Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time

Mikhail Batsyn, Boris Goldengorin, Panos M. Pardalos, and Pavel Sukhov

1 Laboratory of Algorithms and Technologies for Networks Analysis
National Research University Higher School of Economics
136 Rodionova street, Nizhny Novgorod 603093, Russia
2 Department of Industrial and Systems Engineering, University of Florida
303 Weil Hall, Gainesville, FL, 32611, USA
3 Department of Operations Management & Operations Research, University of Groningen
Nettelbosje 2, 9747 AE Groningen, The Netherlands

(v1.2 released June 2013)

The preemptive single machine scheduling problem of minimizing the total weighted completion time with arbitrary processing times and release dates is an important NP-hard problem in scheduling theory. In this paper we present an efficient high-quality heuristic for this problem based on the WSRPT (Weighted Shortest Remaining Processing Time) rule. The running time of the suggested algorithm increases only as a square of the number of jobs. Our computational study shows that very large size instances might be treated within extremely small CPU times and the average error is always less than 0.1%.

Keywords: single machine scheduling; weighted shortest remaining processing time; WSRPT rule; efficient heuristic

AMS Subject Classification: 90B35; 90C59; 68M20

1. Introduction

In this paper we present an efficient heuristic which returns high-quality solutions for the preemptive single machine scheduling problem of minimizing the total weighted completion time with arbitrary processing times and release dates. The problem is defined as $1|\text{pmtn}; \text{r}j| \sum w_jC_j$ in Graham's notation [9]. The suggested heuristic is flexible and can potentially be applied to more complex scheduling problems with many constraints. It is very efficient and can be performed many times for solving subproblems in metaheuristic approaches and exact branch-and-bound, branch-and-cut, branch-and-price, data correcting [15], tolerance-based [14], and other enumeration type algorithms (see e.g. [16]). It is also applicable for very large size instances which cannot be treated by either general purpose software or the most efficient available specialized algorithms [1]. The considered $1|\text{pmtn}; \text{r}j| \sum w_jC_j$ problem is known to be NP-hard [18], and solving it for large number of jobs or long processing times might be CPU time consuming.

Heuristic approaches for theoretical problems are also important for more complicated practical problems in scheduling theory. For instance, there are a lot of single machine approaches that are successfully applied in multi-machine environment [26], [23]. There are many examples when solving of an NP-hard scheduling problem is reduced to solving of many relaxed problem instances by means of exact enumeration algorithms [25],[21].
Calculation of an exact solution can be significantly sped up by using a good heuristic in branch-and-bound approaches [24]. Heuristic algorithms for not complicated scheduling problems usually have a guaranteed analytical estimation of their accuracy [8], [27].

Many practical applications of scheduling problems has been indicated by the so-called online scheduling algorithms [28]. Nowadays online scheduling algorithms are of significant importance in the field of manufacturing and service industries due to the volatile competitive industrial environment. In the online scheduling environment new jobs appear at random time moments unknown beforehand. The number of jobs is not known in advance, and no information is known about any future jobs.

Our heuristic is an online scheduling algorithm. It applies the weighted shortest remaining processing time (WSRPT) rule to find high-quality solutions to the \(1\mid pmtn; r_j\mid \sum w_j C_j\) scheduling problem. This means that the schedule is built consequentially and for every time moment we take the job which currently has the shortest weighted remaining processing time among all the jobs available at this time moment. The computational study shows that our WSRPT heuristic finds solutions extremely close to an optimal one.

The suggested algorithm is also applicable to scheduling problems with availability constraints like the \(1; NC\mid pmtn; r_j\mid \sum w_j C_j\) problem that is also NP-hard ([23], [29]). In this case the remaining processing time in the WSRPT rule should be increased by the total length of periods of unavailability intersecting with the processing period.

