A Comparison of Solution Procedures

for

the Flow Shop Scheduling Problem with Late Work Criterion

Jacek Błażewicz *)

Erwin Pesch 1)

Małgorzata Sterna 2)

Frank Werner 3)

*) Institute of Computing Science, Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland
phone: +48 (61) 8790 790
fax: +48 (61) 8771 525
blazewic@sol.put.poznan.pl

1) Institute of Information Systems, FB 5 - Faculty of Economics, University of Siegen
Hölderlinstrasse 3, 57068 Siegen, Germany
pesch@fb5.uni-siegen.de

2) Institute of Computing Science, Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland
Malgorzata.Sterna@cs.put.poznan.pl

3) Faculty of Mathematics, Otto-von-Guericke-University
PSF 4120, 39016 Magdeburg, Germany
Frank.Werner@mathematik.uni-magdeburg.de

A Comparison of Solution Procedures for the Flow Shop Scheduling Problem with Late Work Criterion

Abstract

In this paper, we analyze different solution procedures for the two-machine flow shop scheduling problem with a common due date and the weighted late work criterion, i.e. for problem $F2 \mid d_j = d \mid Y_w$, which is known to be binary NP-hard. In computational experiments, we compare the practical efficiency of a dynamic programming approach, an enumerative method and a heuristic list scheduling procedure. Test results show that each solution method has its advantages and none of them can be rejected from the consideration a priori.

Keywords: flow shop, late work, dynamic programming, enumerative method, list scheduling

1. Introduction

The process of investigating every new optimization problem follows the same well known scheme (cf. e.g. (Błażewicz, Ecker, Pesch, Schmidt & Węglarz, 2001), (Brucker, 1998)). If researchers have failed in constructing an optimal polynomial-time method solving the problem under consideration, they attempt to prove the hardness of the problem. For NP-hard problems, exact but exponential time methods can be proposed (including pseudo-polynomial time approaches for binary NP-hard problems) or efficient but approximate ones. We have followed the same route in our research on the two-machine flow shop problem with a common due date and the weighted late work criterion, $F2 \mid d_j = d \mid Y_w$, which appeared to be binary NP-hard (Błażewicz, Pesch, Sterna & Werner, 2004b).

In the problem considered in this paper, we have to find an optimal schedule for a set of jobs $J = \{J_1, \ldots, J_j, \ldots, J_n\}$ on two dedicated machines M_1, M_2. Each job $J_j \in J$ consists of two tasks T_{1j} and T_{2j}, executed on machines M_1, M_2 for p_{1j}, p_{2j} time units, respectively. Moreover, to each job J_j a weight w_j is assigned. Each job has to be performed, without preemptions, first on machine M_1 and then on M_2. Furthermore, each job can be processed on at most one machine at the same time and each machine can perform at most one task at the same time. Solving the problem we are looking for a schedule minimizing the total weighted late work in the system (Y_w). The late work (Y_j) for job $J_j \in J$ is determined as the sum of late parts of its tasks, i.e. the task parts executed after the common due date d. Denoting with C_{ij} the completion time of task T_{ij}, Y_j equals to $\sum_{i=1,2} \min\{\max\{0, C_{ij} - d\}, p_{ij}\}$. The criterion value is determined as $Y_w = \sum_{j=1}^{n} w_j Y_j$ (cf. Figure 1).

Figure 1. An illustrative instance of problem $F2 \mid d_j = d \mid Y_w$

The late work objective function is a due date involving criterion, which was proposed in the context of parallel machines ((Błażewicz,1984), (Błażewicz & Finke, 1987)) and then applied to the one-machine scheduling problem (Potts & Van Wassenhove, 1991a, 1991b). Recently, this performance measure has been analyzed in the shop environment ((Błażewicz, Pesch, Sterna & Werner, 2000, 2003, 2004a, 2004b), (Sterna, 2000)). Moreover, the late work criterion can be also considered as a special case of the imprecise computation model (cf. e.g. (Ho, Leung & Wei, 1994), (Shih, Liu & Chung, 1991)) applied in real time scheduling. Minimizing the amount of late work in a system finds many practical applications, e.g. in data collecting in control systems (Błażewicz,1984), agriculture technologies, e.g. when perishable goods are considered (Potts & Van Wassenhove, 1991), or designing production plans within predefined time periods in flexible manufacturing systems (Sterna, 2000).

Within the earlier research (Błażewicz et al., 2004b), binary NP-hardness of problem $F2 \mid d_j = d \mid Y_w$ has been proved by presenting a transformation from the partition problem (Garey & Johnson, 1979) and proposing an extensive pseudo-polynomial time dynamic programming method (DP) of complexity $O(n^2d^4)$. Usually, pseudopolynomial time approaches are considered only theoretically. We have decided to implement the dynamic programming method in order to validate the correctness proof by computational experiments results and, first of all, to compare in practice some basic approaches to solve hard scheduling problems. A dynamic programming method with a pseudo-polynomial time complexity is intuitively considered as more efficient than other enumerative methods. The computational analysis, presented in this paper, shows that even a simple enumerative approach cannot be a priori rejected form the consideration. On the other hand, a heuristic procedure based on a list scheduling framework appeared to be a really effective and efficient approximate

method. This frequently used scheduling technique (e.g. (Błażewicz et al., 2001), (Brucker, 1998), (Haupt, 1989)) confirms once more its advantages.

The paper is organized as follows. In Section 2, we sketch a dynamic programming method, while in Sections 3 and 4, an enumerative approach and a heuristic procedure are presented, respectively. Computational experiments results are provided and discussed in Section 5. The paper is concluded in Section 6.

2. Dynamic Programming Method

Problem $F2 \mid d_j = d \mid Y_w$, as a binary NP-hard case, can be solved optimally by a dynamic programming approach (Błażewicz et al., 2004b) in pseudo-polynomial time $O(n^2d^4)$. This approach is based on the property (Błażewicz et al., 2004b) that in an optimal solution, all early jobs have to be scheduled before the common due date in Johnson's order (Johnson, 1954). Johnson's order, which is optimal from the schedule length viewpoint, requires that jobs with the first task shorter than the second one ($p_{1j} < p_{2j}$) are scheduled in non-decreasing order of p_{1j}, while the remaining ones, with a task on M_1 requiring at least as many time units as a task on M_2 ($p_{1j} \geq p_{2j}$), in non-increasing order of p_{2j}. As a consequence of this property of the problem under consideration, the proposed DP method determines the first late job in the system and, then, it divides the set of the remaining jobs into two subsets containing activities being totally early and partially or totally late. All early jobs have to be scheduled in Johnson's order, while the sequence of totally late activities can be arbitrary. Because the sequence of analyzing jobs is determined by Johnson's order, the method does not need to investigate all possible permutations of jobs on machines.

