Noname manuscript No.
(will be inserted by the editor)

Heuristic algorithms to maximize the weighted revenue and weighted number of jobs processed on parallel uniform machines

Omid Gholami * · **Yuri N. Sotskov** · **Saeedeh Bakhoda** · **Frank Werner**

Received: date / Accepted: date

Abstract A set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of jobs has to be processed on a set of parallel uniform machines. For each job $J_i \in \mathcal{J}$, a release time $r_i \geq 0$ and a due date $d_i > r_i$ are given. Each job may be processed without interruptions on any of the given machines having different speeds. If job $J_i \in \mathcal{J}$ will be started and then completed within the time segment $[r_i, d_i]$, the benefit $b_i > 0$ will be earned. Otherwise, this job will be rejected and the benefit b_i will be discarded. Let $\mathcal{J}(S)$ denote the subset of all jobs $J_i \in \mathcal{J}$ processed within their intervals $[r_i, d_i]$ in the schedule S. The set $\mathcal{J} \setminus \mathcal{J}(S)$ includes all the jobs rejected from the schedule S. The criterion under consideration is to maximize the weighted sum of the benefits $w_1 \sum_{i \in \mathcal{J}(S)} b_i$ and the weighted number of jobs $w_2|\mathcal{J}(S)|$ processed according to the schedule S, where both weights w_1 and w_2 are non-negative with the assumption that $w_1 + w_2 = 1$. We investigate some properties of the objective function, develop a simulated annealing algorithm, a tabu search algorithm, and a genetic algorithm for solving the above scheduling problem heuristically. The developed algorithms are tested on moderate instances with $n = 50$ and five uniform machines and on large instances with up to 500 jobs and 50 uniform machines. It is demonstrated how to use the obtained results in scheduling practice.

Keywords Scheduling · Uniform machines · Revenue maximization · Genetic algorithm · Simulated annealing · Tabu search

Omid Gholami *
Islamic Azad University - Mahmudabad Center, Nour Branch, Mahmudabad, Iran.
E-mail: gholami@iaumah.ac.ir

Yuri N. Sotskov
United Institute of Informatics Problems, National Academy of Sciences of Belarus, Surganova Str 6, Minsk 220012, Belarus.
E-mail: sotskov@newman.bas-net.by

Saeedeh Bakhoda
Islamic Azad University, Amol Branch, Amol, Iran.
E-mail: saeedehbakhoda@iauamol.ac.ir

Frank Werner
Faculty of Mathematics, Otto-von-Guericke-University, Magdeburg, Germany.
E-mail: frank.werner@ovgu.de

MSC classification: 90 B 35

June 29, 2016

1 Introduction

One of the main scheduling goals is to bind optimally a set of limited resources (machines) to competitive activities (jobs) along time. An improved production schedule is an important factor for the effectiveness of the company since a better schedule leads to a reduction of wastes, shorter response times to customer demands, a timely supply of raw materials and spare parts needed for the factory. An improved production schedule prevents the factory from capital accumulation and idle machinery. For a small scheduling problem, an optimal schedule may be obtained by an universal optimization method like a branch-and-bound method, dynamic programming or mathematical programming. However, these methods are not appropriate for a real-world scheduling problem having a larger size since the most scheduling problems are NP-hard even with two or three jobs [Brucker et al.]. Therefore, heuristic algorithms are usually applied to a real-world scheduling problem.

In this paper, we consider one of the scheduling problems arising in practice, where a job from the given set needs to be processed on one of the parallel machines, which might be either identical machines or uniform ones. The uniform machines are characterized by different speeds. In [Graham et al.], the former problem is denoted as $P||\Phi$ with Φ determining the objective function, and the latter problem is denoted as $Q||\Phi$. The problem $P||\Phi$ is NP-hard even if there are two machines and the objective is to minimize the makespan: $\Phi = \max\{C_1, C_2, \ldots, C_n\}$, where C_i is the completion time of the job J_i (see [Graham et al.]).

This paper deals with the problem of maximizing both the revenue and the number of jobs processed on parallel uniform machines. The rest of the paper is organized as follows. In Section 2, we survey some papers dealing with scheduling jobs on parallel machines. In Section 3, the problem setting is presented along with analytical results on the extreme values of the objective function. Three heuristic algorithms are described in Section 4. Computational results with an analysis of the developed algorithms are provided in Section 5. Managerial implications of the obtained results are discussed in Section 6. Concluding remarks and perspectives are described in Section 7.

2 Related literature

In [Anglani et al.], a fuzzy mathematical programming was developed to minimize the total setup time for processing the jobs on parallel identical machines, where the numerical data were uncertain. The same problem with deterministic numerical data was investigated in [Feng and Lau].

The work [Berrichi and Yalaoui] dealt with production scheduling and maintenance planning problems. An ant colony algorithm was developed for scheduling a set of jobs on parallel machines. The paper [Lin and Lin] contains results on different dispatching rules for the minimization of the makespan, the total weighted completion time, and the total weighted tardiness. The consideration of non-identical parallel machines makes the scheduling problem harder. In [Balin-2011], a crossover operator was proposed for such a problem in order to adapt the genetic algorithm to scheduling jobs on non-identical parallel machines. The computational results were compared to those obtained with the longest processing time (LPT) dispatching rule. A meta-heuristic approach was proposed in [Feng and Lau] to minimize the sum of the weighted earliness and tardiness on uniform parallel machines. In [Juraszek et al.], results were presented for a simultaneous job selection and an assignment of the jobs to the parallel uniform machines. A simulated annealing algorithm was developed and the computational results were compared with those obtained by a branch-and-bound algorithm. In [Balin-2012], a genetic algorithm embedded into a simulation model was proposed to minimize the makespan. The obtained results were compared with those obtained by using the LPT dispatching rule.

In [Li and Yang], some models and relaxations for non-identical parallel machines were provided for the minimization of the total weighted completion times. The scheduling problem of minimizing the makespan on non-identical parallel machines was solved by integer programming in [Shubin and Bean]. The authors proposed a genetic algorithm based on an encoding scheme using random keys. The work [Rodriguez et al.] dealt with non-identical parallel machines to minimize the total weighted completion times. In [Supithak and Plongon], the minimization of the sum of earliness and tardiness times was considered. To reduce the size of the solution space, the problem was reduced to sequencing the jobs. The authors proposed a memetic algorithm to solve the problem.

A problem with parallel unrelated machines was investigated in [Logendran et al.], where six algorithms based on tabu search were developed to solve the problem. The influence of different initial solutions on the quality of the final schedule was investigated. The same problem was solved by a simulated annealing algorithm in [Anagnostopoulos and Rabadi]. It was shown that simulated annealing can be promising to obtain a better makespan value. For a parallel machine scheduling problem with sequence-dependent setup times and release times, [Sivrikaya and Ulusoy] suggested a genetic algorithm that minimizes the sum of the earliness and tardiness costs. In [Bilge et al.], a tabu search was proposed for minimizing total job tardiness. The authors claimed that the tabu search algorithm outperformed the genetic algorithm suggested in [Sivrikaya and Ulusoy].

A multi-objective model for parallel machine scheduling with minimizing the number of tardy jobs and total completion time of the jobs was investigated in [Moghaddam et al.], where the parallel machines were unrelated. There are given due dates and release times for the jobs as well as

precedence relations between the jobs. Sequence-dependent setup times embedded in the proposed model may vary for different machines based on their characteristics. This problem was solved by binary linear programming with two levels and goal programming. In [Dunstall and Wirth], the problem of scheduling jobs divided into G families for processing on parallel identical machines was considered. A setup time must be considered when switching from the processing of a family i of jobs to those of another family j. Several heuristic algorithms were proposed and their performance was evaluated by comparing the results with the proven lower bounds and the solutions obtained by an exact algorithm.

Another important area for practical research is related to parallel machine scheduling with revenue maximization. In this case, each job has a due date and the aim is to complete the job before the given due date to gain a benefit. Exceeding the due date for the job completion may decrease the revenue with a penalty depending on the job tardiness. In [Sterna et al.], a simulated annealing algorithm was proposed for the revenue maximization on parallel machines. The efficiency of the algorithm was compared in computational experiments with a branch-and-bound method and a list scheduling approach. The work [Islam et al.] was addressed to the problem of market-based batch scheduling for parallel jobs running on supercomputer centers with the consideration of revenue maximization. The authors proposed a value based scheduling algorithm. It was shown that, in terms of maximizing the revenue, while achieving a better performance with respect to standard performance metrics such as slowdown and utilization, the proposed algorithm was efficient.

