Minimizing the weighted number of tardy jobs on a single machine: strongly correlated instances

Lukáš Hejl[a,b], Přemysl Šůcha[a,*], Antonín Novák[a,b], Zdeněk Hanzálek[a]

[a] *Czech Technical University in Prague*
Czech Institute of Informatics, Robotics, and Cybernetics
Jugoslávských partyzánů 1580/3, 160 00, Prague, Czech Republic

[b] *Czech Technical University in Prague*
Faculty of Electrical Engineering
Technická 2, 166 27, Prague, Czech Republic

Abstract

This paper addresses a single machine scheduling problem minimizing the weighted number of tardy jobs, where each job is characterized by processing time, due date, deadline, and weight. It is known from the existing literature that so-called strongly correlated instances, i.e., instances where each job has the weight equal to its processing time plus a constant, are significantly harder to solve compared to instances without this relation. In this work, we extend an exact algorithm proposed by Baptiste *et al.* [2] with the aim of solving the strongly correlated instances significantly faster. The main improvement is the new integer linear programming model for strongly correlated instances utilizing a decomposition according to the number of tardy jobs. Other proposed improvements are tighter lower and upper bounds which can be applied to all types of instances. The best-known algorithm proposed by Baptiste *et al.* [2] cannot solve all instances with 250 jobs to the optimum within an hour. On the same hardware, our relatively simple improvements implemented into the algorithm proposed by Baptiste *et al.* enable solving all examined strongly correlated instances to the optimum within an hour for up to 5,000 jobs and reduce the computational time on other instances as well.

Keywords: scheduling, single machine, weighted number of tardy jobs, strongly correlated instances

*Corresponding author

Email addresses: hejl.lukas@gmail.com (Lukáš Hejl), suchap@cvut.cz (Přemysl Šůcha), antonin.novak@cvut.cz (Antonín Novák), zdenek.hanzalek@cvut.cz (Zdeněk Hanzálek)

1. Introduction

In this paper, we deal with a particular case of instances of the scheduling problem which is formally defined by a set of n jobs $N = \{1, \ldots, n\}$. Each job $j \in N$ is defined using four non-negative integer parameters: processing time p_j, weight w_j, due date d_j, and deadline $\tilde{d}_j$, $d_j \leq \tilde{d}_j$. A solution to the problem is a schedule, i.e., assignment of the jobs to the start times such that the jobs do not overlap while the completion of each job is not greater than its deadline $\tilde{d}_j$. In addition, jobs are scheduled without preemption. The goal is to find a feasible schedule minimizing $\sum w_j U_j$ where $U_j = 1$ if the job is tardy, i.e., the completion time of the job is greater than its due date, and $U_j = 0$ otherwise. In Graham's scheduling notation, the problem is denoted as $1|\tilde{d}_j|\sum w_j U_j$ and is known to be $\mathcal{NP}$-hard [11].

Potts and Van Wassenhove [19] have shown that certain classes of instances of $1||\sum w_j U_j$ and $1|\tilde{d}_j|\sum w_j U_j$ are significantly harder to solve. The authors defined three classes of instances for $1||\sum w_j U_j$ regarding the relationship between the weights and processing times as (i) *strongly correlated*, (ii) *weakly correlated*, and (iii) *uncorrelated*. Strongly correlated instances are those where $w_j = p_j + C$ for some constant $C \in \mathbb{N}$. The typical choice in the literature is $C = 20$ [19], and its value is related to the distribution of processing times and due dates. Nevertheless, similar difficulty of strongly correlated instances is observed for different values of C as well, which will be shown later in this paper. For weakly correlated instances, an integer weight w_j is drawn from the uniform distribution $w_j \sim [p_j, p_j + C]$, while uncorrelated instances do not have any specific relation between p_j and w_j.

A real-world situation where one can often find a correlation between the weights and processing times is the production. It is quite frequent that more complex orders have a longer duration and a higher cost. The higher cost automatically implicates a higher penalty for late delivery. Another more specific example is bandwidth sharing and scan scheduling in passive surveillance systems used to detect, locate, and track air, ground, and naval targets. Such a system repetitively scans different frequency bands in order to identify or track a target. The more important target, the more data the system needs to acquire. Therefore, a frequency scanning (corresponding to a job) has a longer duration for targets having higher priority and thus a more important due date, i.e., larger weight.

The hardness of strongly correlated instances of problem $1|\tilde{d}_j|\sum w_j U_j$ and $1||\sum w_j U_j$ can be illustrated on the state-of-the-art algorithm by Baptiste *et al.* [2]. Their algorithm is a very efficient depth-first branch and bound procedure formulated such that it maximizes the weighted number of early jobs which is equivalent to minimization of $\sum w_j U_j$. The algorithm exploits a memory-efficient method of solving the problem relaxation, i.e, the upper bound to the optimal solution, and a heuristic solution based on an Integer Linear Programming (ILP) model, i.e. the lower bound. The results in [2] show that this exact algorithm can solve uncorrelated instances with up to $30,000$ jobs with deadlines and $50,000$ jobs in case of deadline-free variant. In contrast, the same algorithm is not able to solve all strongly correlated instances with only 250 jobs to the optimum (3% of instances were not solved to optimum within one hour). The same phenomenon was observed with strongly correlated

instance of Knapsack Problem [15, 18] as well.

Based on our experiments and observations, the limiting factor for solving uncorrelated and correlated instances by the algorithm published in [2] is not the same. The principal limiting factor in the case of uncorrelated instances is the memory limit. It is caused by the ILP model, whose size grows quadratically with n. On the other hand, for strongly correlated instances, the CPU time of the algorithm grows much faster with the size of the instance. In case of strongly correlated instances, the algorithm requires much more CPU time even for smaller instances and eventually time outs, before the memory becomes the limiting factor. Therefore, the algorithm is not capable of solving as large instances as in the previous case, thus the size of memory is not limiting. Based on our analysis, there are three reasons why the algorithm published in [2] cannot solve larger strongly correlated instances.

The principal reason of the algorithm's relatively low performance is caused by job's dominance property (see Theorem 1 in Section 3). The property states that if certain conditions for a pair of jobs are met, then the two jobs are either both early or both tardy. Thus, the property can significantly reduce the number of jobs in the lower bound calculation, depending on how many pairs of jobs satisfy the conditions. However, for the case of strongly correlated instances, the conditions of the job's dominance property translate into the requirement that the jobs have to have identical processing times and weights, which is far more restrictive than in the case of uncorrelated instances. For this reason, it is not possible to significantly reduce the number of jobs involved in the lower bound calculation, and, hence, the ILP model used inside the algorithm tends to be larger (in terms of variables as well as constraints). Therefore, even smaller instances remain intractable for the algorithm.

The second reason is similar to the previous one. The algorithm uses a variable-fixing technique from the Integer Linear Programming domain to *a priori* decide whether some job is early or tardy [13]. The technique requires the knowledge of upper and lower bounds on the objective function. Even though the gap between the bounds is mostly narrow, the variable-fixing technique does reduce far fewer jobs than in the case of uncorrelated instances. Specifically, the number of jobs without a decision in the root node before branching is on average four times higher in the case of correlated instances and 1.5 times higher in the case of weakly correlated both w.r.t. uncorrelated instances.

Lastly, the higher complexity of strongly correlated instances can also be confirmed experimentally. The same ILP model employed inside the algorithm from [2] used just alone with a state-of-the-art solver can handle uncorrelated instances up to size $5,000$ jobs, whereas for the strongly correlated it is less than 150 jobs (see our experiment in Section 5.2).

In this paper, we suggest several improvements to the state-of-the-art algorithm proposed by Baptiste *et al.* [2] for problems $1||\sum w_j U_j$ and $1|\tilde{d}_j|\sum w_j U_j$. The suggested improvements are relatively simple, but, as the experimental results show, their impact on the algorithm performance is surprising. In this work, we focus particularly on the strongly correlated instances; however, the results demonstrate appreciable improvement concerning the other two instance classes as well. We present four main contributions:

1. We reformulate the ILP model from [2] to a model more suitable for strongly correlated instances.

The reformulated model, together with our problem decomposition, allows transforming an instance having $w_j = p_j + C$ (C is a non-negative constant) to several instances with $w_j = p_j$ for which empirical evidence suggests that they are much easier to solve (Section 4.1).

2. We provide improved lower and upper bounds to the problem. The new bounds allow to reduce up to 20% more decision variables, depending on the instance parameters, using variable-fixing techniques, and thus the algorithm is less memory demanding.
3. Our experimental results provide a detailed analysis of strongly correlated instances w.r.t. parameter C which is fundamental for understanding the efficiency of the proposed algorithm. In addition, we compare our implementation of the algorithm suggested in [2] with and without the improvements suggested in this paper.
4. The proposed improvements led to a more efficient algorithm capable of solving more than 20 times larger problem instances than ever before. In addition, the source code of our algorithm is freely available at GitHub `https://github.com/CTU-IIG/SingleMachine-NumberOfTardyJobs`.