Denoting with $\hat{J}_n$ the job selected as the first late job and numbering the remaining jobs $J\backslash\{\hat{J}_n\}$ from $\hat{J}_1$ to $\hat{J}_{n-1}$ in Johnson's order, the dynamic programming algorithm takes decisions based on an initial condition $f_n(A, B, t, a)$ and a recurrence function $f_k(A, B, t, a)$. The value of

these functions denotes the maximum amount of early work of jobs $\hat{J}_k, \hat{J}_{k+1}, ..., \hat{J}_n$ assuming that $\hat{J}_n$ is the first late job, the first job among $\hat{J}_k, \hat{J}_{k+1}, ..., \hat{J}_n$ starts on machine M_1 exactly at time A and not earlier than at time B on M_2. Moreover, exactly t time units are reserved for executing jobs $\hat{J}_1$ to $\hat{J}_{k-1}$ after $\hat{J}_n$ on M_1 before the common due date d. Moreover, it is assumed that no job ($a = 0$) or exactly one job ($a = 1$) among $\hat{J}_1$ to $\hat{J}_{k-1}$ is partially executed on machine M_1 after $\hat{J}_n$ before d. The criterion value corresponding to a certain selection of the first late job is represented by $f_1(0, 0, 0, 0)$. Obviously, maximizing the weighted early work is equivalent to minimizing the weighted late work in the system. To find an optimal solution of the problem, each job is considered as the first late job in order to determine the best first late job, J_*, ensuring the optimal criterion value $F^* = f_1(0, 0, 0, 0)$.

The general scheme of the dynamic programming approach (Błażewicz et al., 2004b) running in $O(n^2d^4)$ time is presented below.

$J = \{J_1, ..., J_n\}$;

for $j = 1$ to n do

set $\hat{J} = J \setminus \{J_j\}$ and J_j as the first late job $\hat{J}_n$;

renumber jobs from $\hat{J}$ in Johnson's order as $\hat{J}_1, \ldots, \hat{J}_{n-1}$;

calculate initial conditions $f_n(A, B, t, a)$ for $\hat{J}_n$, for $0 \le A, B, t \le d$ and $a \in \{0, 1\}$;

for $k = n-1$ to 1 do calculate recurrence relations $f_k(A, B, t, a)$ for $\hat{J}_k$, for $0 \le A, B, t \le d$ and $a \in \{0, 1\}$;

set $F_j = f_1(0, 0, 0, 0)$ as the total weighted early work subject to the first late job $\hat{J}_n$;

set $F^* = \max_{j=1,..n.} \{F_j\}$ as the optimal total weighted early work and set J_* to be a job with $F_j = F^*$;

based on dynamic programming results for the first late job J_* determine: J^E - the set of early jobs, J^P - the set of jobs performed between J_* and d on M_1, J^L - the set of late jobs;

construct an optimal schedule by: executing jobs from J^E in Johnson's order followed by J_*, performing the first tasks of jobs from J^P after J_* before d on M_1 and executing jobs from J^L after d in an arbitrary order.

3. Enumerative Method

To check the correctness of the rather complicated dynamic programming method sketched in the previous section, we have designed an enumerative approach too, which finds an optimal solution by systematic exploring the solution space. This algorithm checks all possible subsets of early jobs (E), executed in Johnson's order. Then, it considers each job among the remaining ones as the first late job J_x (if possible) and it completes a partial schedule with other jobs from $J\backslash E$ sequenced according to the non-increasing weights. Obviously particular sets of early jobs (and, in consequence, sequences of late ones) are considered only once in the order of non-decreasing cardinalities. Thus, not all possible permutations of n jobs are explicitly checked by the method, which, despite this fact, is obviously an exponential one with complexity $O(n2^n)$. The outline of an enumerative method mentioned is presented 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 already found one.

4. List Scheduling Method

In practice, the exact methods presented in the previous sections can be applied only for small problem instances, because of their exponential complexities. To obtain good solutions for instances of bigger sizes in acceptable time, heuristic algorithms have to be applied. Within this research, we compare the exact methods with Johnson's algorithm and a list scheduling method.

Johnson's algorithm (JA) designed for problem $F2 \mid \mid C_{max}$ (Johnson, 1954) can be used as a fast heuristic for the problem under consideration. It runs in O($n$log$n$) time necessary for sorting two subsets of jobs with $p_{1j} < p_{2j}$ and $p_{1j} \geq p_{2j}$ according to non-decreasing p_{1j} values and to non-increasing p_{2j} values, respectively. With regard to the fact that all early jobs are sequenced in Johnson's order in an optimal solution for $F2 \mid d_j = d \mid Y_w$, Johnson's algorithm may construct feasible schedules of this problem of rather good quality.

The list approach is a technique commonly used in the scheduling area, especially for practical applications (Haupt, 1989). It constructs a feasible solution by scheduling jobs on available machines, one by one, in a certain order determined by a given priority dispatching rule. The constructive procedure presented in this paper adds particular jobs, one by one, to a set of executed jobs (E). All jobs from this set are scheduled on machines in Johnson's order. At each step of the method, a new job is selected from the set of the remaining (available) jobs $A = J \backslash E$ according to a certain priority dispatching rule and it is added to E. Then, set E is rescheduled in Johnson's order. In consequence of adding a new job, the set of early jobs may change and the criterion value may be improved. Some jobs, which were early so far, can be shifted to the right by a newly added job, which precedes them in Johnson's order. The solution returned by the heuristic is the best solution obtained for particular sets of early jobs E, for which the partial schedule length exceeds the common due date d. In order to improve the method efficiency, the additional optimization step is performed, when the algorithm has constructed the first feasible schedule. When Johnson's schedule for a certain set of executed jobs E exceeds the common due d for the first time, the newly added job has been removed. Then, similar steps are performed for the remaining jobs from the set of available jobs A, i.e. they are introduced one by one to the set of executed activities E, in order to create a new solution, and then removed before the next job is taken into consideration.

Table 1. Definitions of static (S) and dynamic (D) priority dispatching rules

We have proposed 15 rules for selecting jobs from the set of available jobs A (cf. Table 1). Some of them determine the sequence of adding the jobs to a schedule once, at the beginning of the algorithm (static rules), while others arrange jobs from set A at each step of the method with regard to the length of the partial schedule obtained so far (dynamic rules). Because a selection rule (i.e. a priority dispatching rule) influences the execution order of jobs, applying different strategies makes it possible to construct different feasible solutions of the problem under consideration. In consequence, the list scheduling algorithm can be used for constructing a set of feasible solutions of different quality in a reasonable amount of time. The general framework of the list scheduling approach can be formulated as follows.

set $E = \emptyset$ and $A = J$, where E and A denote the set of executed and available jobs, respectively;

set $R = \emptyset$, where R denotes the set of feasible schedules constructed;

set $S = \emptyset$, where S denotes a partial schedule obtained so far;

for a static selection rule, sort A according to this rule obtaining the sequence in which the jobs are analyzed;

while $A \neq \emptyset$ do

    for a dynamic selection rule update job parameters (x_j or z_j) with regard to S;

    take the job $\hat{J}_1$ from A according to a selection rule;

    $E = E \cup \{\hat{J}_1\}$;

    $A = A \setminus \{\hat{J}_1\}$;

    schedule E by Johnson's algorithm obtaining solution S;

    if the schedule length of S exceeds d, then $R = R \cup \{S\}$;

    if S is the first feasible solution, then

        $E = E \setminus \{\hat{J}_1\}$;

        for each $J_x \in A \setminus \{\hat{J}_1\}$ do

            $E = E \cup \{J_x\}$;

schedule E by Johnson's algorithm obtaining solution S;

if the schedule length of S exceeds d, then $R = R \cup \{S\}$;

$E = E \setminus \{J_x\}$;

$E = E \cup \{\hat{J}_1\}$;

select the best solution from R.