The paper [Dawande et al.] addresses the problem of the selection and screening of movies for a multiplex to maximize the exhibitor's cumulative revenue over a fixed planning horizon. The release times of the movies that can be selected are known. If selected for screening, a movie must be scheduled through its obligatory period, after which its run may or may not be extended. The authors of [Dawande et al.] investigated two basic screening policies: preempt-resume and non-preempt. It was shown that optimizing under the preempt-resume policy is NP-hard while the problem under the non-preempt policy is polynomially solvable. Several algorithms have been developed to solve both problems. It was shown that the revenue obtained from the preempt-resume policy can be significantly higher as compared with that from the non-preempt policy.

The work [Feng et al.] addresses the problem of maximizing the provider's revenue through dynamic resource allocation based on Service Level Agreement that plays a vital role in Cloud computing to bridge service providers and customers. The authors of [Feng et al.] formalized the resource allocation problem using queuing theory and proposed optimal solutions for the problem considering various parameters such as pricing mechanism, arrival rate, service rate and available resources. The experimental results showed that the developed algorithms outperform related ones.

3 Problem setting and preliminaries

The scheduling problem under consideration is denoted as $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ using the three-field notation $\alpha|\beta|\gamma$ introduced in [Graham et al.], where the field α characterizes the type of the processing system, the field β the input parameters, and the field γ the objective function.

3.1 Problem setting

The considered problem can be described as follows. There is a set of m parallel uniform machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ for processing a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of given jobs. A job $J_i \in \mathcal{J}$ may be processed on any machine from the set $\mathcal{M}$ without preemptions. However, the speeds of the machines $M_k \in \mathcal{M}$ are different.

For each job $J_i \in \mathcal{J}$, a release time $r_i \geq 0$ and a strict due date $d_i > r_i$ are given. If the job $J_i \in \mathcal{J}$ is processed within the closed interval $[r_i, d_i]$, then a benefit $b_i > 0$ is earned. Otherwise, the benefit b_i is discarded and the job J_i is deleted from the schedule. Let $w_1 \geq 0$ denote the weight of the revenue maximization (minimization of the total benefit for processing jobs) and $w_2 \geq 0$ the weight of the maximization of the number $|\mathcal{J}(S)|$ of the jobs $J_i \in \mathcal{J}$ processed within their intervals $[r_i, d_i]$ in the schedule S. It is assumed that $w_1 + w_2 = 1$.

The considered problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ is to find such a schedule S for processing the jobs from the set $\mathcal{J}$ on the machines from the set $\mathcal{M}$, which maximizes the sum $w_1 \sum_{J_i \in \mathcal{J}(S)} b_i + w_2|\mathcal{J}(S)|$ of the weighted total benefit for processing the subset $\mathcal{J}(S)$ of jobs and the weighted number $|\mathcal{J}(S)|$ of the jobs processed in the schedule S within their intervals $[r_i, d_i]$, $J_i \in \mathcal{J}$.

Since the machines from the set $\mathcal{M}$ are uniform and parallel, they have different speeds to process the same job $J_i \in \mathcal{J}$. Let machine $M_k \in \mathcal{M}$ have a production rate $v_k > 0$, then the time p_{ik} for processing job J_i on machine M_k is determined as follows:

$$p_{ik} = \frac{p_i}{v_k},$$

where $p_i > 0$ denotes the normal time needed for processing the job $J_i \in \mathcal{J}$.

Let job J_i be processed on machine $M_k \in \mathcal{M}$ in the schedule S. The processing of the job J_i on the machine M_k can be started after the release time r_i and after the completion time $c_{lk}(S)$ of the previous job J_l (if any) processed on the same machine M_k in the schedule S. Therefore, the start time $s_{ik}(S)$ of the job J_i on machine M_k in the schedule S can be calculated as follows:

$$s_{ik}(S) = \max\{r_i, c_{lk}(S)\}.$$

Since preemptions in the processing of a job are not allowed, the completion time $c_{ik}(S)$ of the job J_i on the machine M_k in the schedule S can be calculated as the sum of the start time s_{ik} and the processing time p_{ik} of the job J_i on the machine M_k:

$$c_{ik}(S) = s_{ik}(S) + p_{ik}.$$

The set $\mathcal{J}(S)$ is the subset of jobs $J_i \in \mathcal{J}$ processed in the schedule S within the intervals $[r_i, d_i]$, i.e., the inclusion $J_i \in \mathcal{J}(S)$ holds if

$$r_i \leq s_{ik}(S) < s_{ik}(S) + p_{ik} = c_{ik}(S) \leq d_i.$$

The first summand of the objective function

$$\Phi(S) = w_1 \sum_{J_i \in \mathcal{J}(S)} b_i + w_2 |\mathcal{J}(S)|$$

is the sum $\sum b_i(S)$ of the benefits for the processed jobs:

$$\sum b_i(S) := \sum_{J_i \in \mathcal{J}(S)} b_i.$$

The second summand of the objective function Φ is connected with the most customer satisfaction via increasing the number $|\mathcal{J}(S)|$ of the jobs $\mathcal{J}(S)$ to be started and completed in the schedule S. The problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ is to maximize the weighted sum of the above two summands:

$$w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i + w_2 \cdot |\mathcal{J}(S)| = \Phi(S),$$

where w_1 denotes the weight of the revenue maximization and w_2 the weight of the maximization of the number of jobs $\mathcal{J}(S)$ processed in the schedule S. If $w_1 = 1$, then $w_2 = 0$ and the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ under consideration turns out to be the problem $Q|r_i, d_i| \sum b_i$ of revenue maximization.

3.2 Extreme values of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$

The following Theorem 1 characterizes an extreme case of the objective function value, namely, the maximal possible value of the function $\Phi(S)$.

Theorem 1 *The schedule S is optimal and provides the maximal possible value of the objective function $\Phi(S)$ for the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ if and only if $\mathcal{J}(S) = \mathcal{J}$.*

Proof Sufficiency. Let equality $\mathcal{J}(S) = \mathcal{J}$ hold. Then we obtain the following equality:

$$w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i + w_2 \cdot |\mathcal{J}(S)| = w_1 \cdot \sum_{J_i \in \mathcal{J}} b_i + w_2 \cdot n. \quad (1)$$

It is easy to convince that the value $w_1 \cdot \sum_{J_i \in \mathcal{J}} b_i + w_2 \cdot n$ on the right-hand side of equality (1) is equal to the maximal possible value of the objective function $\Phi(S)$. Thus, the schedule S indicated in (1) provides the maximal possible value of the objective function $\Phi(S)$. Therefore, the schedule S is optimal for the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ under consideration.

Necessity. Let the schedule S be optimal for the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ and let the schedule S provide the maximal possible value of the objective function $\Phi(S)$ which is determined in (1).

By contradiction, we assume that $\mathcal{J}(S) \neq \mathcal{J}$. Therefore, there exists a job $J_k \in \mathcal{J}$, which was not processed within the interval $[r_k, d_k]$ in the schedule S, i.e., the set $\mathcal{J}(S)$ does not contain the job J_k: $J_k \notin \mathcal{J}(S)$.

Since $w_1 \geq 0$, $w_2 \geq 0$, and the equality $w_1 + w_2 = 1$ holds, then at least one of the weights w_1 or w_2 is not equal to zero in the objective function $\Phi(S) = w_1 \sum_{J_i \in \mathcal{J}(S)} b_i + w_2|\mathcal{J}(S)|$.

If $w_1 > 0$, then using the inequality $b_k > 0$, we obtain the following inequality:

$$w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i \leq w_1 \cdot \sum_{J_i \in \mathcal{J}} b_i - w_1 \cdot b_k < w_1 \cdot \sum_{J_i \in \mathcal{J}} b_i.$$

If $w_2 > 0$, then we obtain $w_2 \cdot |\mathcal{J}(S)| \leq w_2 \cdot (|\mathcal{J}| - 1) < w_2 \cdot n$.

Thus, in both possible cases, we obtain a contradiction that the equality (1) does not hold for the schedule S. Hence our above assumption $\mathcal{J}(S) \neq \mathcal{J}$ was wrong, which completes the proof.

