Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 574–599
computers & operations research
www.elsevier.com/locate/cor

Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date

Jacek Blazewicz[a], Erwin Pesch[b], Malgorzata Sterna[a,*], Frank Werner[c]

[a] *Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland*
[b] *Institute of Information Systems, FB 5-Faculty of Economics, University of Siegen, Hoelderlinstrasse 3, 57068 Siegen, Germany*
[c] *Faculty of Mathematics, Otto-von-Guericke-University, PSF 4120, 39016 Magdeburg, Germany*

Available online 5 May 2006

Abstract

In this paper, metaheuristic approaches for the two-machine flow-shop problem with a common due date and the weighted late work performance measure ($F2|d_j = d|Y_w$) are presented. The late work criterion estimates the quality of a solution with regard to the duration of the late parts of jobs, not taking into account the quantity of the delay for the fully late activities. Since the problem mentioned is known to be NP-hard, three trajectory methods, namely simulated annealing, tabu search and variable neighborhood search are proposed based on the special features of the case under consideration. Then, the results of computational experiments are reported, in which the metaheuristics were compared one to each other, as well to an exact approach and a list scheduling algorithm.

Keywords: Scheduling; Flow-shop; Late work criteria; Metaheuristic; Tabu search; Simulated annealing; Variable neighborhood search

1. Introduction

The rapid development of real-time systems makes the due date involving criteria especially useful and important, increasing the interest devoted by researchers to this branch of the scheduling theory. The quality of solutions in real systems is usually estimated from different points of view, which can be modeled by different performance measures (cf. e.g. [1–4]).

The late work criteria are relatively new objective functions, which have not been so intensively explored as the maximum lateness or tardiness ones, for example. They estimate the quality of a solution with regard to the number of tardy units of particular activities executed in a system. The late work concept was introduced in the context of a scheduling problem on identical parallel machines [5] and, then, applied to uniform [6] and single [7–13] machine cases. Recently, practical motivations have directed the research to the shop environment, i.e. to systems with dedicated machines [14–21].

For example, modern control systems [5] utilize a great amount of information on the managed environment, gathered from different sensing devices. Hard real-time restrictions cause that some pieces of information are lost, if they are

* Corresponding author. Tel.: +48 61 6652982; fax: +48 61 8771 525.
E-mail address: Malgorzata.Sterna@cs.put.poznan.pl (M. Sterna).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.03.021

exposed beyond feasible periods, due to e.g. system overloading or failures. Any control algorithm has to work out its steering strategy based only on the information available. The information loss, modeled as late work, should be minimized in order to increase the accuracy and quality of a control process.

The late work minimization can be also considered at different levels of managing flexible manufacturing systems [21]. These production environments usually work in a shift manner, which complicates planning a production. The shift length, or more generally speaking, a planning horizon, may be considered as a common due date for all tasks realized in a system. Activities which cannot be scheduled within a certain planning horizon, modeled as late work, have to be assigned to the following ones. Obviously, planners tend to minimize late work, i.e. the amount of work which has to be performed in the incoming planning slot in addition to orders newly appearing in a system. The late work parameter allows to validate a production plan also from a slightly different point of view. Interpreting customer orders as tasks to be executed, minimizing the late work is equivalent minimizing parts of orders, which are delayed. Obviously, every customer is interested in minimizing late parts of his/her orders. Moreover, taking into account a fine, which has to be usually paid for delays, an owner of a system is also concerned in minimizing financial loss caused by the work not finished on time.

Other interesting applications for the late work criteria arise in agriculture. The late work models, in a very natural way, all issues concerning perishable goods [12]. Modeling stretches of land with tasks, one can consider the process of collecting crops as a late work minimization problem. Particular stretches of land differ in their climate and soil conditions, which influence a feasible interval for harvesting. Crops perish, if they are not collected before a given deadline, causing a financial loss. Minimizing late work is equivalent to minimizing the amount of wasted crops, which are not harvested on time. Moreover, the late work criteria can also support optimizing a process of land cultivation, requiring delivering different fertilizers and plant protection substances [16,21]. Particular stretches of land correspond to jobs, while different cultivation actions, which should be performed for these stretches, are represented by tasks constituting jobs. Obviously, particular actions have to be done with certain specialized agriculture machines, and nature conditions determine time intervals in which they should be completed. The actions not performed before a given deadline have to be abandoned (e.g. spreading fertilizers at a certain vegetation stage is forbidden) influencing the quantity of crops. Minimizing the amount of cultivation actions not executed on time is indirectly equivalent to minimizing the loss in the income from harvesting.

Finally, it is worth to be mentioned that the late work idea is a special case of the imprecise computation model (cf. e.g. [20]), in which tasks are divided into two parts: a mandatory and an optional one. This model has its own extensive application field, especially in real-time systems (e.g. flight control systems).

In this paper, we continue the research [17,18] on the two-machine flow-shop problem with the weighted late work criterion and a common due date, which is known to be NP-hard [22], presenting metaheuristics approaches for its solution. In Section 2, the formal definition of the case under consideration is given. In Section 3, the tabu search, simulated annealing and variable neighborhood search methods are described, while Section 4 contains the results of computational experiments performed for these search strategies. Some conclusions are given in Section 5.

2. Problem formulation

The two-machine flow-shop problem with the weighted late work criterion and a common due date, $F2|d_j = d|Y_w$, concerns the scheduling of a set of jobs $J = \{J_1, \ldots, J_j, \ldots, J_n\}$ on two dedicated machines M_1, M_2. Each job J_j has to be performed first on machine M_1 and then on M_2 for p_{1j} and p_{2j} time units, respectively. Each machine can process only one job at any time and, analogously, each job can be executed by only one machine at any time. We look for a non-preemptive schedule minimizing the late work in the system, i.e. minimizing the amount of work executed after a given common due date d.

Denoting by C_{ij} the completion time of job J_j on machine M_i, the late work Y_j for this job is determined as (cf. Fig. 1):

$$Y_j = \sum_{i=1,2} \min\{\max\{0, C_{ij} - d\}, p_{ij}\}.$$

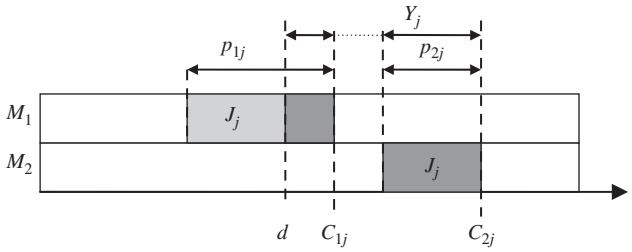

Fig. 1. The late work parameter Y_j for job J_j in the two-machine flow shop environment.

The criterion value to be minimized, estimating the quality of a complete schedule for the whole set of n jobs, taking into account their given weights w_j, $j = 1, \ldots, n$, is determined as:

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

Problem $F2|d_j = d|Y_w$ stated above is NP-hard [17], since a polynomial transformation was constructed from the set partition problem to its decision counterpart. Moreover, it cannot be NP-hard in the strong sense, because a dynamic programming (DP) method with pseudo-polynomial time complexity, $O(n^2d^4)$, was proposed. This exact approach is important mainly from a theoretical point of view, because it determines the complexity status of the case under consideration, as NP-hard in the ordinary sense.

The theoretical studies on problem $F2|d_j = d|Y_w$ were followed by the computational analysis of exact methods solving it [18]. The complex DP approach was compared to an enumerative approach in order to validate its correctness in practice, as well as to validate their efficiency. Unfortunately, DP found optimal solutions in reasonable time only for small problem instances. Thus, to solve the problem efficiently, heuristic approaches had to be proposed. First, a list scheduling method was implemented for the problem under consideration and compared to DP and enumerative algorithms. The list approach is a scheduling technique commonly used, especially for practical applications, because of its ease of implementation and the low time complexity. The computational experiments showed a very high efficiency of the list scheduling algorithm from the run time and the solution quality points of view in comparison to exact strategies. This heuristic constructed solutions with a criterion value of only 2.5% worse, on average, than the optimum. The high efficiency of the list approach resulted from the particularities of the problem under consideration, whose optimal solution has a very specific structure.

Taking into account the encouraging results of the list scheduling algorithm, it was interesting, whether a further improvement of the solution quality could be obtained by extending this approach with an additional search engine, i.e. by proposing metaheuristic methods.

3. Metaheuristic approaches

In this paper, we present three metaheuristics (cf. e.g. [23,24]) for problem $F2|d_j = d|Y_w$, i.e. simulated annealing (SA), tabu search (TS) and variable neighborhood search (VNS) methods. Similarly to the DP approach [17] and the list scheduling algorithm [18], these approaches are based on the specific structure of an optimal solution of the problem under consideration. It was proven [17,21] that in an optimal schedule, the set of early jobs, executed before a common due date, has to be sequenced in Johnson's order [25], which is optimal from the schedule length point of view. Thus, any method solving problem $F2|d_j = d|Y_w$ (cf. Fig. 2) has to select the first late job (L_1) in the system and to divide the remaining activities into two sets of early (E) and late jobs (L). Early jobs are scheduled by Johnson's algorithm, while the late activities are executed in non-increasing order of their weights w_j. In consequence, the crucial element of any method solving the problem is taking the decision whether a particular job is processed early or late in a schedule.

The metaheuristic approaches investigated (SA, TS and VNS) are trajectory methods, which start their search from a certain initial solution and explore the solution space by moving from a current solution to a new one, picked up from its neighborhood, until the termination condition is met.

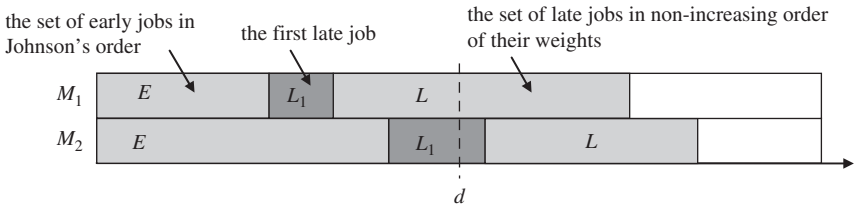

Fig. 2. The general structure of an optimal solution for problem $F2|d_j = d|Y_w$ (with the first late job partially late on machine M_2).

3.1. Initial solution and termination condition

For all metaheuristic approaches proposed, the initial solution is determined either by Johnson's algorithm or by the list scheduling algorithm.

Johnson's method, designed for problem $F2\|C_{\max}$ [25], can be applied as a simple heuristic for the case under consideration, $F2|d_j = d|Y_w$. It orders jobs with a task on machine M_1 no longer than on M_2, i.e. with $p_{1j} \leqslant p_{2j}$, in non-decreasing order of p_{1j}, while the remaining jobs, with $p_{1j} > p_{2j}$, are sequenced in non-increasing order of p_{2j}. Because of the need of sorting jobs, the complexity of Johnson's method is bounded by $O(n \log n)$.

On the contrary, the list scheduling algorithm is a constructive method, which builds a solution by executing jobs, selected according to a given priority dispatching rule [26], one by one on machines. Based on the results obtained within the previous research [18], the maximum weight rule was applied, since it allowed constructing schedules of highest quality. The framework of this procedure, which works in $O(n^2 \log n)$ time, is presented below.

Step 1: set $F = \emptyset$ and $A = J$, where F and A denote the set of executed and available jobs, respectively; set $R = \emptyset$, where R denotes the set of feasible schedules constructed by the algorithm;

Step 2: if $A = \emptyset$ then go to Step 5;

Step 3: take a job $\hat{J}_1$ from A with the maximum weight and set $A = A\backslash\{\hat{J}_1\}$; set $F = F \cup \{\hat{J}_1\}$ and schedule F with Johnson's algorithm obtaining solution S; if the schedule length of S exceeds d, then set $R = R \cup \{S\}$ assuming that the jobs in A are late; if S is the first feasible solution found, then go to Step 4, otherwise go to Step 2;

Step 4: for each $J_x \in A$ schedule $F\backslash\{\hat{J}_1\} \cup \{J_x\}$ by Johnson's algorithm and execute jobs from $A\backslash\{J_x\} \cup \{\hat{J}_1\}$ late obtaining solution S; if the schedule length of S exceeds d, then set $R = R \cup \{S\}$; go to Step 2;

Step 5: select the best schedule from R to be the final solution and stop.