The paper is structured as follows. The next section provides an overview of existing work addressing problems $1||\sum w_jU_j$ and $1|\tilde{d}_j|\sum w_jU_j$. Section 3 summarizes the essential parts of the algorithm proposed by Baptiste *et al.* [2]. The core of the paper, i.e., the improvements of the algorithm, are described in Section 4, while their assessments can be found in Section 5. The last section concludes the work.

2. Literature review

The problem of minimizing the weighted number of tardy jobs on a single machine has been studied for many years. For the deadline-free variant, i.e., $1||\sum w_jU_j$, Karp [9] proved that the problem is $\mathcal{NP}$-hard even if all jobs have a common due date. On the other hand, Hermelin et al. [8] showed that if the job's processing times or weights are equal to a constant, then the problem can be solved in polynomial time. However, problem $1|\tilde{d}_j|\sum w_jU_j$ remains $\mathcal{NP}$-hard even if all jobs have the same weight, e.g., $w_j = 1$ [11].

The deadline-free variant of the problem is studied in more papers. Lawler [12], Moore [16], and Sanhi [20] propose dynamic programming-based algorithms with pseudopolynomial time complexity. The subsequent papers use branch-and-bound based algorithms to find the exact solution. The first paper based on this approach was published by Villarreal *et al.* [22]. The authors reduce the size of the problem instances by the application of a dominance theorem, which enabled solving instances having up to 50 jobs. Tang [21] introduces a new job's dominance rules that allow the algorithm to solve instances of up to 85 jobs. The first work that allowed to solve larger instances was published by Potts and Van Wassenhove [19]. They describe an algorithm for solving a linear relaxation of $1||\sum w_jU_j$ with time complexity $\mathcal{O}(n\cdot\log n)$ as a sequence of n fractional problems where k-th problem considers first $k-1$ due dates of jobs in the earliest due date order with the constraint for k-th job being modified based on the solution of the previous problem. The lower bound obtained from the

relaxation is used inside a branch-and-bound algorithm which is able to solve instances with $1,000$ jobs. M'Hallah and Bulfin [17] show an algorithm that solves instances having up to $2,500$ jobs. It uses Knapsack problem to compute a lower bound on the objective function of $1||\sum w_jU_j$.

Only a few papers study exact algorithms for the problem $1|\tilde{d_j}|\sum w_jU_j$. One of the first exact algorithms for this problem is introduced in [7]. The algorithm is based on the branch-and-bound with dynamic programming for computing bounds and can solve instances having up to 300 jobs. The state-of-the-art algorithm is introduced by Baptiste *et al.* [2]. The algorithm is a very efficient branch-and-bound method formulated as a maximization problem (i.e., the weighted number of early jobs). It uses variable-fixing techniques to reduce the problem size, and the authors also introduce methods for computing very tight lower and upper bound. The upper bound is based on a transformation to the maximum profit flow problem (a relaxation allowing preemption); the lower bound computation uses a dominance theorem for reducing jobs set and an ILP model. The algorithm solves uncorrelated instances for both problems $1|\tilde{d_j}|\sum w_jU_j$ and $1||\sum w_jU_j$ assuming up to $30,000$ and $50,000$, respectively, but only 200 jobs in the case of correlated instances.

Recent research papers addressing the single machine total number of (weighted) tardy jobs problem concentrate on more specific variants of this problem with the additional job's characteristics. Liu *et al.* [14] focus on a periodic maintenance (PM) problem denoted as $1|PM|\sum U_j$. For this problem, they present an exact algorithm based on an improved branch-and-bound method with effective lower bound and several dominance properties. They also show that this problem is $\mathcal{NP}$-hard in the strong sense. Wang *et al.* [23] developed two heuristic algorithms for the problem $1|p_{j,r} = (1 + p_{[1]} + p_{[2]} + \ldots + p_{[r-1]})^a|\sum U_j$, denoting a single machine scheduling problem with time-dependent learning effect. Both heuristics are capable of finding near-optimal solutions. The authors also show an exact branch-and-bound algorithm for which they present lower bound and two dominance properties. Paper [1] focuses on two problems $1|cos, prec|\sum w_jU_j$ and $Q_m|cos, p = 1|\sum w_jU_j$, where *cos* means common operation scheduling, i.e., the situation when jobs share operations. They formulate both problems as the set covering problem with an exponential number of constraints and use the branch-and-cut method to solve these formulations. Zhao and Yuan [24] present an improved algorithm with time complexity $\mathcal{O}(n \cdot \log n)$ for a problem dealing with a trade-off between the number of tardy jobs and the start time of a machine, denoted as $1||^{\#}(\sum U_j, A)$. They also present a new algorithm for solving problem $1||\sum U_j$, with time complexity $\mathcal{O}(n \cdot \log n)$.

For problem $1|r_j|\sum U_j$, Briand and Ourari [5] developed an ILP model that assumes only a set of dominant job sequences. The model does not guarantee the solution's optimality, but it can be used for computing tight upper and lower bounds. Laalaoui and M'Hallah [10] address a single machine scheduling problem with machine unavailability periods and a common due date denoted as $1, h_{m-1}|nr - a, d_j = d|\sum w_jU_j$. They use binary multiple knapsack problem to formulate the problem, and show that some large instances can be easily solved with an off-the-shelf solver for binary multiple knapsack problem. They also developed a heuristic based on the variable neighborhood search technique for instances that are difficult for the solver.

3. Preliminaries

In this section, we summarize the fundamental properties of the scheduling problem and outline the basic ideas of the state-of-the-art algorithm proposed by Baptiste *et al.* [2]. The authors propose the algorithm which maximizes $\sum_{i \in N} w_i(1 - U_i) = \sum_{i \in N} w_i - \sum_{i \in N} w_i U_i$, which is identical to the minimization of the original objective $\sum_{i \in N} w_i U_i$.

An important property of problem $1|\tilde{d}_j|\sum w_j U_j$ is that its solution (i.e., a schedule) can be expressed by a set of *early jobs* $E \subseteq N$ with the meaning that each $i \in E$ is completed before due date d_i, i.e., $U_i = 0 \iff i \in E$. If job i exceeds the due date in the solution, then $i \in N \setminus E$ and the job is called a *tardy job*. The set E defines for each job its maximum completion time D_i as

$$D_i = \begin{cases} d_i & \text{if } i \in E, \\ \tilde{d}_i & \text{if } i \in N \setminus E. \end{cases}$$

Therefore, for a given E, the problem $1|\tilde{d}_j|\sum w_j U_j$ reduces to $1|\tilde{d}_j = D_j|-$, which can be solved in a polynomial time by sequencing jobs N in a non-decreasing order of their maximum completion times D_i.

The algorithm proposed in [2] is based on three essential components, which are an ILP model of the problem, and two algorithms computing the upper and lower bound to the objective function. The upper bound is based on a problem relaxation, and a heuristic algorithm computes the lower bound. Additionally, their algorithm exploits two fundamental theorems capturing the structure of $1|\tilde{d}_j|\sum w_j U_j$ problem. The first theorem is a dominance theorem. It states that in an optimal schedule, a job must be early (or tardy) if another job is early (or tardy) provided that certain conditions hold:

Theorem 1 ([2]). *Let $p_i \leq p_j$, $d_i \geq d_j$, $\tilde{d}_i \leq \tilde{d}_j$ and $w_i \geq w_j$, and at least one inequality is strict. Then,*

- *if job i is tardy, then job j must be tardy too,*
- *if job j is early, then job i must be early too.*

The second theorem is the reduction theorem [19, 2], which is essential for reducing the size of an instance. When the algorithm decides that job i is early or tardy, it implies that $D_i = d_i$ for early jobs and $D_i = \tilde{d}_i$ for tardy jobs. Then, the theorem defines a *reduced problem* represented by set $N' = N \setminus \{i\}$ as

$$p'_j = p_j,\ w'_j = w_j,\ j \in N'$$

$$d'_j = \begin{cases} \min\{d_j, D_i - p_i\} & \text{if } d_j \leq D_i, \\ d_j - p_i & \text{if } d_j > D_i \end{cases} \qquad j \in N',$$

$$\tilde{d}'_j = \begin{cases} \min\{\tilde{d}_j, D_i - p_i\} & \text{if } \tilde{d}_j \leq D_i, \\ \tilde{d}_j - p_i & \text{if } \tilde{d}_j > D_i \end{cases} \quad j \in N'.$$

The theorem allows removing job i from N and solving the reduced problem, without losing the optimal solution.

Theorem 2 ([2]). *There exists a feasible schedule with early set E if and only if there exists a feasible schedule with early set $E' = E \setminus \{i\}$ for the reduced problem.*

Therefore, a job that is identified as early or tardy in the algorithm is excluded from N by Theorem 2. Proofs of both theorems can be found in [2].