Theorem 2, which follows, characterizes another extreme case of the objective function value, namely, the minimal possible value of the objective function $\Phi(S)$.

Theorem 2 *The optimal value of the objective function $\Phi(S)$ is equal to zero for the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ if and only if there is no job $J_i \in \mathcal{J}$ which may be processed within the interval $[r_i, d_i]$.*

Proof Sufficiency. It is assumed that there is no job $J_i \in \mathcal{J}$ that may be processed within the interval $[r_i, d_i]$. Then it follows that equality $\mathcal{J}(S) = \emptyset$ must hold for any schedule S. Therefore, the optimal value of the objective function $\Phi(S)$ is equal to zero (it is the minimal possible value of the objective function $\Phi(S)$):

$$\Phi(S) = w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i + w_2 \cdot |\mathcal{J}(S)| = w_1 \cdot 0 + w_2 \cdot 0 = 0.$$

Necessity. Let the optimal value of the objective function $\Phi(S)$ be equal to zero for the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ under consideration.

By contradiction, we assume that there is a job $J_i \in \mathcal{J}$, which may be processed within the interval $[r_i, d_i]$. Therefore, there exists a schedule S, for which the job $J_i \in \mathcal{J}$ is processed within the interval $[r_i, d_i]$. Next, we estimate the objective function value $\Phi(S)$ for such a schedule S.

Since $w_1 \geq 0$, $w_2 \geq 0$, and the equality $w_1 + w_2 = 1$ holds, then at least one of the weights w_1 or w_2 is not equal to zero in the objective function $\Phi(S) = w_1 \sum_{J_i \in \mathcal{J}(S)} b_i + w_2|\mathcal{J}(S)|$.

If $w_1 > 0$, then using inequality $b_i > 0$, we obtain

$$w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i \geq w_1 \cdot b_i > 0.$$

If $w_2 > 0$, then we obtain $w_2 \cdot |\mathcal{J}(S)| \geq w_2 > 0$.

Thus, in both cases, we obtain a contradiction that the optimal value of the objective function $\Phi(S)$ is equal to zero. Thus, our assumption that there is a job $J_i \in \mathcal{J}$, which may be processed within the interval $[r_i, d_i]$, was wrong. This completes the proof.

Let us assume that the machines in the set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ are indexed in an non-decreasing order of their rates v_k, i.e., inequality $v_k \geq v_{k+1}$ holds for each index $k \in \{1, 2, \ldots, m-1\}$. Using this indexing, we show that testing the condition of Theorem 2 may be realized in $O(n)$ time.

Indeed, for testing the condition of Theorem 2, one can select a job J_i from the set $\mathcal{J}$ with the maximal difference $d_i - r_i$ (this takes $O(n)$ time). Since machine M_1 has the largest rate v_1, where $v_1 = \max\{v_k \,|\, M_k \in \mathcal{M}\}$, machine M_1 needs the minimal time equal to $p_{i1} = \frac{p_i}{v_1}$ for processing the job J_i, for which a time interval $[r_i, d_i]$ of maximal length is allowed for processing. From this argument, the following claim follows: *There is no job $J_i \in \mathcal{J}$, which can be processed within the interval $[r_i, d_i]$, if and only if* $p_{i1} = \frac{p_i}{v_1} \geq d_i - r_i$. As a result, one can verify the condition of Theorem 2 in $O(n)$ time provided that the machines in the set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ are indexed in an non-decreasing order of their rates v_k.

4 Three heuristic algorithms for solving the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$

In this section, a simulated annealing algorithm, a tabu search algorithm and a genetic algorithm are developed for solving the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$. These algorithms may start with any randomly generated heuristic solution to the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$. An investigation of the different aspects of the developed algorithms (like the number of iterations, the number of neighbors, the number of generations, and the running time) allowed the algorithms to increase the obtained values of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

4.1 A genetic algorithm

A genetic algorithm is an evolutionary one, which represents a solution (an individual) of the problem by a code known as a *chromosome* or *genome*. To imitate the evolution process, the chromosomes are combined and mutated to create a new individuals in each generation. In a crossover operation, an offspring's chromosome is created by joining segments chosen alternately from each of the two parents' chromosomes, which are of a fixed length. By recombining portions of good individuals (crossover), this process is likely to create a better individual. By a mutation operator, a few portions of the current generation will be selected and some transformations will occur in their genome. Algorithm 1 presents a pseudo code for the genetic algorithm to solve the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

Algorithm 1 Genetic algorithm

Require: Maximum iteration number (itr_{max}); number of chromosomes in each generation (n_{pop}); number $m = N$ of machines; list of jobs J; X percentage of crossover products; Y percentage of mutation products; Z percentage of elites in the new generation;

$n_{itr} \leftarrow 1$;// generation counter.
$C_{n_1} = null$; // $C_{n_{itr}}$ is a set of chromosomes in the generation n_{itr}.
for i=1 **to** n_{pop} **do**
  $q_c \leftarrow$ random-solution(J);
  $C_{n_{itr}} \leftarrow C_{n_{itr}} \cup (q_c, evaluate(q_c))$;
end for
while ($n_{itr} \leq itr_{max}$) **do**
  $n_{itr} \leftarrow n_{itr} + 1$;
  for i=1 **to** $n_{pop} \times X\%$ **do**
    $q_c \leftarrow$ crossover(roulette-wheel($C_{n_{itr}-1}$), roulette-wheel($C_{n_{itr}-1}$));
    $C_{n_{itr}} \leftarrow C_{n_{itr}} \cup (q_c, evaluate(q_c))$;
  end for
  for i=1 **to** $n_{pop} \times Y\%$ **do**
    $q_c \leftarrow$ mutate(roulette-wheel($C_{n_{itr}-1}$));
    $C_{n_{itr}} \leftarrow C_{n_{itr}} \cup (q_c, evaluate(q_c))$;
  end for
  $C_{n_{itr}} \leftarrow C_{n_{itr}} \cup elites(C_{n_{itr}-1}, Z\%)$; //Add to the new population $C_{n_{itr}}$, $Z\%$ of the best $C_{n_{itr}-1}$ chromosomes as elites directly.
  if not-changed(best-solution($C_{n_{itr}}$, $itr_{max} \times 10\%$)) **then**
    $exit()$;//if the best solution did not change for 10 % of n_{itr} iterations, then stop.
  end if
end while
return best-solution(n_{itr});

Several parameters must be set in the genetic algorithm, which are as follows: the number of chromosomes in the first generation n_{pop}, the probability of the crossover operation p_c, the probability of the mutation operation p_m, the maximal number of allowed generations itr_{max}.

In our experiments, the number of chromosomes in the first generation was 500, the probability of the crossover operation was $p_c = 0.6$, the probability of the mutation operation was $p_m = 0.04$, and the maximal

number of iterations was 1000. The first generation consists of n_{pop} chromosomes. The length of a chromosome n_{var} was equal to the sum of the number of jobs and the number of machines minus one: $n_{var} = n+m-1$. The numbers between one and n_{var} were randomly distributed in the chromosome.

After the first step, a fitness function is calculated. For selecting the candidate parents, a roulette wheel function is used. A two-point crossover is used in the developed genetic algorithm. After that an error covering function is used to repair the damaged chromosomes in the crossover function. A swap function is used for the mutation, where two places in the chromosome are selected randomly and the code inside these places is exchanged.

The natural numbers in the chromosome are between 1 and n and represent the index i of the corresponding job $J_i \in \mathcal{J}$: $1 \leq i \leq n$. If the number g in a chromosome is larger than n, $g > n$, then this number g determines the index k of the used machine M_k as follows: $k = g - n$. Thus, the largest gene is restricted by the number $n + m$.

The set $\mathcal{J}(S)$ of jobs is processed in the schedule S, i.e., the inequalities $r_i \leq s_{i,k}(S)$ and $c_{i,k}(S) \leq d_i$ hold for each job $J_i \in \mathcal{J}(S)$. To maximize the first summand $w_1 \sum b_i(S)$ of the objective function Φ, it is necessary to construct a schedule S with the maximal value $\sum b_i(S) = \sum_{J_i \in \mathcal{J}(S)} b_i$. The maximization of the second summand $w_2|\mathcal{J}(S)|$ of the objective function Φ is connected with the most customer satisfaction via increasing the number $|\mathcal{J}(S)|$ of the jobs $\mathcal{J}(S)$ to be completely processed in the schedule S. Therefore, a final fitness is calculated as follows:

$$Fitness = w_1 \cdot \sum_{J_i \in \mathcal{J}(S)} b_i + w_2 \cdot |\mathcal{J}(S)|,$$

which corresponds to the value of the objective function.