Since two methods of determining the initial solution are available, it is possible for the metaheuristics to start the solution space exploration from two different points, which may lead to final schedules of a different quality.

In the research reported, the search is terminated after exceeding a certain run time limit or after exceeding a given number of iterations without an improvement in the quality of a schedule.

3.2. Neighborhood structures

As it was mentioned, a solution of problem $F2|d_j = d|Y_w$ is described as a sequence (E, L_1, L), where E denotes the set of early jobs scheduled in Johnson's order, L_1 is the first late job (executed partially late either on M_1 or on M_2) and L denotes the set of late jobs performed in non-increasing order of their weights. This feature of an optimal solution is the basis for the neighborhood structures applied within all three metaheuristics considered in this paper. In order to explore the solution space two move definitions were proposed which involve a job shift (N_1) and an interchange of jobs (N_2).

According to the first move strategy, N_1, a new solution is generated by selecting a late job in a current schedule and shifting it to the set of early activities. Since all early jobs are scheduled in Johnson's order, it might happen that the position of the selected activity in the new Johnson's sequence does not ensure executing it totally early. In such a situation, it is necessary to move some early jobs preceding the chosen one in Johnson's sequence after the common due date d. On the other hand, after these modifications, some late jobs succeeding the selected activity in Johnson's order might need shifting before d to complete a schedule. The selection of particular activities may be done at random (selection rule S_1) or according to the weighted processing times of the jobs (selection rule S_2). This leads to two

different neighborhood variants, which can be applied within metaheuristics. The idea of move N_1 is sketched below.

Step 1: move a job J_j selected from $L \cup \{L_1\}$ to the set E and calculate Johnson's schedule for E;

Step 2: if the job J_j is late in the new subschedule for E, then move to the set L some early jobs J_i selected in non-decreasing order of $(p_{1i} + p_{2i})w_i$ values or at random, until J_j becomes early or there are no more early jobs in E to be moved to L;

Step 3: add to the subschedule for E some jobs J_i from the set $L \cup \{L_1\}$ succeeding J_j in Johnson's order, selected in non-increasing order of $(p_{1i} + p_{2i})w_i$ values or at random, until the last job in E exceeds the common due date d becoming L_1;

Step 4: remove the job included to E as the last one and re-include it into L, then try all jobs from L as the first late job and select as L_1 the job for which the best criterion value has been obtained;

Step 5: schedule all late jobs J_i from the set L in non-increasing order of w_i values.

Based on the definition of move N_2 provided below, a new schedule is obtained by choosing a pair of jobs (one from the set of late ones and one from the set of early ones) and interchanging them between these sets. The further modification of a solution is performed in a similar way as for move N_1.

Step 1: move a job J_j selected from the set E to the set L and disable it (i.e. exclude it from the further analysis, such that J_j cannot become early in Step 2);

Step 2: apply move N_1 to a job J_i selected from the set $L \cup \{L_1\}$.

The schedule modification for moves N_1 as well as N_2 requires the application of Johnson's algorithm, which takes $O(n \log n)$ time. In consequence, both types of moves, enabling the exploration of the solution space, are performed in $O(n \log n)$ time.

3.3. Simulated annealing method

The SA method proposed in the paper is based on the classical framework of this metaheuristic (cf. e.g. [23,24,27]), and it can be sketched as follows.

set the iteration number i to 0;
construct an initial schedule S_0;
set an initial temperature T_0 based on the total processing time of the jobs;
set S_0 to be the current solution S;
while termination conditions are not met do
    select at random a solution $\hat{S}$ from the neighborhood $N_k(S)$;
    calculate the change in the criterion value $\Delta Y_w = Y_w(\hat{S}) - Y_w(S)$;
    if $\Delta Y_w < 0$, then replace S by $\hat{S}$ else replace S by $\hat{S}$ with probability $P(\Delta Y_w, T_i)$;
    update temperature T_i according to the cooling scheme Q;
    set $i = i + 1$.

The algorithm starts the search from an initial solution S_0 with an initial temperature T_0, whose value is set to the multiplied total processing time of the jobs (where the multiplication factor is a control parameter). The initial temperature is settled with regard to the number of jobs in order to adjust the search process to the size of a problem instance. For larger instances, the solution process can last longer, which may increase the quality of a final solution. At each iteration numbered with i, the method constructs a new solution $\hat{S}$ from a current one S according to the move definition N_1 or N_2, by shifting or interchanging jobs, respectively. This means that SA, performing move N_1 or N_2, picks at random one solution $\hat{S}$ from the neighborhood of the current solution S, $N_k(S)$, where $k \in \{1, 2\}$. The move configuration, i.e. the move type and the job selection rule within the move (cf. Section 3.2), are control parameters for the SA algorithm. If the new schedule improves the criterion value (i.e. if $Y_w(\hat{S}) - Y_w(S) < 0$, where $Y_w(s)$ denotes the criterion value for a schedule s), then it is accepted, otherwise it replaces the previous solution with a probability depending on the criterion value deterioration ΔY_w and the current temperature value at the ith iteration T_i, i.e. with $P(\Delta Y_w, T_i) = \exp(-\Delta Y_w / T_i)$. At the end of each iteration, the temperature is decreased according to a geometric reduction scheme Q, i.e. $T_{i+1} = \alpha T_i$, where $\alpha \in (0, 1)$ is a control parameter. The preliminary computational

experiments showed that a arithmetic reduction scheme ($T_{i+1} = T_i - \alpha$) is much less efficient for the problem under consideration, so it has been excluded from the further experimental analysis. SA finishes its search after reaching the termination condition mentioned in Section 3.1 or after decreasing the temperature to zero (more precisely speaking, to a value small enough).

3.4. Tabu search method

The TS method implemented for problem $F2|d_j = d|Y_w$ follows the classical scheme of this approach (cf. e.g. [23,24,28]) as it is sketched below.

construct an initial schedule S_0;
set S_0 to be the current solution S;
initialize tabu list T;
while termination conditions are not met do
    determine C as the neighborhood $N_k(S, \beta)$ restricted to the solutions without tabu status;
    replace S with the best solution $\hat{S}$ from C;
    update tabu list T.

In the TS algorithm, a new solution $\hat{S}$ is selected as the best not forbidden schedule from the neighborhood $N_k(S)$ generated from a current schedule S by applying move N_1 or N_2. The complete neighborhood $N_1(S)$ contains all schedules obtained by shifting every late job in S early, while the complete neighborhood $N_2(S)$ consists of solutions determined by interchanging every early job with every late job in S. However, it is possible to run the TS method with the restricted neighborhoods, $N_k(S, \beta)$, in which only β percent of jobs are considered for shifting or interchanging in particular definitions of a move. Obviously, for $\beta = 100\%$ the complete neighborhood is generated. In the restricted neighborhoods with $\beta < 100\%$, the jobs for shifting and interchanging are chosen at random. As we have mentioned, the best not forbidden schedule generated from the neighborhood becomes the starting point for the next iteration. In order to prevent the TS method from returning to the solutions already visited, the move leading to the schedule newly accepted is stored in the tabu list T. The attempt to reverse this move will cause the tabu status for related solutions in the neighborhoods which are considered in the following iterations. Consequently, only solutions without tabu status (constituting a candidate set C) are taken into account in the search process. The tabu list is managed according to the first in first out rule (FIFO) and its length is determined with regard to the number of jobs (as the multiplied cardinality of the set of jobs). The TS method stops after reaching the termination condition mentioned in Section 3.1 or when there is no solution in the neighborhood without the tabu status.

3.5. Variable neighborhood search method

The VNS method is a strategy using a local search algorithm and dynamically changing neighborhood structures (cf. e.g. [23,24,29]). The general idea of this approach applied within the presented research is given below.

construct an initial schedule S_0;
set S_0 to be the current solution S;
while termination conditions are not met do
    set $k = 1$;
    while $k \leqslant 3$ do begin
        pick at random S' from the neighborhood $\breve{N}_k(S)$;
        improve S' to S'' with the TS or SA algorithm starting from a schedule S' as an initial solution;
        if $Y_w(S'') < Y_w(S)$, then replace S with S'' and set $k = 1$,
        else set $k = k + 1$.

The VNS algorithm starts the search from an initial schedule S_0 as the current schedule S and it repeatedly applies three neighborhood definitions $\breve{N}_k(S)$, $k = 1, 2, 3$. Particular neighborhoods $\breve{N}_1(S)$, $\breve{N}_2(S)$, $\breve{N}_3(S)$ have an increasing cardinality, i.e. $|\breve{N}_1(S)| < |\breve{N}_2(S)| < |\breve{N}_3(S)|$. $\breve{N}_1(S)$ is determined as the complete neighborhood $N_1(S)$ obtained by shifting all late jobs in S before the common due date (cf. Section 3.4), while $\breve{N}_2(S)$ and $\breve{N}_3(S)$ correspond to the

neighborhood $N_2(S)$ restricted to schedules obtained by interchanging only β_2 and β_3 percent of early jobs and late jobs, where $\beta_2 < \beta_3$. The values β_2 and β_3 are obviously control parameters of the VNS method.

At every iteration, VNS picks at random a solution S' from the neighborhood $\check{N}_k(S)$ generated from a current solution S. This selected schedule becomes the starting point for a local search method. The role of this local search procedure is played by the SA or the TS algorithm. That results in two versions of the variable neighborhood method: VNS–SA and VNS–TS, respectively. If a solution generated by the local search procedure, S'', improves the criterion value, then it is accepted and VNS returns to the first neighborhood definition. Otherwise, the local search is restarted from another neighbor solution generated according to the next neighborhood definition. The VNS algorithm finishes the solution space exploration after reaching the termination conditions mentioned in Section 3.1. Obviously, because the method calls SA or TS as a subprocedure, which performs a certain number of its own iterations, the number of VNS iterations without an improvement, as the termination condition, has to be set to a rather small value.

4. Computational experiments

Computational experiments performed within the reported research were devoted to comparing the efficiency of particular metaheuristic methods, as well as to evaluating the quality of their solutions with regard to optimal schedules. Moreover, the analysis of the test results made it possible to determine some specific features of the approaches proposed. The main computational experiments were obviously preceded by a careful tuning process (cf. e.g. [30,31]), during which we tested the efficiency of the metaheuristics for different values of control parameters in order to determine their best settings. The results of these preliminary experiments are summarized in Section 4.1. Then, in Section 4.2, the tuned metaheuristics are compared one to each other for large problem instances, as well as to the exact methods for small problem instances in Section 4.3.

4.1. Tuning of metaheuristic methods

In the process of tuning of all three metaheuristics, we used Johnson's algorithm to obtain the initial solution. Obviously, the list scheduling algorithm generated initial schedules of a higher quality than Johnson's one. Hence we used the list scheduling method in the main phase of the computational experiments, when the highest possible efficiency of the algorithms implemented was required. But during the tuning process, we wanted to check sensitivity of the methods to different settings of control parameters. Starting the search from a slightly worse solution (Johnson's one) we made some dependencies between control parameters values and the efficiency of the algorithms more visible. (The detailed results of computational experiments performed within tuning process can be found in [19].)

4.1.1. Tuning of simulated annealing

The main objective of the tuning process for SA, as well as for the remaining metaheuristics, was to determine the setting of the control parameters that ensures the highest method efficiency in terms of the solution quality. The preliminary tests for SA showed that an arithmetic cooling scheme is much less efficient than a geometric one. Therefore, the following experiments for SA were performed only for this latter (geometric) cooling rule.

At the first phase of the tuning process, we used a time limit equal to 1 or 3 s, respectively, as the termination condition (then the number of iterations without an improvement was taken into account). For both termination values, we used different values of the initial temperature T_0 (equal to the total processing time of the jobs multiplied by factor 10, 20, 50, 100) and a cooling factor α equal to 0.9, 0.95, 0.999. For each pair of these parameters, we tested two possible moves N_1 (shifting a late job early) and N_2 (interchanging an early job with a late one) with two selection rules applied within them (the random rule S_1 and the rule based on the weighted processing times of the jobs S_2).