The algorithm proposed in [2] is a branch-and-bound algorithm with very efficient solution space pruning. Assuming the objective maximizing $\sum_{i \in N} w_i(1 - U_i)$, every partial solution of the branch-and-bound algorithm is processed in the following way:

1. (Upper bound $\bar{z}$). The algorithm solves the LP (Linear Programming) relaxation of the problem using a transformation to maximum profit flow problem. The flow problem is called a *relaxed problem*, and its solution defines the upper bound on the objective function of the original problem denoted by $\bar{z}$.
2. (Lower bound $\underline{z}$). The heuristic computing $\underline{z}$ uses the dominance theorem (Theorem 1) to transform the relaxed solution (obtained in the previous step) to a feasible solution. The jobs that are not decided by Theorem 1 are resolved by the ILP formulation described in Section 3.1.
3. (Fixing of decisions). Subsequently, the algorithm uses variable fixing techniques from [13] to decide whether $i \in E$ or $i \in N \setminus E$. The efficiency of this technique depends on the tightness of bounds $\bar{z}$ and $\underline{z}$.
4. (Branching). The algorithm selects job i for which it cannot decide whether it is early or tardy and recursively branches with $i \in E$ and $i \in N \setminus E$.

In the following two subsections, we summarize the key parts of the algorithm proposed in [2], i.e., the ILP problem formulation, and the algorithm computing the upper bound. Both parts are essential for understanding our improvements described in Section 4.

3.1. ILP model

The ILP model solving problem $1|\tilde{d}_j|\sum w_j U_j$ described in paper [2] decides whether job i is in E or $N \setminus E$; therefore, it introduces a binary variable x_i which equals to one if $i \in E$ and zero otherwise. The ILP formulation is

$$\max_{\boldsymbol{x}} \sum_{i \in N} w_i x_i, \tag{1}$$

subject to

$$\sum_{i \in A_t} p_i + \sum_{i \in B_t} p_i x_i \leq t, \quad t \in T, \tag{2}$$

$$x_i \in \{0, 1\}, \quad i \in N. \tag{3}$$

Instead of minimizing the weighted number of tardy jobs, the objective (1) maximizes the weighted number of early jobs. To ensure that all jobs are completed before their deadlines and the jobs belonging to set E are completed before their due dates, the ILP model contains constraints (2). The constraints are defined over a set of time points $T = \left\{t : t = d_i \vee t = \tilde{d}_i, \forall i \in N\right\}$. For each $t \in T$, the constraint defines two sets of jobs $A_t = \left\{i \in N : \tilde{d}_i \leq t\right\}$ and $B_t = \left\{i \in N : d_i \leq t \wedge \tilde{d}_i > t\right\}$, i.e., the set of jobs that must be completed before t and the set of jobs that will be early if they are scheduled before t, respectively. The term $\sum_{i \in A_t} p_i$ in (2) represents the sum of processing times of jobs that must be completed within time t. Conversely, $\sum_{i \in B_t} p_i x_i$ represents the sum of processing times of jobs that can be completed within time t, i.e., before their due dates. All these jobs may not be completed within time t because their deadlines are after $t \in T$.

As it is pointed out by authors in [2], and our experiments also confirm that, the issue with this ILP formulation is that it is significantly memory-demanding for larger instances. For example, Gurobi ILP solver consumes approximately 8 GB of memory for an instance containing $5,000$ jobs. This is because the constraints (2) in the matrix form contain a large number of non-zero coefficients, where the number of coefficients increases with $\mathcal{O}(n^2)$. The size of the ILP model affects all kinds of instances, but especially for strongly correlated instances, its computational complexity becomes a significant problem. For example, Gurobi is unable to solve some of the instances with 150 jobs to the optimum in a time limit of 1 hour.

3.2. Upper bound

The upper bound to the objective function (1) is a crucial part of the algorithm presented in [2], where it is used across the whole algorithm. It is used in the branch-and-bound for cutting off unfavorable solutions, in the variable-fixing technique reducing the number of jobs, and the solution of the relaxed problem is also used in the heuristic which computes the lower bound.

The easiest way to obtain the upper bound would be to compute the LP relaxation of the ILP formulation (1) – (3). However, such LP relaxation suffers from the same issue with the model size as the ILP model. Therefore, the authors of [2] presented a compact linear relaxation formulated as the maximum profit flow problem. The main advantage of this formulation is memory complexity, which is $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ of the LP relaxation. This formulation allowed solving the relaxation even for instances with $50,000$ jobs, which would not be possible with the LP model.

The maximum profit flow problem is defined on a directed graph having two types of nodes, i.e., nodes that represent every job $i \in N$ and nodes that represent every time point $t \in T$. Each node corresponding to job $i \in N$ is a source node injecting p_i units of flow into the network. This flow can

be divided between two edges from node i to nodes corresponding to time points d_i and $\tilde{d}_i$. Thus, principal decision variables in this formulation are y_{i,d_i} and $y_{i,\tilde{d}_i}$ representing the portion of job i which is executed before the due date and after the due date, respectively. The rest of the graph models the capacity of the machine. For more details, we refer readers to paper [2]. The profit of the edges associated with flow y_{i,d_i} is $\frac{w_i}{p_i}$, while the profit gained from all other edges is 0. Then the relaxed problem maximizes $\max_{\boldsymbol{y}} \sum_{i \in N} \frac{w_i}{p_i} y_{i,d_i}$, which is an upper bound to the objective (1) of the ILP formulation. Since the algorithm proposed by Baptiste *et al.* does not use only the upper bound but also the solution of the relaxed problem, in the rest of the paper, we use $\hat{y}_{i,d_i}$ to denote the optimal value of flow y_{i,d_i}.

4. Improved algorithm

We introduce three enhancements of the algorithm outlined in the previous section. The most significant one is a modified ILP formulation, which admits a decomposition by the number of early jobs, and we show a method for its efficient solution. The other two improvements relate to the computation of lower bound $\underline{z}$ and upper bound $\bar{z}$ on the objective. The first improvement can be used for strongly correlated instances only while the other two can be applied to any class of instances.

4.1. Improved ILP model

We have performed extensive experiments with ILP formulation (1) – (3) to investigate whether the C value affects the computational difficulty of an instance. Our experiments revealed an exception that occurs for $C = 0$. Instances where $C = 0$ (i.e., $w_i = p_i$) are significantly easier than instances that have $C > 0$, such as the widely used $C = 20$ (i.e., $w_i = p_i + 20$). Notice that when $C = 0$, the problem becomes a generalized subset sum problem (for the common due date $d_j = D$ it is the ordinary subset sum problem). The experiments described in Section 5.2 show that the time needed for solving instances with $C = 0$ is similar to the time to solve uncorrelated instances. With this observation, we propose a simple but yet powerful reformulation of the original model.

The new ILP model uses constraints (2) and (3) from the original ILP model. In addition, we introduce variable e, which specifies the number of early jobs via constraint (6). Using variable e, one can reformulate the original objective function for strongly correlated instances as $\max_{\boldsymbol{x}} \sum_{i \in N} w_i x_i = \max_{\boldsymbol{x}} \sum_{i \in N} (p_i + C) x_i = \max_{\boldsymbol{x}} \left(C \cdot e + \sum_{i \in N} p_i x_i \right)$. Then, the new formulation is stated as

$$\max_{e \in \{0,\ldots,n\}} \max_{\boldsymbol{x}} \left(C \cdot e + \sum_{i \in N} p_i x_i \right), \tag{4}$$

subject to

$$\sum_{i \in A_t} p_i + \sum_{i \in B_t} p_i x_i \leq t, \quad t \in T, \tag{5}$$

$$\sum_{i \in N} x_i = e, \tag{6}$$

$$x_i \in \{0, 1\}, \quad i \in N. \tag{7}$$

The reformulated ILP model allows decomposition according to variable e. If we select some specific value of variable e, then the objective function can be written as $\max_{\boldsymbol{x}} \left(\sum_{i \in N} p_i x_i + C \cdot e \right)$. The expression $C \cdot e$ is a constant, and therefore only $\max_{\boldsymbol{x}} \sum_{i \in N} p_i x_i$ can be considered. The constant expression $C \cdot e$ is added after the solution is found. Thus, for each value of e and $C > 0$, we have an ILP model essentially identical to the one with $C = 0$. The decomposed ILP model, called *sub-problem*, differs in that it contains one extra constraint (6) to limit the number of early jobs.

One of the reasons why solving several sub-problems with objective $\max \sum_{i \in N} p_i x_i$ is faster than solving the original ILP model (1) – (3) follows from Theorem 1. Each sub-problem can be seen as a problem instance where $w_i = p_i \quad \forall i \in N$. Thus, the condition defined in Theorem 1 reduces to $d_i \geq d_j$ and $\tilde{d}_i \leq \tilde{d}_j$. The reduced condition is typically satisfied for many more pairs of jobs. This property, together with condition $\sum_{i \in N} x_i = e$, makes the branching inside the ILP solver much stronger on the sub-problems compared to the original formulation. Therefore, the optimal solution can be found much faster.