The genetic algorithm is continued up to the maximum number of generations (itr_{max}). However, if the obtained best schedule is not improved within a specified number of generations the genetic algorithm may be stopped.

In the computational experiments, the weight $w_1 = 0.7$ for revenue maximization and the weight $w_2 = 1 - w_1 = 0.3$ for the number $|\mathcal{J}(S)|$ of jobs being processed in the schedule S were used.

4.2 A simulated annealing algorithm

Due to its simplicity, a simulated annealing algorithm is often used in optimization. This algorithm starts from a single point in the solution space (a heuristic schedule S) and tries to investigate the solution space in order to find a better schedule S' [Carter and Price]. If the newly constructed schedule S' will have a larger objective function value than the previous

best schedule S, then the schedule S' will be used as the new best schedule ($S \longleftarrow S'$). If the new schedule S' is not better than the previous best schedule S, then the schedule S will be accepted with the probability $P(S, S', T)$. After each step, the system cools down by reducing the "temperature" and the probability $P(S, S', T)$, which is determined for the acceptance of a new schedule.

The pseudo-code for the simulated annealing algorithm for solving the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ is presented as Algorithm 2.

Algorithm 2 The pseudo code for simulated annealing algorithm to solve the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

Require: Number of new neighbors NN; start temperature T_0; final temperature T_f; maximal number of iterations itr_{max}; number $m = N$ of machines; list of jobs J;

```
n_itr ← 1;// iteration counter.
T ← T_0;
S ← random-solution(J);
while (n_itr ≤ itr_max) do
   n_itr ← n_itr + 1;
   S' ← swap(S);//or reversion(S);
   ΔF ← (Fit(S') − Fit(S))/Fit(S);
   if (ΔF < 0) then
      S ← S';
   else
      r ← random(0, 1);
      P_r ← e^(−ΔF)/T;
      if (r < P_r) then
         S ← S';
      end if
   end if
   T_rf = (T_0 − T_f)/itr_max;
   T = T − T_rf;//controlling the temperature;
   if not-changed(S, itr_max × 10%) then
      exit();//if the solution did not change in a specified number of iterations (e.g., for 10
      % of n_itr iterations), then exit.
   end if
end while
return S;
```

Several parameters must be chosen in the simulated annealing algorithm before its execution. These parameters are the number of new neighbors NN per iteration, the start "temperature" T_0, the final "temperature" T_f, the procedure for the "temperature" reduction T_{rf}, and the maximal number of iterations itr_{max} allowed for the simulated annealing algorithm.

Swap procedures are used to generate a new neighbor. After generating NN neighbors, the best solution will be selected as the new schedule S'. Then the acceptance probability $P(S, S', T)$ will be checked as follows:

$$E = (best\_new\_sol.fit - pop(i).fit) \ / \ pop(i).fit$$

$$P_r = e^{(-E)/T}.$$

After this step, a random number between zero and one will be selected and if this number is less than P_r, the constructed schedule will be accepted. After finishing this step, the "temperature" will be reduced to cool down as follows:

$$T := T - T_{rf},$$

where we use the following reduction factor:

$$T_{rf} = (T_0 - T_f)/itr_{max}.$$

The algorithm is continued until the maximal number (itr_{max}) of allowed iterations is reached. However, if the obtained best schedule is not improved within a specified number of iterations, the simulated annealing algorithm may be stopped.

4.3 A tabu search algorithm

The tabu search algorithm is a meta-heuristic algorithm introduced by Glover [Glover] in 1986. This algorithm starts from an initial solution S and like a simulated annealing algorithm, it determines a best neighbor solution S'. If the new solution S' is not in the tabu list, it will be selected as the current solution: $S \leftarrow S'$. Otherwise, the algorithm will check the acceptance condition. Based on this acceptance condition, if the new neighbor of the schedule S' is better than the current solution S, then the algorithm will accept the new solution S' as the current solution, although it may be in the tabu list.

Actually, the tabu list is used to escape from a local optimum. When a move is added to the tabu list, some other moves will be removed from the tabu list. A parameter called tabu list TL will set the the number of iterations that a move must be in the tabu list. This move to the new neighbor will continue until the stopping condition is satisfied. After the realization of each move, the tabu list will be updated by adding this new move in order to prevent the algorithm from returning to a previous solution (a loop prevention).

A pseudo code for the tabu search algorithm to solve the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ is presented as Algorithm 3.

The number of neighbors NN per iteration, the restriction of moves in the tabu list lim_t, and the maximum iteration number itr_{max} are three important parameters in the tabu search algorithm. The tabu moves are the moves that have been used in the past, if the algorithm reached an upper limit for the number of these moves in the tabu list, the elements with an index higher than one, $TL > 1$, are tabu moves. If a move is added to the list of tabu moves, the parameter lim_t shows that in the next lim_t iterations this move is forbidden.

For example, by assigning $lim_t = 5$ for a move, in the next five iterations that move is forbidden. In our tabu search algorithm, we assume $NN = 1000$, $itr_{max} = 1000$ and the parameter lim_t is calculated as follows: $lim_t = \lceil itr_{max} \times 0.05 \rceil$.

Algorithm 3 The pseudo code for tabu search algorithm to solve the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

Require: Number of permutation neighbors, NN; the maximal number lim_t of forbidden moves in the tabu list; maximal iteration number, itr_{max}; number $m = N$ of machines; list of jobs J;

```
n_itr ← 1; //iteration counter.
set neighbors ← NULL; //set of neighbors.
S_now ← random-solution(J);
clear-history(H);
while (n_itr ≤ itr_max) do
  n_itr ← n_itr + 1;
  for i=1 to NN do
    S' ← swap(S_now); //or reversion(S_now);
    neighbors ← neighbors ∪ S';
  end for
  for all S_candid ∈ neighbors do
    if (Fit(S_candid) < Fit(S_new)) and not (S_candid ∈ H) then
      S_new ← S_candid;
    end if
  end for
  if (Fit(S_new) < Fit(S_now)) then
    S_now ← S_new;
    update-history(H, S_now);
  end if
  if not-changed(S_now, itr_max × 10%) then
    exit(); //if the best solution did not changed in the sufficient number of iterations (e.g.,
    for 10 % of n_itr iterations), then exit.
  end if
end while
return S_now;
```

The initial solution is generated heuristically. Then the tabu search algorithm creates some neighbors (by applying the swap function). After generating NN neighbors, the best allowed neighbor and the best forbidden neighbor are selected. If the best allowed neighbor is better than the best forbidden neighbor, it will be selected as a new solution (even if it is not better than the current solution).

On the other hand, if the best forbidden neighbor is better than both the best allowed neighbor and the best known solution, then it will be accepted as a new solution. If all the generated solutions are in the tabu list, the best neighbor will be selected as a new solution. After exchanging the current solution with its neighbor, the tabu list must be updated. It means that the previous move action that made this new solution will be added to the tabu list. This insertion helps the algorithm to prevent the current solution to go back to its previous state. Then, the indexes of all moves in the tabu list will be decreased by one unit (non-zero indexes are forbidden moves, zero indexes are free moves) as follows:

$$TL_i := \max\{TL_i - 1,\ 0\}.$$

After this modification, the new move will be added to the tabu list with the following index considered for limitation:

$$TL_k(sol_{m1}, sol_{m2}) = Limit;$$

$$TL_l(sol_{m2}, sol_{m1}) = Limit.$$

The execution of the algorithm is continued until it reaches the maximal number itr_{max} of iterations, but if the previous best schedule is not improved during a predetermined number of generations, the procedure may be stopped.

5 Computational results and discussion

To evaluate the efficiency of the developed algorithms (genetic algorithm, simulated annealing, and tabu search), 20 instances of the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ with the same medium sizes $n \times m$ with $n = 50$ and $m = 5$ have been randomly generated. To compare the algorithms for large scheduling problems arising in practice, 10 large instances with up to $n = 500$ jobs and $m = 50$ parallel uniform machines have been randomly generated.

