An exact algorithm for the uncertain version of parallel machines scheduling problem with the total completion time criterion

Marcin Siepak[1]

1 Wroclaw University of Technology/Institute of Informatics
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw
`Marcin.Siepak@pwr.wroc.pl`

Abstract

An uncertain version of parallel and identical machines scheduling problem with total completion time criterion is considered. It is assumed that the execution times of tasks are not known a priori but they belong to the intervals of known bounds. The absolute regret based approach for coping with such an uncertainty is applied. This problem is known to be NP-hard and a branch and bound algorithm (B&B) for finding the exact solution is developed. The results of computational experiments show that for the tested instances of the uncertain problem — B&B works significantly faster than the exact procedure based on enumeration of all the solutions. The algorithm proposed has application for further research of quality evaluation for heuristic and approximate solution approaches for the considered problem (in order to check how far from the optimality are solutions generated by them) and also in the cases where the requirement is to obtain the exact solutions for the uncertain problem.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Task scheduling, Interval uncertainty, Minmax regret, Branch and Bound

Digital Object Identifier 10.4230/OASIcs.SCOR.2012.53

1 Introduction

$P\|\sum C_i$ is classical scheduling problem polynomially solvable ([3], [5]). It consists in scheduling the set of I tasks on the set of J parallel and identical machines. The execution times p_i, $i = 1, \ldots, I$ are given. The optimal schedule minimizes the total flow time $\sum_i C_i$, i.e. the sum of tasks completion times where C_i is the completion time of the ith task. Such a schedule, at each point of time maximizes the number of tasks already processed, by first scheduling those which execution time is the shortest. In order to solve the problem, it is required to assign tasks to the machines and also fix the tasks execution order on each machine.

In this paper we consider the uncertain version of $P\|\sum C_i$ where the execution times p_i are imprecise. The uncertainty is modelled by the concept of a *scenario* which is an assignment of possible values into the imprecise parameters of the problem [1]. The set of all possible scenarios can be described in two ways — in the discrete scenario case, the possible values of uncertain parameters are presented explicitely and in the interval scenario case they can take any value between a lower and upper bound. Hereinafter we consider the interval case where we assume that for each of tasks only the borders of the intervals $\underline{p}_i$ and $\overline{p}_i$ are given where $\underline{p}_i \leqslant \overline{p}_i$ and $p_i \in \left[\underline{p}_i, \overline{p}_i\right]$. Such a way of uncertainty description is useful in the cases where no historical data is available regarding the imprecise parameters, which would be required in order to obtain the probability distribution and apply the stochastic approach and also when there is lack of experts opinions which would be a source of other representations of uncertain execution times, e.g. in the form of a membership function for the fuzzy approach.

To solve the nondeterministic case, one can assume the specific values of uncertain parameters and use the deterministic approach. The quality of such a solution however, may be poor because assumed values may differ greatly from the actual ones. For the considered nondeterministic problem, we use therefore the approach based on the *absolute opportunity loss* (*regret*) introduced by Savage in [12] and the concept of *minmax regret* ([2], [9]) which requires finding a feasible solution that is ϵ-optimal for any possible scenario with ϵ as small as possible [1].

Minmax regret versions of many classical optimization problems were studied in [1] and [6]. Newer results were recently obtained in [2], [7], [13], [14].

The paper is organised as follows. The deterministic and uncertain versions of the considered problem are formulated in Section 2. The solution algorithm proposed is presented in Section 3. Section 4 is devoted to description of the computational experiments and their results. In Section 5, the conclusions are presented.

2 Problem formulation

2.1 Deterministic case

Let us introduce the following notation:
$\boldsymbol{I} = \{1, 2, \ldots, I\}$ — set of tasks,
$\boldsymbol{J} = \{1, 2, \ldots, J\}$ — set of machines,
$p = [p_i]_{i=1,\ldots,I}$ — vector of task execution times.
Moreover, let $x = [x_{ikj}]_{i=1,\ldots,I;k=1,\ldots,I;j=1,\ldots,J}$ be the matrix of binary optimization variables where $x_{ikj} = 1$ if the ith task is scheduled as the kth to the last task on machine j, and 0 otherwise. Each machine has therefore I virtual positions, where tasks can be assigned to and parameter k specifies index of the position to the last where a task can be performed.