Based on the solution quality improvement with regard to an initial schedule, we observed that the selection rule S_2 involving the weighted processing times of the jobs always dominated the random rule S_1 ensuring a larger improvement of the criterion value (9.72% vs. 3.33%). Moreover, the lower standard deviation value showed that the rule S_2 was more stable than S_1 (0.3% vs. 2.61%). Comparing the two types of moves N_1 and N_2, one could conclude that the move based on interchanging jobs (N_2) was slightly less efficient than the move shifting a late job early (N_1). However, the smaller average criterion improvement obtained for N_2 for all tests (6.26% vs. 6.79%) was caused by the very poor quality of schedules generated only with the random rule S_1. Taking both move components into account, we observed that solutions constructed by interchanging two jobs selected at random, $N_2 + S_1$, were much worse in terms of a solution

improvement (2.56% vs. 4.11%) than schedules obtained by $N_1 + S_1$, shifting a single activity selected at random early. On the contrary, for the weighted rule, $N_2 + S_2$ dominated $N_1 + S_2$, but the difference between them was very small (9.96% vs. 9.47%). The most important observation was that a current solution modification by interchanging jobs selected according to their weighted processing times ($N_2 + S_2$) always guaranteed the highest criterion value improvement for all possible settings of the remaining control parameter values.

The test results for the two termination conditions used in the experiments (i.e. 1- or 3-s time limit) showed rather weak influence of this termination value on the quality of the solutions (5.82% vs. 7.23%). For worse move + rule configurations, the higher time limit ensured a slightly better method performance of the method than the lower time limit. This analysis showed that the move type and especially the selection rule applied within the move influenced the behavior of SA most.

Looking at the time moment at which SA found the best solution, we saw that using the rule S_2, the method achieved the final solution faster than using S_1 (independently of the move type). As we could expect, the rule S_2 based on the instance description, i.e. on the weighted processing times of the jobs, appeared to be more efficient than the random rule S_1. It modified solutions in a systematic way leading the search to a local optimum more easily. In the experiments for S_2, the best solution was found quite early, after about 16% and 10%, respectively, of iterations, depending on the move type. It never happened that the method constructed the best schedule just before its termination (at most after about 67% of iterations). On the contrary, the random character of the rule S_1 spread the search within the solution space—the final schedule was generated after about 64% and 44%, respectively, of iterations, depending on the move type. Sometimes, the best solution was determined even in the last step of SA. On average, the SA method found the best schedule quite fast, after about 34% of iterations, because initial solutions were rather close to the optimum. For this reason, the metaheuristic fell into a local optimum quite easily.

The temperature analysis confirmed the conclusions already formulated. In 87.5% of tests, the method using rule S_2 stopped because of cooling the system down. In the remaining cases, the final temperature was nearly equal to zero. SA with S_2, independently of the move variant, constructed a good solution at early steps of the search, and it was usually terminated by the cooling mechanism, before exceeding the time limit. On the contrary, with the random selection rule S_1, the system was cooled down in only about 4% and 16%, respectively, of tests (depending on the move type). In the remaining tests, the method was stopped because of consuming all computation time allowed and the temperature decreased to about 30% of the initial value. Using the random rule, SA generated solutions of various quality and performed more time-consuming undirected search in the solution space, than applying the weighted processing time rule S_2. On average, for nearly 49% of the tests the SA finished because of cooling the system down (i.e. as we have mentioned, mostly for the rule S_2).

The earlier research on problem $F2|d_j = d|Y_w$ (cf. [18]) showed that heuristic solutions, based on the special features of an optimal solution, were very close to it in terms of the criterion value. This fact was confirmed by the quality of the solutions generated by SA in the tuning process. Most solutions constructed by SA at particular iterations (about 78%) were worse than the current schedule. An initial solution quality close to the optimum really made an improvement of the criterion value very difficult and rare. For the rule S_2, the initial schedule was improved at early iterations and a further decrease of the late work criterion was rather unlikely (about 95% of worse solutions). In the case of the random rule S_1, a current schedule was improved more often (in about 60% of iterations), because the method did not converge to a local optimum as fast as for S_2. It generated and accepted worse solutions and, then, improved them again, however, the final solution was usually worse than the one obtained with the S_2 rule.

The higher convergence of SA applying the rule S_2 instead of S_1 was additionally confirmed by the number of worse solutions which were accepted during the search. For rule S_2, this number was equal to only about 6% of worse solutions. Because SA found the best schedule relatively early, the system cooled down fast, and the low temperature prevented it from accepting bad schedules. In the case of the random rule S_1, solutions of poor quality appeared very often at the beginning of the search, when the temperature and the probability of their acceptance were high (about 76% and 68%, respectively, of acceptances, depending on the move type).

In consequence of the fast convergence to a local optimum observed for SA with the rule S_2, its average iteration time was much shorter than for the rule S_1. SA with S_2 generated a good solution at the beginning of the search. Because schedules constructed in the later steps were usually worse than the current one and the criterion value deterioration made their acceptance very unlikely, most of them were rejected. As we have already mentioned, for the random rule S_1, the search process was less stable and more time consuming. Hence, SA with S_2 required only about 50 μs per

iteration, while SA with S_1 consumed about 15 ms. The difference in the computational times between S_1 and S_2 results mainly from the implementation issues. In the implementation used, accepting a new solution (more often for S_1) is much more time consuming than rejecting it (more often for S_2). Moreover, selecting jobs according to the weighted processing time rule (S_2) is supported by some specialized fast data structures, which could not be applied for the random selection rule (S_1).

Taking into account the very vague differences among particular control parameter settings, we performed additional computational experiments for the best move configuration with another termination condition—a given number of iterations without an improvement. The average run time equal to 0.15 s showed that the 3-s time limit used in the previous stage of the experiments was sufficiently long. Moreover, in 69.8% of tests, the system stopped with zero temperature. This means that the new termination condition used was large enough to cool the system down. Changing the termination condition from the time limit to the number of iterations without an improvement caused that the number of iterations to the best solution increased from 17% to about 50% of the total number of iterations. In the case of a time limit, the total number of iterations was larger because the method continued the search without improving the solution, till consuming the whole computation time assigned to it.

Finally, based on a careful analysis of the results of the tuning process, we selected the following values of control parameters for the SA method:

- the solution modification based on the move interchanging jobs selected with the weighted processing time rule: $N_2 + S_2$;
- the termination condition set as the number of iterations without an improvement 5 times as large as the number of jobs;
- the initial temperature 20 times as large as the total processing time of the jobs;
- the geometrical cooling scheme with a cooling factor equal to 0.933.

4.1.2. Tuning of tabu search

The TS method is a trajectory method constructing a set of neighborhood solutions at each iteration instead of generating a single solution as the SA approach does. Similarly as for SA, we performed the tuning process using Johnson's solution as the initial one, with a run time limit set to 1 or 3 s and with different neighborhood definitions based on shifting a job (N_1) and an interchange of two jobs (N_2) with the random selection rule (S_1) or the rule based on the weighted processing times of the jobs (S_2). Moreover, during the tests either the whole neighborhood was investigated or the number of shifted/interchanged jobs was restricted to a certain percent of a total number of candidate jobs (β). Finally, various lengths of tabu list was investigated, determined as a percentage of the number of jobs in the system.

In the case of the TS method the dominance of the selection rule S_2 (based on the weighted processing times of the jobs) over the random rule S_1 was not so visible as for the SA algorithm. Nevertheless, the selection rule based on the instance description S_2 ensured a slightly higher solution quality than the random one (15.61% vs. 14.29%). Moreover, S_2 was much more stable than S_1 in terms of the standard deviation (0.59% vs. 2.02%). These results disclosed the differences between TS and SA. Since SA constructed a single solution at each iteration, the quality of the current solution influenced significantly the quality of the final one. The schedule modification based on the instance features (i.e. on the weighted processing times of the jobs) resulted in a better current solution and, consequently, in a better schedule being the result of the whole search process. In the case of TS, a whole set of neighborhood solutions was constructed at a single iteration, and a new current solution was selected from this set. Therefore, the procedure according to which jobs are chosen to build new schedules did not influence the quality of a new current solution as much as one could observe for the SA method. Taking into account the fact, that the initial solution of the problem under consideration was not far from the optimum, different move concepts resulted in similar neighborhoods. Thus, the efficiency of particular selection rules S_1 and S_2 applied within TS appeared to be nearly alike. Similar conclusions were drawn from the comparison of the two neighborhood concepts proposed. The move N_2 of interchanging jobs resulted in only slightly better schedules than the move N_1 of shifting a selected job early (15.39% vs. 14.51%).

As for the SA approach, the comparison of particular combinations of the move and the selection rule showed that interchanging jobs selected according to their weighted processing times ($N_2 + S_2$) always ensured the highest solution quality, independently on other control parameter settings of the algorithm, which was indicated by the zero standard deviation.

Furthermore, similarly as for SA, the influence of the time limit on the solution quality was not significant. TS with N_2 and S_2, which appeared to be the best neighborhood configuration, generated exactly the same results within the 1- and 3-s time limits. For worse control parameter settings, the longer run time resulted in a slightly higher improvement of the criterion value.

Comparing the percentage of the number of iterations till the moment when the best solution was determined, we noticed that for the neighborhood $N_2 + S_2$, a final solution was found very fast, i.e. at the beginning of the search. The move interchanging early and late jobs taking into account their weighted processing times constructed good solutions, which could be hardly improved in the following iterations. For the remaining move + rule configurations reaching the local optimum was more difficult and the best solutions were found in the later iterations of the search process.

The influence of the move type applied within the procedure constructing a neighborhood on the search process was better visible, when the number of solutions analyzed per single iteration was investigated. Interchanging jobs, independently of the rule selecting those jobs, lead to a larger number of distinct neighbor solutions than moving some late activities early. It is obvious that there were at least as many pairs of early and late jobs to be interchanged (N_2) as late jobs to be moved early (N_1).

In the additional computational experiments of the tuning process, the termination condition was determined by the number of iterations without an improvement. Since the average run time was equal to 0.199 s, we could state that the 3-s time limit used in the previous experiments was large enough similarly as for the SA algorithm.

Taking into account all computational experiments, we selected the following control parameter values as the result of the tuning process for the TS method:

- the neighborhood generated by interchanging jobs based on the weighted processing times of the jobs, $N_2 + S_2$;
- the neighborhood generated from 33% of candidate jobs;
- the termination condition set as the number of iterations without an improvement equal to the doubled number of jobs;
- the tabu list length equal to 300% of the number of jobs.

It is worth to be mentioned, that the rather large tabu list length chosen resulted from the specificity of the problem under consideration. Taking into account the fact that the initial solution was nearly optimal, we could not allow the method to return to a solution already visited, if we wanted to force it to get closer to the optimum.

4.1.3. Tuning of variable neighborhood search

In the preliminary experiments for the variable neighborhood method, we used the SA or the TS as a local search procedure, using the control parameter settings determined in the tuning process reported in Sections 4.1.1 and 4.1.2, respectively. Actually, there are only two parameters controlling the behavior of the VNS method implemented: the termination condition and the neighborhood configuration.

The termination condition decides how many times the main loop of the algorithm is performed and it does not influence the termination condition of an embedded local search procedure. Taking into account the good quality of the initial solution and the high efficiency of SA and TS used as a subroutine, enforcing long-lasting runs of VNS seems to be pointless. Therefore, during the tuning process, the VNS termination condition was settled to only one, two or three iterations without an improvement.

The VNS method is based on the same neighborhood definitions, which are applied within SA and TS. In order to lead the search from a narrower to a wider neighborhood, VNS starts with the neighborhood shifting a job (N_1) and, then, it passes to two neighborhoods interchanging jobs (N_2). The size of these neighborhoods is controlled by the size of the set of jobs being candidates for shifting/interchanging. For example, the first neighborhood (based on the move N_1) can be generated for all candidate jobs (100%), the second neighborhood (based on the move N_2) interchanges 20% of the number of candidate jobs, and the third neighborhood (based on the move N_2 too) interchanges 50% of the number of candidate jobs.

The tuning process showed that the VNS method was insensitive to the control parameter settings. VNS with SA as well as with TS found the best solution in its first iteration, that means after the first run of the embedded local search method. In consequence, particular sets of control parameters could not be distinguished from the efficiency point of view. The time efficiency obviously reflected the termination condition—the larger the number of iterations without an improvement, the longer the run time of the method. Taking into account the fact that the initial solution was close to the optimum as well as the efficiency of the TS and SA method, a further solution improvement by the VNS algorithm

was very difficult to be obtained. It was rather unlikely to found significantly better schedules by restarting SA or TS from different initial points in this case. Thus, in the main experiments, we decided to use the control parameter setting ensuring the highest degree of freedom for the VNS method, i.e.:

- the termination condition set to three iterations without an improvement,
- the neighborhood configuration with the candidate sets restricted to 100%, 50%, 100% of the number of candidate jobs.