4.2. Problem decomposition

When decomposing the new ILP model according to variable e, up to $n + 1$ sub-problems with different numbers of early jobs may emerge. Then in the worst case, the algorithm needs to solve all of them to ensure that the optimal solution for the original model is found. To reduce this number, we define exact lower and upper bounds of the number of early jobs denoted $\underline{e}$ and $\bar{e}$, respectively.

Lower bound $\underline{e}$ is computed by solving problem $1|\tilde{d}_j, w'_j = p_j| \sum w'_j U_j$ with modified weights w'_j. Even though this problem is still $\mathcal{NP}$-hard, it is computationally significantly less demanding than the original problem (see Section 5.2).

Proposition 1. *Let $E^=$ be the set of early jobs obtained by solving problem $1|\tilde{d}_j, w'_j = p_j| \sum w'_j U_j$ and let E be the set of early jobs obtained by solving original problem $1|\tilde{d}_j, w_j = p_j + C| \sum w_j U_j$. Then, we have $\underline{e} = |E^=| \leq |E|$.*

Proof. To prove the proposition by contradiction, we assume $|E^=| > |E|$. Since any solution of the problem where $w'_j = p_j$ is feasible for the original problem and vice versa, for the objective of the original problem, we can write that

$$C|E| + \sum_{i \in E} p_i x_i \geq C|E^=| + \sum_{i \in E^=} p_i x_i,$$

$$C(|E| - |E^{=}|) \geq \sum_{i \in E^{=}} p_i x_i - \sum_{i \in E} p_i x_i.$$

Using the assumption, we can see that the left-hand side of the equation is negative. On the other hand, expression $\sum_{i \in E^{=}} p_i x_i$ on the right-hand side is the optimal value of the objective function assuming $w'_j = p_j$, and thus $\sum_{i \in E^{=}} p_i x_i \geq \sum_{i \in E} p_i x_i$. Therefore the right side of the equation is greater or equal to zero, which leads to a contradiction. □

To strengthen the lower bound, problem $1|\tilde{d}_j, w'_j = p_j|\sum w'_j U_j$ is solved with a secondary objective maximizing $|E^{=}| = \sum_{i \in N} x_i$. The secondary objective is not in contradiction with Proposition 1 since it holds for any optimal solution of the problem with $w'_j = p_j$. An advantage of the lower bound $\underline{e}$ is that it also provides a solution to the sub-problem for $e = \underline{e}$, hence it is not necessary to solve this sub-problem in the decomposition again.

Lower bound $\underline{e}$ can be further strengthened by imposing constraint $\sum_{i \in N} w'_i x_i \geq \underline{z}$, where $\underline{z}$ is a lower bound on the optimal objective value (see Section 3 and Section 4.3). However, we have observed that it significantly increases the computational time, and thus the improved algorithm does not use it.

Upper bound $\bar{e}$ is computed similarly. We use problem $1|\tilde{d}_j|\sum U_j$, which is a relaxation of the original problem. We denote the set of early jobs in its optimal solution as E^1. As in the case of the lower bound, this problem is still $\mathcal{NP}$-hard, but it is computationally even easier than $1|\tilde{d}_j, w'_j = p_j|\sum w'_j U_j$ which is used to find $\underline{e}$. Since the original problem and the relaxed problem consider the same solution space and the relaxed problem maximizes the number of early jobs, its solution E^1 defines the upper bound as $\bar{e} = |E^1|$. Both bounds are formulated and solved as an ILP.

With the values $\underline{e}$ and $\bar{e}$, we define a set $K = \{\underline{e}, \underline{e}+1, \ldots, \bar{e}\}$ of candidates for the number of early jobs $|E|$ in an optimal solution. Subsequently, for each $e \in K$, we solve the sub-problem with the constraint $\sum_{i \in N} x_i = e$. The optimal solution is then obtained as the maximum over all the optimal solutions of the sub-problems with a fixed value of $e \in K$. The natural question is how large K can occur. Unluckily, some instances yield $|K| = n - 1$. For example, consider an instance with a single *long job* with processing time $p_1 = k$ and additional $n - 1$ *short jobs* with unit processing times, where $k > n - 1$. If all due dates are set to k, then it can be seen that using the bounds on the number of early jobs described above, we have that $\underline{e} = 1$ and $\bar{e} = n - 1$. However, we note that the bounds are very tight in practice, thus the size of K is typically very small.

The efficiency of the decomposition algorithm is further improved by using the maximum value of the objective function of sub-problems solved with previous values of $e \in K$ as the *cut off* parameter, i.e., the solver is told to search only for solutions with objectives better than the cut-off value. Hence, the solver often searches significantly smaller solution space as many solutions are quickly cut off by a previously found solution.

4.3. Improved lower bound $\underline{z}$

The tightness of the lower bound $\underline{z}$ (and upper bound $\bar{z}$ as well) has a crucial impact on the size of the job set the branch-and-bound algorithm must handle. Both bounds are used in the variable fixing technique, where together with the reduction theorem (Theorem 2) allow reducing the size of N significantly. When it comes to solving strongly correlated instances of large sizes, the original algorithm from [2] fixes significantly fewer variables than in the case of uncorrelated instances. The smaller number of fixed variables results in extensive branching in the branch-and-bound method and the explosion of run time. Hence, the goal is to improve the lower bound in order to fix more variables and to prevent extensive branching.

The heuristic computing lower bound $\underline{z}$ proposed in [2] is based on the solution of the relaxed problem outlined in Section 3.2. The heuristic does not use the objective of the relaxed solution $\bar{z}$, but it uses its solution $\hat{y}_{i,t}$. According to the solution, the set of jobs N is split into a set of jobs L that will be optimally scheduled by (1) – (3), and the other jobs $N \setminus L$ are scheduled heuristically using the information from the solution of the relaxed problem. The quality of the heuristics depends on the jobs selected into set L. In [2], they propose that all jobs with $0 < \hat{y}_{j,d} < p_j$ are inserted into set L. Furthermore, they include all jobs j that are tardy (i.e., $\hat{y}_{j,d} = 0$) for which there is no job i dominating the job j by the dominance Theorem 1. Similarly, their set L includes all jobs j that are early (i.e., $\hat{y}_{j,d} = p_j$) for which there is no job i that is dominated by job j according to the same theorem.

Our approach uses the same structure but has three main differences. The first one is that the jobs in set L are scheduled by the decomposed ILP model from Section 4.1. The second one is a different definition of L. We have carried out several experiments to investigate which jobs must be in L in order to find a tighter lower bound. The results showed that the heuristic provides a much tighter bound if the rule for adding early jobs to set L is modified to the following: set L includes all jobs j that are early (i.e., $\hat{y}_{j,d} = p_j$) for which there is no job i dominating the job j by the dominance theorem. In other words, the condition was reversed. Set L also includes all jobs j that are tardy (i.e., $\hat{y}_{j,d} = 0$) for which there is no job i dominating the job j by the dominance theorem and all jobs with $0 < \hat{y}_{j,d} < p_j$, which is the same with the original heuristic. The last difference is that we do not use the additional local search used in the original algorithm. In the original heuristics, the local neighborhood is defined by expression $\sum_{i \in N} |\bar{x}_i - \tilde{x}_i| = 2$, where $\bar{x}_i$ represents an original solution, and $\tilde{x}_i$ represents a new solution. This neighborhood allows swapping only pairs of jobs, where one of them is early, and the other is tardy. In our improved heuristic, this local search rarely led to an improvement regardless of the class of instances.

These adjustments resulted in a better lower bound heuristic providing a significantly tighter bound than the original heuristic. Modifying this rule increases the size of set L by approximately 13%. The increase of the CPU time introduced by the larger job set L is compensated by leaving out the additional local search.

4.4. Improved upper bound $\bar{z}$

Another way to reduce the number of jobs with Theorem 2 and the variable fixing technique is to improve the upper bound. In paper [2], the authors use continuous relaxation of the ILP model as the upper bound. This relaxation is calculated using the formulation as the maximum profit flow problem. Although this upper bound is relatively tight, we present its improvement by a limited branching procedure.

In our approach, we first solve the problem relaxation (see Section 3.2). Then we tighten up the obtained upper bound by branching on a selected job. In every node of this branch-and-bound procedure, we select job $i = \arg_{i' \in N} \min\{\tilde{d}_{i'} : 0 < \hat{y}_{i',d} < p_{i'}\}$, i.e., a job with the smallest deadline that is partly scheduled before the due date and partly after the due date in the relaxation. In case of instances without deadlines, deadlines are substituted by due dates. For this job we assume two cases: (i) the job is early ($\hat{y}_{i,d_i} = p_i$), and (ii) is tardy ($\hat{y}_{i,d_i} = 0$). For each case, we solve the relaxed problem while fixing flow y_{i,d_i} accordingly, and apply the branching procedure for each branch again. The branching is repeated up to a small fixed depth of the branching tree. Finally, from all solutions in the leaves of the limited branching tree, the algorithm selects the one with the highest objective value. This solution is then used as a new upper bound $\bar{z}$.