The objective function is expressed as the sum of tasks completion times [11]:

$$F(p, x) = \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \sum_{k=1}^{I} k x_{ikj}. \tag{1}$$

The following constraints are imposed on optimization variables x_{ikj}. Each task is performed on exactly one position of exactly one machine:

$$\sum_{j=1}^{J} \sum_{k=1}^{I} x_{ikj} = 1; \ i = 1, \ldots, I. \tag{2}$$

Maximally one task can be performed on each position of each machine:

$$\sum_{i=1}^{I} x_{ikj} \leqslant 1; \ j = 1, \ldots, J; \ k = 1, \ldots, I. \tag{3}$$

The elements of matrix x are binary optimization variables:

$$x_{ikj} \in \{0, 1\}; \ i = 1, \ldots, I; \ j = 1, \ldots, J; \ k = 1, \ldots, I. \tag{4}$$

Therefore, formulation of deterministic version of $P\|\sum C_i$ is as follows

$$F^{'}(p) \triangleq F(p, x^{'}) = \min_{x} F(p, x) = \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \sum_{k=1}^{I} k x^{'}_{ikj} \tag{5}$$

subject to (2), (3) and (4).

2.2 Uncertain case

For the uncertain case, we assume that the exact value of execution time p_i is not given, and this parameter belongs to the interval $\left[\underline{p}_i, \overline{p}_i\right]$ where $\underline{p}_i$ and $\overline{p}_i$ are known. It means that we consider uncertain parameters p_i described by a set of their possible values in the form of intervals. No other characteristics of such an uncertainty are assumed or used. A particular vector p expresses fixed configuration of the execution times and is called a scenario. A set

$$\boldsymbol{P} = \left[\underline{p}_1, \overline{p}_1\right] \times \ldots \times \left[\underline{p}_I, \overline{p}_I\right] \tag{6}$$

of all scenarios is characterized by the Cartesian product of all the intervals. A scenario where completion times of all tasks are equal borders of the corresponding intervals $\underline{p}_i$ or $\overline{p}_i$ is called an *extreme scenario.*

In order to measure solution quality for the nondeterministic problem, we apply absolute regret as introduced by Savage in [12]. It expresses the difference between value of the total flow time criterion for given solution x and specified scenario p as well as optimal value of the total flow time criterion for p:

$$F(p, x) - F^{'}(p). \tag{7}$$

To find solution of the problem, the robust approach ([1], [9]) is applied. For specified solution x the uncertain parameters are determined by obtaining so called *worst case scenario* denoted by p^x, i.e. the one which maximizes absolute regret:

$$z(x) = \max_{p \in \boldsymbol{P}} \left[F(p, x) - F^{'}(p)\right]. \tag{8}$$

The optimal solution x^* for the uncertain version of $P\|\sum C_i$ minimizes (8), i.e. $z^* \triangleq z(x^*) = \min_x z(x)$ subject to (2), (3) and (4).

3 Solution algorithm

3.1 Deterministic case

The deterministic version of $P\|\sum C_i$ is polynomially solvable and the optimal solution can be obtained using SPT (Shortest Processing Time) rule [11]. This procedure sorts all the tasks in nonincreasing order according to their execution times. They are assigned iteratively then, i.e. the i'th task is assigned into the first available position to the last (i.e. not occupied position, such that the value of parameter k is the smallest) on the n'th machine, where

$$n = \begin{cases} J, & \text{when } (i \bmod J) = 0 \\ (i \bmod J), & \text{otherwise.} \end{cases} \tag{9}$$

3.2 Uncertain case

The uncertain version of $P\|\sum C_i$ is NP-hard which results directly from NP-hardness of the nondeterministic version of a single machine scheduling problem, i.e. $1\|\sum C_i$ [10].

