

Genetic Algorithm for Late Work Minimization in a Flow Shop System

Malgorzata Sterna, Jacek Blazewicz

Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland,
{malgorzata.sterna, jacek.blazewicz}@cs.put.poznan.pl

Erwin Pesch

Institute of Information Systems, FB 5 - Faculty of Economics, University of Siegen, Hoelderlinstrasse 3,
57068 Siegen, Germany, erwin.pesch@uni-siegen.de

The work concerns a genetic algorithm for the flow shop scheduling problem with release times and the late work criterion, which is NP-hard. The proposed meta-heuristic method minimizes the number of units of job processing times executed after their given due dates. We describe the components of this approach and present the analysis of test results. Computational experiments, preceded by an extensive tuning process, were performed for different classes of problem instances in terms of the distribution of release times and due dates over time.

Keywords: Shop Scheduling, Late Work Criterion, Meta-heuristic Search, Genetic Algorithm.

1. Introduction

The late work objective function [1], minimizing the total number of tardy units of particular activities executed in the system, takes into account mostly the amount of late work not the quantity of the delay. It combines the idea of two classical performance measures [2, 9]: the total tardiness and the number of late jobs. Similarly as tardiness, late work increases with a job delay, but when the job becomes totally late, the penalty remains on the certain level (determined by the job processing time) as for the number of late jobs parameter. The concept of late work is strictly related to the imprecise computation model (e.g. [20]). It can be applied for all cases when jobs or activities can be delayed, but the size of their delayed parts influences the quality of a schedule.

First, the criteria based on late work (cf. [26] for a survey) were investigated for parallel identical and uniform [1, 3] and for single machine(s) environments [14–16, 18, 19, 22, 23]. Then, these performance measures were applied for the dedicated machines cases. The studies on the shop systems concentrated mainly on the two-machine weighted cases with a common due date and they resulted in proving the ordinary NP-hardness of open [4], flow [5] and job [8] shop problems. Results of computational experiments are available only for the two-machine flow shop case for which list scheduling approach [6] and three meta-heuristics (tabu search, simulated annealing and variable neighborhood search) were proposed [7]. The non-weighted late work criterion was investigated only for the open-shop problem [4, 25] and, recently, for the flow shop case [21, 28]. There is also an example of a more practical application of the total late work performance measure to the flexible manufacturing environment, modeled as the extended job shop system [27].

In the presented work, we propose a genetic algorithm [12, 17, 24] for the flow shop scheduling problem with the arbitrary number of machines, job release times and the total late work criterion, which is known to be NP-hard [21]. Evolutionary algorithms have not been applied for the late work minimization so far.

In Section 2, the problem under consideration is defined formally. Section 3 describes particular components of the proposed genetic algorithm. Section 4 presents results of computational experiments. The work is concluded in Section 5.

2. Problem formulation

The flow shop problem (cf. e.g. [2, 9]) investigated in the work, $F | r_j | Y$, concerns executing a set of n jobs $J = \{J_1, J_2, \dots, J_n\}$ on a set of m dedicated machines $M = \{M_1, M_2, \dots, M_m\}$. Each job $J_j \in J$

consists of m tasks T_{ij} ($i = 1, \dots, m$), which have to be processed without preemption for p_{ij} time units by machine M_i . Each job J_j has to be first executed by machine M_1 , then, M_2, M_3, \dots, M_m and its execution can start at earliest at release time r_j . Moreover, each job can be processed on at most one machine at the same time and each machine can perform at most one job at the same time. Since we consider the permutation flow shop model, the jobs execution order on all machines is identical, and a schedule can be described as a single sequence of jobs from set J .

The goal is to minimize the total late work in the system. The late work Y_j for job J_j is determined as the sum of late parts of tasks T_{1j}, \dots, T_{mj} , executed after due date d_j . Denoting the completion time of task T_{ij} as C_{ij} , the late work for job J_j is given by:

$$Y_j = \sum_{i=1}^m \min\{\max\{0, C_{ij} - d_j\}, p_{ij}\}.$$

Consequently, the total late work for the entire flow shop system is determined as:

$$Y = \sum_{j=1}^n Y_j.$$

The problem stated above is NP-hard, due to NP-hardness of the two-machine flow shop case $F2 | d_j = d | Y$, which has been proven [21] by a transformation from the partition problem. Due to the intractability of problem $F | r_j | Y$, it can be solved efficiently only by approximation approaches.