4.2. Comparison of metaheuristic methods

In order to compare the efficiency of particular metaheuristics one to each other as well as to the list scheduling approach (LA), we analyzed 20 instances of the problem under consideration with the number of jobs equal to 20 and 200 (two instances for each number of jobs), as well as 50, 80, 110 and 140 (four instances for each number of jobs). For each test instance, particular metaheuristic methods were run four times: starting from Johnson's schedule or from the list schedule as the initial solution, and with a limited number of iterations without an improvement or with a time limit as the termination condition. The remaining control parameter settings were determined during the tuning process.

In general (cf. Table 1), the best average efficiency was achieved by SA and VNS–SA. TS and VNS–TS behaved quite well when the search started from a list schedule. LA confirmed its good performance observed within the former research on the flow-shop problem (cf. [18]). In order to compare the efficiency of the metaheuristic methods implemented more deeply, we restricted the analysis to only those instances for which a particular method did not find the best solution (cf. Table 2).

SA and VNS–SA were not the best metaheuristic only for one or two instances depending on the control parameter values, for these tests TS or VNS–TS were more efficient. However, in the cases mentioned, the quality of SA and VNS–SA solutions differed from the best result of only fractions of per mill. On the contrary, TS and VNS–TS were less efficient than SA and VNS–SA for about 10 instances, generating solutions of about 7‰ and 2.5‰ worse starting from Johnson's and the list schedule, respectively.

Taking into account the best results of particular metaheuristics among the four runs mentioned above, these methods were able to improve the list solution in 70% of the tests (14 instances among 20) with 2.82‰ on average. The smallest improvement was equal to 0.2‰, while the maximum one equals 6.8‰. Taking into account the fact that the list scheduling algorithm generated solutions of a very high quality, i.e. a nearly optimal one, the further improvement achieved by the metaheuristic methods should be regarded as a considerable one, especially, taking into account the rather simple structure of these methods and their short run time. The computational experiments showed that even a very good schedule can be still improved without a huge time effort, by performing a systematic search in the solution space.

Table 1
The distance to the best criterion value for particular instances (for 20 tests) for the list scheduling algorithm and the metaheuristics starting from Johnson's schedule (J) or a list schedule (L) with the number of iterations without an improvement and a 5-s time limit as the termination condition

	Distance to the best result in (‰)	LA	SA(J)	SA(L)	TS(J)	TS(L)	VNS–SA(J)	VNS–SA(L)	VNS–TS(J)	VNS–TS(L)
No. of iterations	Minimum	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Average	1.98	0.01	0.05	3.69	1.18	0.03	0.00	3.40	1.16
	Maximum	6.80	0.15	0.74	31.72	6.80	0.70	0.00	31.72	6.80
	Standard deviation	2.04	0.03	0.17	7.72	1.87	0.15	0.00	7.54	1.89
	No. of tests among 20 with the best result	6	19	18	10	10	19	20	10	11
5-s time limit	Minimum	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Average	1.98	0.00	0.00	3.67	1.21	0.00	0.00	3.34	1.16
	Maximum	6.80	0.00	0.00	31.72	6.80	0.00	0.00	27.59	6.80
	Standard deviation	2.04	0.00	0.00	7.73	1.86	0.00	0.00	6.73	1.89
	No. of tests among 20 with the best result	6	20	20	10	9	20	20	9	11

Table 2
The distance to the best criterion value for particular instances for the list scheduling algorithm and the metaheuristics starting from Johnson's schedule (J) or a list schedule (L) for two termination conditions restricted to those instances for which a particular method did not find the best solution

	Distance to the best result in (‰)	LA	SA(J)	SA(L)	TS(J)	TS(L)	VNS–SA(J)	VNS–SA(L)	VNS–TS(J)	VNS–TS(L)
No. of iterations	Minimum	0.20	0.15	0.20	0.15	0.20	0.70	0.00	0.38	0.20
	Average	2.82	0.15	0.47	7.37	2.37	0.70	0.00	6.81	2.59
	Maximum	6.80	0.15	0.74	31.72	6.80	0.70	0.00	31.72	6.80
	Standard deviation	1.89	0.00	0.27	9.59	2.05	0.00	0.00	0.95	0.21
	No. of tests among 20 without the best result	14	1	2	10	10	1	0	10	9
5-S time limit	Minimum	0.20	0.00	0.00	0.28	0.20	0.00	0.00	0.28	0.20
	Average	2.82	0.00	0.00	7.34	2.21	0.00	0.00	6.06	2.59
	Maximum	6.80	0.00	0.00	31.72	6.80	0.00	0.00	27.59	6.80
	Standard deviation	1.89	0.00	0.00	9.61	2.03	0.00	0.00	8.11	2.05
	No. of tests among 20 without the best result	14	0	0	10	11	0	0	11	9

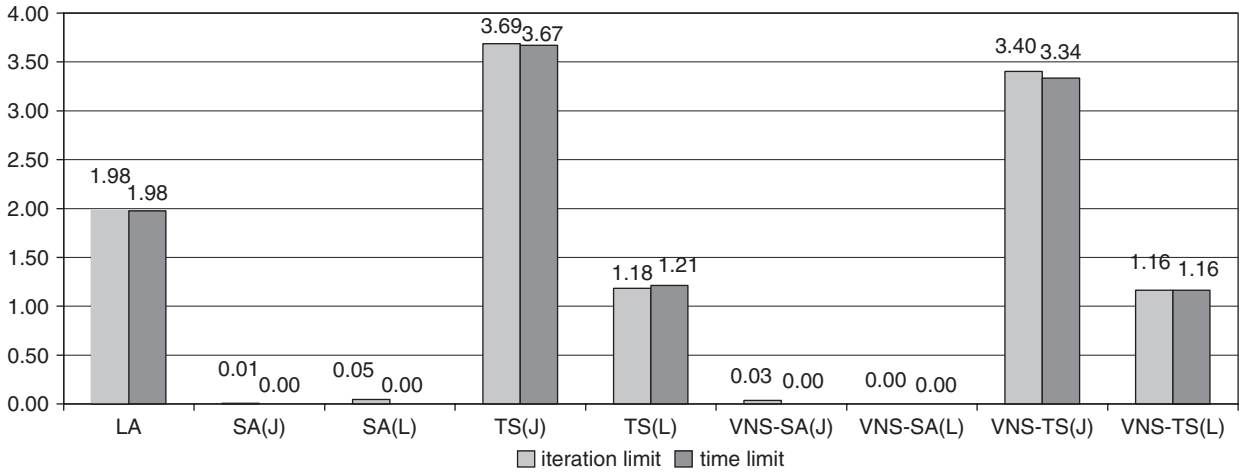

Fig. 3. The average distance to the best solution in (‰) for the list scheduling algorithm and particular metaheuristics (for 20 tests) starting from Johnson's schedule (J) or a list schedule (L) for the number of iterations without an improvement and a 5-s time limit as the termination condition.

The following detailed analysis of the computational experiments disclosed some interesting features of the metaheuristics implemented within the presented research.

As we have mentioned, comparing particular methods, one could notice the superiority of the SA one (cf. Table 1 and Fig. 3, Table 2 and Fig. 5). The SA algorithm constructed the best schedule for all instances analyzed, when the termination condition was set to a 5-s limit (cf. Table 1 and Fig. 4). In this case, restarting SA from different initial solutions by the variable neighborhood algorithm did not result in an improvement of the solution quality, because of the high quality of the schedule found in the first run of SA. When the number of iterations without an improvement was applied as the termination condition, SA found the best solution for only two instances less with a list schedule as the initial solution and for only one instance less than starting from Johnson's schedule. (In both cases SA still significantly dominated the TS approach in terms of the solution quality, cf. Figs. 3 and 5.) Since the SA algorithm was extremely fast, a 5-s time limit enforced this method to explore the solution space in time 2 and even 3 factors longer than the time resulting from the termination condition based on the number of iterations without an improvement (cf. an average run time in Table 7 with a 5-s time limit). The long run time limit allowed SA to leave the local optimum within the

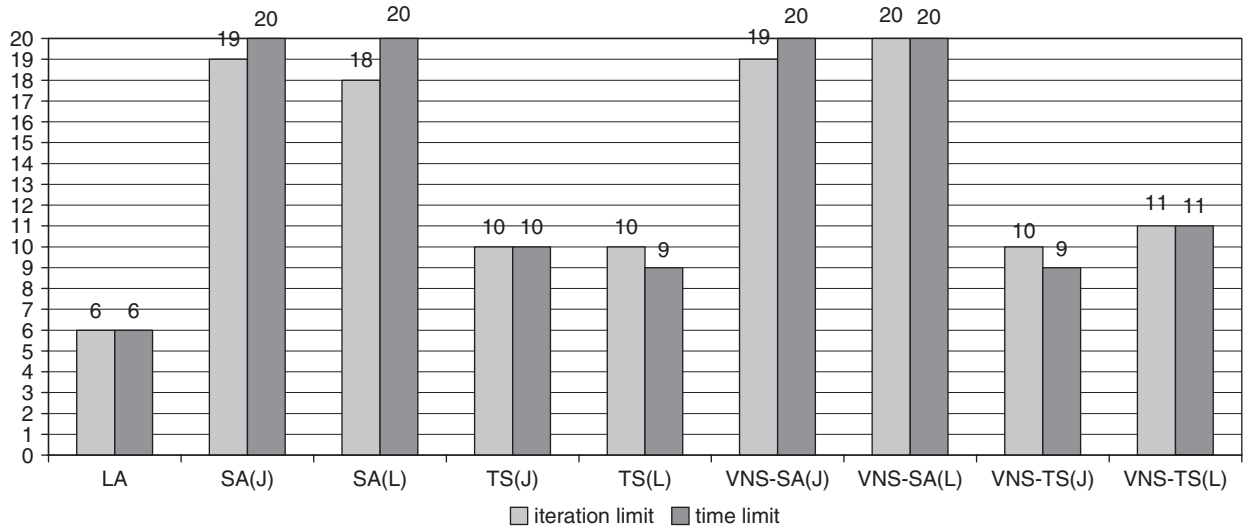

Fig. 4. The number of test instances in which particular methods constructed a solution with best criterion value (for 20 tests) starting from Johnson's schedule (J) or a list schedule (L) for the number of iterations without an improvement and a 5-s time limit as the termination condition.

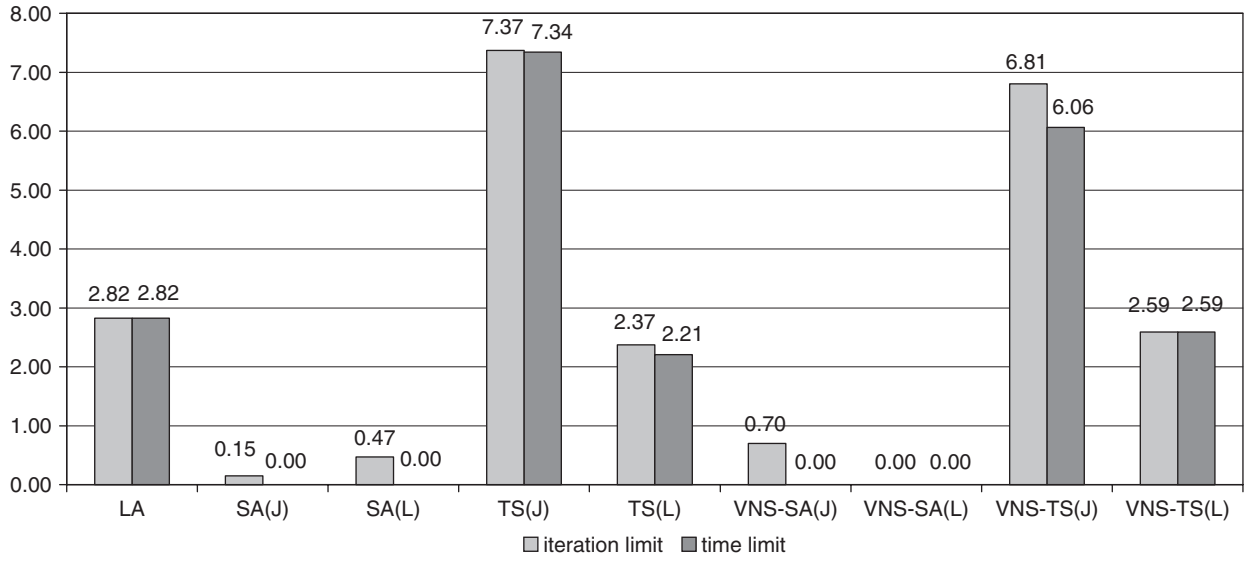

Fig. 5. The average distance to the best solution in (‰) for the list scheduling algorithm and particular metaheuristics starting from Johnson's schedule (J) or a list schedule (L) for two termination conditions restricted to those instances for which a particular method did not find the best solution.

number of iterations which would be not possible for the latter termination condition. These results showed that forcing a fast metaheuristic to a long-lasting search might cause an additional improvement of the criterion value. In general, a similar effect could be obtained by restarting a method from different initial solutions, as it was performed within the VNS algorithm. However, the computational experiments showed that the efficiency of the local search procedure was crucial for the efficiency of VNS for the problem under consideration. This method rarely improved the solution constructed by SA at the first iteration of VNS.