The presented method performs n iterations adding every job to a partial schedule. Moreover, for each set of executed jobs E, the algorithm constructs Johnson's schedule in $O(n\log n)$ time. If a static job selection rule is applied, then all activities are sorted once, at the beginning of the algorithm in $O(n\log n)$ time. In the case of a dynamic selection rule, the selection order is determined at each iteration in $O(n)$ time, necessary for updating the job parameters (z_j or x_j) and selecting the job with the minimal/maximal value of the parameter being the subject of a certain priority dispatching rule (z_j or x_j). Consequently, the algorithm runs in $O(n^2\log n)$ time. The additional optimization step executed after obtaining the first feasible solution, requires $O(n^2\log n)$ time too, but because it is performed only once, it does not increase the overall complexity of the whole approach.

5. Computational Results

Computational experiments are devoted to check the time efficiency of particular methods, described in the paper, solving problem $F2 \mid d_j = d \mid Y_w$ and to compare the quality of solutions generated by them. Intuitionally, the dynamic programming approach with its pseudopolynomial time complexity should be more time efficient than an enumerative method of exponential time complexity (obviously, the quality of solutions determined by these two exact methods must be the same). On the other hand, the performance list scheduling approach and Johnson's algorithm applied as a heuristic for $F2 \mid d_j=d \mid Y_w$ is difficult to be predicted. Similarly, the best selection rule (priority dispatching rule) cannot be

determined in advance based only on a theoretical analysis. All algorithms proposed have been implemented in ANSI C++ and run on AMD Duron Morgan 1GHz PC (Kowalski, 2003).

The test results obtained for the exact algorithms support with numerical examples the correctness proof for the dynamic programming method (Błażewicz et al., 2004b). DP and the enumerative method can be applied only for small problem instances. During the experiments, we analyzed 25 small instances of 5 different sizes for the number of jobs n = 10, 12, 14, 16, 18. The task processing times were generated from the range [1, 10], while job weights were taken from the range [1, 5]. All instances contain an equal number of jobs with $p_{1j} < p_{2j}$ and $p_{1j} \geq p_{2j}$. The common due date value was settled to 30% of the mean machine load (i.e. to 30% of a half of the total processing time of all jobs).

Table 2. Average running times [μs] and standard deviation of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with S1 rule for different numbers of jobs n

The running times of all methods, i.e. the dynamic programming approach (DP), the enumerative method (EM), Johnson's algorithm (JA) applied as a heuristic for $F2 \mid d_j = d \mid Y_w$ and, finally, the list scheduling algorithm with a static selection rule S1 (the time differences between particular selections rule are neglectedly small) are compared in Table 2 and in Figure 2. Surprisingly, the pseudopolynomial time method appeared to be less efficient than the enumerative one. Obviously, DP is important form a theoretical viewpoint, because it made it possible to classify problem $F2 \mid d_j=d \mid Y_w$ as NP-hard in the ordinary sense, but the enumerative method seems to be a better approach for constructing exact solutions for the problem. The time complexity of DP depends mostly on the due date value ($O(n^2d^4)$), while the efficiency of the enumerative method is mainly influenced by the number of jobs ($O(n2^n)$). The DP method is more time-oriented, because looking for an optimal solution, it

considers job sequences with regard to particular time moments between zero and the common due date. In contrast, the EM method operates with sets of early jobs and sequences of late ones. Because the sequence of early jobs is determined by Johnson's order, the enumerative method does not need to generate permutations of all jobs. The special structure of the problem under consideration makes this approach more efficient than the DP one. Obviously, both exact algorithms required much more computational time than the heuristic ones.

Figure 2. Average running times [μs] of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with S1 rule for different numbers of jobs n

The test results show that enumerative methods cannot be rejected from the consideration a priori, even if pseudopolynomial approaches are available. Taking into account the specific structure of the problem analyzed, we constructed the exact method, which was relatively easy to implement and which delivers reference solutions for heuristic approaches in shorter time than DP. However, the EM algorithm appeared to be less stable than the latter one. The running time of EM depends strictly on an instance of the problem not only on its size. The longest running time was detected for an instance with 16 jobs, although instances with 18 jobs have been considered as well. The structure of the dynamic programming method makes it less sensitive to instance parameters in comparison to the enumerative one.

In order to investigate the influence of the problem parameters on exact methods more carefully, we performed additional experiments for a selected instance with 10 jobs changing the due date value from 10% to 90% of a half of the total processing time of all jobs, cf. Table 3. As one can expect, the due date value does not influence the running times of heuristic approaches: Johnson's method and the list scheduling algorithm, whose time

complexity and real efficiency depend only on the number of jobs analyzed. In contrast, the influence of the due date on the efficiency of the exact approaches is very strong.

Table 3. Running times [μs] of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with the S1 rule for different due dates values d (as a percentage of the half of the total processing time of all jobs)

The running times of the dynamic programming method significantly increases with the due date value (cf. Table 3 and Figure 3). This algorithm explores the solution space in $O(n^2d^4)$ time by considering particular time moments between zero and d. Thus, the due date value is the main factor in the complexity function, responsible for its pseudopolynomial status.

Figure 3. Average running times [μs] of the dynamic programming method (DP) for different due date values d (as a percentage of the half of the total processing time of all jobs)

Although the complexity function of the enumerative approach depends on the number of jobs ($O(n2^n)$), its running time depends strongly on the problem parameters. Looking for an optimal solution, the enumerative method generates explicitly only a small number from all possible schedules for n jobs. But the part of the solution space explicitly explored, and, in consequence EM running time, decreases for very small and very big due date values (with regard to the total processing time of all jobs, cf. Table 3 and Figure 4). If the due date value is small, then only a few jobs can be executed before d and the number of possible feasible schedules is small. These feasible schedules are generated by the enumerative method in a short time (the sequence of activities executed after d is irrelevant). Similarly, if the due date value is big, then most jobs are executed early in Johnson's order, and only a few jobs can be late.

Figure 4. Average running times [μs] of the enumerative method (EM) for different due dates values d (as a percentage of the half of the total processing time of all jobs)

The running times required by the exact approaches are usually unacceptable from the practical point of view. To solve a hard problem efficiently, one has to resign from optimal solutions and to apply heuristic methods. The test results show a significant difference between the running times of exact and approximate approaches (cf. Table 2 and 3). However, heuristic methods are evaluated not only from the running time point of view. The quality of solutions constructed by such algorithms is a very important factor in their analysis.

Because exact methods can be applied only for small instances, the comparison of heuristic solutions obtained for different priority dispatching rules with optimal schedules was possible for a small set of test examples (cf. Table 4, Columns 1-3). For bigger instances, only the relative efficiency of the list algorithm for particular priority dispatching rules could be estimated with regard to the best heuristic solution (cf. Table 4, Columns 4-6). We analyzed 25 instances of 5 different sizes for $n = 50, 100, 150, 200, 250$, with the processing time and weight ranges settled to $[1, 20]$, and the due date value determined as 50% of the mean machine load. In both cases (i.e. small and big instances), the priority dispatching rule performance is expressed as a percentage of the optimal or maximal weighted early work obtained by the considered rule with regard to the exact or best heuristic solution, respectively.