In this paper we are not going to overview the computational complexity of scheduling problems related to \(1\mid pmtn; r_j\mid \sum w_j C_j\). Such details can be found at the website of Brucker and Knust [7] (see also [17]). Historical roots of this problem can be found in [9], [10], [11], and overviews in [13], [12]. Our flexible and efficient heuristic approach is motivated by practical applications which really need this flexibility and efficiency [4].

The paper is organized as follows. In the next section we provide the boolean linear programming (BLP) model for the considered problem and report the threshold parameter values for which the largest size instances can be solved to optimality by means of the CPLEX 12 software. Section 3 contains a description of our WSRPT heuristic and the local optimality theorem as a motivation for this approach. The computational study of the heuristic is presented in Section 4 and the final section concludes the paper with a short summary.

2. Problem formulation

The problem \(1\mid pmtn; r_j\mid \sum w_j C_j\) can be described as follows. We are given \(n \geq 2\) jobs that need to be processed on one machine. Each job \(j\) has an arbitrary processing time \(p_j\), release date \(r_j\) and priority weight \(w_j\). The release date \(r_j\) is the time moment at which job \(j\) becomes available for processing. The weight \(w_j\) can be seen as a priority factor of job \(j\). Preemptions are allowed, which means that the processing of any job can be interrupted at any time and any number of times in favor of other jobs. The objective is to schedule the jobs such that the total weighted completion time \(\sum w_j C_j\) is minimized, where \(C_j\) is the completion time of job \(j\). Also we assume that there are no idle time intervals. This means that the release dates should be such that there exists a solution in which at every time moment \(t = 1, 2, \ldots, \sum_{j=1}^{n} p_j\) some job is processed on the machine.

To find an exact solution to the \(1\mid pmtn; r_j\mid \sum w_j C_j\) problem we present our Boolean Linear Programming (BLP) model. Let us define \(T = \sum_{j=1}^{n} p_j\) and \(p_{\text{max}} = \max p_j\). In our BLP model parameters \(w_j\) are replaced with \(w_{jk}\) parameters. We divide every job \(j\) into \(p_j\) unit parts \(k \in \{1, 2, \ldots, p_j\}\), and each of these parts \(k\) should be processed at some time moment \(t \in \{1, 2, \ldots, T\}\). The following indices and parameters are used in the BLP model.
Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time

\[ j \in \{1, 2, \ldots, n\} \quad \text{job index;} \]
\[ k \in \{1, 2, \ldots, p_j\} \quad \text{job part index;} \]
\[ t \in \{1, 2, \ldots, T\} \quad \text{time index;} \]

\[ \forall k = 1, 2, \ldots, p_j - 1 \quad w_{jk} = \begin{cases} 0 & \text{for } r_j + k \leq t \leq T - p_j + k; \\ \infty & \text{otherwise.} \end{cases} \]

\[ w_{jp} = \begin{cases} w_{jt} & \text{for } r_j + p_j \leq t \leq T; \\ \infty & \text{otherwise.} \end{cases} \]

The decision variables are:

\[ x_{jkt} = \begin{cases} 1 & \text{if the } k\text{-th part of job } j \text{ is assigned to time moment } t; \\ 0 & \text{otherwise.} \end{cases} \]

The BLP model is as follows:

\[
\text{min} \quad \sum_{j=1}^{n} \sum_{k=1}^{p_j} \sum_{t=1}^{T} w_{jk}x_{jkt}
\]

subject to

\[
\sum_{t=1}^{T} x_{jkt} = 1, \quad j = 1, \ldots, n, \quad k = 1, \ldots, p_j; \quad (2)
\]

\[
\sum_{n}^{p_j} \sum_{j=1}^{n} x_{jkt} = 1, \quad t = 1, \ldots, T; \quad (3)
\]

\[
\sum_{i=t+1}^{T} \sum_{k=1}^{p_j-1} x_{jki} \leq p_j(1-x_{jtp_j}), \quad j = 1, \ldots, n, \quad t = 1, \ldots, T - 1; \quad (4)
\]

\[ x_{jkt} \in \{0, 1\}, \quad j = 1, \ldots, n, \quad k = 1, \ldots, p_j, \quad t = 1, \ldots, T. \quad (5) \]

The objective function (1) is the total weighted completion time. Constraints (2) require that every part \( k \) of every job \( j \) is assigned to exactly one time moment \( t \in \{1, 2, \ldots, T\} \). Constraints (3) require that at every time moment \( t \) only one job part \( k \) is processed. Constraints (4) require that the last \( (p_j)\text{-th} \) part of every job \( j \) is scheduled after all its previous parts \( k = 1, 2, \ldots, p_j - 1 \).