The limited branch-and-bound procedure selects jobs with the smallest deadline or due date (for instances without deadlines) for two reasons. First, it is a computationally cheap rule, and second, one can expect that the job with the earliest deadline (resp. the earliest due date) can have the most significant impact on the objective function. Even though this method has increased the computational complexity over solving just a single relaxation, overall, it pays off. The main reason is the same as in the lower bound computation. The tighter lower and upper bounds we provide to the variable fixing technique, the fewer nodes it is necessary to explore in the main branch-and-bound method. This is advantageous, especially for strongly correlated instances, where a large number of explored nodes often leads to time outs.

5. Experiments

This section's principal aim is to compare the algorithm presented in this paper, further denoted as DECOMP-SC, with the algorithm published in [2]. Moreover, the experiments documented below illustrate the key properties of strongly correlated instances essential for the design of DECOMP-SC.

All experiments are performed on a computer containing two Intel Xeon E5-2690 v4 CPUs with 512 GB RAM running CentOS Linux 7. The algorithm of Baptiste *et al.* [2] was reimplemented and executed on the same computer in order to obtain a comparison not affected by hardware and solvers improvements. Our algorithm DECOMP-SC and algorithm from [2] are implemented in C++, and both use Gurobi 8.1.1 ILP solver. For solving the minimum cost flow problem (maximum profit flow problem), we use a dual ascent method RELAXIV [3], specifically its implementation in the MCFCLASS project [6], which proved to be the most suitable for implementation of both

algorithms. The source code of our algorithm is freely available at GitHub `https://github.com/CTU-IIG/SingleMachine-NumberOfTardyJobs`.

The algorithm was tested with the following parameters. The maximum depth of the branch-and-bound tree in the improved procedure computing the upper bound was set to 8. The threshold for solving instances directly by the ILP solver was $1.4 \cdot 10^7$ non-zero elements in the constraints matrix, which is the same as used by [2], and it corresponds to instances with approximately $4,000$ jobs.

All the experiments described below assume time limit 3,600 seconds to solve a problem instance. The ILP solver is allowed to use only a single physical CPU core. The reason why we limit the number of CPU cores is that in paper [2] the algorithm was benchmarked on a computer with a single core as well.

This section consists of five parts. First, we describe the way we generate benchmark instances. Then, we analyze the computational complexity of strongly correlated instances, and we benchmark ILP formulations, i.e., ILP formulation from [2] and the decomposed formulation from Section 4.1. Our DECOMP-SC is compared with our implementation of the algorithm [2] in Section 5.3, and individual improvements used in DECOMP-SC are benchmarked in the subsequent section. The last section presents the experimental results on the heaviest problem instances defined in [2].

5.1. Instances generation

To compare the outcomes of our experiments with the paper [2], we generate instances using the same method. The default value of C is 20, but some experiments use other values. For each job i, the processing time p_i is an integer randomly drawn from the interval $[1, 100]$. The weight for each job is generated based on the instance type as follows:

- In the case of uncorrelated instances, for each job i, the weight w_i is an integer randomly drawn from interval $[1, 100]$;
- In the case of weakly correlated instances with $C = 20$, for each job i, the weight w_i is an integer randomly drawn from interval $[p_i, p_i + 20]$;
- In the case of strongly correlated instances with parameters C, each job i has its weight $w_i = p_i + C$. If it is not stated otherwise, $C = 20$. This value is chosen for the comparability of results with paper [2].

Furthermore, we define a set of parameters $D = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ which is used to create pairs $(u, v) \in D \times D$ that meet condition $u < v$, which are used for generating due dates. Then, for each job i, and the selected pair (u, v), its due date d_i is an integer randomly drawn from interval $[P \cdot u, P \cdot v]$ where $P = \sum_{i \in N} p_i$. For instances in which jobs contain deadlines, deadline $\tilde{d}_i$ is an integer randomly drawn from interval $[d_i, P \cdot 1.1]$.

The benchmark instances are created such that for the given n, there are always 20 randomly generated instances for each pair (u, v), i.e., 200 instances for the given n. All generated instances

are guaranteed to be feasible. Feasibility of each generated instance assuming deadlines is tested by omitting the objective function and solving the feasibility problem by the earliest deadline first rule. Hence, infeasible instances are disregarded as in paper [2].

5.2. Performance of ILP models and the impact of parameter C

In this section, we analyze the performance of the original ILP model (1) – (3) and the decomposed ILP model (4) – (7) proposed in this paper. The test are carried out on uncorrelated instances and correlated instances with various C and the results are compared.

5.2.1. Performance of the original ILP model on uncorrelated and correlated instances

The experiments presented in this section analyze the run time required to solve the original ILP model depending on the size of the instance and its type. We focus on three different types of instances, which are uncorrelated instances, strongly correlated instances with a constant $C = 20$, and strongly correlated instances with a constant $C = 0$. We present only results on instances without deadlines since results on instances with deadlines are very similar.

	CPU time			unsolved
n	avg [s]	max [s]	max† [s]	out of 200 [-]
50	0.02	0.12	0.12	0
100	0.04	0.16	0.16	0
150	0.07	0.22	0.22	0
200	0.11	0.31	0.31	0
250	0.16	0.37	0.37	0
500	0.61	1.29	1.29	0
1000	2.84	5.90	5.90	0
2000	11.70	23.84	23.84	0
3000	27.04	61.93	61.93	0
4000	48.63	126.78	126.78	0
5000	77.62	206.10	206.10	0

(a) Uncorrelated instances.

	CPU time			unsolved
n	avg [s]	max [s]	max† [s]	out of 200 [-]
50	0.03	0.31	0.31	0
100	1.42	221.72	221.72	0
150	32.18	3600.00	915.59	1
200	174.81	3600.00	2905.94	6
250	444.72	3600.00	3254.45	22
500	579.14	3600.00	2485.67	30
1000	1153.90	3600.00	2323.80	63
2000	1123.90	3600.00	2853.58	61
3000	1328.78	3600.00	2914.73	72
4000	1642.63	3600.00	458.25	90
5000	1586.28	3600.00	211.35	86

(b) Strongly correlated instances with $C = 20$.

	CPU time			unsolved
n	avg [s]	max [s]	max† [s]	out of 200 [-]
50	0.01	0.09	0.09	0
100	0.03	0.13	0.13	0
150	0.05	0.16	0.16	0
200	0.09	0.23	0.23	0
250	0.13	0.33	0.33	0
500	0.42	1.21	1.21	0
1000	1.87	5.19	5.19	0
2000	7.45	12.68	12.68	0
3000	16.19	28.39	28.39	0
4000	28.97	74.33	74.33	0
5000	45.55	76.11	76.11	0

(c) Strongly correlated instances with $C = 0$.

Table 1: Results of the original ILP model solved with Gurobi 8.1.1.

The results related to the original ILP model are summarized in tables 1a – 1c. This data is primarily used to demonstrate striking difficulty of strongly correlated instances with $C = 20$ compared to uncorrelated instances. In each table, the *unsolved* column indicates the number of instances

where the solver did not prove the optimality of the solution within the time limit. Columns *CPU time avg* and *max* indicate the average and maximum run time required to solve an instance to the optimum. These columns include instances not solved to the optimum; therefore, column *CPU time max*† represents the maximum run time over the optimally solved instances only.

Table 1a contains the measured results for uncorrelated instances. The results concerning strongly correlated instances with $C = 20$ and $C = 0$ are shown in tables 1b and 1c, respectively. We can observe a significant difference between the run time required to solve strongly correlated instances with $C = 20$ in Table 1b and uncorrelated instances in Table 1a. One can also notice that for uncorrelated instances, all instances up to $5,000$ jobs, are solved within the time limit. In contrast, for strongly correlated instances with $C = 20$ with 150 jobs and more, we start to observe some instances not being solved within the time limit. Conversely, if we compare the results in tables 1b and 1c, we can observe that strongly correlated instances with $C = 0$ are easier to solve than both uncorrelated and the strongly correlated instances with $C > 0$.

5.2.2. Impact of C on the performance of ILP models

To further analyze the influence of parameter C on the CPU time, we have performed an additional experiment. We generated several sets of strongly correlated instances differing in the value of C having 250 jobs with the distribution of processing times and due dates following the scheme given in Section 5.1. Each set has 400 instances that were solved by the original ILP model and new decomposed ILP model.