3. Genetic algorithm

The genetic algorithm (GA) proposed for the problem under consideration is based on the general framework for this meta-heuristic [12, 17, 24]. The method starts with an initial population of chromosomes, where each chromosome represents a single solution to the problem. In each iteration, a current population is modified by applying two operators: crossover and mutation, then a new population is selected for the next iteration of the algorithm. The method stops after reaching a termination condition.

3.1. Solution representation

A population analyzed in each iteration of GA is a set of chromosomes, which indirectly represent solutions to problem $F | r_j | Y$. A chromosome is transformed to a schedule by applying the list scheduling method (cf. e.g. [2]).

The list scheduling algorithm (LA) is a constructive procedure, which builds a feasible solution iteratively extending a partial schedule and selecting jobs according to a given priority dispatching rule (PDR, cf. e.g. [13]). In each iteration, the method determines the set of available jobs (i.e. jobs whose release times have been exceeded) and it assigns the job with the highest priority (with regard to PDR) to the machines, as soon as they are ready for executing another job. Consequently, the list scheduling algorithm performs n steps to construct a feasible solution for a set of n jobs.

In the presented approach, the priority dispatching rule used in LA is not fixed. In each iteration a different rule can be applied. It is determined by a chromosome, which is a sequence of $(n-1)$ priority dispatching rules $(R_1, R_2, \dots, R_{n-1})$ [11]. A schedule corresponding to a particular chromosome is constructed by the list scheduling algorithm by assigning to the machines a job selected from a set of available jobs according to priority dispatching rule R_i at iteration i (the last selection is obvious).

As it was mentioned, chromosomes consist of static or dynamic priority dispatching rules. Among static priority dispatching rules, based only on the problem's parameters, we apply the following ones:

- LPT – the longest job processing time $\sum_{i=1}^m p_{ij}$,
- SPT – the shortest job processing time $\sum_{i=1}^m p_{ij}$,
- LPTP – the longest processing time of the first task in a job p_{1j} ,

- SPTP – the shortest processing time of the first task in a job p_{1j} ,
- EDD – the earliest job due date d_j ,
- SA – the smallest allowance $(d_j - r_j)$,
- SA/PT – the smallest allowance per job processing time $(d_j - r_j) / \sum_{i=1}^m p_{ij}$.

Assuming that t_1 denotes the partial schedule length on machine M_1 and c_{ij} denotes the completion time of task T_{ij} , if job J_j would be assigned to the machines as the next one, we use the following dynamic priority dispatching rules:

- ML – the minimum predicted lateness $(d_j - (t_1 + \sum_{i=1}^m p_{ij}))$,
- MS – the minimum slack $(d_j - t_1) / \sum_{i=1}^m p_{ij}$,
- MnCT – the minimum job completion time c_{mj} ,
- MxCT – the maximum job completion time c_{mj} ,
- STR – the smallest time remaining $(d_j - c_{mj})$.

In each iteration, the genetic algorithm analyzes a population of p chromosomes. The initial population contains twelve uniform chromosomes, i.e. chromosomes which are built from only one priority dispatching rule, and $(p - 12)$ chromosomes generated randomly.

3.2. Genetic operators

The implemented genetic algorithm uses a two-parent crossover operator, which constructs two new chromosomes from two parent ones. We propose two operators: one-point and two-point crossover. In the first case (C1), prefixes of a randomly chosen length are interchanged between two chromosomes. In the latter one (C2), subsequences between two randomly chosen positions are taken into account.

The mutation operator modifies a single chromosome by changing the sequence of priority dispatching rules randomly. We propose two mutation operators. The first one (M1) interchanges PDRs between two randomly chosen positions. The latter one (M2) influences the structure of a chromosome significantly – it randomly selects one PDR in a chromosome and replaces all its instances with another randomly selected PDR.

The number of chromosomes selected from a population for the recombination (as parents for crossover operator) is determined by crossover rate (the percentage of the best chromosomes which form pairs for crossover). Similarly, the number of chromosomes being an input for the second genetic operator – mutation – is determined by mutation rate (the probability of being selected for mutation).

3.3. Search process

In each iteration of the genetic algorithm, a population of chromosomes is extended with new chromosomes obtained by the recombination. The quality of all individuals in the population is determined by the criterion value, i.e. the total late work for a schedule corresponding to a certain chromosome. We apply GA with the constant size of the population and the steady state evolution process with the ranking approach. At the end of each iteration, we select the constant number of the best chromosomes from a current population to be analyzed in the following steps of the method.