The TS algorithm constructed solutions worse than those found by SA (cf. Figs. 3 and 5). TS achieved the best criterion value for only 9 and 10, respectively, instances (depending on the control parameter values) improving a

list schedule for three and four instances from the test set (cf. Fig. 4). Embedding TS within VNS sometimes made it possible to obtain better results than in a single run of TS (for at most two instances more, when the method started from a list schedule and worked within a 5-s time limit). It is interesting, that for one test instance VNS–TS generated a schedule worse than TS. Within VNS, TS started its search from a solution taken at random from the neighborhood of the initial solution, which might be worse than this initial one. For this reason, a single run of TS starting from a good schedule, could result in a better solution than a few runs of TS within VNS initialized with a worse schedule.

Summing up, the VNS method with SA as a local search procedure and the list schedule as the initial solution (VNS–SA(L)) appeared to be the best choice for the problem under consideration. It constructed the best schedule for all test instances independently of the termination condition applied. The superiority of VNS–SA resulted from the high efficiency of the SA algorithm. It was possible to achieve the same solution quality level, by extending the run time limit for SA to 5 s or by restarting SA from different initial solutions within VNS. The TS approach appeared to be much less efficient, and introducing a time limit as the termination condition as well as multiple restarting TS within VNS did not result in a significant solution quality improvement. Taking into account the fact that the initial schedule was close to the optimum, searching the solution space by a single modification of a current schedule (as in SA) appeared to be a better approach than generating the whole neighborhood containing schedules mostly worse than the current one (as in TS). Moreover, the computational experiments showed that forcing a long run time limit, longer than the time determined by the tuned number of iterations without an improvement as the termination condition, one could additionally increase the quality of the final solution.

The test results showed that the efficiency of the method depended not only on the termination condition but also on the quality of an initial schedule (cf. Tables 3 and 4), although this influence is rather weak. In the experiments, two starting solutions were compared: Johnson's and list ones. Both schedules were constructed in neglectedly short time and allowed us to formulate some conclusions on the behavior of particular metaheurstics without using artificial random solutions, which are rather difficult to be justified from the viewpoint of scheduling theory.

All methods under consideration achieved a larger improvement of the criterion value starting from Johnson's schedule than from the list scheduling one (cf. Table 3). The difference between the quality of initial and final solutions for SA and VNS–SA was nearly 6 times, for TS and VNS–TS about 9.5 times larger, when these approaches startedfrom

Table 3
The average improvement of the criterion value from two different initial solutions Johnson's schedule (J) and a list schedule (L) to the final solution in (%)

Average improvement of the solution quality in (%)	For Johnson's schedule J	Standard deviation for J	For list schedule L	Standard deviation for L	Difference factor J/L
SA	1.13	1.05	0.19	0.21	5.86
TS	0.76	1.02	0.08	0.16	9.55
VNS–SA	1.14	1.05	0.19	0.20	5.82
VNS–TS	0.79	1.00	0.08	0.16	9.66

Table 4
The average difference of the criterion value of the final solution (%) between two different initial solutions Johnson's schedule and a list schedule, when Johnson's schedule (J/L) and a list schedule (L/J) resulted in a better final solution

Average difference in criterion value in (%)	Johnson's schedule dominating list one			List schedule dominating Johnson's one		
	No. of tests (%)	J/L	Standard dev. for J/L	No. of tests (%)	L/J	Standard dev. for L/J
J vs. L	10	0.1318	0.0114	80	1.1849	1.0028
SA	10	0.0396	0.0200	0	0	0
TS	15	0.0010	0.0004	40	0.3594	0.9410
VNS–SA	5	0.0007	0	0	0	0
VNS–TS	15	0.0013	0.0001	35	0.4099	0.9958

Johnson's instead of list schedules. But these factors reflect rather the difference in the quality of two initial schedules than in the performance of metaheuristics. In the most cases, particular methods constructed the same final schedule independently of an initial one (cf. Fig. 4). Since Johnson's schedule was usually worse than a list scheduling one, the improvement achieved for this latter one was smaller. In the case of TS and VNS–TS the influence of the quality of a initial solution was a bit more visible than for SA and VNS–SA. TS applied as a stand alone method or embedded within VNS, was not able to improve a good list schedule as much as SA. It achieved a considerable increase of the solution quality starting from worse Johnson's solution only. For this reason the difference between the average improvements of the criterion value for various initial sequences of jobs was larger for TS and TS–VNS than for SA and VNS–SA.

The observations provided above were confirmed by the analysis of the quality of particular schedules: initial and final ones (Table 4). For 80% of the test instances, a list schedule dominated Johnson's one by 1.2% on average. For only 10% of the tests, Johnson's solution was better than a list scheduling one by about 0.1%. In 90% of the tests, SA constructed a final schedule of exactly the same quality independently of the quality of the initial solution, while VNS–SA in 95% of the tests. In the remaining cases the difference in the performance measure was insignificant, equal to a few units per a few thousands of units of late work. These results showed that SA and VNS–SA were insensitive to the quality of an initial solution and their efficiency depended more on the search strategy, than on the starting point for a search process. On the contrary, the TS strategy was influenced by an initial solution in a more apparent way. The performance of TS was independent of a starting schedule in 65% of the tests, while VNS–TS in 50% of the tests. In the remaining cases, the poorer quality of the first solution resulted in a worse final one. Because the difference between initial schedules was bigger than between the final schedules corresponding to them (e.g. 1.2% between the list and Johnson's ones, and about 0.4% between the corresponding final schedules), this means that TS decreased the distance between the starting and final solutions nearly 3 times, but it was not able to achieve the same quality level for various starting points of the search process.

Besides the initial solution, the termination condition is another important control parameter influencing the efficiency of metaheuristics. Comparing the results obtained for different termination conditions, one could notice, that although a 5-s time limit allowed the methods to achieve better results in most cases, this improvement was not significant (i.e. equal to fractions of per mill). Actually, for the majority of tests, the metaheuristics generated a schedule of the same quality independently of the termination condition applied (cf. Table 5, Figs. 6 and 7).

For SA and VNS–SA, the time limit was always slightly more profitable than the number of iterations without an improvement limit, however such a difference between these termination conditions was detected for only one or two instances depending on the initial solution. Since SA was a very fast algorithm, the large run time limit sometimes allowed it to leave a local optimum and to find a better final schedule, which would not be possible within a certain number of iterations without an improvement as the termination condition. On the contrary, the TS method and the VNS with TS, whose iterations were much more time consuming, behaved similarly for both termination conditions—at least no correlation could be observed.

Table 5
The difference between the solution quality (‰) between experiments with a 5-s time limit and the number of iterations without an improvement as the termination condition for particular metaheuristics starting from Johnson's schedule (J) or a list schedule (L) and the number of the experiments (among 20) in which the run with a particular termination condition dominated the other one

	Difference in solutions quality in (‰)	SA(J)	SA(L)	TS(J)	TS(L)	VNS–SA(J)	VNS–SA(L)	VNS–TS(J)	VNS–TS(L)
Iteration limit dominating time limit	Minimum	–	–	0.149	–	–	–	0.753	–
	Average	–	–	0.214	0.557	–	–	1.467	–
	Maximum	–	–	0.279	–	–	–	2.179	–
	No. of tests	0	0	2	1	0	0	2	0
Time limit dominating iteration limit	Minimum	–	0.196	–	–	–	–	0.139	–
	Average	0.149	0.471	0.753	–	–	0.697	2.206	–
	Maximum	–	0.745	–	–	–	–	4.274	–
	No. of tests	1	2	1	0	0	1	2	0

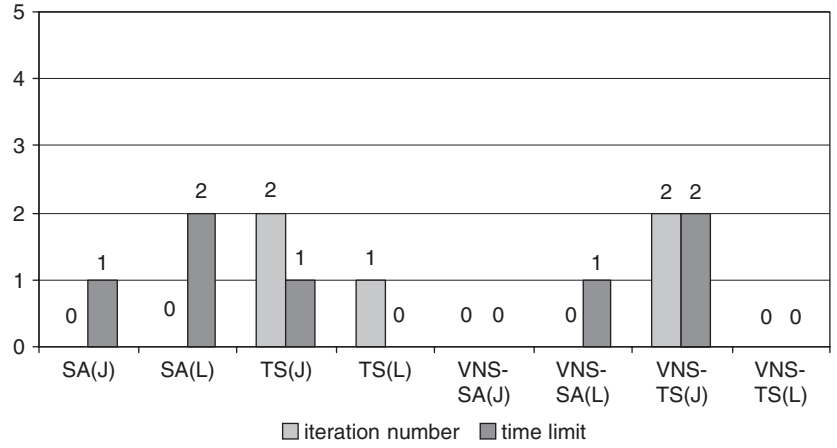

Fig. 6. The number of experiments (among 20) with a better result obtained for particular termination conditions, i.e. the number of iterations without an improvement and a 5-s time limit, for particular metaheuristics starting from Johnson's schedule (J) or a list schedule (L).

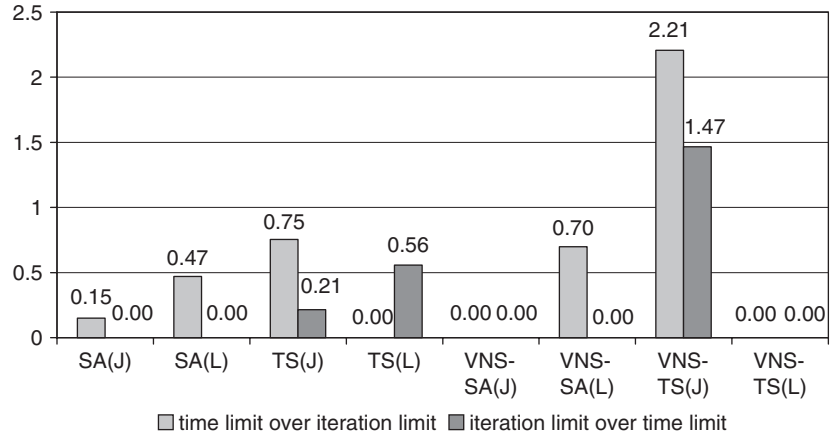

Fig. 7. The average difference in (‰) between the criterion values obtained for particular termination conditions, i.e. the number of iterations without an improvement and a 5-s time limit, for particular metaheuristics starting from Johnson's schedule (J) or a list schedule (L).

Summing up, the distance between the criterion values achieved for the two termination conditions investigated was very small. Taking into account only these computational experiments in which the results obtained for a 5-s time limit dominated the results achieved for the number of iterations without an improvement limit, the average difference in the criterion value over all metaheuristics was equal to 0.99‰. For the experiments in which the iteration limit was more profitable than the time limit, this average difference was a bit smaller and it equals 0.78‰. This fact can be explained by the tuning process performed for all metaheuristics investigated. It made it possible to determine a termination condition value (i.e. the number of iterations without an improvement) large enough to explore the solution space. Enforcing a longer run time could not considerably increase the efficiency of the methods (although it sometimes allowed the algorithms to leave a local optimum).