To find out how large instances could be solved exactly by means of this BLP model we test it on randomly generated instances. These instances are generated in the same way as suggested in [1]:

1. \( w_j \) is randomly selected from interval \([1, 100]\);
2. \( p_j \) is randomly selected from interval \([1, 100]\);
3. \( r_j \) is randomly selected from interval \([0, T - p_j]\);
4. If the generated instance has no solutions without idle time intervals, it is regenerated until it has such a solution.

For different number of jobs \( n = 5, 10, 15, 20, 25 \) we generate 50 instances, solve every instance using the CPLEX software with our BLP model, and measure an average time in seconds. The results for different values of \( n \) are given in Table 7 ("exact model" column). It is easy to see that the exact solution of this problem requires considerable time which quickly increases when \( n \) is increased. While all the 50 generated instances for \( n = 5 \) and
$n = 10$ are solved in less than 30000 seconds, for $n = 15$ there are 3 instances which have failed to be solved to optimality within this time limit (see “timeouts” column in Table 7), for $n = 20$ there are already 6 timeouts, and for $n = 25$ there are 24 timeouts. The average time for the instances which have been solved within 30000 seconds is given in column “mean”. It quickly increases from 1 second for $n = 5$ to more than 20000 seconds for $n = 25$.

3. The WSRPT heuristic

In this section we justify the weighted shortest remaining processing time (WSRPT) rule based on which we design our heuristic. Below we provide some theoretical results showing the local optimality of this rule for the considered problem $1|\text{pmtn}; r_j| \sum w_j C_j$.

**Proposition 3.1** In an optimal schedule there are no "intersecting" jobs: such jobs $i$ and $j$, that at first job $j$ interrupts $i$ and then $i$ interrupts $j$. (See two examples in Table 1, where ellipses mean some jobs other than $i$ and $j$).

<table>
<thead>
<tr>
<th>Table 1. Two instances with intersecting jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
</tr>
<tr>
<td>$i$</td>
</tr>
<tr>
<td>$i$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. The two instances after swapping of parts of the intersecting jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
</tr>
<tr>
<td>$i$</td>
</tr>
<tr>
<td>$i$</td>
</tr>
</tbody>
</table>

**Proof** We prove the proposition by contradiction. Assume that the statement of the proposition is wrong and an optimal solution has intersecting jobs $i$ and $j$ as shown in Table 1. Note that in the general case there could be a number of other jobs in different parts of an optimal schedule between jobs $i$ and $j$. Such jobs are shown by ellipses in the table.

Let us swap the first two parts of job $j$ with the second part of job $i$ in order to complete job $i$ earlier. We do this swap without moving any of the other jobs shown by ellipses in Table 1. There are two different cases shown in this table. In the first case the second part of job $i$ is shorter than the first part of job $j$. In the second case it is longer. It is always possible to make this swapping because we satisfy all the same release date constraints. For job $i$ we do not move its first part so its release date constraint is satisfied. For job $j$ we move its first part righter so its release date constraint is also satisfied. All the other jobs shown by ellipses are not moved.