The search process is continued until the termination condition is reached. We use two stopping criteria: the number of generated populations (i.e. the number of iterations) and the number of populations without improvement in the solution quality.

4. Computational experiments

The genetic algorithm for problem $F | r_j | Y$ was implemented in Java 1.5 and tested on PC (Intel® Pentium® M740 1.73GHz 1GB RAM). The computational experiments were divided into two

phases. First, the method was tuned to determine appropriate values of its control parameters, then the behavior of the tuned GA was analyzed for different classes of input data.

4.1. Input data

The proposed genetic algorithm was tested for the classical benchmarks for the flow shop scheduling problem [29], which were completed with additional parameters, i.e. job release times and due dates (following the idea from [10]).

Release times r_j were taken from uniform distribution over $[0, \alpha LB]$, where LB denotes the lower bound of the schedule length, known for particular benchmarks [29], and α is a control parameter. Due dates d_j were determined with regard to job release times as:

$$d_j = r_j + \sum_{i=1}^m p_{ij} + U[N - NR/2, N + NR/2],$$

where $N = (1-T)LB$, T and R are control parameters, U denotes a uniform distribution. Thus, due date d_j for a certain job was equal to its earliest feasible completion time ($r_j + \sum_{i=1}^m p_{ij}$) increased with a certain factor.

Depending on the parameters values (α , R , T), it is possible to generate instances with tight and loose r_j and d_j values. The tight values of r_j (TR) correspond to the situation when all release times are taken from a small range of values around time zero, while the loose values of r_j (LR) model the situation when release times are spread in time. Similarly, tight values of d_j (TD) correspond to the case when due dates are very close to the smallest feasible completion time of jobs. On the contrary, the loose values of d_j (LD) model the situation when due dates are not so restrictive.

Based on 36 selected classical benchmarks [29] of 12 different sizes (in terms of the numbers of jobs and machines, $n \in [20, 500]$ and $m \in [5, 20]$), four groups of instances were generated differing in due dates and release times of jobs:

- TR-TD with $\alpha = 0.1$, $R = 1$, $T = 0.8$,
- LR-TD with $\alpha = 0.4$, $R = 1$, $T = 0.8$,
- TR-LD with $\alpha = 0.1$, $R = 0.5$, $T = 0.5$,
- LR-LD with $\alpha = 0.4$, $R = 0.5$, $T = 0.5$.

Consequently, the computational experiments were performed for 144 instances of various characteristics.

4.2. Tuning process

The tuning process was devoted to determining the values of control parameters for the genetic algorithm, which result in its highest efficiency. Obviously, the efficiency of a meta-heuristic depends on input instances and the optimal setting of control values cannot be determined. However, the tuning process makes it possible to adjust a meta-heuristic to input data.

Tuning was performed with all available instances for which the average improvement of the criterion value with regard to its initial value was calculated. We tested 32 configurations with different crossover operators (C1 or C2), mutation operators (M1 or M2), crossover rate (5% or 10%), mutation rate (5% or 10%) and the size of the population (100 or n). The search was stopped after 100 iterations.

Computational experiments showed that depending on the control parameters values, the average criterion improvement varied from 7.89% to 17.77% of its initial value. Similarly the average running time varied from 1801 to 16273 ms. This diversity of results confirms the usefulness of the tuning process. After the careful analysis of its results, during the main experiments, we used two-point crossover operator (C2), mutation operator introducing deep changes into a chromosome (M2), higher crossover and mutation rates (10%) and the size of population equal to 100 individuals. In additional experiments, we compared two termination conditions, i.e. the number of iterations without improvement equal to 100 and to n . For the further experiments, we chose

the first value, which ensured the same quality of obtained solutions within significantly shorter running time than the second one.

4.3. Results of computational experiments

The computational experiments confirmed the strong influence of the problem data on the efficiency of the genetic algorithm (Tables 1 and 2). The large standard deviation values resulted from including into particular classes of instances, instances of various sizes.