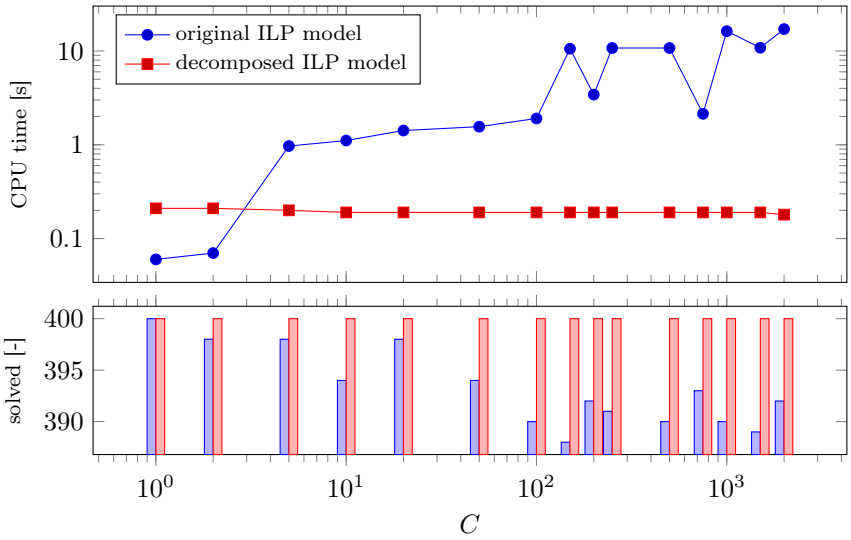

Figure 1: Average CPU time of the solved instances with respect to C value on strongly correlated instances with $n = 250$.

The results are illustrated in Figure 1, showing the relation between C and the mean CPU time over the solved instances. The number of solved instances for each value of C is depicted below. The results related to the original ILP are depicted in blue, and results related to the decomposed ILP are red. Due to the logarithmic scale on the x axis, the graph does not show the result for $C = 0$.

Nevertheless, results for $C = 0$ and $C = 1$ are very similar while the case with $C = 0$ is slightly faster. An important observation one can make is that the CPU time starts sharply growing at $C = 2$ for the original model. For example, the scenario with $C = 5$ requires 379 times more CPU time to solve the instances compared to the scenario with $C = 0$. Also, note that the mean time tends to fluctuate, as the mean is taken over the solved instances, thus it is susceptible to sudden deviations caused by different number of solved instances. On the other hand, the decomposed ILP model shows consistent runtimes across all tested C values, as expected. In addition, the decomposed ILP model has solved all instances for any value of C.

To summarize, the performance of the ILP model illustrated in Table 1c and Figure 1 was the key factor for the design of the decomposed ILP model, which is analyzed in the following subsection.

5.2.3. Comparison of ILP models

The last subsection compares the performance of the original ILP model with the new decomposed one. The experiment demonstrates the capabilities of the decomposed ILP model exploiting the decomposition without using other parts of the algorithm. This experiment was performed on strongly correlated instances with $C = 20$, which are the same as the instances used to measure the results for the original ILP model in Table 1b. The results of this experiment are displayed in Table 2. The table shows that our decomposed ILP model can solve all these instances to optimum within $3,600$ seconds. On the contrary, the original ILP model does not solve all these instances to the optimum within the time limit, even for the instances with 150 jobs. When we compare the ILP models' results, one can also notice that the run times are significantly shorter for our ILP model.

	original ILP model [2]			our ILP model		
	CPU time		unsolved	CPU time		unsolved
n	avg [s]	max [s]	out of 200 [-]	avg [s]	max [s]	out of 200 [-]
50	0.03	0.31	0	0.10	0.40	0
100	1.42	221.72	0	0.17	1.17	0
150	32.18	3600.00	1	0.28	1.42	0
200	174.81	3600.00	6	0.37	3.00	0
250	444.72	3600.00	22	0.48	1.80	0
500	579.14	3600.00	30	1.82	11.65	0
1000	1153.90	3600.00	63	9.01	64.01	0
2000	1123.90	3600.00	61	46.79	423.79	0
3000	1328.78	3600.00	72	153.44	1173.26	0
4000	1642.63	3600.00	90	285.79	2567.63	0
5000	1586.28	3600.00	86	432.04	3022.05	0

Table 2: Strongly correlated instances with $C = 20$ solved using ILP models.

5.3. Comparison of complete algorithms

The next set of experiments focuses on comparing the original algorithm to our improved algorithm DECOMP-SC. The experiments are primarily focused on strongly correlated instances with $C = 20$ with and without deadlines. Nevertheless, we also provide experimental results on weakly correlated and uncorrelated instances.

In each table presented in this section, column *Nodes* indicates the number of visited nodes in the branch-and-bound method. Column *CPU time* indicates the run time required to solve an instance. Column *Root gap* indicates the relative difference between the objectives of the upper bound and lower bound algorithms in the root node denoted $\overline{z}_{root}$ and $\underline{z}_{root}$ respectively. The value is computed as $\frac{\overline{z}_{root}-\underline{z}_{root}}{\underline{z}_{root}} \cdot 100$. The average ($avg$) and maximum values ($max$) of this indicator are in percentages. Column *unsolved* indicates the number of instances that are not solved to the optimum within the time limit of $3,600$ seconds. Finally, in tables 4a and 4b column *Red. prob. size* indicates the size of the reduced instance, i.e., the number of jobs after the application of the reduction theorem in the root node. This quantity is computed only over solved instances.

5.3.1. Strongly correlated instances with deadlines

The first part of the experiments is focused on strongly correlated instances with deadlines. Tables 3a and 3b summarize the results of experiments focused on the relationship between the size of the instance and the time required to find the optimal value for both algorithms.

From the results in tables 3a and 3b, it can be seen that the original algorithm has difficulties to solve to optimality all strongly correlated instances with 250 jobs. On the other hand, DECOMP-SC can solve to optimality all instances having up to $5,000$ jobs and most of the instances with $6,000$ jobs. For the larger instances, DECOMP-SC algorithm solves significantly more instances than the original algorithm.

Comparing the results in tables 4a and 4b, it is apparent that instances with parameters $(0.1, 0.3)$ and $(0.1, 0.5)$ are the most difficult ones. Hence, most of the unresolved instances come from these two pairs of parameters. This behavior is in line with the result of [2], where this phenomenon was observed for uncorrelated and weakly correlated instances as well. On those instances, DECOMP-SC has significantly fewer unsolved instances compared to the original one, and its CPU time is considerably lower. In some cases, *Red. prob. size* is larger for DECOMP-SC than for the original algorithm. This is caused by a different number of solved instances for each algorithm.

The difficult instances with parameters $(0.1, 0.3)$ and $(0.1, 0.5)$ also have their *root gap* larger than the other pairs, and as a consequence of this, there is little ability to reduce the instance size by the variable fixing technique. The larger number of unfixed variables also leads to a large number of nodes being visited in the branch-and-bound method because the instances cannot be reduced to the size, which can be solved directly by the ILP solver.

5.3.2. Strongly correlated instances without deadlines

The experiments with strongly correlated instances with deadlines are provided in the Online Appendix. The absence of deadlines makes the ILP model smaller than in the case with deadlines, which leads to a smaller number of instances unsolved within the time limit. The results show that DECOMP-SC algorithm solved all instances of up to $6,000$ jobs and also reduced the average solving time for them. In contrast, the original algorithm did not solve all the instances with $2,000$ jobs.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
150	0	1.18	98.79	1.00	1.00	0.1901	0.8330
200	0	3.56	353.49	1.00	1.00	0.1528	0.6929
250	6	131.15	3600.00	1.00	1.00	0.1243	0.4853
300	4	74.67	3600.00	1.00	1.00	0.0938	0.4895
350	6	126.28	3600.00	1.00	1.00	0.0820	0.3251
400	5	111.18	3600.00	1.00	1.00	0.0687	0.4350
450	5	119.28	3600.00	1.00	1.00	0.0655	0.3495
500	9	176.88	3600.00	1.00	1.00	0.0579	0.2826
600	14	261.42	3600.00	1.00	1.00	0.0484	0.2559
700	16	294.11	3600.00	1.00	1.00	0.0425	0.1868
800	17	321.06	3600.00	1.00	1.00	0.0347	0.1489
900	18	364.62	3600.00	1.00	1.00	0.0310	0.1348
1000	18	330.91	3600.00	1.00	1.00	0.0284	0.1196
2000	29	558.35	3600.00	1.00	1.00	0.0140	0.0789
3000	47	867.64	3600.00	1.00	1.00	0.0092	0.0345
4000	44	849.17	3600.00	1.00	1.00	0.0066	0.0299
5000	47	875.12	3600.00	1.06	7.00	0.0053	0.0239
6000	49	918.11	3600.00	5.96	477.00	0.0044	0.0150
7000	68	1249.09	3600.00	42.23	2236.00	0.0038	0.0215
8000	71	1318.14	3600.00	60.98	1921.00	0.0033	0.0132
9000	97	1775.35	3600.00	114.42	1987.00	0.0031	0.0135
10000	103	1870.18	3600.00	141.26	2696.00	0.0027	0.0103

(a) Results of the original algorithm.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
1000	0	1.57	16.15	1.00	1.00	0.0255	0.1087
2000	0	3.93	46.52	1.00	1.00	0.0133	0.0604
3000	0	7.38	48.37	1.00	1.00	0.0090	0.0357
4000	0	14.33	332.31	1.00	1.00	0.0067	0.0297
5000	0	17.53	94.21	1.05	5.00	0.0053	0.0239
6000	10	218.03	3600.00	20.84	1165.00	0.0045	0.0176
7000	32	598.27	3600.00	55.66	1540.00	0.0038	0.0148
8000	38	733.45	3600.00	89.36	1128.00	0.0033	0.0172
9000	64	1189.44	3600.00	140.18	1444.00	0.0030	0.0122
10000	68	1255.93	3600.00	164.66	2007.00	0.0027	0.0103

(b) Results of DECOMP-SC.