Let us denote by $\boldsymbol{P}_e$ the set of extreme scenarios, i.e.

$$\boldsymbol{P}_e = \left\{\underline{p}_1, \overline{p}_1\right\} \times \ldots \times \left\{\underline{p}_I, \overline{p}_I\right\}. \tag{10}$$

and by p^x the worst-case scenario for solution x.

▶ **Lemma 1.** *$p^x \in \mathbf{P}_e$ for any feasible solution x.*

Proof. It is easy to see, that for two feasible solutions x^1 and x^2 the following equality holds:

$$\begin{aligned} F(p,x^1) - F(p,x^2) &= \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \sum_{k=1}^{I} k x^1_{ikj} - \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \sum_{k=1}^{I} k x^2_{ikj} \\ &= \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \left(k^{x^1}_{ij} - k^{x^2}_{ij}\right) \end{aligned} \tag{11}$$

where for any feasible solution x^s:

$$k^{x^s}_{ij} = \begin{cases} k \in \{1,\ldots,J\} & \text{if } \exists k : x^s_{ikj} = 1 \\ 0 & \text{if } \forall k = 1,\ldots,I \ x^s_{ikj} = 0. \end{cases} \tag{12}$$

As a result of (11) we get:

$$F(p,x) - F^{'}(p) = \sum_{i=1}^{I} p_i \sum_{j=1}^{J} \left(k^{x}_{ij} - k^{x^{'}}_{ij}\right). \tag{13}$$

Now, in order to maximize (13) and obtain the worst-case scenario, it is enough to assume:

$$p_i = \begin{cases} \overline{p}_i, & \text{when } \exists j_x, j_{x'} \in \{1,\ldots,J\} : \ k^{x}_{ij_x} > 0 \ \wedge \ k^{x^{'}}_{ij_{x'}} > 0 \ \wedge \ k^{x}_{ij_x} \geqslant k^{x^{'}}_{ij_{x'}} \\ \underline{p}_i, & \text{otherwise.} \end{cases} \tag{14}$$

From (14) it results immediately, that $\forall i$ either $\overline{p}_i \in p^x$ or $\underline{p}_i \in p^x$ and consequently $p^x \subset \boldsymbol{P}_e$ for each solution x. ◀

3.3 Branch and bound algorithm

As a consequence of NP-hardness of the uncertain version of $P\|\sum C_i$, we apply Branch and Bound procedure (B&B) [4] which allows to obtain the exact solution for small instances of the problem faster than the algorithm based on a simple enumeration (denoted as EXCT). B&B can therefore be applied to larger instances of the uncertain problem, which solving using EXCT procedure would take unacceptably long time in the real life systems. According to B&B, the main complex and hard to solve input problem denoted as OPT is decomposed into a finite number of subproblems $OPT_1, OPT_2, ..., OPT_q$ where the corresponding subsets of feasible solutions $Q_1, \ldots, Q_q$, fulfill conditions $Q_l \cap Q_m = \emptyset$ for $l \neq m$ and $Q_1 \cup \ldots \cup Q_q = Q$, where Q is a set of feasible solutions for OPT. The process of decomposition can represented in a form of partition tree where the root corresponds to the input problem OPT and the nodes represent individual subproblems. For each subproblem, until it is fulfilled by maximally one solution — a further decomposition can be performed.

B&B remembers in each iteration the best solution $\acute{x}$ found so far and the corresponding value of the objective function $z(\acute{x})$ or its upper bound $z_{\text{UB}}(\acute{x})$. This algorithm also requires knowledge of the feasible initial solution which can be obtained by applying any approximate or heuristic algorithm. For the purpose of considered uncertain problem, we obtain it by assuming the execution times are equal middles of the corresponding intervals and solving the deterministic version of the problem.

Let $\hat{x}_l$ be the optimal solution for OPT_l. Solving the subproblem directly can still be a process of high complexity, therefore instead of finding $\hat{x}_l$ — a relaxation of OPT_l is performed and optimal solution $\tilde{x}_l$ for the relaxed subproblem is sought. The relaxation generally consists in decreasing subproblem's complexity by weakening some of its restrictions and making it easier to solve. As a consequence of relaxation — solution $\tilde{x}_l$ may not necessirely be feasible for OPT_l, however it fulfills condition $z(\tilde{x}_l) \leq z(\hat{x}_l)$, therefore it can be treated as a lower bound for the subproblem, i.e. $z(\tilde{x}_l) = z_{\text{LB}}(\hat{x}_l)$.

