Discrete Optimization

An exact approach for single machine subproblems in shifting bottleneck procedures for job shops with total weighted tardiness objective

R. Braune, G. Zäpfel, M. Affenzeller

Institute for Production and Logistics Management, Johannes Kepler University, Altenbergerstrasse 69, 4040 Linz, Austria
Upper Austria University of Applied Sciences, 4232 Hagenberg, Austria

1. Introduction

Job shop scheduling consists of finding an optimal sequence and associated starting times for a finite set \( J = \{1, \ldots, n\} \) of jobs on a finite set \( M = \{1, \ldots, m\} \) of machines. In the standard case, each job \( j \in J \) has to be processed on each machine \( k \in M \), suggesting the fragmentation of jobs into separate operations. Let \( o_k \) denote the operation of job \( j \) to be processed on machine \( k \) and \( p_{ik} \) its non-negative processing time.

The order in which operations pass the machines may be different for each job and is pre-defined by precedence constraints. The machines on the shop floor are subject to capacity constraints with regard to the maximum number of operations processed simultaneously. In the disjunctive case, as considered in this paper, only one operation may be processed at a time. Apart from that, no preemption is allowed, meaning that, once started, the processing of an operation cannot be interrupted until it is finished.

A feasible schedule \( S \) is a set of (integer) starting times \( t_{ik} \) for each job \( j \) on machine \( k \) which satisfies the precedence constraints and the capacity constraints. The objective is to find a schedule which minimizes a given cost function.

Assumed that each job is assigned a due date \( d_j \) and a weight \( w_j \), the total weighted tardiness (TWT) cost function can be formulated as a function of job completion times \( C_j \), resulting from a particular schedule:

\[
\text{TWT}(C_1, \ldots, C_n) := \sum_{i=1}^{n} w_i \max(0, C_i - d_i).
\]

In the widely used three-field notation of Graham et al. (1979), the total weighted tardiness job shop scheduling problem including unequal job release dates \( r_i \) is written as

\[
J[r_i] \sum w_i T_i.
\]

where \( T_i = \max(0, C_i - d_i) \).

Contrary to job shops with makespan \( (C_{\text{max}} = \max(C_i)) \) objective, literature on solution approaches to \( J[r_i] \sum w_i T_i \) is scarce. Besides the only branch-and-bound algorithm for this problem proposed by Singer and Pinedo (1997), the remaining approaches mainly consist in local search (cf. e.g. Kreipl, 2000; De Bontridder, 2005; Essafi et al., 2008; Mati et al., 2011) and shifting bottleneck based methods. Initially proposed by Adams et al. (1988) for \( J|C_{\text{max}} \) problem decomposition principle has first been applied to total weighted tardiness job shops by Pinedo and Singer (1999) and later been integrated into a time-window-based approach for large job shops (Singer, 2001). Modified shifting bottleneck procedures have been applied to complex job shops with weighted tardiness objective by Mason et al. (2002) and Mönch et al. (2007). In a recently published journal preprint, Bülbül (2011) describes a hybridization approach with Tabu Search as a re-optimization procedure.

Concerning the single machine subproblems which arise during weighted tardiness oriented bottleneck scheduling, an enumerative approach has been proposed by Pinedo and Singer (1999). The remaining approaches are built upon priority rules or meta-heuristics, e.g. genetic algorithms (Mönch et al., 2007). A brief
comparison of exact and heuristic subproblem solution procedures is provided by Braune (2008). In the paper of Bülbül (2011), an alternative heuristic approach for such problems based on a time-indexed preemptive relaxation is described, initially developed for a single-machine earliness-tardiness problem (Bülbül et al., 2007).

However, the existing subproblem solution methods exhibit two main drawbacks. Either they can hardly be applied to problems involving significantly more than 15 operations due to excessive run time behavior, as it is the case for the Pinedo and Singer’s algorithm (Pinedo and Singer, 1999), or optimality gaps of heuristically built solutions are highly erratic (cf. e.g. Bülbül, 2011).

The primary goal of our research has therefore been the development of an efficient branch-and-bound algorithm for subproblem solving in the total weighted tardiness context in order to enable the applicability to problems involving up to 30 operations per machine. The applicability of such an algorithm is indeed not limited to shifting bottleneck scheduling. The minimum cost induced by a particular machine could be used for example as a lower bound in an enumerative algorithm for the overall job shop as outlined by Singer and Pinedo (1997).

Section 2 focuses on the structural properties of the single machine subproblems and provides an appropriate mathematical formulation. Section 3 describes dedicated constraint programming (CP) based techniques to tighten the subproblem formulation. Concepts for a well-defined comparison of partial sequences with respect to their quality are introduced in Section 4. These concepts are then used in a dominance rule as described in Section 5. Dedicated lower bounds for the considered single machine problem are covered in Section 6. All developed techniques are finally incorporated into a branch-and-bound algorithm (cf. Section 7). Sections 8 and 9 are dedicated to the discussion of computational experience obtained from the application of the developed approach to benchmark problem instances and their concluding interpretation.