The solutions for the two cases obtained after swapping the jobs are shown in Table 2. In the optimal solution (Table 1) jobs $i$ and $j$ make the following contribution to the objective function: $w_i t_i + w_j t_j$. In the solution in Table 2 their contribution is $w_i t_i' + w_j t_j$. Since $t_i' < t_i$ this contribution is less than in the optimal solution: $w_i t_i' + w_j t_j < w_i t_i + w_j t_j$. All the other jobs stay on the same positions in the schedule. So we have decreased the value of the objective function. This contradicts with the optimality of the original solution and so our assumption is wrong and the statement of the proposition is true. ■
Proposition 3.2 In an optimal schedule any job $i$ can be interrupted by another job $j$ only at time moment $r_j$.

Table 3. Two cases when job $j$ interrupts job $i$ after time moment $r_j$

<table>
<thead>
<tr>
<th>$r_j$</th>
<th>$j$</th>
<th>$t_j$</th>
<th>$i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$r_j$</th>
<th>$j$</th>
<th>$t_j$</th>
<th>$i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Proof We prove the proposition by contradiction. Assume that the statement of the proposition is wrong and in an optimal solution job $j$ interrupts job $i$ after time moment $r_j$. According to the proposition 3.1 job $i$ cannot intersect with job $j$. So the two cases shown in Table 3 are possible. Note that there could be a number of other jobs in different parts of an optimal schedule between jobs $i$ and $j$. Such jobs are shown by ellipses in the table.

Table 4. The two cases after moving parts of jobs $i$ and $j$

<table>
<thead>
<tr>
<th>$r_j$</th>
<th>$t_j$</th>
<th>$t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$i$</td>
<td>$j$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$r_j$</th>
<th>$t_j$</th>
<th>$t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$i$</td>
<td>$j$</td>
</tr>
</tbody>
</table>

Let us move all the parts of job $j$ to the left so that it starts from time moment $r_j$ in order to complete job $j$ earlier. The part of job $i$ in the interval from $r_j$ to the first part of job $j$ should be moved to the time moments freed after moving job $j$. We do this move without moving any of the other jobs shown by ellipses in Table 3. There are two different cases shown in this table. In the first case the second part of job $j$ is shorter than the interval from $r_j$ to the first part of job $j$. In the second case it is longer. It is always possible to make this movement because we satisfy all the same release date constraints. For job $i$ we do not move its first part so its release date constraint is satisfied. For job $j$ we move its first part to start from its release date $r_j$. All the other jobs shown by ellipses are not moved.

The solutions for the two cases obtained after moving the jobs are shown in Table 4. In the optimal solution (Table 3) jobs $i$ and $j$ make the following contribution to the objective function: $w_it_i + w_jt_j$. In the solution in Table 4 their contribution is $w_{ij}t_{ij} + w_jt_j'$. Since $t_j' < t_j$ this contribution is less than in the optimal solution: $w_{ij}t_{ij} + w_jt_j' < w_it_i + w_jt_j$. All the other jobs stay on the same positions in the schedule. So we have decreased the value of the objective function. This contradicts with the optimality of the original solution. So our assumption is wrong and the statement of the proposition is true.

From proposition 3.2 it follows that after a job is started it should not be interrupted up to the closest release date. So in our algorithm we do not need to consider every time moment and apply WSRPT rule to it. We should consider only time moments corresponding to release dates and to completion dates. The next theorem shows the local optimality of the WSRPT rule.

Definition 3.3 Scheduling of jobs $j_1, \ldots, j_k$ available at time moment $t$ is called locally optimal if it makes the smallest possible contribution to the objective function provided that all the other jobs available only after time moment $t$ are ignored.

Theorem 3.4 Consider an optimal schedule given up to time moment $t$ at which there are $k$ jobs $j_1, \ldots, j_k$ available for processing (both already started and not yet started
Proof. Since jobs $j_1, \ldots, j_k$ are available from time moment $t$ and there are no jobs available after it up to time moment $t + \sum_{i=1}^{k} p_i$, then according to proposition 3.2 there will be no preemptions in an optimal schedule at interval from $t$ to $t + \sum_{i=1}^{k} p_i$. This means that the problem of optimal scheduling of these jobs is equivalent to the scheduling problem $1|| \sum w_j C_j$ without preemptions and release dates where the processing time of job $j$ is equal to the remaining processing time $\rho_j$ in the original problem. The optimal solution for this problem is obtained by the WSPT rule [22]. So the jobs should be scheduled in the decreasing order of the weight to remaining processing time ratio $w_j/\rho_j$.