Listing 1: Partition tree browse procedure

```
Generate subproblem $CurrentSubproblem$ at the first level of partition tree by
  assigning the first task into the last position of the first machine;
repeat
    if $CurrentSubproblem$ is not closed then
        // If subproblem is not closed - then we generate a child node.
        $CurrentSubproblem \leftarrow$ GenerateChild($CurrentSubproblem$);
        $i \leftarrow i + 1$;
        $\acute{x} \leftarrow$ TryToCloseSubproblem($CurrentSubproblem, i, I, \acute{x}$);
    else
        // If subproblem is closed - we try to generate its neighbour
           then.
        $NeighbourSubproblem \leftarrow$ GenerateNeighbour($CurrentSubproblem$);
        if $NeighbourSubproblem$ was created then
            $CurrentSubproblem \leftarrow NeighbourSubproblem$;
            $\acute{x} \leftarrow$ TryToCloseSubproblem($CurrentSubproblem, i, I, \acute{x}$);
        else
            $CurrentSubproblem \leftarrow ParentSolution$;
            $i \leftarrow i - 1$;
            Close $CurrentSubproblem$ ;
        end
    end
until $CurrentSubproblem$ is root node;
```

All the subproblems are generated dynamically during execution of B&B, so their total number denoted as q is known after execution of the algorithm is completed. In order to solve OPT, it is required to close each of generated subproblems. The closure of a subproblem implies that that it will not be decomposed anymore, as doing so — would not lead to finding the optimal solution. The individual subproblem OPT_l can be closed when any of the following conditions is fulfilled:

(a) No feasible solution exists that fulfills OPT_l.
(b) Any of conditions $z_{\text{LB}}(\hat{x}_l) > z(\acute{x})$ or $z_{\text{LB}}(\hat{x}_l) > z_{\text{UB}}(\acute{x})$ hold, so as a result optimal solution of OPT_l cannot be optimal for OPT.
(c) A solution $\tilde{x}_l$ is found which is feasible for the non relaxed version of OPT_l.

If none of the above conditions occurs, then OPT_l must be decomposed into further subproblems. Closing the subproblem based on condition (c) when additionally $z(\tilde{x}_l) < z(\acute{x})$ occurs, means that a new best solution was found, therefore substitution $\acute{x} = \tilde{x}_l$ can be performed.

Each node of the partition tree specifies unambiguously the partial allocation matrix $\tilde{x}$, which at the last, I'th level indicates the complete solution x of the scheduling problem. At the i'th level of the partition tree ($i = 1, \ldots, I$), $i - 1$ tasks have already been assigned to the machines and the allocation of the i'th task is performed, while $(I - i)$ tasks will be assigned on the next levels $i + 1, i + 2, \ldots$ of the tree (unless the subproblem can be closed before that). The procedure presented on Listing 1 generates and browses the partition tree. The following notation is used: *CurrentSubproblem* — subproblem corresponding to the currently processed node, i — level of partition tree corresponding to *CurrentSubproblem*, I — total number of tasks.

According to the procedure presented on Listing 1 — if a current subproblem could not be closed (Line 3), then we decompose it (*GenerateChild* procedure — Line 4). Otherwise — we try to generate its neighbour (*GenerateNeighbour* procedure — Line 8), and if that succeds, we assign neighbour as the current node and try to close it (*TryToCloseSubproblem* — Line 6).