Table 3: Strongly correlated instances with $C = 20$ and deadlines.

u	v	unsolved out of 200 [-]	Red. prob. size avg [-]	Red. prob. size max [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
0.1	0.3	8	3498.28	5241	1487.76	3600.00	7.75	98.00	0.0115	0.0150
0.1	0.5	16	2675.45	4257	2893.31	3600.00	1.10	3.00	0.0057	0.0088
0.1	0.7	16	1916.33	3056	2990.03	3600.00	1.00	1.00	0.0040	0.0083
0.1	0.9	4	1256.39	2222	743.56	3600.00	1.00	1.00	0.0024	0.0052
0.3	0.5	1	2994.40	5213	224.44	3600.00	25.90	477.00	0.0049	0.0097
0.3	0.7	2	2753.11	4256	377.63	3600.00	1.15	4.00	0.0038	0.0070
0.3	0.9	0	1405.53	2757	7.64	16.61	1.00	1.00	0.0025	0.0053
0.5	0.7	1	2710.90	4811	247.01	3600.00	18.70	353.00	0.0034	0.0062
0.5	0.9	0	2034.05	3054	9.49	16.52	1.00	1.00	0.0029	0.0048
0.7	0.9	1	3035.85	4168	200.28	3600.00	1.00	1.00	0.0037	0.0052

(a) Results of the original algorithm.

u	v	unsolved out of 200 [-]	Red. prob. size avg [-]	Red. prob. size max [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
0.1	0.3	4	3376.85	5392	844.67	3600.00	96.75	1165.00	0.0113	0.0176
0.1	0.5	4	3416.40	4707	808.24	3600.00	68.75	464.00	0.0066	0.0107
0.1	0.7	0	2460.90	3666	42.55	308.54	1.00	1.00	0.0042	0.0077
0.1	0.9	0	1298.21	2363	9.24	16.27	1.00	1.00	0.0024	0.0052
0.3	0.5	1	2940.85	5213	221.73	3600.00	22.15	406.00	0.0045	0.0087
0.3	0.7	0	2698.16	4115	18.99	36.62	1.00	1.00	0.0034	0.0063
0.3	0.9	0	1490.44	2757	8.53	20.14	1.00	1.00	0.0024	0.0051
0.5	0.7	1	2617.60	4811	196.16	3600.00	14.70	275.00	0.0032	0.0062
0.5	0.9	0	2034.05	3054	11.49	18.50	1.00	1.00	0.0029	0.0048
0.7	0.9	0	3027.05	4168	18.70	31.28	1.00	1.00	0.0037	0.0052

(b) Results of DECOMP-SC.

Table 4: Strongly correlated instances with $C = 20$ and deadlines with 6000 jobs.

Instances above $6,000$ jobs are equally hard for both algorithms in terms of the number of solved instances and the average solving time.

5.3.3. Weakly correlated and uncorrelated instances with deadlines

A detailed comparison of the original algorithm and DECOMP-SC on weakly correlated and uncorrelated instances with deadlines is provided in the Online Appendix. This section only provides conclusions following from the results. Both algorithms solved all weakly correlated instances up to the size of $15,000$ jobs, while DECOMP-SC needs about a half of the CPU time on average. Concerning uncorrelated instances, both algorithms can solve instances with up to $30,000$ jobs. Algorithm DECOMP-SC reduced the needed CPU time by about 19%.

When one compares the results for uncorrelated, weakly correlated, and strongly correlated instances, there are significant differences in the difficulty of solving these instances that are dependant on the type of the instance. Differences in difficulty between types of instances are mainly due to the reduction of the size of instances by variable-fixing techniques. For strongly correlated instances, fewer jobs can be reduced than for uncorrelated and weakly correlated instances.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
1000	0	1.57	16.15	1.00	1.00	0.0255	0.1087
2000	0	3.93	46.52	1.00	1.00	0.0133	0.0604
3000	0	7.38	48.37	1.00	1.00	0.0090	0.0357
4000	0	14.33	332.31	1.00	1.00	0.0067	0.0297
5000	0	17.53	94.21	1.05	5.00	0.0053	0.0239
6000	10	218.03	3600.00	20.84	1165.00	0.0045	0.0176
7000	32	598.27	3600.00	55.66	1540.00	0.0038	0.0148
8000	38	733.45	3600.00	89.36	1128.00	0.0033	0.0172
9000	64	1189.44	3600.00	140.18	1444.00	0.0030	0.0122
10000	68	1255.93	3600.00	164.66	2007.00	0.0027	0.0103

(a) Results of DECOMP-SC with all improvements.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
1000	0	2.16	27.70	1.00	1.00	0.0290	0.1196
2000	0	5.14	38.66	1.00	1.00	0.0144	0.0789
3000	0	9.51	71.71	1.00	1.00	0.0096	0.0357
4000	0	18.22	396.51	1.00	1.00	0.0072	0.0309
5000	0	21.50	135.03	1.06	7.00	0.0056	0.0246
6000	10	225.18	3600.00	19.46	847.00	0.0047	0.0176
7000	34	645.68	3600.00	70.31	2019.00	0.0041	0.0215
8000	40	786.65	3600.00	108.10	1839.00	0.0036	0.0172
9000	68	1252.85	3600.00	151.71	1842.00	0.0033	0.0135
10000	72	1337.61	3600.00	184.90	2553.00	0.0029	0.0115

(b) Results of DECOMP-SC only with the decomposed ILP solver.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
1000	0	2.35	26.72	1.00	1.00	0.0258	0.1087
2000	0	5.22	46.95	1.00	1.00	0.0133	0.0604
3000	0	9.09	53.50	1.00	1.00	0.0091	0.0357
4000	0	18.10	397.19	1.00	1.00	0.0069	0.0307
5000	0	21.99	137.00	1.05	5.00	0.0054	0.0246
6000	10	224.09	3600.00	16.95	636.00	0.0045	0.0176
7000	33	619.98	3600.00	55.72	1288.00	0.0039	0.0148
8000	38	742.92	3600.00	69.95	792.00	0.0034	0.0172
9000	64	1197.32	3600.00	116.41	1165.00	0.0031	0.0127
10000	69	1287.62	3600.00	136.38	1605.00	0.0028	0.0115

(c) Results of DECOMP-SC only with the improved upper bound and decomposed ILP model.

n	unsolved out of 200 [-]	CPU time avg [s]	CPU time max [s]	Nodes avg [-]	Nodes max [-]	Root gap avg [%]	Root gap max [%]
1000	0	1.72	16.98	1.00	1.00	0.0288	0.1196
2000	0	3.86	38.87	1.00	1.00	0.0143	0.0789
3000	0	7.37	68.66	1.00	1.00	0.0095	0.0357
4000	0	13.96	332.87	1.00	1.00	0.0070	0.0299
5000	0	16.79	92.25	1.06	7.00	0.0055	0.0239
6000	10	216.26	3600.00	26.48	1997.00	0.0047	0.0176
7000	33	618.86	3600.00	77.28	3196.00	0.0040	0.0215
8000	40	773.41	3600.00	150.09	2461.00	0.0035	0.0172
9000	67	1246.16	3600.00	204.26	2614.00	0.0032	0.0135
10000	71	1309.17	3600.00	248.40	3557.00	0.0028	0.0103

(d) Results of DECOMP-SC with improved lower bound and decomposed ILP model.

Table 5: Individual improvements in DECOMP-SC on the strongly correlated instances with $C = 20$ and deadlines.

5.4. Comparison of individual improvements

Experiments in subsequent sections focus on comparing each improvement of DECOMP-SC with the original algorithm on strongly correlated instances with $C = 20$. The same instances with deadlines used in the previous section are utilized for the experiments described below. Comparison of the algorithm with all improvements (Table 5a) and the algorithm using only the decomposed ILP model (Table 5b) shows that the decomposed ILP model has a major impact on the performance of DECOMP-SC. Both variants are able to solve all instances with $5,000$ jobs, while a significant difference is noticeable only on larger instances.