Table 4. Ranking of priority dispatching rules based on the solution quality compared to the optimal criterion value for small instances (Columns 1-3) and to the best heuristic solution for big instances (Columns 4-6) – the average rule efficiency and the standard deviation

The computational experiments show that list scheduling algorithm constructs heuristic solutions of very high quality (cf. Figure 5). The best priority dispatching rule (S1) selecting jobs according to non-decreasing weights constructed solutions with almost 98% of the optimal criterion value. Moreover, the standard deviation was the lowest for this selection strategy among all implemented ones. It has to be underlined that such a high performance is

obtained with incomparably small computational effort with regard to exact algorithms (cf. Table. 2)

Figure 5. Ranking of priority dispatching rules for small instances based on the solution quality compared to the optimal criterion value

As we expected, the most effective priority dispatching rules selected jobs with regard to their weights (S1) and, additionally, to their processing times on both machines (S2) or on only one machine (S6, S7). These parameters – weights and processing times – are the most important ones because in an optimal solution, the weighted early work has to be maximized. They are involved in the process of determining the criterion value. All the static rules mentioned above ensure a solution quality near to 90% of the optimum. Surprisingly, the dynamic selection rules, taking into account the job weights and a partial schedule structure, appeared to be less efficient (i.e. the rule determining the current late work for particular jobs – D4, as well as, the rule calculating the ratio between the job processing time and the gap between the partial schedule length and the common due date – D2). The test results show that simple static rules are more efficient than more time-consuming dynamic ones. Nevertheless, these dynamic rules involving job weights constructed better schedules than any other rule not taking into account job weights. When job weights are ignored, dynamic rules made it possible to construct solutions of higher quality than static ones (D1, D3, D5, D6 dominated S3, S4, S8 and S9).

Johnson's algorithm applied as a heuristic to $F2 \mid d_j = d \mid Y_w$ appeared to be a less efficient heuristic for the problem under consideration. This algorithm constructs schedules within an extremely short time (cf. Table 2) but their quality is of almost 30% worse than the ones determined by the list scheduling approach (for the best selection rule). Test results confirmed that the strategy devoted to the schedule length minimization cannot be efficient for minimizing the weighted late work. However, taking into account the extremely short

running time, the performance about 70% of the optimum might be enough for some specific applications.

As we have mentioned, heuristic solutions were compared to the optimal ones only for small problem instances. In computational experiments performed for a larger number of jobs, only the relative performance of particular priority dispatching rules could be validated with regard to the most efficient selection rule. However, the rule ranking obtained based on the comparison with the optimal and to the best heuristic solution is almost identical (cf. Figure 5 and Figure 6). The best solutions were always generated by selecting jobs according to their weights (S1). The static selection rules involving weights appeared to be more efficient than the dynamic rules. In addition, the latter ones were less stable, that is reflected in the higher values of the standard deviation. The test results confirmed once more the observation that designing more complicated rules, requiring more computational effort may not be rewarded with the solution quality.

Figure 6. Ranking of priority dispatching rules for big instances based on the solution quality compared to the best heuristic solution

Similarly as for small instances, Johnson's algorithm generated schedules of the poorest quality, however, for a larger number of jobs its performance was closer to the worst priority dispatching rule. Taking into account the huge difference in running times (cf. Table 5), this algorithm could be competitive to the list scheduling one (if the selection rule is not properly chosen).

Table 5. Average running times [μs] and standard deviation of the list scheduling algorithm with the S1 rule and of Johnson's algorithm (JA), and the minimal and maximal running times for the list algorithm for all selection rules and the difference between them and S1

The running times of the list scheduling algorithm does not vary too much for particular priority dispatching rules (see Table 5). We compared the running time of the best selection

rule (S1) to the minimal and maximal running times for particular instance sizes (the average value for 5 instances with the same number of jobs was calculated). The S1 rule was about 7% slower on average than the fastest one, and it was 11% faster on average than the slowest one (the dynamic selection rules). Hence, a priority dispatching rule determines the sequence of analyzing jobs and it does not affect the number of steps performed by the algorithm, its influence on the running time is rather inconspicuous.

Figure 7. Average running times [μs] of the list scheduling algorithm with S1 rule and the minimal (MIN) and maximal (MAX) running times for the list algorithm for different number of jobs n

The relation between running times and the number of jobs perfectly illustrates the complexity functions of both heuristic methods: the list scheduling algorithm and Johnson's one. In the first case, the parabolic function reflects $O(n^2 \log n)$, cf. Figure 7, in the latter, the linear function corresponds to $O(n \log n)$, cf. Figure 8.

Figure 8. Average running times [μs] of Johnson's algorithm for different number of jobs n

Investigating the features of the list scheduling algorithm applied to problem $F2 \mid d_j=d \mid Y_w$, we have also studied the influence of different problem parameters on its behavior (a similar analysis has been done for Johnson's method as well).

The running times of the heuristics depend mostly on the number of jobs n. As one could predict from the structures of the algorithms, their time efficiency is not influenced by the common due date value (cf. Table 3), the maximum weight value (cf. Table 7) as well as by the percentage of jobs with shorter first task than the second one (cf. Table 9). However, the quality of solutions obtained is dependent on the problem parameters values.

Table 6. Solution quality for Johnson's algorithm (JA), the worst (S8), the best (S1) priority dispatching rule and the average rule efficiency for different due date values d (as a percentage of the half of the total processing time of all jobs)

For large due date values, all heuristic methods determine solutions of a similar quality (cf. Table 6 and Figure 9). If the value of *d* increases (it is getting closer to the half of the total processing time of all jobs), then almost all jobs can be executed early and only a few of them are executed late. In consequence, the difference between Johnson's algorithm and the list scheduling one becomes less visible. Moreover, almost all jobs are early, the sequence of their analysis (i.e. the sequence of adding jobs into the set of executed jobs) does not influence a final solution too much. Therefore, the selection rule does not affect the performance of the list algorithm. For the largest due date values, schedules generated by the worst (S8) and the best rule (S1) as well as by Johnson's algorithm are almost identical. For strict due date values, the differences in the solution quality are much bigger, because only a few jobs can be processed early and their selection affects the criterion value significantly.

Figure 9. Solution quality for Johnson's algorithm (JA), the worst (S8), the best (S1) priority dispatching rule and the average rule efficiency for different due dates values d (as a percentage of the half of the total processing time of all jobs)

Similar computational experiments were performed in order to investigate the influence of the maximum job weight on the heuristic methods. 25 instances with 5 different values of the maximum weight w_{max} = 2, 5, 10, 20, 50 were analyzed for 100 jobs, the due date value was set to 50% of the mean machine load and the processing times were taken from the range [1,10].

Table 7. Running times for Johnson's algorithm (JA), the best rule (S1), the second rule in ranking (S2) and the worst rule (S8) for different values of the maximum job weight w_{max}

As we have already mentioned, the maximum job weight does not influence the running times of the heuristic methods (cf. Table 7), but it affects the solution quality obtained for particular priority dispatching rules (cf. Table 8 and Figure 10).

Table 8. Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the maximum job weights w_{max}

Increasing the value of w_{max} does not affect the efficiency of the selection rules involving job weights, but it deepens the difference between the rules selecting jobs with regard to w_j and the rules not taking w_j into account. In all tests, the priority dispatching rule analyzing jobs in non-decreasing order of w_j generated the best results. The second rule in the ranking, S2, based on the weighted job processing time, constructed schedules that are 5% worse and its performance did not depend on w_{max}. However, the solution quality for the worst rule applying Johnson's order (S8) deteriorates with the increase of w_{max}. A similar effect was observed for Johnson's algorithm and for the average performance of the list algorithm (only 6 rules among 15 involve job weights). In the case of the weighted performance measure, the larger the weights, the more important is to take them into account in the process of scheduling jobs on machines.

Figure 10. Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the maximum job weight w_{max}

Finally, we checked the influence of the structure of the job set on the heuristic methods, changing the percentage of jobs with the first task shorter than the second one. The experiments were performed for 15 different instances with 10%, 50% and 90% of jobs with the first task shorter than the second one, for 100 jobs with the due date set to 50% of the mean machine load and the processing times and weights from the range [1,10].