The procedure *GenerateChild* applied for *CurrentSubproblem* at the i'th level of partition tree generates the first child only which is allocated at level $(i + 1)$ and the attempt to close it is performed immediately after it has been created (Line 6). If closure fails to succeed, then *GenerateChild* procedure is applied in the next iteration (Line 4), in order to generate child node at level $(i + 2)$. Otherwise — if it was closed successfully, then its neighbour is tried to be generated in the next iteration using *GenerateNeighbour* procedure (Line 8). Each execution of *GenerateNeighbour* applied for any node generates at most one its direct neighbour or returns empty value when no neighbour node could be generated. Returning empty value causes that the procedure goes to the parent node which is automatically closed then (Line 15) and its neighbour is tried to be generated in the next iteration of the algorithm (Line 8). The existence of neighbour node is determined by the analysis of *CurrentSubproblem* to which *GenerateNeighbour* procedure is applied.

Listing 2: TryToCloseSubproblem procedure

```
Input  : CurrentSubproblem, $i$, $I$, $\acute{x}$
Output : $\acute{x}$
1  Find solution $\tilde{x}$ for the relaxed version of CurrentSubproblem;
2  if $z_{LB}(\tilde{x}) \geq z_{UB}(\acute{x})$ then
3      Close CurrentSubproblem ; // We close the current subproblem
4  else
5      if $i == I$ then
           // That means $\tilde{x}$ is a feasible solution for the nonrelaxed
              subproblem
6          Close CurrentSubproblem;
7          if $z_{UB}(\tilde{x}) < z_{UB}(\acute{x})$ then
8              $\acute{x} \leftarrow \tilde{x}$ ; // $\tilde{x}$ is the best solution we currently have
9          end
10     end
11 end
12 return $\acute{x}$
```

The subprocedure $TryToCloseSubproblem$ presented in Listing 2 evaluates the partial solution related to the current node and checks if the subproblem related to it can be closed, so no more children of that node would be generated.

The child node is generated according to the following property which improves solution quality and is true for each optimal solution [11]:

Property: If job i is assigned to position $k > 1$ on machine j, then there is also a job assigned to position $k-1$ on the machine j. Otherwise scheduling job i on position $k-1$ would improve the total assignment cost.

While generating child at i'th level of the tree (*GenerateChild* procedure), we try to assign task $i+1$ into the latest available position (i.e. the one where value of corresponding parameter k is the smallest, as parameter k specifies the position to the last) of the first available machine. The indexes of available machines and positions depend however on the assignment of tasks $1, \dots, i$ on the corresponding previous levels of the partition tree. Let us denote by k_j the value of k for the earliest position occupied (by any task) on machine j. That means, k_j equals the highest value of k corresponding to position occupied on machine j in the current partial solution. As an example, let's assume that $I = 10$, $J = 2$ and $k_2 = 4$. So, the task currently assigned as the earliest on machine 2 is performed as the 4'th to the last. Let us denote by $kMax_j$ the earliest possible position where the current task can be assigned to on machine j in order to fulfill the above property. In case of the presented example, $kMax_1$ must equal 6, which means that current task can be performed on machine 1 the earliest at position 6'th to the last. Assigning this task any earlier on machine 1 would cause unnecessary increase of the total completion time criterion (1), as the property presented above would not be fulfilled then. Therefore, while performing *GenerateChild* procedure, first we determine parameters $k_j, (j = 1, ..., J)$ for each machine. Then we try to assign task $i+1$ into machine j, starting from $j = 1$. We iteratively analyse position k'th to the last on the j'th machine within range $k = 1, \dots, kMax_j$ and check if it is available. If the position is free, then we assign there the current task, and *GenerateChild* procedure stops. Otherwise we try to perform the assignment of task $(i+1)$ on machine $(j+1)$. Parameter $kMax_j$ is calculated according to the following formula:

$$kMax_j = \max\{k_j, I - k_1 - \ldots - k_{j-1} - k_{j+1} - \ldots - k_J\}\,. \tag{15}$$

While building neighbour node for the $CurrentSubproblem$ at the i'th level of the tree we use the knowledge of indexes for machine and position of where the i'th task has been assigned in $CurrentSubproblem$. Let us denote these indexes as $jCurr$ and $kCurr$ respectively. The procedure starts from trying to assign the task into the next available (i.e. not occupied) position on machine $jCurr$ where parameter k corresponding to that position is greater than $kCurr$. So, the i'th task is tried to be assigned iteratively on machine $jCurr$ into the position earlier than it has already been assigned on this machine in the node corresponding to $CurrentSubproblem$. If all the positions up to the one where $k = kMax_{jCurr}$ were occupied, then the assignment is tried to be performed starting from the last position (i.e. $k = 1$) of the next machine $(jCurr + 1)$ until $k = kMax_{jCurr+1}$. The procedure stops, when the task assignment was successfully made or when position $kMax_J$ on the J'th machine was processed and no assignment could be performed due to the occupancy of all the processed positions (the empty value is returned then).