**Corollary 3.5** In an optimal schedule it is locally optimal not to interrupt processing of job $j$ at time moment $t$ if and only if it has the maximum ratio $w_j/\rho_j$ among all the jobs available at this time moment.

Proof. According to theorem 3.4 we should schedule job $j$ first (which means that it is not interrupted at time moment $t$) if it has the maximum ratio $w_j/\rho_j$ among all the available jobs. Otherwise, we should interrupt it with another job which has the maximum weight to the remaining processing time ratio.

We illustrate the WSRPT rule by means of an example which also shows that the WSRPT heuristic does not always return an optimal solution. Let the number of jobs be $n = 4$ and let the processing times, weights and release dates be defined by Table 5. The WSRPT solution is also shown in Table 5 (numbers show the scheduled jobs). For the first two time moments $t = 1, 2$ the only available job is job 1, so it occupies the first two cells. For time moment $t = 3$ we calculate the weight to remaining time ratio for jobs 1 and 2: $w_1/\rho_1 = 1/1 = 1.0, w_2/\rho_2 = 3/2 = 1.5$. So job 2 is scheduled at $t = 3$. For $t = 4$ $w_2/\rho_2 = 3/1 = 3.0, w_3/\rho_3 = 7/2 = 3.5, w_4/\rho_4 = 7/2 = 3.5$. We do not calculate the ratio for job 1 here because according to proposition 3.1 it cannot intersect with job 2. So job 3 is scheduled at $t = 3$ and according to proposition 3.2 we complete it at $t = 5$ without interruption. For $t = 6$ $w_2/\rho_2 = 3/1 = 3.0, w_4/\rho_4 = 7/2 = 3.5$ (again job 1 is not considered because of the intersection with job 2). So job 4 is scheduled at $t = 6$ up to its completion at $t = 7$. At $t = 8$ job 2 can be scheduled without intersections and at $t = 9$ job 1 is completed. The value of the objective function for this solution is $1 \cdot 9 + 3 \cdot 8 + 7 \cdot 5 + 7 \cdot 7 = 117$. But the optimal schedule is $(1, 1, 1, 3, 4, 2, 2)$ (see Table 5) with the total weighted completion time $1 \cdot 3 + 3 \cdot 9 + 7 \cdot 5 + 7 \cdot 7 = 114$.

<table>
<thead>
<tr>
<th>$j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_j$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$w_j$</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>$r_j$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**WSRPT solution**

| 1 | 1 | 2 | 3 | 4 | 4 | 2 | 1 |

**Optimal solution**

| 1 | 1 | 1 | 3 | 3 | 4 | 4 | 2 | 2 |
4. Computational experiments

The computational complexity of our heuristic is determined by the computational complexity of the WSRPT rule which is $O(n)$ on each step of the heuristic. Since there are no idle time intervals the total number of steps is not greater than $n p_{\text{max}}$. The WSRPT heuristic stores in memory at most $n \frac{w_j}{\rho_j}$ ratios. So the heuristic has time complexity of $O(n^2 p_{\text{max}})$ and space complexity of $O(n)$.