The moderate instances were numbered as $1, 2, \ldots, 20$ and the large ones as $21, 22, \ldots, 30$. Each input data for the moderate instance consists of $n = 50$ jobs $J_i \in \mathcal{J} = \{J_1, J_2, \ldots, J_{50}\}$ that must be processed on $m = 5$ parallel uniform machines. The production rates (machine speeds) $v_k \in \{0.5, 0.8, 1, 1.1, 1.3\}$ were randomly selected for the machines $M_k \in \mathcal{M} = \{M_1, M_2, \ldots, M_5\}$.

The 20 randomly generated input data for the moderate instances $1, 2, \ldots, 20$ were categorized into four classes with different characteristics. The intervals for the processing times of the jobs $J_i \in \{J_1, J_2, \ldots, J_{50}\}$, and the intervals for their release times r_i and due dates d_i are given in Table 1.

In the first class (instances 1 – 5), the release times and due dates are tight. In the second class (instances 6 – 10), the release times are loose while the due dates are tight. In the third class (instances 11 – 15), the release times are tight while the due dates are loose. In the fourth class (instances 16 – 20), the release times and due dates are loose. The job processing times were randomly chosen from the set of integers $\{1, 2, \ldots, 10\}$. The release times are generated randomly in the allowed intervals, given in Table 1.

The due date was determined as the sum of the release time, the processing time, and a random number in the allowed positive ranges, i.e., $d_i = r_i + p_i + random.range()$. From this it follows that the condition of Theorem 2 does not hold for any instance tested in the computational experiments. Therefore, the optimal value of the objective function $\Phi(S)$ is strictly positive, $\Phi(S) > 0$, for any tested instance of the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

Table 1 Characteristics of four classes of the input data for the problems with 50 jobs and 5 machines.

Problem class	Processing times	Release times	Due dates	Job benefits
1	[1–10]	[0–40]	[1–10]	[1-20]
2	[1–10]	[0–70]	[1–10]	[1-20]
3	[1–10]	[0–40]	[1–20]	[1-20]
4	[1–10]	[0–70]	[1–20]	[1-20]

W=(1 0) n_{pop}=100 P_c=0.4 P_m=0.6

Obj 1	Obj 2
415	34
406	32
416	32
398	30
361	27
402	30
424	33
413	33
418	32
428	33

W=(0 1) n_{pop}=1000 P_c=0.6 P_m=0.8

Obj 1	Obj 2
430	40
443	44
429	44
423	44
395	39
471	44
392	40
420	40
439	44
425	43

W=(0 1) n_{pop}=1000 P_c=0.8 P_m=1

Obj 1	Obj2
483	46
481	45
427	43
425	42
450	45
448	44
450	42
488	46
476	45
453	43

W=(0.7 0.3) n_{pop}=1000 P_c=1 P_m=1

Obj 1	Obj 2
507	47
470	39
499	44
478	40
492	42
488	45
496	43
488	46
476	45
496	43

W=(1 0) n_{pop}=1000 P_c=1 P_m=1

obj1	obj2
491	42
492	42
491	42
476	38
498	44
487	43
507	47
498	43
491	43
502	46

Fig. 1 The quality of schedules with different input parameters for the genetic algorithm.

The computational experiments were realized on the computer with the following characteristics: Pentium 4, Dual Core, 1.8 G.Hz, and Ram 4 GB.

5.1 Finding suitable parameters of the algorithms

Different aspects of the developed algorithms like the number of iterations, the number of neighbors, the population size, the running time (CPU-time), and the quality of schedules were monitored in the preliminary experiments in order to increase the objective function values of the schedules constructed by the developed heuristic algorithms.

The preliminary computational results for the genetic algorithm confirm that increasing the population size (n_{pop}) and mutation probability (p_m) cause the construction of the better schedules. Increasing the population size (n_{pop}) until 1000 parents could improve the quality of constructed schedules, but after a specified limit of generations, it had no further effect. In Fig. 1, the first summand ($Obj1 = w_1 \sum b_i$) and the second summand ($Obj2 = w_2|\mathcal{J}(S)|$) of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)| = Obj1 + Obj2$ are presented for ten runs of the genetic algorithm with different input parameters.

Table 2 The computational results for the instances with $(w_1, w_2) = (1, 0)$ and $(w_1, w_2) = (0, 1)$ achieved by the simulated annealing algorithm with $n_{itr} = 100$, $NN = 100$, $T_0 = 100$ and $T_f = 0$.

$(w_1, w_2) = (1, 0)$				$(w_1, w_2) = (0, 1)$			
$\sum b_i$	J_{ok}	CPU-time	n_{itr}	$\sum b_i$	J_{ok}	CPU-time	n_{itr}
502	45	2572	539	495	47	2802	548
496	43	2485	638	456	45	1729	384
499	44	4985	661	455	46	1901	436
495	43	1543	389	472	46	1421	335
499	44	2005	500	455	45	1627	406

For tuning the simulated annealing algorithm, the numbers NN of neighbors per iteration were started from 5 and the number of iterations n_{itr} was started from 10. A lot of repeated executions of this algorithm showed that the best variants for the number NN of neighbors and the number n_{itr} of iterations are both 100. This means that by increasing these numbers, the quality of the schedule is not considerably changed while the running time of the algorithm is increased.

From the preliminary computational experiments, it follows that there is a high correlation between the two summands $w_1 \sum b_i$ and $w_2|\mathcal{J}(S)|$ of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$ provided that the weights w_1 and w_2 are close. This means that, by increasing the number

$$|\mathcal{J}(S)| := J_{ok}$$

of the jobs $J_i \in \mathcal{J}(S)$ processed in the given time intervals $[r_i, d_i]$ in the schedule S, it is expected to observe an increase of the benefit as well. Similarly, for increasing the benefit, the number J_{ok} of the jobs $\mathcal{J}(S)$ increases as well, if the weights w_1 and w_2 are close.

However, this correlation is not often true, if the weights w_1 and w_2 are more different one from another. In Table 2, the results from the experiment with $n_{itr} = 100$, $NN = 100$, $(w_1, w_2) = (1, 0)$ and $(w_1, w_2) = (0, 1)$ are presented. By comparing these cases, it can be observed that a larger benefit may come when more jobs are rejected from the set $\mathcal{J}(S)$.

5.2 Discussion of the computational results

The maximal number itr_{max} of the allowed iterations (generations) of the heuristic algorithm is an important factor for the quality of the best constructed schedule. Our computational experiments showed that for the genetic algorithm, after 200 generations the value of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$ for the best constructed schedule S is not increased in a meaningful manner (see Fig. 2). For the other two tested algorithms, a bound also exists such that increasing the number itr_{max} of the allowed iterations more than this bound has no further effect on the quality of the obtained schedule while the CPU-time being increased.

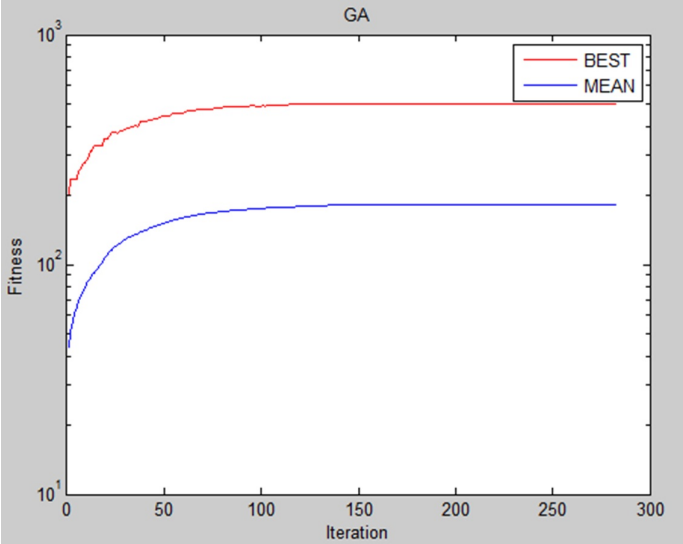

Fig. 2 The quality of the objective function values after each iteration for the genetic algorithm.

The rate of changes in the value of objective function $\Phi(S)$ after each iterations was measured. Different executions of the tested algorithms showed that a suitable number of iterations for the simulated annealing algorithm and the tabu search one was about 500 (see Fig. 3 and Fig. 4). In Fig. 2, for the tabu search algorithm, the mean and best objective function values are the same since this algorithm deals with only one solution of the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ at each iteration.