Obviously, the behavior of metaheuristics depends not only on the values of the control parameters but also on the input data. For problem $F2|d_j = d|Y_w$, the due date value is especially important for the efficiency of approaches solving it. When the due date value is small with regard to the schedule length, this means that only a few jobs can be processed early and they have to be carefully selected. Moreover, the differences between the criterion values for particular solutions might be quite large, because the set of early job contains only a few activities, whose processing times influence the performance measure. On the contrary, when a due date is large with regard to the schedule length,

Table 6
The average improvement of the criterion value for the data set with different due date values equal to 10%, 25% and 50% of the half of the total processing time of the jobs for particular metaheuristics starting from a list schedule and Johnson's schedule

Metaheuristics method	Due date	List schedule		Johnson's schedule	
		Average improvement (%)	Standard deviation (%)	Average improvement (%)	Standard deviation (%)
SA	10	1.71	2.12	24.73	4.68
	25	0.52	1.72	21.18	3.94
	50	0.19	0.20	1.13	1.05
TS	10	1.61	1.94	24.82	5.21
	25	0.79	1.84	21.37	4.05
	50	0.07	0.14	0.86	1.04
VNS–SA	10	1.73	2.12	24.89	4.60
	25	0.68	1.81	21.25	3.85
	50	0.20	0.21	1.13	1.05
VNS–TS	10	1.78	1.97	24.83	5.20
	25	0.82	1.90	21.44	4.01
	50	0.08	0.15	0.89	1.02

then almost all jobs are early and particular feasible schedules only slightly differ in the quality. To investigate the impact of the due date on the particular metaheuristics, computational experiments were performed with three sets of instances, differing in the value of the common due date. It was equal to 10%, 25% and 50% of the half of the total processing time of the jobs (Table 6).

The main conclusion was that the average improvement of the criterion value obtained by particular metaheuristics deteriorated with the increase of the value of the due date. When the due date was strict then an initial solution was usually more distant from an optimum. There were only a few early jobs and they had to be selected very carefully. Johnson's algorithm as well as the list scheduling procedure were not able to find such a good selection, while metaheuristics, changing the sequence of the jobs, could increase the quality of a solution considerably and quite easily. In the case of Johnson's algorithm the average improvement was significantly larger than for the list scheduling algorithm applied for constructing an initial solution. However, this large improvement resulted from a much worse quality of this initial schedule, not from a better quality of a final one, because as we observed before, all metaheuristics finished with nearly the same final solution, independently of an initial schedule. Johnson's procedure did not take into account weights of the jobs, which were especially important, when only a few jobs could be processed before the due date. Hence, the difference between the quality of initial Johnson's and list schedules was more apparent for instances with strict due dates.

With the increase of the due date value, instances became more difficult for metaheuristics, because there existed many subsets of jobs, which could be executed before the common due date and which should have been investigated in the search for an optimal solution. On the other hand, the quality of initial solutions increased, because for larger due dates, the set of early activities contained much more jobs and the differences between particular sets were not so large. In consequence, the average improvement decreased with the value of the due date (cf. Fig. 8 for a list schedule as an initial one). Obviously, if the due date was large enough, then all activities would be early and an initial solution would be optimal.

The computational experiments showed that for strict due dates the average improvement of the criterion value was larger, but all metaheuristics behaved similarly. Because the initial solution could be improved quite easily, the search strategy according to which a solution space was explored appeared to be not so important. The dominance of SA over TS (as a stand alone method as well as an embedded one within VNS) became more visible with the increase of the due date value. The set of instances with the due date equal to the half of the total processing time of the jobs, disclosed the differences among particular metaheuristics most. For this reason, computational results for this data set were mainly investigated in this paper.

The evaluation of the performance of any metaheuristics should be performed from two viewpoints, taking into account the quality of solutions as well as the computational time necessary to construct them. Obviously, in this case only to the computational experiments with the number of iterations without an improvement as the termination

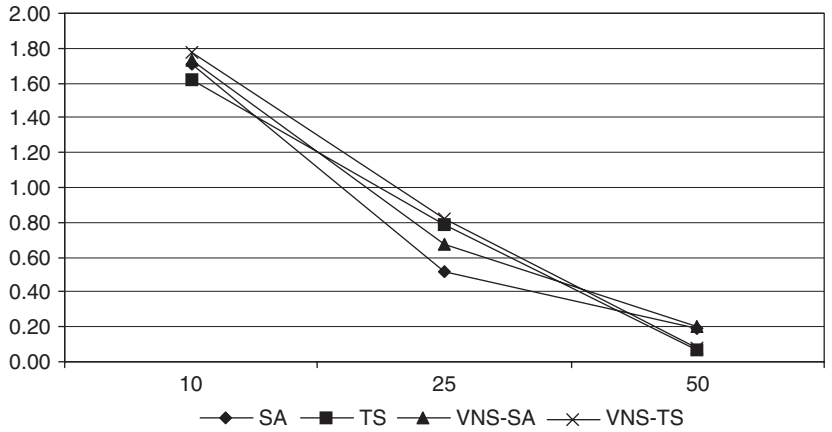

Fig. 8. The average criterion value improvement for the data set with different due dates values equal to 10%, 25% and 50% of the half of the total processing time of the jobs for particular metaheuristics starting from a list schedule.

Table 7
The average run time in (s) for particular numbers of jobs n with the standard deviation value

n	Average run time (s)				Standard deviation			
	SA	TS	VNS–SA	VNS–TS	SA	TS	VNS–SA	VNS–TS
20	0.003	0.003	0.006	0.011	0.001	0.002	0.002	0.002
50	0.011	0.041	0.036	0.205	0.003	0.023	0.008	0.078
80	0.019	0.089	0.062	0.572	0.002	0.007	0.008	0.103
110	0.057	0.557	0.147	3.178	0.018	0.339	0.030	2.217
140	0.034	0.222	0.110	1.176	0.003	0.005	0.014	0.152

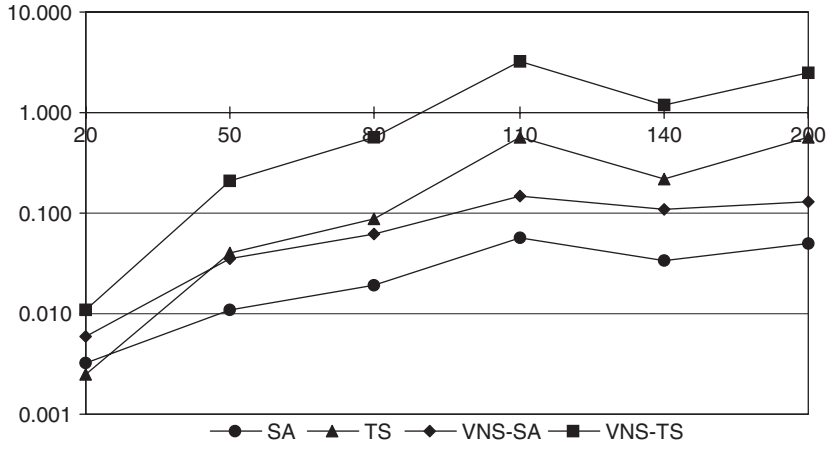

Fig. 9. The average run time in (s) for particular numbers of jobs with the logarithmic time axis.

condition are interesting. Moreover, the most difficult instances with a due date equal to 50% of the half of the total processing time of the jobs were taken into account. Table 7 and Figs. 9 and 10 present the results for particular numbers of jobs n as average values obtained for a few instances with the same value of n investigated twice with two different initial solutions.

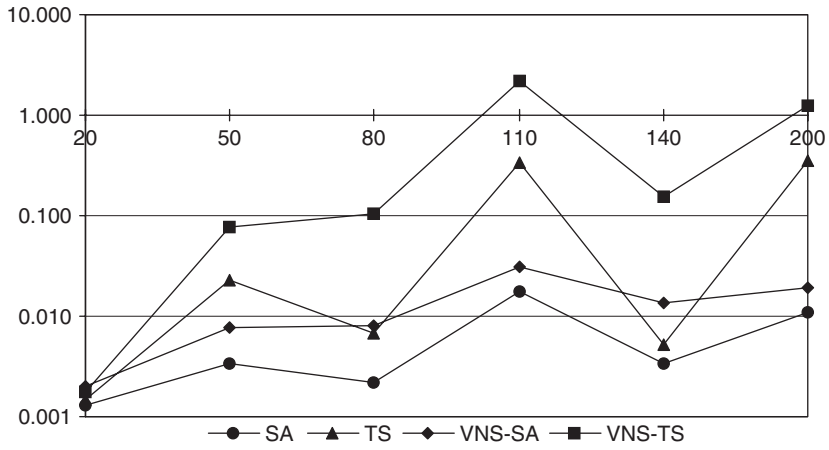

Fig. 10. The standard deviation for the run time of particular numbers of jobs with logarithmic y-axis.

Analyzing the test results, one could notice that the run time of all metaheuristics increased with the number of jobs (cf. Fig. 9), which was caused by the termination condition applied for SA and TS (also for SA and TS embedded in VNS). Both algorithms terminated after reaching the given number of iterations without an improvement, which was settled after the tuning process as the number of jobs multiplied by some number (cf. Sections 4.1.1 and 4.1.2).

Moreover, the computational experiments showed that the SA method was much more time efficient than the TS algorithm. A single modification of a current solution, applied within SA, was less time consuming than generating the whole neighborhood in TS (cf. Fig. 9). Moreover, the SA behavior appeared to be more stable and more independent of the problem data which was reflected in smaller standard deviation values than those achieved for TS (cf. Fig. 10).

Similar observations could be formulated for the VNS method with two different local search procedures SA and TS, i.e. for VNS–SA compared to VNS–TS. Obviously the VNS algorithm required more computational time than SA or TS applied as stand alone methods, because it repeated SA or TS a few times in order to restart the search from different initial solutions. According to the termination condition imposed (cf. Section 4.1.3), VNS stopped after performing three iterations without an improvement. This means that SA or TS were applied within VNS at least 3 times. Actually, on average, VNS–SA worked 3.08 times longer than SA, while VNS–TS run 5.79 times longer than TS. Because VNS–SA was usually not able to improve the solution found after the first SA run, it terminated after three iterations in total. In the case of VNS–TS, the schedule improvement happened more often than for VNS–SA and the total number of iterations was larger than the minimum possible one. Although VNS with TS performed more iterations during its search, this additional computational effort did not result in a significant solution quality improvement (cf. Tables 1 and 2).

Comparing the average run time for the whole test set containing 20 instances of a different size (cf. Table 8, Figs. 11–13), the superiority of SA became even more visible. TS overtook SA for only one (cf. Fig. 12) small problem instance (with 20 jobs), for which the computational time was almost immeasurable (cf. Fig. 11). For the remaining 19 instances, SA found better schedules requiring about 7 times shorter time than TS (cf. Fig. 13).

The repeated application of SA within VNS consumed about 3 times more run time, while VNS–TS worked 30–40 times longer than SA. As we have already mentioned, VNS–SA usually performed the minimal possible number of iterations, i.e. three iterations if no improvement in the SA solution was achieved. This means that VNS–SA run SA as a local search procedure 3 times on average. In the consequence, VNS–SA consumed a bit more than the tripled run time of SA applied as a stand alone approach. On the contrary VNS–TS sometimes improved the solution generated by TS and the total number of iterations was larger than the allowed number of iterations without an improvement (i.e. three iterations). Moreover, the TS approach was much more time consuming, more than 7 times, and less stable than SA (cf. Fig. 10). For this reason, the computational time for particular TS runs within VNS–TS could vary significantly.