Table 1. The differences in the criterion value for different classes of input instances

| Class of instances | Difference between initial and final solutions [%] | | Difference between initial solution and best uniform chromosome [%] | |
|--------------------|--|--------------------|---|--------------------|
| | Average value | Standard deviation | Average value | Standard deviation |
| TR-TD | 5.45 | 3.75 | 0.95 | 1.71 |
| LR-TD | 6.44 | 4.44 | 1.09 | 3.08 |
| TR-LD | 18.71 | 7.08 | 2.17 | 4.45 |
| LR-LD | 45.05 | 25.27 | 3.26 | 8.32 |
| all tests | 18.91 | 20.82 | 1.87 | 5.07 |

The genetic algorithm improved the quality of initial solutions by nearly 19% on average, but its efficiency strictly depended on the type of problem instance.

Class TR contains instances in which most jobs become available at the same time – there are small differences in the release times. Thus, the set of available jobs is numerous and, additionally, most jobs have a similar priority, especially with regard to the PDRs involving the job release times (this effect is additionally strengthened by tight due dates). In consequence, the genetic algorithm was less efficient in improving the solution's quality than for the other classes of instances. On the contrary, class LR includes instances for which release times are spread in time – and there are fewer jobs available at the same time in comparison to LR and the priorities of jobs are more diverse (especially for loose due dates). Since jobs have various priorities for particular priority dispatching rules, modifying the sequence of these rules in chromosomes allowed GA to achieve larger improvements of the criterion value.

However, the influence of due dates on the method's efficiency seems to be more important than release times (which is understandable, since the criterion value is calculated with regard to d_j). Class TD contains instances with tight due dates, in which there is a very small allowance window for particular jobs. In consequence, most of jobs are late, independently of the sequence of their execution, and GA could only slightly improve the criterion value. In the case of loose due dates the allowance windows are much wider and the criterion value is strongly influenced by the sequence of jobs. Thus it was possible to obtain higher improvement in the value of the performance measure by changing the execution order of jobs.

Summing up, the largest criterion value improvement (about 45%) was obtained for instances with loose release times and due dates (LR-LD), but, simultaneously, the efficiency of GA was the least stable, which is expressed in the large standard deviation value.

The similar observations can be formulated with regard to the quality of uniform chromosomes, i.e. list solutions for particular priority dispatching rules. The quality of uniform chromosomes was slightly worse (of 1.87%) than the quality of random individuals in the initial population (Table 1). Surprisingly applying the fixed strategy of assigning jobs to machines was less efficient than a random selection. The less restrictive release times and due dates are, the difference is more visible (the difference increases from 0.95% to 3.26%). For the instances with the tight due dates and release times (TR-TD), the structure of a chromosome was less important – the quality of all solutions in the initial population was similar. The test results showed that uniform chromosomes were worse than random ones, so list scheduling solutions for fixed PDRs were worse than initial ones and, consequently, they were worse than final schedules. This means that the genetic algorithm constructed much better solutions than the list scheduling heuristic (by more than 20% of the criterion value on average).

Table 2. The time efficiency of GA for different classes of input instances

| Class of instances | Number of iterations | | Running time [ms] | |
|--------------------|----------------------|--------------------|-------------------|--------------------|
| | Average | Standard deviation | Average | Standard deviation |
| TR-TD | 371 | 344 | 9510 | 21068 |
| LR-TD | 591 | 1042 | 20590 | 81221 |
| TR-LD | 728 | 831 | 27509 | 61012 |
| LR-LD | 424 | 416 | 21432 | 60788 |
| all tests | 529 | 725 | 19760 | 59843 |

The instance type influenced not only the quality of a schedule constructed by GA, but also its time efficiency (cf. Table 2). The running times of GA obviously reflect the number of populations generated by the method. For class TR-LD, GA performed the smallest number of iterations, since it was not able to improve the initial solution significantly. For LR-LD, the criterion improvement was very large, but it was achieved quite fast, then the algorithm fell into a local optimum and terminated. For the remaining two classes of instances, GA explored more solutions, especially for instances with loose due dates (TR-LD), performing more iterations. Similarly as in the case of the quality of solutions, the large standard deviation values for running times resulted from analyzing instances of various sizes within particular classes. The efficiency of the genetic algorithm depends not only on the characteristic of data set, but it is also significantly influenced by the instance size.

criteria improvement [%]