Table 9. The running times for Johnson's algorithm (JA), the best rule (S1), the second rule in ranking (S2) and the worst rule (S8) for different values of the percentage of jobs with the first task shorter than the second one

The structure of the job set does not affect the running time of Johnson's algorithm and its influence on the time requirements of the list scheduling method is rather inconspicuous – a small increase of the running time was observed for instances including mostly jobs with

shorter first task than the second one (cf. Table 9). This effect is caused by an additional optimization done on the implementation level, because, theoretically, there should be no influence of the job set structure on the method efficiency. In the implementation, when a job newly added to the set of executed jobs succeeds early jobs, the list method does not run Johnson's algorithm to construct a new partial schedule for a new set of selected jobs. When an instance contains many jobs with shorter first task, a newly added job has to be often executed before the common due date and it precedes jobs which were early in a previous solution. In consequence, Johnson's algorithm has to be executed for the set of selected jobs more frequently increasing the running time of the list method.

Table 10. Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the percentage of jobs with the first task shorter than the second one

Comparing the solution quality obtained for different structures of the job set (cf. Table 10 and Figure 11), one could conclude that the heuristic methods slightly prefer instances containing jobs of the same type. But bearing in mind the fact, that the solution quality was compared to the best heuristic schedule, the test results showed, in fact, that the analyzed heuristics construct similar schedules, when the job set is dominated by one type of jobs. In this case, the solution quality is comparable for particular priority dispatching rules and they achieve a performance closer to the best selection rule. For a more heterogeneous job set, there exist more distinct schedules with regard to the criterion value and the distance between good and bad priority dispatching rules increases.

Figure 11. Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the percentage of jobs with the first task shorter than the second one

4. Conclusions

In this paper, we have compared three different solution methods for problem $F2 \mid d_j = d \mid Y_w$, which is known to be binary NP-hard: a dynamic programming approach, an enumerative method and a list scheduling algorithm. Moreover, we tested Johnson's algorithm devoted to problem $F2 \mid\mid C_{max}$ as a heuristic for the problem under consideration.

The computational experiments showed that intuitive assumptions on time efficiency and quality performance of different solving methods may not be always confirmed in practice. Based on the complexity functions, one could suppose that the pseudopolynomial time dynamic programming method would be more efficient than the enumerative algorithm. The test results disclosed that the enumerative method is more efficient than the DP one owing to some special features of the problem under consideration. Looking for an optimal solution by analyzing job sequences, as in the enumerative approach, appeared to more efficient than searching for the best schedule from a time-oriented point of view, as in the DP algorithm investigating time moments between zero and the common due date. Taking into account the computational experiments performed within this research, when a binary NP-hard problem is investigated, an enumerative method should not be rejected from the consideration a priori as being less efficient than a pseudopolynomial one. Our studies confirms that the simplest solutions are not always the worst ones. Of course, the dynamic programming method is still very important from a theoretical point of view, because its existence made it possible to classify problem $F2 \mid d_j = d \mid Y_w$ as binary NP-hard. But, for determining optimal solutions, the enumerative method is a better choice in this case.

Computational experiments confirmed the well known fact, that exact approaches can only be applied for problem instances of small size. In order to find feasible solutions in reasonable time, we proposed a list scheduling algorithm. This technique is commonly used in scheduling theory, especially for practical applications, first of all, because of its easy

implementation. The test results supported the general opinion on the high utility of the list scheduling method. It constructed solutions of a good quality in a reasonable running time. Hence we proposed a number of static and dynamic priority dispatching rules, it was possible to perform extensive computational experiments in order to determine the influence of the selection rule on the solution quality and to detect the influence of problem parameters on the method behavior. Moreover, we checked the efficiency of Johnson's algorithm, dedicated originally to the schedule length minimization, as a heuristic approach to the weighted late work minimization. This idea was inspired by the special feature of the problem under consideration, according to which all early jobs have to be sequenced in Johnson's order. The quite high performance of Johnson's algorithm for problem $F2 \mid d_j = d \mid Y_w$ showed that searching for similarities between scheduling problems may be advantageous. The existing methods can act as heuristics for similar problems.

Within the future research, we are designing metaheuristic approaches for problem $F2 \mid d_j = d \mid Y_w$. The exact methods presented in this paper will be used for validating the solution quality for small problem instances, while the list algorithm will be applied as a method of generating initial schedules for metaheuristics. Since we have proposed a set of different priority dispatching rules, the list method can generate initial solutions of different quality, that makes it possible to start the solution space exploration from distinct points. Moreover, one can obtain a set of various feasible solutions, which may be used as an initial population for a population-based metaheuristic. Additionally, the schedules constructed by the list scheduling algorithm can be reference solutions for validating the metaheuristics performance for big problem instances.

Acknowledgment
We would like to express our appreciation to Krzysztof Kowalski for implementing the methods proposed and performing the computational experiments designed within this research.

References

Błażewicz, J. (1984). Scheduling preemptible tasks on parallel processors with information loss. *Recherche Technique et Science Informatiques*, 3(6), 415-420.

Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2001). *Scheduling computer and manufacturing processes*. Berlin, Heidelberg, New York: Springer-Verlag.

Błażewicz, J., & Finke, G. (1987). Minimizing mean weighted execution time loss on identical and uniform processors. *Information Processing Letters*, 24, 259-263.

Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2000). Total late work criteria for shop scheduling problems. In: K. Inderfurth, G. Schwödiauer, W. Domschke, F. Juhnke, P. Kleinschmidt, & G. Wäscher, *Operations Research Proceedings 1999* (pp. 354-359) Berlin: Springer-Verlag.

Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2003). *Revenue management in a job-shop: a dynamic programming approach* (pp. 1-21). Preprint Nr. 40/03, Magdeburg: Otto-von-Guericke-University.

Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2004a). Open shop scheduling problems with late work criteria. *Discrete Applied Mathematics*, 134, 1-24.

Błażewicz, J., Pesch, E., Sterna, M., & Werner, F. (2004b). The two-machine flow-shop problem with weighted late work criterion and common due date. *European Journal of Operational Research*, to appear.

Brucker, P. (1998). *Scheduling algorithms.* Berlin, Heidelberg, New York: Springer-Verlag.

Garey, M.R., & Johnson, D.S. (1979). *Computers and intractability*. San Francisco: W.H. Freeman and Co.

Haupt, R. (1989). A survey of priority rule based scheduling. *OR Spectrum*, 11, 3-16.

Ho, K., Leung, J.Y.T., & Wei, W.D. (1994). Minimizing maximum weighted error for imprecise computation tasks. *Journal of Algorithms*,16, 431-452.

Johnson, S.M. (1954). Optimal two- and three-stage production schedules. *Naval Research Logistics Quarterly*, 1, 61-68.

Kowalski, K. (2003*) Scheduling algorithms for the shop scheduling problems with the late work criteria*. Master thesis. Poznań University of Technology.

Potts, C.N., & Van Wassenhove, L.N. (1991a). Single machine scheduling to minimize total late work. *Operations Research*, 40(3), 586-595.

Potts, C.N., & Van Wassenhove, L.N. (1991b). Approximation algorithms for scheduling a single machine to minimize total late work. *Operations Research Letters*, 11, 261-266.