The next factor for evaluating the quality of the developed algorithms is the probability to get a good schedule in each execution of the algorithm. The computational results for ten executions of the three tested algorithms for the same instance 1 from the first class (instances 1 – 5) is presented in Table 3.

It can be seen that the dispersion of the results in the simulated annealing algorithm is lower than for the other two tested algorithms. For example, if a desired benefit $w_1 \sum b_i$ of more than 495 is sufficient for the user, then the simulated annealing algorithm could achieve 8 times this sufficient results in 10 executions of the algorithm. The genetic algorithm could achieve 5 times this sufficient result in 10 executions of the algorithm. The tabu search algorithm could achieve 4 times this sufficient results in 10 executions of the algorithm. Thus, we can conclude that for the tested moderate instances, *the probability of achieving an appropriate schedule by the simulated annealing algorithm is higher.*

The other difference between the tested algorithms is the CPU-time needed to find a good schedule. For the instance 1, from the first class, all algorithms could achieve a benefit of $\sum b_i \geq 502$ and we obtain a number of the completed jobs $|\mathcal{J}(S)| \geq 45$ for the best constructed schedule.

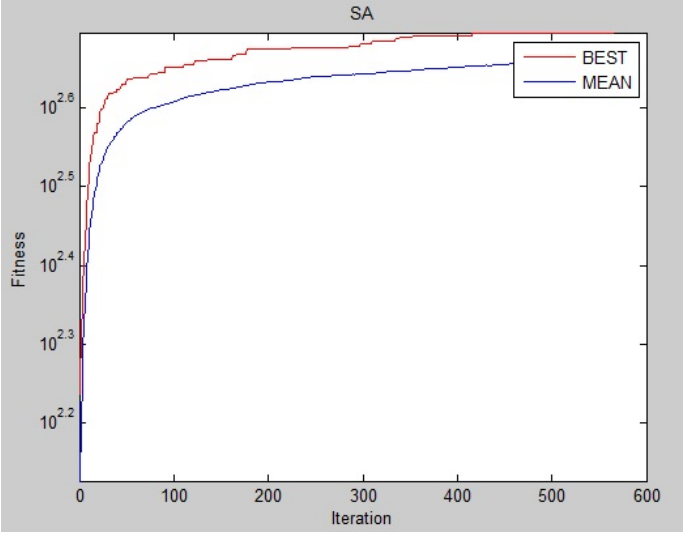

Fig. 3 The quality of the objective function values after each iteration for the simulated annealing algorithm.

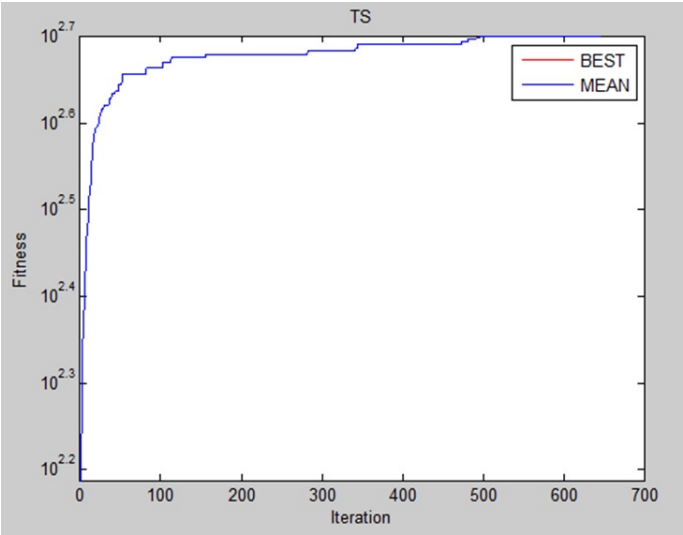

Fig. 4 The quality of the objective function values after each iteration for the tabu search algorithm.

However, such a schedule was constructed after 385 seconds by the genetic algorithm, after 370 seconds by the tabu search algorithm, and after 2572 seconds by the simulated annealing algorithm (almost 6 times more time than by the other two tested algorithms). Note that such results were achieved after 283 generations of the genetic algorithm, 647 iterations of

Table 3 The computational results obtained in ten runs of the three algorithms for instance 1.

	Genetic algorithm				*Simulated annealing*				*Tabu search*			
Runs	$\sum b_i$	J_{ok}	CPU-time	n_{itr}	$\sum b_i$	J_{ok}	CPU-time	n_{itr}	$\sum b_i$	J_{ok}	CPU-time	n_{itr}
1	491	42	294	264	495	43	*1279*	614	502	45	370	648
2	492	42	307	262	488	42	*1162*	576	471	38	370	761
3	498	44	334	395	496	44	*1733*	543	491	42	213	421
4	487	42	404	347	472	46	*1421*	335	483	40	220	452
5	488	42	36	282	495	<u>47</u>	*2802*	548	500	45	317	627
6	495	43	298	250	502	45	*2572*	539	484	40	328	385
7	502	46	720	227	496	43	*2485*	638	506	<u>47</u>	884	1000
8	499	44	416	371	499	44	*4985*	661	495	43	552	388
9	<u>507</u>	<u>47</u>	315	279	495	43	*1543*	389	444	34	404	441
10	491	43	269	237	499	44	*2005*	500	491	42	669	462

the tabu search algorithm, and 539 iterations of the simulated annealing algorithm.

In Table 2, the largest CPU-time (in seconds) used for running the three algorithms are given in italic. Similar relations on the CPU-times were observed for the other solved moderate instances from the second class (instances 6 – 10), third class (instances 11 – 15) and fourth class (instances 16 – 20) (see Table 4) and for all large instances 21 - 30 (Table 5).

The quality of the results for the solved instances were also measured. The algorithms were tuned for the best performance based on the preliminary computational experiments. Table 4 represents the best results from ten runs for 20 moderate instances by the genetic algorithm, simulated annealing, and tabu search. By comparing the results from simulated annealing, tabu search and the genetic algorithm, it can be seen that all three algorithms are rather efficient for several solved instances. For example, the genetic algorithm has a large value J_{ok} for instance 1 and instance 2. Simulated annealing was the best for solving instance 5 and instance 13. Tabu search was the best for solving instance 4 and instance 15.

In Tables 2 – 4, the best obtained values $\sum b_i$ and $|\mathcal{J}(S)| = J_{ok}$ are underlined. In Table 4, the obtained largest (i.e., optimal) values $|\mathcal{J}(S)| = J_{ok} = |\mathcal{J}| = 50$ are given in bold fashion. Due to Theorem 1, equality $|\mathcal{J}(S)| = |\mathcal{J}|$ implies that the benefit $\sum b_i$ is also the largest (optimal) for such a schedule S. These optimal values of the benefits $\sum b_i$ are given in bold fashion in Table 4.

We compared how many times each tested algorithm constructed schedules with the best values $\sum b_i$ and $|\mathcal{J}(S)| = J_{ok}$. The optimal value $J_{ok} = 50$ was obtained by the genetic algorithms 6 times (instances 9, 16 – 20), 7 times (instances 9, 10, 16 – 20) by the simulated annealing algorithm, and 7 times by the tabu search algorithm (instances 8, 9, 16 – 20). From Table 4, it follows that the simulated annealing algo-

Table 4 The results obtained for 20 moderate instances by the genetic algorithm ($n_{itr} = 1000$), the simulated annealing algorithm ($n_{itr} = 50$ and $NN = 100$), and the tabu search algorithm ($NN = 1000$ and $lim_t = 50$).