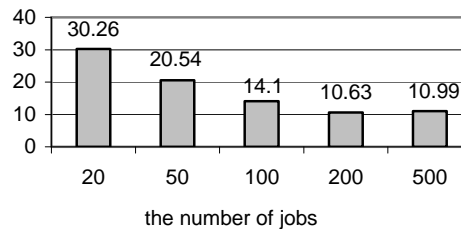


Fig. 1. The criteria value improvement with regard to the initial solution for different sizes of input instances

The improvement of the criteria value with regard to an initial solution decreased with the size of instances, from about 30% to about 11% (Fig. 1). The larger instances were analyzed, the smaller part of the solution space was explored by the genetic algorithm, which influenced the quality of the final solution. Moreover, for large sets of jobs modifications of schedules, performed within a single iteration, caused smaller changes in the criteria value, than it was observed for sets of jobs with a lower cardinality.

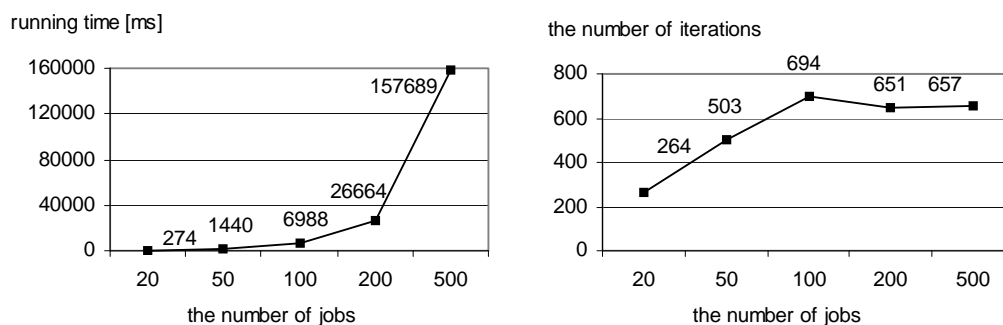


Fig. 2. The time efficiency of GA for different sizes of input instances

The running time of GA increased significantly with the size of instances, but the number of iterations performed by the method (i.e. generated populations) became stabilized for large prob-

lems (Fig. 2). The behavior of GA for large instances was dominated by the list scheduling method (cf. Table 3). It has to be applied for every chromosome in a population and it requires $O(n^2)$ time. For this reason for large problem instances, the list scheduling algorithm consumed the majority of computational time. For smaller instances, the recombination was the second most time-consuming operation. Applying the mutation operator required incomparably shorter running time than using the crossover operator. In this latter case, the genetic operator was applied to a subset of the best chromosomes, which required to rank all individuals in the population, and moreover, after recombination, duplicates were detected and removed from a current population.

Table 3. The usage of running time for particular components of the genetic algorithm

| The number of jobs | The usage of running time for particular GA's components [%] | | | |
|--------------------|--|----------|---------------------------|------------------|
| | Crossover | Mutation | List scheduling algorithm | Other components |
| 20 | 45.88 | 1.11 | 47.83 | 5.18 |
| 50 | 23.27 | 0.44 | 74.55 | 1.74 |
| 100 | 10.9 | 0.18 | 88.25 | 0.67 |
| 200 | 4.16 | 0.07 | 95.54 | 0.23 |
| 500 | 1.26 | 0.02 | 98.65 | 0.06 |

5. Conclusions

In the paper we presented the genetic algorithm for the flow shop problem with release times and the total late work criterion. Computational experiments performed for different classes of input instances allowed us to formulate interesting conclusions on the efficiency of the method under consideration.

The proposed genetic algorithm represents a solution to the problem as a sequence of priority dispatching rules used by the list scheduling algorithm during constructing a schedule. Within the future research, we want to perform additional computational experiments in order to investigate the influence of particular priority dispatching rules on the solution quality. Moreover, the method will be improved and completed with additional components.

Acknowledgements. We thank Bartosz Stefaniak for his effort in implementing and testing the approach investigated within the reported research. The first author was partially supported by KBN grant.

References

- [1] J. Blazewicz (1984), Scheduling preemptible tasks on parallel processors with information loss, *Technique et Science Informatiques* **3(6)**, 415 – 420.
- [2] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz (2001), *Scheduling computer and manufacturing processes*, 2 ed., Springer, Berlin-Heidelberg-New York.
- [3] J. Blazewicz, G. Finke (1987), Minimizing mean weighted execution time loss on identical and uniform processors, *Information Processing Letters* **24**, 259 – 263.
- [4] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2004), Open shop scheduling problems with late work criteria, *Discrete Applied Mathematics* **134**, 1 – 24.
- [5] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2005), The two-machine flow-shop problem with weighted late work criterion and common due date, *European Journal of Operational Research* **165(2)**, 408 – 415.
- [6] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2005), A comparison of solution procedures for two-machine flow shop scheduling with late work criterion, *Computers and Industrial Engineering* **49**, 611 – 624.
- [7] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2007), Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date, *Computers and Operations Research*, doi:10.1016/j.cor.2006.03.021, in press.

