtained optimal values of objective function (15)–(16). Optimality of these values was proven by a complete enumeration of all available $n!$ permutations. Results are reported in Table 2.

Table 2
Comparative results for the examples in the case of flow-shop without buffers.

	$s = m/2$				$s = 1, 2, \ldots, m-1$				
n	ε	N_p	N_0	N^*	ε	N_p	N_0	N^*	N_g
4	4.9	2	1	0	0	10	6	6	3
5	3.9	7	2	0	0	26	15	3	3
6	0	11	4	3	0	72	30	3	1

Here ε is the relative error of the solution (in %); N_p is the overall number of permutations; N_0 is the number of different permutations from N_p; N^* is the number of optimal permutations from N_p; N_g is the number of global minima of the objective function (15)–(16); s is the number of machines in the first set of the partition, i.e. m–s machines are in the second set.

The results in Table 2 are those obtained before the application of the sequential search method. Table 2 shows that N_p increases exponentially with the increase of n. A great number of permutations is doubled ($N_0 \approx N_p/2$). This allows us to propose a modification that brings this algorithm under a polynomial complexity. This modification is the following: do not study all the possible combinations when the Gilmore-Gomory algorithm gives several equivalent solutions. For this, at step *ii)* in Algorithm 4, a simple rule is applied:

i) if there are two values for P_j^I or P_j^{II} which are equal, then they will be ordered according the value of j;
ii) if in the multi-graph there are two arcs with the same weight, then the one with the smallest number will be included in the minimal cost spanning tree.

With the application of this rule, the results obtained are the same as those presented in Table 2, ex-

cept for $n = 6$ where the error became 13%. This error was eliminated by the application of the sequential search method. It can be concluded that the complexity of Algorithm 4 can be limited to $O(m^2\ n^3)$ without a perceptible degradation of the results.

Conclusion

An interesting way for solving scheduling problem was explored in this paper. The goal was to find heuristics with solid theoretical foundations and with a possibility to estimate subsequently their error rate. The approach developed is based on some idea of global optimization for continuous functions. In addition, some new techniques inspired from Ψ-transform were used.

The approach proposed in this paper was tested on several random generated examples as well as on examples from literature. The numerical results show a relative efficacity of the techniques proposed. The improvements obtained for heuristics used is interesting. From the results of the experiments presented, it follows that the algorithms developed are effective and can be used to solve scheduling problems in production and logistics.

Evidently, this work is limited to some specific algorithms and need to be expanded to a broader field and tested on a larger set of benchmarks as well as on real life cases.

The authors thanks to Chris Yukna for the help in English.

References

[1] Tanaev V.S., Gordon V.S. and Shafranski Y.M., *Scheduling Theory. Single-Stage Systems*, Kluwer Academic Publishers, 1994.

[2] Nigmatullin R.G., Complexity of Approximate Methods for Combinatorial Problems, *Soviet Mathematical Doklady*, **16**, 1199–1203, 1975.

[3] Chu C. and Proth J.-M., *L'ordonnancement et ses applications*, Masson, Paris, 1996.

[4] Johnson S.M., Optimal two-and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, **1**, 61–68, 1954.

[5] Proust C., Influence des idées de S.M. Johnson sur la résolution de problèmes d'ordonnancement de type n/m/F, contraintes diverses/Cmax, *Working paper*, Laboratoire d'Informatique, Université de Tours, France, 1989.

[6] Osman I.H., Potts C.N., Simulated annealing for permutation flow-shop scheduling, *Omega*, **17**, (6), 551–557, 1989.

[7] Ladhari T., and Haouari M., A computational study of the permutation flow shop problem based on a tight lower bound, *Computers & Operations Research*, **32** (7), 1831–1847, 2005.

[8] Zobolas G.I., Tarantilis C.D., and Ioannou G., Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, *Computers & Operations Research*, **36** (4), 1249–1267, 2009.

[9] Laha D., and Sarin S.C., A heuristic to minimize total flow time in permutation flow shop, *Omega*, **37** (3), 734–739, 2009.

[10] Ribas I., Companys R., and Tort-Martorell X., Comparing three-step heuristics for the permutation flow shop problem, *Computers & Operations Research*, **37** (12), 2062–2070, 2010.

[11] Tanaev V.S., Sotskov Y.N. and Strusevitch V.A., *Scheduling Theory. Multi-Stage Systems*, Kluwer Academic Publishers, 1994.

[12] Petrov V., *Manufacturing System Planning*, Mashinostroenie, Leningrad, 1966a – in Russian.

[13] Petrov V., *Flow Line Group Production Planning*. Business Publications, London, 1966b.

[14] Lvov Y., Research of Optimal Flow-shop Scheduling Using Stochastic Methods, *Works of the Institute for Engineering and Economy of Leningrad*, **62**/2, 15–33, 1966 – in Russian.

[15] Campbell H.G., Dudek R.A. and Smith M.L., A Heuristic Algorithm for the n Job, m Machine Sequencing Problem, *Management Science*, **16**, B630–B637, 1970.

[16] Ofitserov D., *Computer-Aided Production Planning in an Automatic Factory*, Ph.D. Dissertation, Institute of Engineering Cybernetics, Minsk, Belarus, 1987.

[17] Taha H.A., *Operations research: An Introduction*, MacMillan Publishing Co, Inc. New York, 1982.

[18] Feigin L., Statistic Estimation of Optimum in the General Scheduling Problem, *Izvestia AN SSSR, Engineering Cybernetics*, **2**, 221–224, 1982 – in Russian.

[19] Chichinadze V.K., The ψ-transform for Solving Linear and Non-Linear Programming Problems, *Automatica*, **5**, 347–355, 1969.

[20] Dolgui A. and Ofitserov D., A Stochastic Method for Discrete and Continuous Optimization in Manufacturing Systems, *Journal of Intelligent Manufacturing*, **8** (5), 405–413, 1997.

[21] Dolgui A., and Sysoev V., Une heuristique d'optimisation globale basée sur la ψ- transformation, *RAIRO Operations Research*, **37**, 119–141, 2003.

[22] Boender C.G.E., Rinnooy Kan A.H.G., Timmer G.T. and Stougie L., A Stochastic Method for Global Optimization, *Mathematical Programming*, **22**, 125–140, 1982.

[23] Artamonov I.M., An Algorithm for the Resolution of the Johnson-Bellman Problem, *Research in Mathematics*, Kichinev, **68**, 3–10, 1982 – in Russian.

[24] Gilmore P.C. and Gomory R.E., Sequencing a one State-Variable Machine: A Solvable Case of the Travelling Salesman Problem, *Operations Research*, **12**, 655–659, 1964.

[25] Levner E.B., Optimal Planning for Many Machines, *Automation and Remote Control*, Moskow, **12**, 94–100, 1973 – in Russian.

[26] Papadimitriou C.H. and Kanellakis P.C., Flow-Shop Scheduling with Limited Temporary Storage, *Journal of the Association for Computing Machinery*, **27**, 533–549, 1980.