Shih, W.K., Liu, J.W.S., & Chung, J.Y. (1991). Algorithms for scheduling imprecise computations with timing constraints. *SIAM Journal on Computing*, 20, 537-552.

Sterna, M. (2000). *Problems and algorithms in non-classical shop scheduling*. Ph.D. thesis. Poznań: Scientific Publishers of the Polish Academy of Sciences.

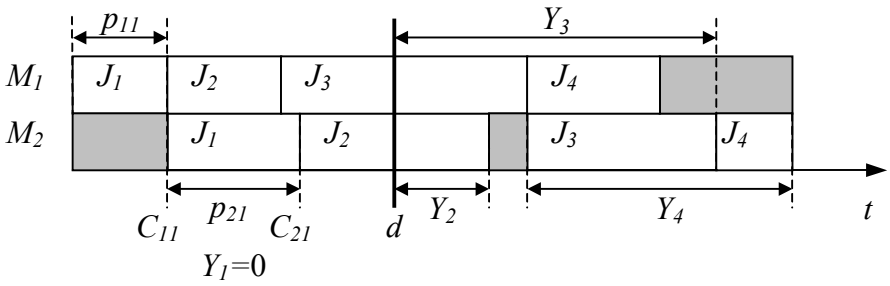

Figure 1.

An illustrative instance of problem $F2 \mid d_j = d \mid Y_w$

Notation	Priority Dispatching Rule (PDR) – selection rule
S1	$max\{w_j\}$ for $J_j \in A$
S2	$max\{w_j\ (p_{1j} + p_{2j})\}$ for $J_j \in A$
S3	$min\{p_{1j}\}$ for $J_j \in A$
S4	$max\{p_{1j}\}$ for $J_j \in A$
S5	$min\{p_{1j} + p_{2j}\}$ for $J_j \in A$
S6	$max\{w_j\, p_{1j}\}$ for $J_j \in A$
S7	$max\{w_j\, p_{2j}\}$ for $J_j \in A$
S8	job selection in Johnson's order
S9	random job selection
D1	$min\{x_j\}$ for $J_j \in A$, where $x_j = p_{1j}/(d - T_1) + p_{2j}/(d - T_2)$ and T_i denotes the partial schedule length on machine M_i
D2	$min\{w_j\, x_j\}$ for $J_j \in A$
D3	$max\{x_j\}$ for $J_j \in A$
D4	$max\{w_j\, x_j\}$ for $J_j \in A$
D5	$max\{z_j\}$ for $J_j \in A$, where $z_j = max\{0, d - T_1 - p_{1j}\} + max\{0, d - T_2 - p_{2j}\}$
D6	$min\{z_j\}$ for $J_j \in A$

Table 1.

Definitions of static (S) and dynamic (D) priority dispatching rules

n	DP time [μs]		EM time [μs]		JA time [μs]		S1 time [μs]	
	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.
10	451 782	191 576	1 939	492	11	0,50	323	20
12	1 081 864	326 380	20 357	22 592	11	0,50	448	24
14	2 821 465	844 267	127 221	100 223	13	0,00	614	26
16	4 101 604	1 666 507	7 928 479	14 959 867	14	0,50	758	51
18	12 445 751	3 489 505	6 205 041	5 564 582	15	0,58	1039	39

Table 2.

Average running times [μs] and standard deviation of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with S1 rule for different numbers of jobs *n*

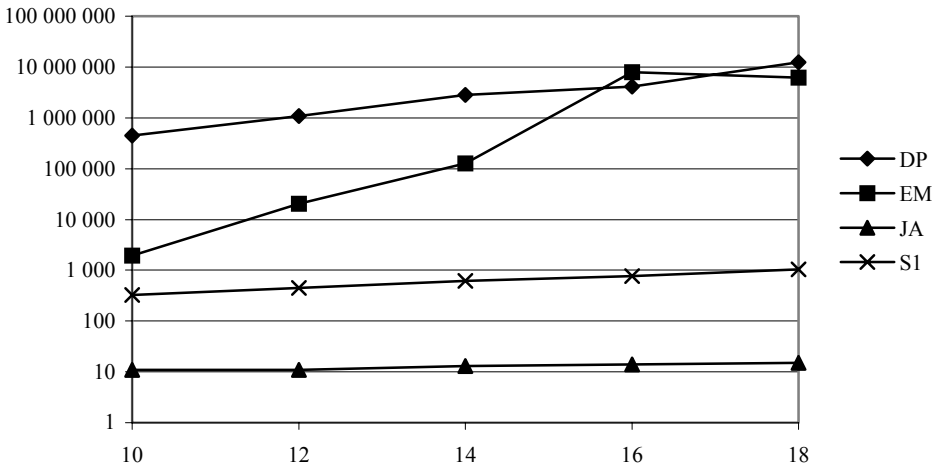

Figure 2.

Average running times [μs] of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with S1 rule for different numbers of jobs *n*

d [%]	DP [μs]	EM [μs]	JA [μs]	S1 [μs]
10	31 580	63	12	340
20	238 335	138	11	343
30	648 829	547	12	344
40	1 791 391	3 654	12	347
50	3 490 018	7 289	12	359
60	5 512 625	8 845	11	358
70	8 790 002	9 311	12	341
80	14 079 739	4 841	11	333
90	20 049 948	1 991	11	336

Table 3.

Running times [μs] of the dynamic programming (DP), enumerative (EM) methods, Johnson's algorithm (JA) and the list algorithm with the S1 rule for different due dates values *d* (as a percentage of the half of the total processing time of all jobs)

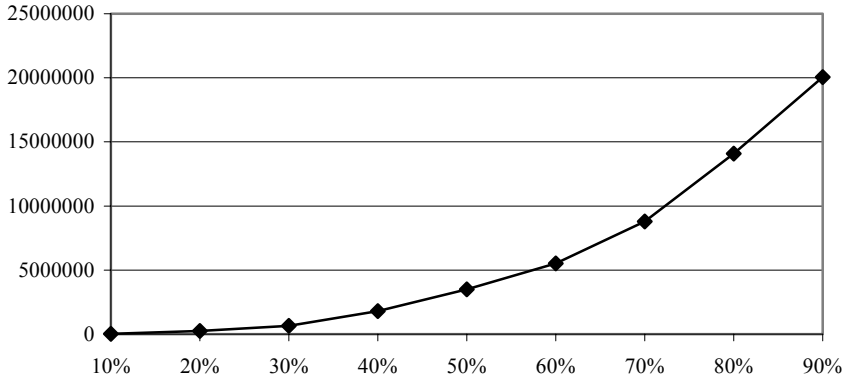

Figure 3.

Average running times [μs] of the dynamic programming method (DP) for different due date values *d* (as a percentage of the half of the total processing time of all jobs)

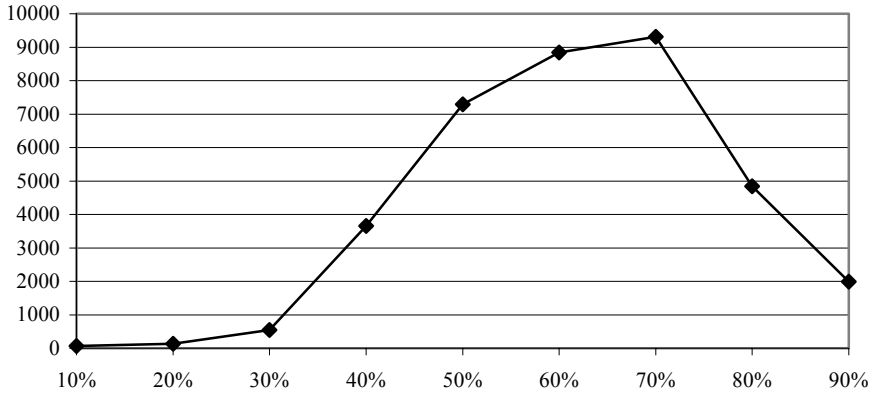

Figure 4.