Finally, based on the above analysis of the time requirements for particular metaheuristics (cf. Figs. 11 and 13), it was possible to observe a slight influence of the initial schedule on the duration of the search process. Starting the

Table 8
The average run time for the 20-instance test set in (s) and the average value of the run time divided by the shortest run time for a particular instance starting from Johnson's schedule (J) or a list schedule (L) and the number of iterations without an improvement as the termination condition

		SA(J)	SA(L)	TS(J)	TS(L)	VNS–SA(J)	VNS–SA(L)	VNS–TS(J)	VNS–TS(L)
Run time for 20 tests in (s)	Minimum	0.004	0.001	0.001	0.001	0.004	0.008	0.012	0.008
	Average	0.031	0.028	0.232	0.246	0.088	0.082	1.423	1.124
	Maximum	0.074	0.086	0.934	0.984	0.172	0.199	7.180	5.566
	Stand. dev.	0.020	0.021	0.270	0.310	0.050	0.051	1.786	1.326
Run time divided by the shortest run time	Minimum	1.11	1.15	3.38	2.88	2.16	2.48	12.00	8.00
	Average	1.85	1.88	7.21	7.18	3.65	3.66	40.04	32.40
	Maximum	4.00	4.00	26.69	26.00	6.38	8.00	118.31	88.83
	Stand. dev.	1.04	1.22	5.10	5.37	0.84	1.52	27.03	18.97
	No. of tests among 20 with the shortest run time	9	16	1	1	0	0	0	0

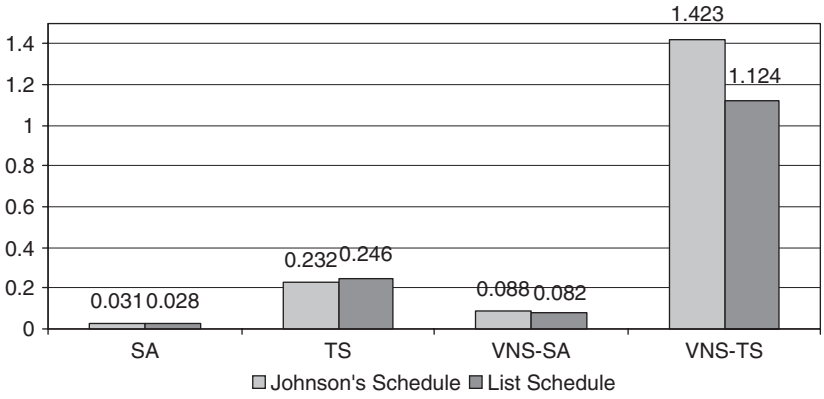

Fig. 11. The average run time for the 20-instance test set in (s) for the two different initial solutions and the number of iterations without an improvement as the termination condition.

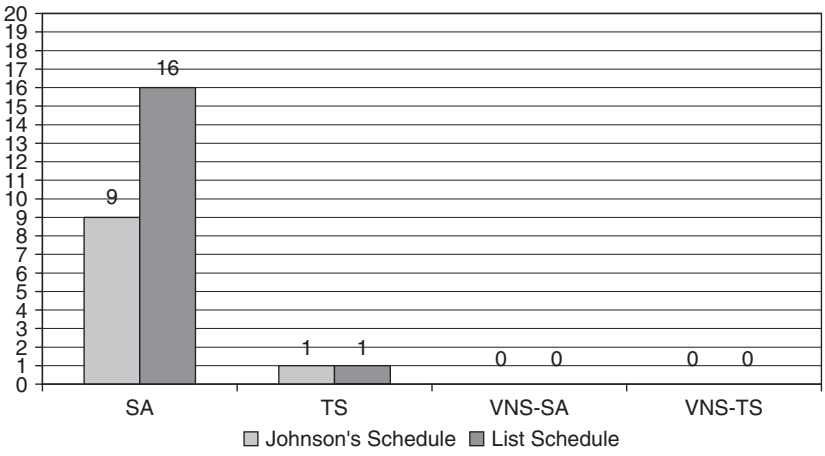

Fig. 12. The number of test instances in which a particular method constructed a solution in shortest time (for 20 experiments) for two different initial solutions and the number of iterations without an improvement as the termination condition.

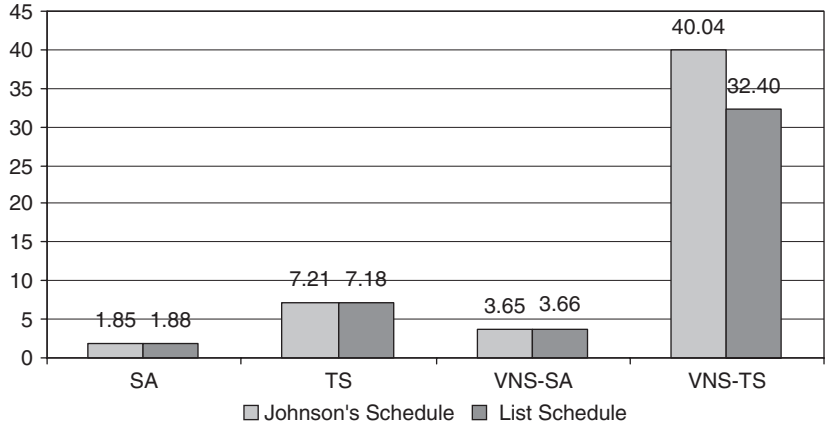

Fig. 13. The average run time difference for the 20-instance test set calculated as the quotient of the run time of the method and the shortest run time for particular instances for the two different initial solutions and the number of iterations without an improvement as the termination condition.

exploration of the solution space from the list schedule, the metaheuristics terminated faster than starting their search from Johnson's schedule. The initial schedule of a better quality (the list schedule) made the solution improvement more difficult and caused that the given limit of iterations without an improvement was exceeded quite early.

4.3. Comparison of heuristic methods to an exact approach

The second stage of the computational experiments was devoted to the comparison of the heuristic approaches with an exact enumerative method. Earlier research results obtained for problem $F2|d_j = d|Y_w$ (cf. [18]) showed that the enumerative method (EM) was more time efficient than a pseudo-polynomial time DP one. EM investigates in $O(n2^n)$ time all possible solutions for the problem under consideration. It checks all possible subsets of early jobs (E) and executes them in Johnson's order. Then, EM considers each job among the remaining ones as the first late job J_x and completes a partial schedule with the remaining jobs from $J\backslash(E \cup \{J_x\})$ sequenced according to non-increasing weights. The outline of the presented enumerative method is given below.

for each set $E \subseteq J$ such that a partial schedule obtained by sequencing jobs from E in Johnson's order does not exceed d and Johnson's schedule for $E \cup \{J_x\}$ where $J_x \in J\backslash E$ exceeds d do
  for each job $J_x \in J\backslash E$ do
    construct a schedule by executing jobs from E in Johnson's order, followed by J_x and jobs from $J\backslash(E \cup \{J_x\})$ sequenced according to non-increasing weights;
  store the best solution constructed for set E, if it is better than the best already found.

The exact approach was tested against four metaheuristics: SA, TS, VNS–SA and VNS–TS as well as two other simple heuristics: the list scheduling algorithm with the maximum weight priority dispatching rule (LA) and Johnson's procedure used as an approximate method for $F2|d_j = d|Y_w$ (JA). Taking into account an exponential time complexity of the enumerative method, 16 small problem instances were used with the number of jobs equal from 5 to 20 (with a unit increment).

The average quality of particular heuristic solutions is presented in Table 9. Johnson's algorithm appeared to be the weakest method generating schedules of nearly 11% worse than the optimum. On the other hand, taking into account its neglectedly short run time, the simplicity of this procedure and the fact that it was designed for a different scheduling problem ($F2\|C_{\max}$), the results of JA might be quite satisfying for certain applications.

The list scheduling algorithm was able to construct an optimal solution for only six instances among 16 test problems (cf. Fig. 14). Moreover, it found an optimum mostly for small instances (with 6–9 jobs). The SA and TS methods reached the optimal criterion value for 10 instances. Applying these procedures within the VNS framework increased the number of optima found to be 11 and 12, respectively. In general, the best metaheuristic solution had the same criterion value as the optimal solution constructed by the enumerative algorithm for 13 tests among 16.

Table 9
The distance to the optimal criterion value for particular methods and for the best metaheuristic in a certain test (BMH) in (%) for all tests and for those tests for which a particular method did not find the optimum

		JA	LA	SA	TS	VNS–SA	VNS–TS	BMH
All tests	Minimum	1.26	0.00	0.00	0.00	0.00	0.00	0.00
	Average	10.70	0.81	0.40	0.40	0.27	0.25	0.04
	Maximum	21.15	2.46	2.26	2.46	2.25	2.46	0.20
	Standard dev.	6.45	1.21	0.61	1.25	0.56	0.62	0.07
	No. of optimal solutions in 16 tests	0	6	10	10	11	12	13
Non-optimal results	Minimum	1.26	0.59	0.39	0.18	0.20	0.18	0.18
	Average	10.70	1.35	1.00	1.21	0.81	0.95	0.19
	Maximum	21.15	2.46	2.26	2.46	2.25	2.46	0.20
	Standard dev.	6.45	1.14	0.61	1.50	0.74	0.93	0.01

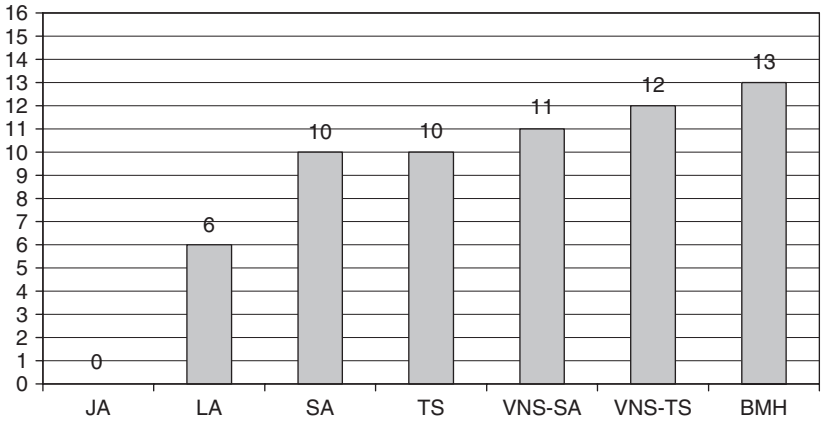

Fig. 14. The number of optima found in 16 tests for particular methods and the best metaheuristic result (BMH).

The specificity of the problem under consideration, $F2|d_j = d|Y_w$, made the search process difficult. Taking into account the fact that all early jobs have to be scheduled in Johnson's order, the crucial decision is to select activities performed before the common due date d. At this stage of the computational experiments, the initial solution for all metaheuristic methods was generated by the list scheduling algorithm, which selected early jobs according to their weights. When particular jobs differ only slightly in their processing times and weights, a further improvement of the criterion value by the metaheuristics is difficult, since an optimal solution is a specific sequence of jobs, which does not differ to much from the initial one. From this point of view, the efficiency of the metaheuristic methods proposed was quite satisfying.

Actually, these observations were confirmed by the analysis of the average distance to the optimal criterion value (cf. Fig. 15) calculated for all tests and, especially, for only these instances for which particular methods did not reach the optimum.

The best metaheuristic solution differed from the optimum by only 0.19%, while for the list scheduling one this value was equal to 1.35%.

Among metaheuristics, the highest performance in terms of the schedule quality was observed for VNS–SA, for which the distance to the optimum was equal to 0.81%. As we noticed in Section 4.2, the repeated use of SA within VNS made it possible to improve slightly the quality of the schedules constructed by the SA approach applied as a stand alone method. The distance to the optimum for SA was equal to only 1%.

Similarly as in the computational experiments with large instances reported in Section 4.2, for small problem instances the TS algorithm appeared to be less efficient than SA. However, the difference was not so significant (about 0.21%).

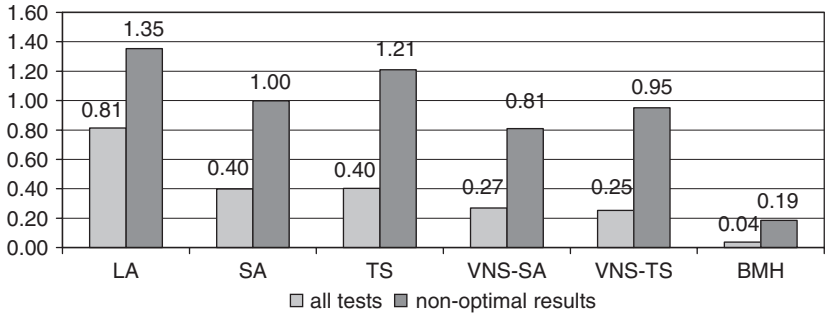

Fig. 15. The distance to the optimal criterion value for particular methods and the best metaheuristic in a certain test (BMH) in (%) for all tests and for those tests for which a particular method did not find the optimum.