	Genetic algorithm				*Simulated annealing*				*Tabu search*			
Problem	$\sum b_i$	J_{ok}	CPU-time	n_{itr}	$\sum b_i$	J_{ok}	CPU-time	n_{itr}	$\sum b_i$	J_{ok}	CPU-time	n_{itr}
1	507	47	315	279	502	47	2572	539	506	47	884	1000
2	554	48	657	295	546	45	1526	669	551	48	299	535
3	525	42	1010	362	525	44	2028	897	524	42	611	608
4	561	45	584	282	561	45	1716	831	565	45	297	523
5	557	47	351	292	560	47	2501	884	557	47	341	576
6	555	47	367	231	555	47	1415	633	557	44	477	890
7	501	45	475	258	505	46	1770	670	505	46	280	419
8	411	48	422	272	411	48	1465	682	**413**	**50**	525	767
9	**506**	**50**	359	282	**506**	**50**	1768	680	**506**	**50**	437	607
10	551	49	559	266	**552**	**50**	1547	571	551	49	221	454
11	526	47	590	288	529	47	1146	587	529	47	340	634
12	573	46	790	307	573	46	2781	699	572	46	401	698
13	512	48	761	337	513	49	1486	452	511	48	303	480
14	566	48	644	239	566	49	1969	510	566	49	432	671
15	485	44	655	260	487	45	1440	436	492	47	399	589
16	**540**	**50**	447	206	**540**	**50**	819	279	**540**	**50**	266	363
17	**515**	**50**	385	206	**515**	**50**	710	229	**515**	**50**	327	454
18	**510**	**50**	589	243	**510**	**50**	1249	291	**510**	**50**	242	405
19	**514**	**50**	601	262	**514**	**50**	1226	347	513	49	196	376
20	**564**	**50**	483	214	**564**	**50**	1264	373	**564**	**50**	207	393

rithm constructed schedules with the best value $\sum b_i$ and (or) best value $|\mathcal{J}(S)| = J_{ok}$ 31 times for the moderate instances 1 – 20, the tabu search algorithm 27 times, and the genetic algorithm 25 times.

Thus, we can conclude that for the tested moderate instances, *the simulated annealing algorithm outperforms the tabu search algorithm in the reached objective function values, while the tabu search algorithm slightly outperforms the genetic algorithm.* In general, all three tested algorithms could show their efficiency for the most moderate instances from the second and fourth classes.

5.3 Computational results for solving the large instances

The developed algorithms were tested on the instances 21 – 30 of the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ with large sizes. The obtained results are presented in Table 5. For the large instances, 5 times the best schedules were constructed by the tabu search algorithm, 4 times by the genetic algorithm, and one time by the simulated annealing algorithm (see Table 5).

We compared how many times each tested algorithm constructed schedules with the best values $\sum b_i$ and (or) $|\mathcal{J}(S)| = J_{ok}$ for the large instances 21 – 30. In Table 5, the best obtained values $\sum b_i$ and $|\mathcal{J}(S)| = J_{ok}$ are underlined. From Table 5, it follows that the simulated annealing algo-

Table 5 Comparison of the obtained results for 10 instances with large sizes.

Large problem	*Problem size* $n \times m$	*Genetic algorithm* J_{ok}	CPU-time	$\sum b_i$	*Simulated annealing* J_{ok}	CPU-time	$\sum b_i$	*Tabu search* J_{ok}	CPU-time	$\sum b_i$
21	100×5	49	718	652	46	2852	639	41	356	571
22	100×10	78	743	872	85	4092	892	85	1265	903
23	200×5	51	1441	831	48	6873	789	44	2026	729
24	200×10	85	1538	1274	88	7453	1316	80	1511	1209
25	200×15	120	2180	1644	115	8582	1624	114	3428	1625
26	200×20	139	2141	1780	145	10375	1807	148	4036	1842
27	300×25	176	5147	2592	180	12964	2573	199	5211	2788
28	300×30	195	5325	2579	199	12430	2593	228	4470	2807
29	500×40	269	8114	4094	270	26368	4012	283	8739	4198
30	500×50	318	8402	4451	319	20601	4385	337	9231	4573

rithm constructed schedules with the best value $\sum b_i$ and (or) the best value $|\mathcal{J}(S)| = J_{ok}$ three times for the large instances 21 – 30, the genetic algorithm 6 times, the tabu search algorithm 12 times. Therefore, for the tested large instances, *the tabu search algorithm outperforms the genetic algorithm, while the genetic algorithm outperforms the simulated annealing algorithm.*

There were no optimal values $J_{ok} = n$ obtained by the tested algorithms for the large instances 21 – 30 (see Table 5). Unfortunately, we have no exact algorithm to find a schedule S with the largest (optimal) value of the objective function $\Phi(S)$. Therefore, if $|\mathcal{J}(S)| < |\mathcal{J}|$, we cannot estimate how far the reached values of the objective function $\Phi(S)$ presented in Tables 2 – 5 are to the largest (optimal) values of the objective function $\Phi(S)$.

6 How to apply the developed algorithms and proven results

In a real-world company, the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ may arise in order to achieve the highest possible income and (or) increase the number of client requests, which are realized by the company in an acceptable time window (the planning horizon). The revenue maximization may be determined as an allocation of resources (in our case, the resources are parallel uniform machines) to realize the most suitable customer requests during the planning horizon in order to retrieve a payment (a corresponding job has to be processed by the company in order for realizing the customer request). Since any real-world company has a limited set of resources and if the stream of client requests exceeds the production capabilities of the company, some requests must be rejected as non-profitable ones.

One of the questions to be solved first in the company is connected with determining appropriate weights w_1 and w_2 of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$ provided that equality $w_1 + w_2 = 1$ holds. It should be noted that the increase of the first summand ($Obj1 = w_1 \sum b_i$)

of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)| = Obj1 + Obj2$ means that the company prefers to increase its revenue, while the number of clients of the company may be fixed or even decreased. On the other hand, the increase of the second summand ($Obj2 = w_2|\mathcal{J}(S)|$) of the objective function Φ means that the company prefers to increase the number of clients, while the revenue may be saved or even decreased.

As it is shown in the computational experiments, if the weights w_1 and w_2 are close to each other, there is a high correlation between the two summands $w_1 \sum b_i$ and $w_2|\mathcal{J}(S)|$ of the objective function $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$. In particular, increasing the number $|\mathcal{J}(S)|$ of the jobs $J_i \in \mathcal{J}(S)$ processed in the given time intervals $[r_i, d_i]$ in the schedule S, it is expected to observe an increase of the benefit. Similarly, for increasing the benefit, the number of the jobs $\mathcal{J}(S)$ increases as well, if the weights w_1 and w_2 are rather close. However, if the weights w_1 and w_2 are rather different, the correlation between these summands in the optimal schedule S is often low.

Based on the problems $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ with nested sets of jobs $\mathcal{J}$ (which correspond to the satisfied customer requests), we propose to use the following iterative scheme for achieving a higher income for the company and (or) a larger number of the clients, whose requests will be satisfied by the company during the planning horizon.

A decision maker has the option to select a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of the jobs, which correspond to a subset of the set $\mathcal{R} = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$ of all available customer requests, where $\lambda \geq n$. For each selected job J_i, the condition of Theorem 2 must be tested in order to avoid a consideration of the customer requests $\rho_i \in \mathcal{R}$, which cannot be realized within their allowed intervals $[r_i, d_i]$. To realize such tests faster, all the jobs corresponding to the set $\mathcal{R}$ of the available customer requests must be ordered in a non-increasing order of their differences $d_i - r_i$. As it was proven in Section 3.2, such a test can be realized in $O(n)$ time if the machines in the set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ are indexed in an non-decreasing order of their rates v_k, i.e., inequality $v_k \geq v_{k+1}$ holds for each index $k \in \{1, 2, \ldots, m-1\}$.

After selecting the job set $\mathcal{J}$, the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ must be solved for this set $\mathcal{J}$ of the jobs using one of the developed heuristic algorithms. If for the constructed schedule S, equality $\mathcal{J}(S) = \mathcal{J}$ holds, then this schedule S is optimal for the selected set $\mathcal{J}$ of the jobs (due to Theorem 1). However, the constructed schedule S may cause idle machinery. In such a case, if the strong inequity $\lambda > n$ holds, i.e., there are customer requests, which are not included into the set $\mathcal{J}$, the decision maker can enlarge the set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ via selecting appropriate costumer requests from the set $\mathcal{R}$ and adding the corresponding jobs to the set $\mathcal{J}$.

Let such an enlarged set of jobs be determined and denoted by $\mathcal{J}'$. Then the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ may be solved again but for the enlarged set of jobs $\mathcal{J}'$ provided that $\mathcal{J} \subset \mathcal{J}'$. Again, if for the

constructed schedule S, equality $\mathcal{J}(S) = \mathcal{J}$ holds, then this schedule S is optimal for the set $\mathcal{J}'$ of the jobs.

The above process may be continued until the inequality $\mathcal{J}^*(S) < \mathcal{J}^*$ will be reached for the first time. Note that it is useful to solve the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ for the last largest set $\mathcal{J}^*$ again but using a more efficient algorithm than the one used at the previous iterations. A more efficient algorithm can provide a schedule S, which is either optimal or at least close to optimal schedule. Of course, the process of solving the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ using a more efficient algorithm may be more time-consuming.