Average running times [μs] of the enumerative method (EM) for different due dates values *d* (as a percentage of the half of the total processing time of all jobs)

PDR ranking I	average perform.[%]	standard deviation [%]	PDR ranking II	average perform.[%]	standard deviation [%]
1	2	3	4	5	6
S1	97.66	2.80	S1	100.0	0.00
S2	92.40	5.40	S2	93.98	1.85
S6	91.55	5.80	S6	92.19	2.10
S7	91.03	5.10	S7	88.36	2.66
D4	85.15	6.70	D2	75.23	4.27
D2	83.25	8.10	D6	73.89	4.36
D1	83.23	7.70	D1	73.64	4.01
D6	81.85	8.10	D4	73.45	4.28
S5	81.64	8.10	D3	73.37	4.28
D5	81.25	8.80	S5	72.84	3.73
D3	80.44	6.80	D5	72.76	3.24
S4	78.68	9.10	S4	72.63	5.05
S9	78.41	6.80	S3	71.44	3.65
S3	76.17	8.30	S9	71.18	4.26
S8	73.87	6.90	S8	70.43	3.95
JA	69.57	8.40	JA	70.06	3.86

Table 4.

Ranking of priority dispatching rules based on the solution quality compared to the optimal criterion value for small instances (Columns 1-3) and to the best heuristic solution for big instances (Columns 4-6) – the average rule efficiency and the standard deviation

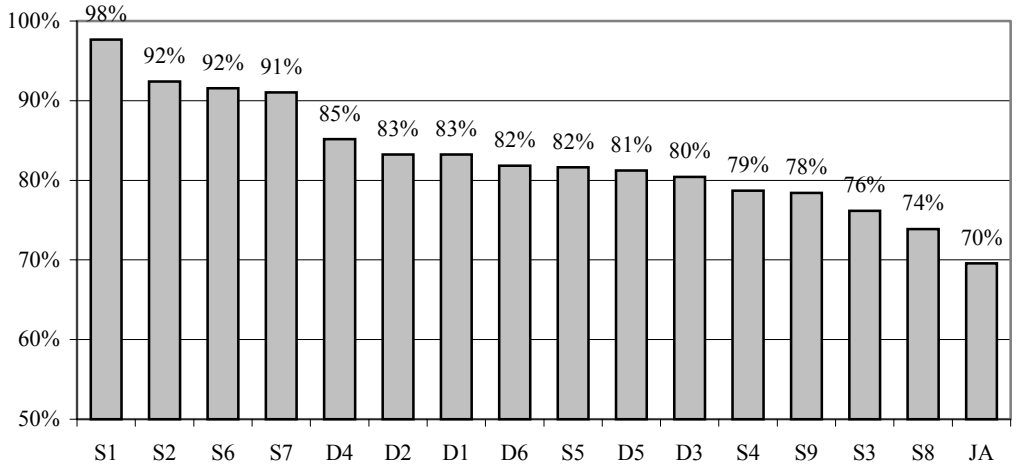

Figure 5.

Ranking of priority dispatching rules for small instances based on the solution quality compared to the optimal criterion value

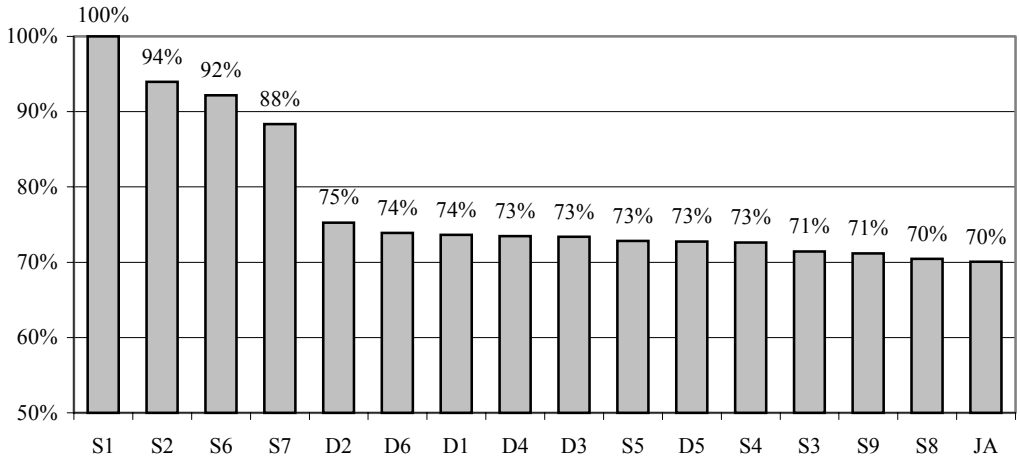

Figure 6.

Ranking of priority dispatching rules for big instances based on the solution quality compared to the best heuristic solution

n	MIN time [μs]		S1 time [μs]		MAX time [μs]		JA time [μs]	
	avg.	vs. S1	avg.	st. dev.	time	vs. S1	avg.	st. dev.
50	6 990	4.87%	7 331	207.22	8 254	12.59%	36	3.71
100	27 366	6.98%	29 275	409.64	32 848	12.20%	72	0.55
150	93 531	7.37%	100 422	3186.13	109 346	8.89%	100	3.81
200	168 984	6.89%	180 624	2921.68	199 472	10.44%	132	3.56
250	269 500	7.59%	289 967	4734.13	321 586	10.90%	165	4.67

\

Table 5.

Average running times [μs] and standard deviation of the list scheduling algorithm with the S1 rule and of Johnson's algorithm (JA), and the minimal and maximal running times for the list algorithm for all selection rules and the difference between them and S1

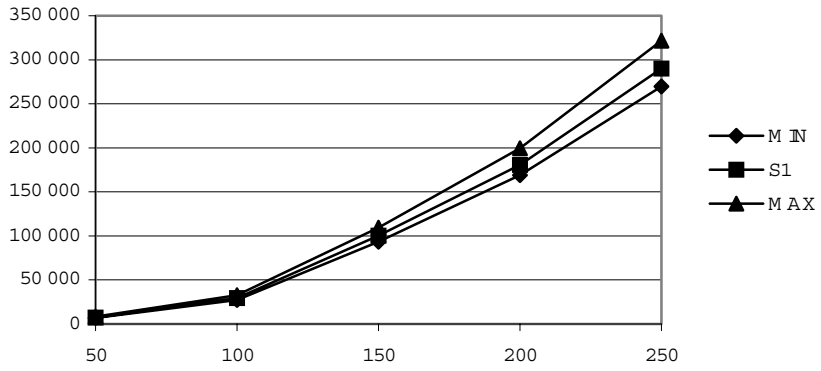

Figure 7.