The computational experiments are performed on Intel i7 machine with 2.50 GHz and 8 GB of memory. Our heuristic is fast enough to solve problems with the number of jobs $n = 1000$ and the processing times $p_j \sim 1000$ in 10 seconds. For the greater number of jobs $n = 10000$ and $p_j \sim 10000$ the algorithm needs about 3 hours. Average computation times for randomly generated instances are given in Table 6. The average time is computed over 50 instances.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p_{\text{max}}$</th>
<th>time (sec)</th>
<th>$n$</th>
<th>$p_{\text{max}}$</th>
<th>time (sec)</th>
<th>$n$</th>
<th>$p_{\text{max}}$</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>400</td>
<td>0</td>
<td>1000</td>
<td>2000</td>
<td>79</td>
<td>10000</td>
<td>400</td>
<td>406</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>2</td>
<td>5000</td>
<td>400</td>
<td>98</td>
<td>10000</td>
<td>1000</td>
<td>1043</td>
</tr>
<tr>
<td>1000</td>
<td>400</td>
<td>4</td>
<td>5000</td>
<td>1000</td>
<td>261</td>
<td>10000</td>
<td>2000</td>
<td>2085</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>10</td>
<td>5000</td>
<td>2000</td>
<td>522</td>
<td>10000</td>
<td>4000</td>
<td>4167</td>
</tr>
<tr>
<td>1000</td>
<td>2000</td>
<td>21</td>
<td>5000</td>
<td>4000</td>
<td>1043</td>
<td>10000</td>
<td>10000</td>
<td>10427</td>
</tr>
<tr>
<td>2000</td>
<td>400</td>
<td>17</td>
<td>5000</td>
<td>10000</td>
<td>2609</td>
<td>10000</td>
<td>20000</td>
<td>20855</td>
</tr>
<tr>
<td>2000</td>
<td>1000</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To test the quality of heuristic solutions we take $n$ from set $\{5, 10, 15, 20, 25\}$ and $p_j \in [1, 100]$. For each $n$ we randomly generate 50 instances in the way described above. Every instance is solved exactly by CPLEX 12 software using the BLP model and by the WSRPT heuristic. Then we compute the minimum, average, and maximum relative error of the heuristic solutions over the 50 instances. The results are presented in Table 7. As it can be seen the WSRPT heuristic finds solutions of high quality and the average error does not exceed 0.08% for any combination of $n$ and $p_j$ values. We have not considered large values for $n$ because even for $n = 25$ half of the instances have not been solved in 30000 seconds (about 8 hours) by the CPLEX. The 50 generated instances for $n = 25$ have required more than 360 hours (15 days) to be solved exactly. We present the results only for $p_j \in [1, 100]$ because for smaller values of $p_j$ the average and maximal error are virtually the same and for greater values the errors are even smaller.

<table>
<thead>
<tr>
<th>$n$</th>
<th>timeouts</th>
<th>time, exact model (sec)</th>
<th>time, heuristic (sec)</th>
<th>error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>min</td>
<td>mean</td>
<td>max</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.01</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>31</td>
<td>280</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>7</td>
<td>2455</td>
<td>&gt;3000</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>15</td>
<td>8870</td>
<td>&gt;3000</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>26</td>
<td>22807</td>
<td>&gt;3000</td>
</tr>
</tbody>
</table>

5. Concluding remarks

In this paper we develop an efficient high-quality heuristic for the $1|\text{pmtn};r_j|\sum w_j C_j$ scheduling problem. The suggested heuristic is based on the WSRPT rule and has the computational complexity of $O(n^2 p_{\text{max}})$ and the space complexity of $O(n)$. The computational experiments show that the average relative error of the solutions found by our heuristic is less than 0.1% for any size of the tested problem instances. The quadratic
computational complexity of our algorithm allows to solve extremely large instances with thousands of jobs in a reasonable time. This provides new avenue of research directions by means of incorporation of our heuristic within the well known exact enumeration type algorithms as well as within metaheuristics to find high quality schedules to more complicated practical scheduling problems including multi-machine scheduling, online scheduling, with availability and other constraints.

6. Acknowledgments

The authors are partially supported by LATNA Laboratory, National Research University Higher School of Economics (NRU HSE), Russian Federation government grant, ag. 11.G34.31.0057.

Boris Goldengorin’s research was partially supported by the Exchange Visiting Program Number P-1-01285 carried out at the Center of Applied Optimization, University of Florida, USA.

References