The above scheme may be used on a higher level of the planing and scheduling, e.g., when it is necessary to achieve the highest income and (or) increase the number of client requests, which are realized by the company including m factories. In such a case, a factory corresponds to one of the m parallel uniform machines.

In the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$, all parameters of the company are assumed to be fixed in advance. However, in a real-world company, there exist parameters, which are uncertain in nature. Therefore, the above proposed scheme needs to be extended by some tools in order to take into account the uncertainty for revenue maximization of the company based on a better planing and scheduling.

7 Conclusion

The problem of maximizing the weighted sum of the benefits $w_1 \sum_{i \in \mathcal{J}(S)} b_i$ and the number of jobs $w_2|\mathcal{J}(S)|$ processed in a schedule S was investigated, where a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ has to be processed on a set of parallel uniform machines. For each job $J_i \in \mathcal{J}$, a release times r_i and a due date d_i, $d_i > r_i$, are given. Each job J_i has to be processed on one of the m uniform parallel machines. If a job J_i will be processed (i.e., started and completed) within the interval $[r_i, d_i]$, then a benefit b_i will be earned. Otherwise, this job will be rejected from the system and the benefit b_i will be discarded. If $\mathcal{J}(S)$ denotes the subset of jobs $\mathcal{J}$ processed in the schedule S. A genetic algorithm, a simulated annealing algorithm and a tabu search algorithm were developed for solving this problem.

We investigated the influence of different parameters on these algorithms, e.g., the number of generations, the number of generated neighbors per iteration, the population size for the genetic algorithm, the running time of the algorithm. The developed algorithms are compared to each other with respect to the obtained objective values $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$.

As a future research, it would be useful to find upper and lower bounds on the objective function value $\Phi = w_1 \sum b_i + w_2|\mathcal{J}(S)|$ for the considered problem. Another interesting research area is connected with other criteria for scheduling a set of jobs optimally on parallel uniform machines. De-

veloping an exact algorithm for the scheduling problem with parallel uniform machines is highly appreciated in order to compare the schedules obtained by the heuristic algorithms with the optimal schedules. Since most practical scheduling problems have uncertain parameters, it would be useful to investigate the problem $Q|r_i, d_i|w_1 \sum b_i + w_2|\mathcal{J}(S)|$ with uncertain job processing times similarly as it was realized in [Sotskov and Lai] for scheduling a set of jobs on a single machine.

References

[Abdolzadeh] M. Abdolzadeh, H. Rashidi, 2010. An approach of cellular learning automata to job-shop scheduling problem, International Journal of Simulation: Systems, Science and Technology, 11, 56-64.

[Anagnostopoulos and Rabadi] G. Anagnostopoulos, G. Rabadi, 2002. A simulated annealing algorithm for the unrelated parallel machine scheduling problem, Automation Congress, 2002 Proceedings of the 5th Biannual World, 14, 115-120.

[Anglani et al.] A. Anglani, A. Grieco, E. Guerriero, R. Musmanno, 2005. Robust scheduling of parallel machines with sequence-dependent set-up cost, European Journal of Operational Research, 161 (3), 704-720.

[Balin-2011] S. Balin, 2011. Non-identical parallel machine scheduling using genetic algorithm, Expert Systems with Applications, 38, 6814-6821.

[Balin-2012] S. Balin, 2012. Non-identical parallel machine scheduling with fuzzy processing times using robust genetic algorithm and simulation, International Journal of Innovative Computing, Information and Control, 8(1-B), 727-745.

[Berrichi and Yalaoui] A. Berrichi, F. Yalaoui, 2013. Efficient bi-objective ant colony approach to minimize total tardiness and system unavailability for a parallel machine scheduling problem, The International Journal of Advanced Manufacturing Technology, 68, 2295-2310.

[Bilge et al.] U. Bilge, F. Kirac, M. Kurtulan, P. Pekgun, 2004. A tabu search algorithm for parallel machine total tardiness problem, Computers and Operations Research, 31, 397-414.

[Brucker et al.] P. Brucker, Y.N. Sotskov, F. Werner, 2007. Complexity of shop-scheduling problem with fixed number of jobs: a survey, Mathematical Methods of Operations Research, 65(3), 461-481.

[Carter and Price] M.W. Carter, C.C. Price, 2001. Operations research: A practical introduction, Textbook, CRC Press, Boca Raton, 416.

[Dawande et al.] M. Dawande, I. Drobouchevitch, T. Rajapakshe, C. Sriskandarajah, 2010. Analysis of revenue maximazation under two movie-screening policies, Production and Operations Management, 19(1), 111-124.

[Dunstall and Wirth] S. Dunstall, A. Wirth, 2005. Heuristic methods for the identical parallel machine problem with set-up times, Computers & Operations Research, 32, 2479-2491.

[Feng et al.] G. Feng, S. Garg, R. Buyya, W. Li, 2012. Revenue maximization using adaptive resource provisioning in cloud computing environments, 2012 ASM/IEEE 13th International Conference in Grid Computing, IEEE Computer Society, 192-200, DOI 10.1109/Grid.2012.16.

[Feng and Lau] G. Feng, H.C. Lau, 2008. Efficient algorithm for machine scheduling problems with earliness and tardiness penalties, Annals of Operations Research, 159(1), 83-95.

[Glover] F. Glover, 1986. Future Paths for Integer Programming and Links to Artificial Intelligence, Computers and Operations Research, 13(5), 533-549.

[Graham et al.] R.E. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, 1979. Optimization and approximation in deterministic sequencing and scheduling a survey, Annals of Discrete Mathematics, 4, 287-326.

[Islam et al.] M. Islam, G. Khanna, P. Sadayappan, 2008. Revenue maximization in market-based parallel job schedulers, Technical Report, Ohio State University, OSU-CISRC-4/08-TR16, Ohio, USA, 1-13.

[Juraszek et al.] J. Juraszek, M. Sterna, E. Pesch, Revenue maximization on parallel machines, Institute of Computing Science, Poznan University of Technology Piotrowo 2, Poznan, Poland, 60-965.

[Li and Yang] Kai Li, Shan-lin Yang, 2009. Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms, Applied Mathematical Modelling, 33(4), 2145-2158.

[Lin and Lin] Y.K. Lin, C.W. Lin, 2013. Dispatching rules for unrelated parallel machine scheduling with release dates, International Journal of Advanced Manufacturing Technology, 67(1-4), 269-279.

[Logendran et al.] R. Logendran, B. McDonell, B. Smuckera, 2007. Scheduling unrelated parallel machines with sequence-dependent setups, Computers & Operations Research, 34, 3420-3438.

[Moghaddam et al.] R. Tavakkoli-Moghaddam, F. Taheri, M. Bazzazi, 2008. Multi-objective unrelated machines scheduling with sequence-dependent setup times and precedence constrains, IJE Transactions A: Basics, 21(3), 269-278.

[Rodriguez et al.] F.J. Rodriguez, C. Blum, C. Garcia-Martinez, M. Lozano, 2012. GRASP with path-relinking for the non-identical parallel machine scheduling problem with minimising total weighted completion times, Annals of Operations Research, 201(1), 383-401.

[Sivrikaya and Ulusoy] F. Sivrikaya-Serifoglu, G. Ulusoy, 1999, Parallel machine scheduling with earliness and tardiness penalties, Computers and Operations Research, 26, 773-787.

[Shubin and Bean] S. Xu, J.C. Bean, 2007. A genetic algorithm for scheduling parallel non-identical batch processing machines, Computational Intelligence in Scheduling, SCIS '07. IEEE Symposium, 143-150.

[Sotskov and Lai] Y.N. Sotskov, T.-C. Lai, 2012. Minimizing total weighted flow time under uncertainty using dominance and a stability box, Computers and Operations Research, 39(6), 1271-1289.

[Sterna et al.] M. Sterna, J. Juraszek, E. Pesch, 2008. Revenue maximization on parallel machines, Book Chapter, Operations Research Proceedings, 153-158.

[Supithak and Plongon] W. Supithak, K. Plongon, 2011. Memetic algorithm for non-identical parallel machines scheduling problem with earliness and tardiness penalties, Int. J. of Manufacturing Technology and Management, 22(1), 26-38.