3.4 Lower bound calculation

Having given partial solution $\dot{x}$ corresponding to a specified node at the ith level of the partition tree, calculation of the lower bound z_{LB} is required for this subproblem. In order to perform it, we start from determining parameters $k_j, j = 1, \ldots, J$ for each machine (similarly like in case of *GenerateChild* procedure) and generate series of finite sequences $m_j = (m_{j_s})$ where each sequence contains values of parameter k of those positions on machine j which have not been occupied by any task yet (starting from $k = 1$ up to k_j). That means, each element of m_j specifies index of a position to the last on machine j, where one of non assigned yet tasks will need to be allocated to (in order to fulfill the property presented in Section 3.3) — and as a result its execution time will be multiplied by the value of k corresponding to that position. Let $\overline{m}_j$ denote the number of elements of m_j and m be a sequence of pairs $(m_{j_s}, j), s = 1, \ldots, \overline{m}_j; j = 1, \ldots, J$ concatenating elements of sequences m_j. In order to calculate the lower bound, we generate all the possible combinations of size $\overline{m}$ ($\overline{m}$ is the length of m) from these tasks which have not been assigned to any machines yet, i.e. tasks $i + 1, \ldots, I$. Then, for each combination set, denoted as $\boldsymbol{N}$ we generate all the permutations of $\boldsymbol{N}$. Let $n = (n_s), s = 1, \ldots, \overline{m}$ be a single permutation sequence. Now, we modify $\dot{x}$ and allocate task n_s into the machine and position to the last specified by element s of sequence m ($s = 1, \ldots, \overline{m}$). For $\dot{x}$ modified in such a way we calculate the value of absolute regret. Then we start again from the initial partial solution $\dot{x}$ and modify it using the next permutation sequence calculating absolute regret right after the modification is done. Such a process is repeated until all the permutation sequences for all the combination sets have been analysed. The smallest calculated value of absolute regret is the lower bound for the relaxed version of subproblem and the corresponding $\dot{x}$ is the optimal solution for the relaxed subproblem (we denote such a solution as $\tilde{x}$).

In order to calculate absolute regret (7) for $\dot{x}$ — knowledge of the worst case scenario $p^{\dot{x}}$ is required. The occurrence of parameter k in every addend of (1) causes however, that finding $p^{\dot{x}}$ is NP-hard problem, as it requires analysis of all the extreme scenarios, which number is $2^{|P_e|}$. Therefore, Listing 3 presents how to efficiently generate $\widehat{p}^{\dot{x}}$ which is approximation of unable to calculate effectively scenario $p^{\dot{x}}$ and as a result — how to obtain lower bound of absolute regret.

Listing 3: Calculation of the lower bound for absolute regret

```
Input  : $\dot{x}$
Output : $\tilde{z}$— lower bound for absolute regret $z$
Obtain scenario $\overline{p} = [\overline{p}_i]_{i=1\ldots,I}$;
for $i \leftarrow 1$ to $I$ do
    Generate scenario $\tilde{p} \leftarrow \overline{p}$;
    Assign $\tilde{p}_i \leftarrow \underline{p}_i$ in scenario $\tilde{p}$;
    // $\tilde{z}(p,\dot{x}) = F(p,\dot{x}) - F'(p)$ for any $p$
    if $\tilde{z}(\tilde{p},\dot{x}) > \tilde{z}(\overline{p},\dot{x})$ then
        $\overline{p} \leftarrow \tilde{p}$;
    end
end
return $\tilde{z}(\overline{p},\dot{x})$
```