Table 10
Run time for particular methods in (ms) (results in italic denote that the run time was immeasurably short)

Number of jobs	EM	SA	TS	VNS–SA	VNS–TS
5	0.015	0.001	*0.001*	*0.001*	*0.001*
6	*0.001*	*0.001*	*0.001*	0.001	0.001
7	*0.001*	0.001	*0.001*	0.001	0.001
8	*0.001*	*0.001*	0.001	0.001	0.001
9	*0.001*	0.001	0.001	0.001	*0.001*
10	*0.001*	0.001	0.001	0.002	0.001
11	0.015	0.001	*0.001*	0.002	0.002
12	0.031	0.001	0.001	0.001	0.003
13	0.187	0.001	0.001	0.002	0.004
14	0.218	0.001	0.001	0.001	0.004
15	0.328	*0.001*	0.001	0.003	0.011
16	0.89	0.001	0.003	0.004	0.012
17	1.359	0.001	0.002	0.003	0.007
18	8.843	0.001	0.003	0.004	0.01
19	52.375	0.002	0.001	0.006	0.01
20	71.281	0.001	0.004	0.003	0.021

VNS–TS behaved for small numbers of jobs much better than for the job sets of larger cardinality. The quality of VNS–TS was comparable to VNS–SA and SA (cf. Fig. 15). When the number of jobs was small, the size of the neighborhood for a current solution was also small and TS, applied as a stand alone algorithm or as a local search procedure within VNS, worked more efficiently. Moreover, the differences between particular heuristics became less visible than in the experiments with instances of large size.

The run time comparison (cf. Table 10 and Fig. 16) confirmed the observations formulated above on the efficiency of particular metaheuristics, but first of all it underlined the necessity of applying this kind of search procedures for hard problems. The enumerative approach ensured the optimality of the solutions constructed, but it required a huge computational effort, increasing rapidly with the problem size. On the contrary, the run time of metaheuristics was low and it increased very slowly with the number of jobs. In the computational experiments reported, the exponential explosion for the exact approach was observed already for 18 jobs.

The efficiency of particular solution methods analyzed within this research depends on the input data used during the computational experiments. This influence is usually more visible in the case of metaheuristics than for the exact approaches, because the tuning process adjusts the values of their control parameters to the specificity of the test instances. The research presented concerned a general scheduling model, which can be considered only as an approximation of a real world environment. However, in the filed of combinatorial optimization, artificial data sets often appear to be more difficult than the real ones. Nevertheless, every practical case has to be considered separately, taking into account its specificity, additional constraints and parameters. Thus, the results obtained for such general theoretical models

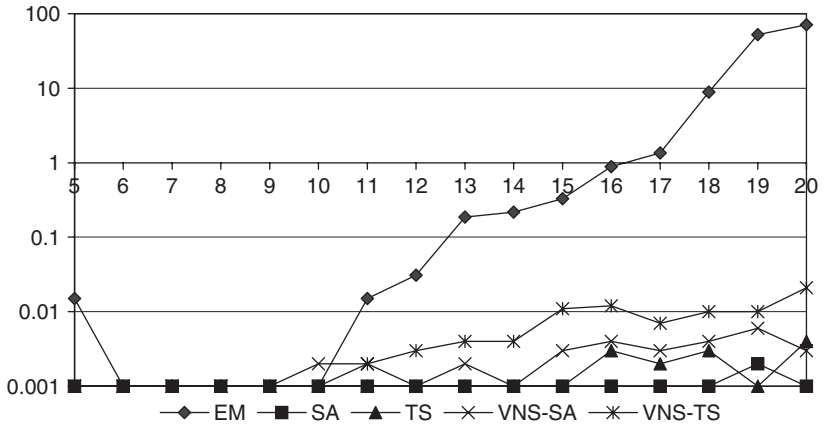

Fig. 16. Run time in (ms) for particular methods.

(as for the flow shop one) cannot be directly applied to real problems, but they provide a valuable background for solving them efficiently.

Actually, the results obtained for the problem $F2|d_j = d|Y_w$ are interesting not only from the viewpoint of scheduling theory but also of combinatorial optimization. The most important results, as far as scheduling theory is taken into account, include an NP-hardness proof, a pseudo-polynomial time algorithm and special features of an optimal sequence of jobs. Unfortunately, a very strict structure of an optimal schedule for $F2|d_j = d|Y_w$ does not leave too much freedom in designing solution methods. Nevertheless, the computational experiments showed that metaheuristic methods can still improve extremely good list scheduling solutions. Besides evaluating the usefulness of particular solution methods, the test results allowed us to draw interesting conclusions on these approaches from the viewpoint of combinatorial optimization. In the case of the problem under consideration, for which an optimal solution has a strict structure and suboptimal ones can be easily constructed, the random character of SA allowed this method to dominate TS, which performs a more systematic search. Moreover, multiple restarting of a local search procedure within VNS appeared to be less important than the efficiency of this local search procedure itself.

5. Conclusions

The research presented completes the studies on the two-machine flow-shop problem with the common due date and the weighted late work criterion, $F2|d_j = d|Y_w$. The theoretical investigation resulted in the NP-completeness proof for its decision counterpart and the proposal of a pseudo-polynomial time dynamic programming approach that allowed us to classify the case as NP-hard in the ordinary sense [17]. The first stage of computational experiments made it possible to verify the applicability of the DP procedure in practice and showed a high efficiency of the list scheduling algorithm [18]. At the second stage of computational studies reported in this paper, more advanced solution techniques were proposed: simulated annealing, tabu search and variable neighborhood search methods, based on the specific structure of an optimal solution for the problem under consideration. These trajectory metaheuristics were compared in the computational experiments one to each other as well as to the list scheduling approach, Johnson's method applied as a heuristic for $F2|d_j = d|Y_w$ and to an exact enumerative method.

The main experiments were preceded by an extensive tuning process in order to determine the best control parameter settings for particular metaheuristic methods. The difference in the performance for a single problem instance and different control parameter settings reached almost 10% for the simulated annealing and 7% for the tabu search approach which confirms the importance of the tuning process. Moreover, these preliminary experimental results made it possible to formulate interesting observations on the SA, TS and VNS–SA as well as VNS–TS behavior.

Then, we compared the metaheuristic approaches to the list scheduling algorithm for large instances in terms of the number of jobs. Despite the high quality of the list solution, the metaheuristic methods were still able to improve its quality. Furthermore, the test results showed that simulated annealing significantly dominates the tabu search strategy for the scheduling case under consideration. SA generated better schedules in shorter time than TS. The fast SA method

moving from one solution to another as a result of a single solution modification was more efficient than TS generating the whole neighborhood in order to select the next schedule for the analysis. Moreover, restarting SA within the variable neighborhood framework made an additional solution improvement possible. A similar effect was observed for TS and VNS–TS, although it was less visible. In the computational experiments with small instances in terms of the number of jobs, the differences among particular metaheuristics became less apparent, since all trajectory methods proposed generated optimal solutions for most test sets. Their time requirements were incomparably small with regard to the enumerative algorithm of exponential time complexity.

The experience gained during the research on problem $F2|d_j = d|Y_w$ provided many useful hints for future work on other scheduling problems with the late work performance measure.

Acknowledgments

We would like to thank Michal Glowinski for his effort in implementing and testing the approaches investigated within the presented research. The fourth author has been supported by INTAS (project 03-51-5501).

References

[1] Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J. Scheduling computer and manufacturing processes. 2nd ed., Berlin, Heidelberg, New York: Springer; 2001.
[2] Brucker P. Scheduling algorithms. 2nd ed., Berlin, Heidelberg, New York: Springer; 1998.
[3] Chen B, Potts CN, Woeginger GJ. A review of machine scheduling. In: Du D-Z, Pardalos PM, editors. Handbook of combinatorial optimization. Boston: Kluwer Academic Publishers; 1998.
[4] Pinedo M, Chao X. Operation scheduling with applications in manufacturing and services. Boston: Irwin/McGraw-Hill; 1999.
[5] Blazewicz J. Scheduling preemptible tasks on parallel processors with information loss. Technique et Science Informatiques 1984;3(6): 415–20.
[6] Blazewicz J, Finke G. Minimizing mean weighted execution time loss on identical and uniform processors. Information Processing Letters 1987;24:259–63.
[7] Hariri AMA, Potts CN, Van Wassenhove LN. Single machine scheduling to minimize total late work. INFORMS Journal on Computing 1995;7:232–42.
[8] Hochbaum DS, Shamir R. Minimizing the number of tardy job unit under release time constraints. Discrete Applied Mathematics 1990;28: 45–57.
[9] Kethley RB, Alidaee B. Single machine scheduling to minimize total late work: a comparison of scheduling rules and search algorithms. Computers and Industrial Engineering 2002;43:509–28.
[10] Kovalyov MY, Potts CN, Van Wassenhove LN. A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work. Mathematics of Operations Research 1994;19(1):86–93.
[11] Leung JY-T, Yu VKM, Wei W-D. Minimizing the weighted number of tardy task units. Discrete Applied Mathematics 1994;51:307–16.
[12] Potts CN, Van Wassenhove LN. Single machine scheduling to minimize total late work. Operations Research 1991;40(3):586–95.
[13] Potts CN, Van Wassenhove LN. Approximation algorithms for scheduling a single machine to minimize total late work. Operations Research Letters 1991;11:261–6.
[14] Blazewicz J, Pesch E, Sterna M, Werner F. Total late work criteria for shop scheduling problems. In: Inderfurth K, Schwoediauer G, Domschke W, Juhnke F, Kleinschmidt P, Waescher G, editors. Operations Research Proceedings 1999. Berlin: Springer; 2000. p. 354–9.
[15] Blazewicz J, Pesch E, Sterna M, Werner F. Revenue management in a job-shop: a dynamic programming approach. Preprint Nr. 40/03, FMA, Otto-von-Guericke-University Magdeburg; 2003.
[16] Blazewicz J, Pesch E, Sterna M, Werner F. Open shop scheduling problems with late work criteria. Discrete Applied Mathematics 2004;134: 1–24.
[17] Blazewicz J, Pesch E, Sterna M, Werner F. The two-machine flow-shop problem with weighted late work criterion and common due date. European Journal of Operational Research 2005;165(2):408–15.
[18] Blazewicz J, Pesch E, Sterna M, Werner F. A comparison of solution procedures for two-machine flow shop scheduling with late work criterion. Computers and Industrial Engineering 2005;49(4):611–24.
[19] Blazewicz J, Pesch E, Sterna M, Werner F. Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date. Preprint Nr. 32, FMA, Otto-von-Guericke-University Magdeburg; 2005.
[20] Leung JY-T. Minimizing total weighted error for imprecise computation tasks and related problems. In: Leung JY-T, editor. Handbook of scheduling: algorithms, models, and performance analysis. Boca Raton, FL: CRC Press; 2004. p. 1–16.
[21] Sterna M. Problems and algorithms in non-classical shop scheduling. Poznan: Scientific Publishers of the Polish Academy of Sciences, Poland; 2000.
[22] Garey MR, Johnson DS. Computers and intractability. San Francisco: W.H. Freeman and Co.; 1979.
[23] Blum Ch, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Computing Surveys 2003;35(3): 268–308.
[24] Crama Y, Kolen A, Pesch E. Local search in combinatorial optimization. Lecture Notes in Computer Science 1995;931:157–74.

[25] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1:61–8.
[26] Haupt R. A survey of priority rule—based scheduling. OR Sektrum 1989;11:3–16.
[27] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80.
[28] Glover F, Laguna M. Tabu search. Boston: Kluwer Academic Publishers; 1997.
[29] Hansen P, Mladenović N. An introduction to variable neighbour search. In: Voss S, Martello S, Osman I, Roucairol C, editors. Metaheuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer Academic Publishers; 1999. p. 433–58.
[30] Barr RS, Golden BL, Kelly JP, Resende MGC, Stewart Jr WR. Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics 1995;1:9–32.
[31] Hooker JN. Testing heuristics: we have it all wrong. Journal of Heuristics 1995;1:33–42.