Average running times [μs] of the list scheduling algorithm with S1 rule and the minimal (MIN) and maximal (MAX) running times for the list algorithm for different number of jobs *n*

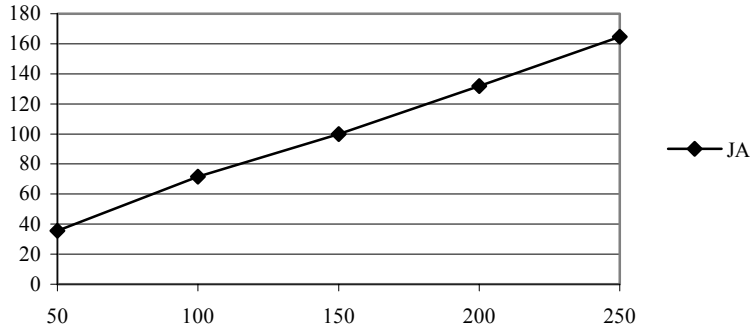

Figure 8.

Average running times [μs] of Johnson's algorithm for different number of jobs *n*

d [%]	JA [%]	S8 [%]	avg. [%]	S1 [%]
10	70.00	70.00	83.13	100.00
20	75.76	75.76	83.08	95.96
30	73.61	81.94	85.42	95.14
40	73.71	77.84	83.02	93.30
50	76.50	76.92	84.27	92.31
60	81.85	81.85	86.53	89.96
70	86.62	86.62	90.76	94.01
80	91.48	91.48	94.80	97.05
90	96.28	96.28	96.40	96.28
avg.	80.65	82.08	87.49	94.89
st.dev.	9.02	8.23	5.21	2.92

Table 6.

Solution quality for Johnson's algorithm (JA), the worst (S8), the best (S1) priority dispatching rule and the average rule efficiency for different due date values *d* (as a percentage of the half of the total processing time of all jobs)

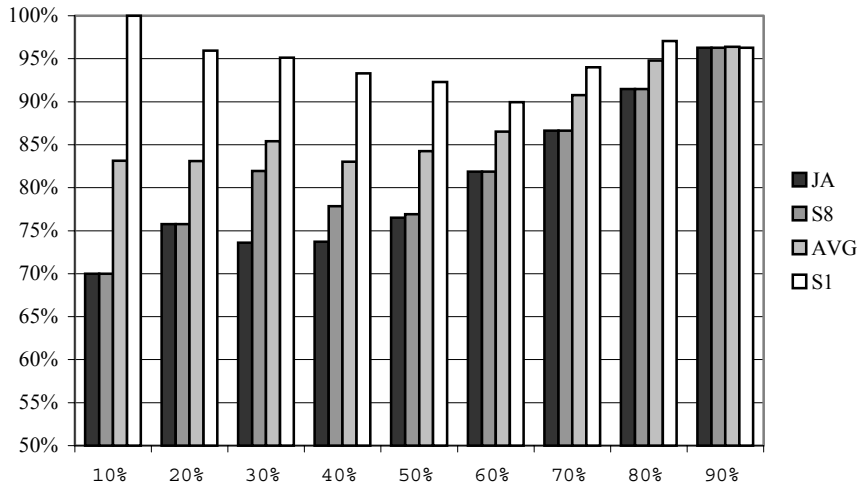

Figure 9.

Solution quality for Johnson's algorithm (JA), the worst (S8), the best (S1) priority dispatching rule and the average rule efficiency for different due dates values *d* (as a percentage of the half of the total processing time of all jobs)

w_{max}	JA time [μs]		S1 time [μs]		S2 time [μs]		S8 time [μs]	
	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.
2	69	4	29467	727	29741	529	29934	572
5	71	1	28483	636	29136	343	29377	290
10	72	1	29033	288	29420	218	29777	212
20	72	1	28529	284	29201	171	29747	369
50	68	4	29111	772	29534	752	29857	677

Table 7.

Running times for Johnson's algorithm (JA), the best rule (S1), the second rule in ranking (S2) and the worst rule (S8) for different values of the maximum job weight w_{max}

w_{max}	JA [%]		S8 [%]		Avg. [%]		S2 [%]	
	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.
2	79.11	2.73	80.03	2.75	82.79	4.12	94.07	0.72
5	71.02	2.72	72.34	2.34	78.76	7.13	94.13	1.60
10	72.93	4.05	74.67	4.22	79.84	6.64	94.10	1.18
20	67.48	2.95	68.09	3.26	75.23	7.84	92.40	2.24
50	68.29	2.90	68.90	2.93	75.31	9.12	94.97	1.88

Table 8.

Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the maximum job weights w_{max}

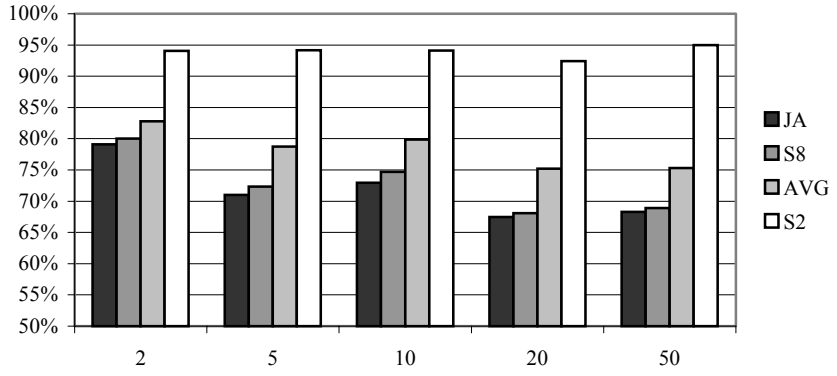

Figure 10.

Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the maximum job weight w_{max}

$p_{1i}<p_{i2}$	JA time [μs]		S1 time [μs]		S2 time [μs]		S8 time [μs]	
	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.
10%	64	1	26903	603	26785	536	26750	408
50%	65	4	29491	1308	29700	980	30068	989
90%	65	0	31955	372	32378	300	32737	389

Table 9.

The running times for Johnson's algorithm (JA), the best rule (S1), the second rule in ranking (S2) and the worst rule (S8) for different values of the percentage of jobs with the first task shorter than the second one

$p_{1i}<p_{i2}$	JA [%]		S8 [%]		Avg. [%]		S2 [%]	
	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.	avg.	st. dev.
10%	80.12	2.94	81.04	2.23	85.29	5.04	95.44%	1.10
50%	70.15	2.53	70.81	2.74	77.74	8.24	95.31%	1.98
90%	76.20	3.75	77.45	4.27	82.16	7.92	99.03%	0.78

Table 10.

Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the percentage of jobs with the first task shorter than the second one

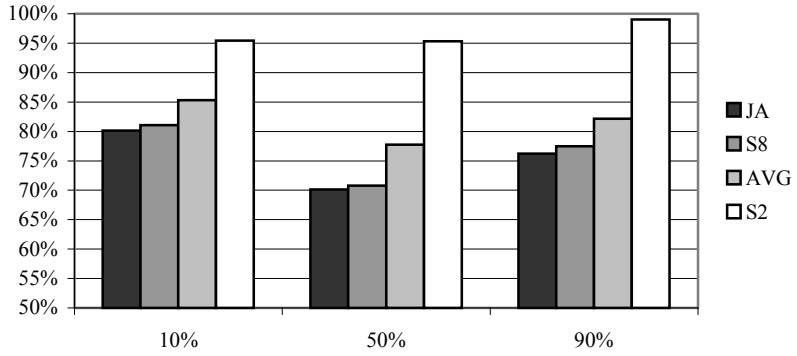

Figure 11.

Solution quality for Johnson's algorithm (JA), the worst (S8), the second best (S2) priority dispatching rule and the average rule efficiency for different values of the percentage of jobs with the first task shorter than the second one