- [8] J. Blazewicz, E. Pesch, M. Sterna, F. Werner (2007), A note on two-machine job shop with weighted late work criterion, *Journal of Scheduling* **10**(2), 87 – 95.
- [9] P. Brucker (1998), *Scheduling algorithms*, 2 ed., Springer, Berlin-Heidelberg-New York.
- [10] E. Demirkol, S. Mehta, R. Uzsoy (1998), Benchmarks for shop problems, *European Journal of Operational Research* **109**, 137 – 141.
- [11] U. Dorndorf, E. Pesch (1995), Evolution based learning in a job shop scheduling environment, *Computers and Operations Research* **22**, 25 – 40.
- [12] D.E. Goldberg (1989) *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading.
- [13] R. Haupt (1989), A survey of priority rule-based scheduling, *OR Spektrum* **11**, 3–16.
- [14] A.M.A. Hariri, C.N. Potts, L.N. Van Wassenhove (1995), Single machine scheduling to minimize total late work, *INFORMS Journal on Computing* **7**, 232 – 242.
- [15] D.S. Hochbaum, R. Shamir (1990), Minimizing the number of tardy job unit under release time constraints, *Discrete Applied Mathematics* **28**, 45 – 57.
- [16] D.S. Hochbaum, R. Shamir (1991), Strongly polynomial algorithms for the high multiplicity scheduling problem, *Operations Research* **39**(4), 648 – 653.
- [17] J.H. Holland (1975), *Adaptation in natural and artificial systems*, University of Michigan, Ann Harbor.
- [18] R.B. Kethley, B. Alidaee (2002), Single machine scheduling to minimize total late work: a comparison of scheduling rules and search algorithms, *Computers and Industrial Engineering* **43**, 509 – 528.
- [19] M.Y. Kovalyov, C.N. Potts, L.N. Van Wassenhove (1994), A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work, *Mathematics of Operations Research* **19**(1), 86 – 93.
- [20] J.Y.-T. Leung (2004), Minimizing total weighted error for imprecise computation tasks and related problems, Chapter 34 in: J.Y.-T. Leung (ed.), *Handbook of scheduling: algorithms, models, and performance analysis*, CRC Press, Boca Raton.
- [21] B.M.T. Lin, F.C. Lin, R.C.T. Lee (2006), Two-machine flow-shop scheduling to minimize total late work, *Engineering Optimization* **38**(4), 501 – 509.
- [22] C.N. Potts, L.N. Van Wassenhove (1991), Single machine scheduling to minimize total late work, *Operations Research* **40**(3), 586 – 595.
- [23] C.N. Potts, L.N. Van Wassenhove (1991), Approximation algorithms for scheduling a single machine to minimize total late work, *Operations Research Letters* **11**, 261 – 266.
- [24] K. Sastry, D. Goldberg, G. Kendall (2005), Genetic algorithms, Chapter 4 in: E.K. Burke, G. Kendall (eds.), *Search methodologies: introductory tutorials in optimization and decision support methodologies*, Springer, New York.
- [25] M. Sterna (2000), *Problems and algorithms in non-classical shop scheduling*, Ph.D. Thesis, Scientific Publishers of the Polish Academy of Sciences, Poznan.
- [26] M. Sterna (2006), *Late work scheduling in shop systems*, Dissertations 405, Publishing House of Poznan University of Technology, Poznan.
- [27] M. Sterna (2007), Late work minimization in a small flexible manufacturing system, *Computers and Industrial Engineering* **52**(2), 210 – 228.
- [28] M. Sterna (2007), Dominance relations for two-machine flow-shop problem with late work criterion, *Bulletin of the Polish Academy of Sciences* **55**(1), 59 – 69.
- [29] E. Taillard (1993), Benchmarks for basic scheduling problems, *European Journal of Operational Research* **64**, 278 – 285.
(<http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html>)