The impact of the improved upper bound on DECOMP-SC algorithm is illustrated in Table 5c. When we compare results of DECOMP-SC with and without the improved upper bound in tables 5c and 5b, it can be observed that tighter upper bound leads to a significant reduction of the average number of visited nodes. This behavior is caused by a smaller *root gap*, which leads to a reduction of more jobs in the instance and also to sooner termination of the branch-and-bound method. It is also possible to see that the average time needed to solve an instance is reduced, even though the improved upper bound's computation requires more time than the original upper bound computation. The other consequence of the improved upper bound is a few extra solved instances.

When we compare the performance of the algorithm with the decomposed ILP and the algorithm that also uses the improved lower bound (tables 5b and 5d respectively), it can be observed that the improved lower bound has a smaller *root gap* compared to the case without this improvement. As a result of smaller *root gap*, the average time required to solve the instance is smaller than without this improvement, and also, the number of solved instances is slightly higher than without this improvement. Since the original lower bound is relatively tight, our improved one results in just a relatively small improvement in the number of solved instances. The increased number of nodes visited is due to the improved lower bound computation omitting the local search, which in some cases reduces the time required to calculate the bound. Secondly, it is also because the improved lower bound results in a different set of jobs to be solved by the ILP solver than the original lower bound.

In summary, when we compare results in tables 5a, 5b, 5c and 5d, it can be seen that the most notable improvement is the decomposed ILP model, which allows instances up to $5,000$ jobs to be solved to the optimality. The next important improvement is the upper bound heuristic, which also enables solving more instances. For a detailed analysis of the improved lower and upper bounds, see the Online Appendix.

5.5. The heaviest strongly correlated instances

The authors of the paper [2] pointed out a special type of strongly correlated instances ($w_i = p_i + 20$) that have only two different due date values for all the jobs. They refer to them as to the heaviest strongly correlated instances, and according to their results, these instances prove to be the most difficult ones among the strongly correlated ones. They also give an example of such instance with 200 jobs, that none of the ILP solvers was able to solve within $3,600$ seconds by that time. However,

the current solvers are much more efficient, and, e.g., ILP solver Gurobi 8.1.1 can solve their example instance in less than 7 seconds (even with their original model).

n	CPU time avg [s]	CPU time max [s]	unsolved out of 200 [-]
50	0.01	0.13	0
100	0.14	24.18	0
150	26.71	3600.00	1
200	36.33	3600.00	2
250	54.05	3600.00	3
500	36.04	3600.00	2
1000	234.09	3600.00	13
2000	283.02	3600.00	15
3000	216.05	3600.00	12
4000	198.25	3600.00	11
5000	198.10	3600.00	11

(a) Original ILP model.

n	CPU time avg [s]	CPU time max [s]	unsolved out of 200 [-]
50	0.06	0.11	0
100	0.06	0.13	0
150	0.07	0.17	0
200	0.05	0.11	0
250	0.06	0.10	0
500	0.06	0.10	0
1000	0.08	0.25	0
2000	0.09	0.21	0
3000	0.11	0.57	0
4000	0.14	0.84	0
5000	0.15	0.30	0

(b) Decomposed ILP model.

Table 6: Results for the strongly correlated instances with $C = 20$ with two due dates values.

Based on this observation, we performed an additional experiment comprising also larger instances than 200 jobs with two due date values. In this experiment, we compared the original ILP model with our decomposed ILP model on these heaviest strongly correlated instances. The results are summarized in tables 6a and 6b. From the results, it can be seen that the original model cannot solve all instances within the time limit of $3,600$ seconds, even for instances with just 150 jobs. On the other hand, our decomposed ILP model has no difficulties solving all instances with $5,000$ jobs within the same time limit. Moreover, it can also be observed that our decomposed ILP model has relatively small differences between average and maximum CPU time, indicating consistent run time for the heaviest correlated instances compared to the original ILP model.

6. Conclusion

This paper studies problem $1|\tilde{d}_j|\sum w_j U_j$ with the focus on a class of instances known as strongly correlated. The main feature of these instances is that they restrict the relation between processing times of jobs and their weights as $w_j = p_j + C$ with $C > 0$. It was a well-documented fact that these represent the most difficult instances to solve with the existing approaches. The results presented in this paper are founded on an empirical observation related to the strongly correlated instances with $C = 0$. The results showed that this particular type of instances could be solved even faster than uncorrelated instances using a contemporary ILP solver. Based on this observation, we proposed an original ILP model based on a decomposition according to the number of early jobs. Essentially, the solution of a single instance with $C > 0$ is transformed into a sequence of problems with $C = 0$. The decomposed model was used in the algorithm proposed in [2], and together with improved lower and upper bounds, also described in our work, we were able to solve radically larger strongly correlated instances, and we reduced the computational time on other classes of instances as well.

Future research should address other scheduling problems, such as $1||\sum w_j T_j$, and investigate whether there are also classes of instances that are much harder to solve. The study of such classes may point out weaknesses of the state-of-the-art approaches and push the present limits further. Moreover, our experiments indicate the current bottleneck of our algorithm lies in the computation of the lower bound. Hence, the design a new heuristic, such as parallel tabu search algorithm [4], producing results with the same quality but in less time might unlock the solution for even larger instances.

Acknowledgements

This work was supported by the European Regional Development Fund under the project AI&Reasoning (Reg. No. CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[1] Arbib, C., Felici, G., and Servilio, M. Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs. *Omega 84* (2019), 18–30.

[2] Baptiste, P., Della Croce, F., Grosso, A., and T'kindt, V. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. *Journal of Scheduling 13*, 1 (2010), 39–47.

[3] Bertsekas, D. P., and Tseng, P. Relax-iv: A faster version of the relax code for solving minimum cost flow problems. In *Research report*. Massachusetts Institute of Technology, Laboratory for Information and and Decision Systems, 1994, pp. 1–18.

[4] Bożejko, W., Gnatowski, A., Pempera, J., and Wodecki, M. Parallel tabu search for the cyclic job shop scheduling problem. *Computers & Industrial Engineering 113* (2017), 512–524.

[5] Briand, C., and Ourari, S. Minimizing the number of tardy jobs for the single machine scheduling problem: Mip-based lower and upper bounds. *RAIRO-Operations Research-Recherche Opérationnelle 47*, 1 (2013), 33–46.

[6] Frangioni, A., and Gentile, C. The MCFClass project. *http://www.di.unipi.it/optimize/Software/MCF.html. Accessed: 09 January 2020* (2001).

[7] Hariri, A., and Potts, C. Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science 40*, 12 (1994), 1712–1719.

[8] Hermelin, D., Karhi, S., Pinedo, M., and Shabtay, D. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research* (2018), 1–17.

[9] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[10] Laalaoui, Y., and M'Hallah, R. A binary multiple knapsack model for single machine scheduling with machine unavailability. *Computers & Operations Research 72* (2016), 71–82.

[11] Lawler, E. L. Scheduling a single machine to minimize the number of late jobs. Tech. Rep. UCB/CSD-83-139, EECS Department, University of California, Berkeley, Oct 1983.

[12] Lawler, E. L., and Moore, J. M. A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 1 (1969), 77–84.

[13] Linderoth, J. T., and Savelsbergh, M. W. P. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing 11*, 2 (1999), 173–187.

[14] Liu, M., Wang, S., Chu, C., and Chu, F. An improved exact algorithm for single-machine scheduling to minimise the number of tardy jobs with periodic maintenance. *International Journal of Production Research 54*, 12 (2016), 3591–3602.

[15] Martello, S., and Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[16] Moore, J. M. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science 15*, 1 (1968), 102–109.

[17] M'Hallah, R., and Bulfin, R. Minimizing the weighted number of tardy jobs on a single machine. *European Journal of Operational Research 145*, 1 (2003), 45–56.

[18] Pisinger, D. Where are the hard knapsack problems? *Computers & Operations Research 32*, 9 (2005), 2271 – 2284.

[19] Potts, C. N., and Van Wassenhove, L. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science 34*, 7 (1988), 843–858.

[20] Sahni, S. K. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM) 23*, 1 (1976), 116–127.

[21] Tang, G. A new branch and bound algorithm for minimizing the weighted number of tardy jobs. *Annals of Operations Research 24*, 1 (1990), 225–232.

[22] Villarreal, F. J., and Bulfin, R. L. Scheduling a single machine to minimize the weighted number of tardy jobs. *IIE Transactions 15*, 4 (1983), 337–343.

[23] Wang, Z., Wei, C.-M., and Sun, L. Solution algorithms for the number of tardy jobs minimisation scheduling with a time-dependent learning effect. *International Journal of Production Research 55*, 11 (2017), 3141–3148.

[24] Zhao, Q., and Yuan, J. A note on single-machine scheduling to tradeoff between the number of tardy jobs and the start time of machine. *Operations Research Letters 47*, 6 (2019), 607–610.