3.5 Upper bound calculation

For the initial solution generated assuming execution times equal middles of the intervals and also for each solution x feasible for the input problem *OPT* (e.g. the one generated at the leaf of partition tree), calculation of the upper bound z_{UB} is necessary. In order to perform it, we propose to find sets P_m and P_s of scenarios fulfilling the following conditions

$\forall p_m \in P_m : \ F(p_m, x) \geq F(p^x, x)$ and $\forall p_s \in P_s : \ F'(p_s) \leq F'(p^x)$. The value of z_{UB} is then calculated using the following formula:

$$z_{\mathrm{UB}}(x) = \min_{p_m, p_s} \left[F(p_m, x) - F'(p_s)\right]. \tag{16}$$

In order to build set P_s, we apply the procedure presented on Listing 4:

Listing 4: Generation of set P_s

```
$z_{\max} \leftarrow 0$;
Generate scenario $\underline{p} \leftarrow \left[\underline{p}_i\right]_{i=1\ldots,I}$;
for $i \leftarrow 1$ to $\lfloor I/2 \rfloor$ do
    repeat
        Get next scenario $p_i$ by modifying $i$ execution times in $\underline{p}$ from the lower to the
        upper bounds of the corresponding intervals;
        // $\tilde{z}(p,x)$=$F(p,x) - F^{'}(p)$ for any $p$
        if $\tilde{z}(p_i,x) \geq z_{\max}$ then
            if $\tilde{z}(p_i,x) > z_{\max}$ then
                Clear set $P_s$;
                $z_{\max} \leftarrow \tilde{z}(p_i,x)$;
            end
            Add $p_i$ to set $P_s$;
        end
    until all possible scenarios $p_i$ have been processed;
end
```

The procedure of finding set P_m follows the one as on Listing 4 with the difference, that scenario $\overline{p} = [\overline{p}_i]_{i=1\ldots,I}$ is used instead of $\underline{p}$ (Line 2) and while performing each iteration, scenarios having i execution times equal lower bounds of the corresponding intervals (Line 5) are processed ($i = 1, \ldots, \lfloor I/2 \rfloor$). After obtaining sets P_s and P_m, the upper bound is calculated according to (16).

4 Computational experiments

All the algorithms have been written in c# and performed on Intel Core i5, 2.40GHz, 4.00 GB of RAM. In order to test the quality of developed B&B, we compared it with the exact algorithm based on enumeration of all the possible solutions. The test instances of the uncertain problem were generated according to the procedure presented in [8]. First we introduced parameter $C = \{10, 50, 100\}$. For each C the experiments were performed separately. Starting from $C = 10$, for the i'th task ($i = 1, \ldots, I$) we generated $\underline{p}_i$ randomly from the interval $[1, C]$ and $\overline{p}_i$ randomly from the interval $\left[\underline{p}_i, \underline{p}_i + C\right]$ according to the uniform probability distribution. Parameter C specifies the influence of uncertainty on the problem — the higher its value is, the bigger may be the difference between the lower and the upper bound of the intervals for the imprecise parameters. For each C and each of $I = 5, \ldots, 14$ five instances of the uncertain problem were generated and solved independently.

The results of computational experiments for different values of C and $J = 2$ are presented in Table 1. Column I reports the total number of tasks, *Enum* and *B&B* represent the average execution time (in seconds) of the algorithm based on enumeration and Branch and Bound procedure while running it 5 times for different instances of the uncertain problem of

Table 1 The results of experiments for B&B and enumeration procedure.

$C = 10$				$C = 50$				$C = 100$			
I	Enum	B&B	ADV	I	Enum	B&B	ADV	I	Enum	B&B	ADV
5	0.02	0.01	50%	5	0.06	0.01	83.3%	5	0.1	0.04	60%
6	0.202	0.124	38.6%	6	0.24	0.13	45.8%	6	0.18	0.12	33.3%
7	2.62	1.66	36.6%	7	2.76	1.74	37%	7	2.3	1.6	30.4%
8	24.13	13.7	43.2%	8	23.11	12.52	45.8%	8	28.42	14.53	48.9%
9	170	95	44.1%	9	242	120	50.4%	9	309	174	43.7%
10	1087	536	50.7%	10	1353	521	61.5%	10	1493	667	55.3%
11	4108	1781	56.6%	11	4714	2014	57.3%	11	5014	1935	61.4%
12	24990	9985	60%	12	26043	10148	61%	12	29091	10306	64.6%
13	—	27730	—	13	—	29990	—	13	—	31237	—
14	—	71470	—	14	—	77353	—	14	—	81382	—

the same size. Column ADV represents the percentage execution time advantage of B&B over the exact algorithm.

While performing experiments we solved the uncertain problem consisting of up to 12 tasks using the exact algorithm. For the higher number of tasks, the calculation time took more than 24 hours and we stopped the computations. The Branch and Bound procedure solved the problem consisting of up to 14 tasks. The experiments generally show that both of the algorithms work longer while increasing the value of C. For $I = 12$, B&B works even 64.6% faster than the exact procedure, so as a result we could retrieve the exact solution in approximately 2 hours 52 minutes instead of 8 hours 5 minutes while using the exact procedure. The results also show that the execution time advantage of B&B over the exact algorithm increases while increasing the total number of tasks (I). The above observations are limited of course to the tested instances of the uncertain problem.

5 Conclusions

In this paper we have developed and tested a Branch and Bound algorithm for the uncertain version of $P||\sum C_i$ where the uncertain parameters are expressed in the forms of intervals and only their lower and upper bounds are known. Such a version of the problem is known to be NP-hard. The computational experiments for small instances of the problem show that this procedure finds an optimal solution significantly faster than the exact algorithm based on analysis of all the possible solutions. The algorithm proposed has therefore applications for further research of developing efficient approximate and heuristic procedures — in order to test how far from the optimality are the solutions generated by those procedures. Moreover the methodology of building and browsing the partition tree can be applied while extending developed B&B algorithm to solve the uncertain version of more complex problem than considered in this paper, i.e. $R||\sum C_i$. We also recommend to use this method in the cases when the requirement of obtaining the optimal solution has priority — and is more important than the computation time.

References

1 H. Aissi, C. Bazgan, and D. Vanderpooten. Minmax and minmax regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197:427–438, 2009.

2 I. Averbakh and J. Pereira. Exact and heuristic algorithms for the interval data robust assignment problem. *Computers & Operations Research*, 38:1153–1163, 2011.

3 J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382–387, 1974.

4 M. J. Brusco and S. Stahl. *Branch-and-bound applications in combinatorial data analysis.* Springer, 2005.

5 W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21:846–847, 1973.

6 A. Kasperski. *Discrete Optimization with Interval Data: Minmax Regret and Fuzzy Approach (Studies in Fuzziness and Soft Computing).* Springer, Berlin, Heidelberg, New York, 2008.

7 A. Kasperski, A. Kurpisz, and P. Zielinski. Approximating a two-machine flow shop scheduling under discrete scenario uncertainty. *European Journal of Operational Research*, 217:36–43, 2012.

8 A. Kasperski and P. Zielinski. Minimizing maximal regret in linear assignment problems with interval data. Technical Report 07, Raport Serii PREPRINTY, Wroclaw, 2004.

9 P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications.* Kluwer Academic Publishers, Dortrecht, Boston, London, 1997.

10 V. Lebedev and I. Averbakh. Complexity of minimizing the total flow time with interval data and minmax regret criterion. *Discrete Applied Mathematics*, 154:2167–2177, 2006.

11 M. L. Pinedo. *Scheduling - Theory, Algorithms and Systems.* Springer, New York, 2008.

12 L. J. Savage. The theory of statistical decision. *Journal of the American Statistical Association*, 46:55–67, 1951.

13 M. Siepak and J. Jozefczyk. Minmax regret algorithms for uncertain p||cmax problem with interval processing times. In *Proceedings of 21'st International Conference on Systems Engineering*, Las Vegas, USA, 2011.

14 A. Volgenant and C. W. Duin. Improved polynomial algorithms for robust bottleneck problems with interval data. *Computers & Operations Research*, 37:900–915, 2010.