2. Problem statement

In this section, we briefly describe the single machine scheduling problems which occur during bottleneck-based scheduling of total weighted tardiness job shops. The basic principle of setting up those subproblems by isolating them from the enclosing job shop has been outlined by Pinedo and Singer (1999), based on a graph model. Therefore we mainly focus on differences in symbol notation and repeat only the most important ideas. Furthermore, we provide a mixed integer programming (MIP) formulation of the problem.

2.1. Model extraction and structural properties

It is of common practice to represent job shop scheduling problems by a disjunctive graph model (Balas, 1969). Other than in the makespan case, the total weighted tardiness objective requires the inclusion of multiple sink or terminal nodes, one for each job. For examples and illustrations of such kind of graphs we refer to Pinedo and Singer (1999) and Kreipl (2000). As far as the graph model and the subproblem setup is concerned, we basically use the notation introduced by Pinedo and Singer (1999), with slight modifications, as summarized in the following overview. Note that in accordance with the given literature references, operation processing times are modeled by edge weights.

\[ r_{uk} \]
release time of operation \( o_k \) on machine \( u \)

\[ d^*_ik \]
local due date of operation \( o_k \) with respect to sink node \( z \), with \( 1 \leq z \leq n \) (cf. Pinedo and Singer, 1999 for a definition)

\[ f_k \]
the set of jobs (job indices) containing an operation to be processed on machine \( k \). In the job shop case, \( |f_k| = n \), for all \( 1 \leq k \leq m \)

A typical situation arising during bottleneck scheduling is when operation sequences on one or more machines have already been fixed while other edges are still undirected. Such existing sequences may give rise to longest paths between operations on unscheduled machines, commonly referred to as delayed precedence constraints (DPCs) (Dauzère-Pérès, 1995; Balas et al., 1995). In this context, let \( P^k \subseteq f^k \times f^k \) denote the set of ordered pairs of job indices with a delayed precedence constraint between them. Then the inequality \( t_{ik} \geq t_{zk} + L(o_k,o_k) \) must hold for all \((i,j) \in P^k\). For ease of notation, we define an equivalent end-to-start delay \( s(i,j) \) between two dependent operations \( o_k \) and \( o_k \) as

\[ v(i,j) := \begin{cases} L(o_k,o_k) - p_k & \text{if } L(o_k,o_k) \text{ exists}, \\ 0 & \text{otherwise}. \end{cases} \] (2.1)

A sequence on a single machine \( k \) which is feasible with respect to capacity and delayed precedence constraints can be evaluated as follows: Given the actual starting time \( t_{zk} \) of an operation \( o_k \) in the single machine schedule, the tardiness of this operation concerning a particular local due date \( z \) is given by \( T^z_{ik} = \max(0, t_{zk} + p_k - d^*_ik) \). Since more than one operation on machine \( k \) may have a longest path to sink \( z \) and thus a corresponding local due date, the overall tardiness of job \( z \) is increased by the maximum of all local tardiness values, i.e. max\( _{i \in f^k} \) \( T^z_{ik} \). The objective function of the single machine problem is then given by

\[ \sum_{z=1}^{n} w_z \max_{i \in f^k} T^z_{ik}. \]

In the three-field notation of Graham et al. (1979), our single machine subproblem can be written as

\[ 1|r,DPC|\sum_z w_z \max_i T^z_i, \]

where the machine index \( k \) has been omitted for the release times and the tardiness values.

2.2. Mathematical formulation

Given the release dates, the local due dates and the delayed precedence relations as described in Section 2.1, we can state a disjunctive mixed integer formulation of the subproblem on machine \( k \):

\[ \min \sum_{z=1}^{n} w_z \max_{i \in f^k} T^z_{ik} \]

subject to

\[ t_{ik} \geq t_{zk} + p_k \quad \text{or} \quad t_{ik} \geq t_{zk} + p_k, \quad \forall \ i,j \in f^k, \ i \neq j. \] (2.2)

\[ t_{ik} \geq t_{ik} + L(o_k,o_k), \quad \forall \ (i,j) \in P^k. \] (2.3)

\[ T^z_{ik} = \max(0, t_{ik} + p_k - d^*_ik), \quad \forall \ i \in f^k. \] (2.4)

\[ t_{ik} \geq r_{ik}, \quad \forall \ i \in f^k. \] (2.5)

The constraints in this formulation are according to Queyranne and Wang (1991), with a slight modification necessary for the delayed precedence constraints. Note that constraints (2.2) have to be linearized using large constants (“big-M” formulation) before the model can be handled by a MIP solver.
We refer to the objective function as the total weighted maximum tardiness \(TWT_{\text{max}}\), since we may introduce \(T^2_{\text{max}} = \max j \in I T^2_{ik}\).

2.3. Complexity status

Pinedo and Singer (1999) stated that \(1|r_j, DPC|\sum w_j \max T^2_j\) is a generalization of \(1|r_j|T_{\text{max}}\) which is in turn equivalent to the minimization of maximum lateness \(L_{\text{max}}\) with release times. Of course the \(TWT_{\text{max}}\) problem is also a generalization of the total weighted tardiness problem. The total weighted tardiness problem is equivalent to a \(TWT_{\text{max}}\) problem where each operation has exactly one single local due date with an index \(z\) which is unique among all operations, i.e. no other operation has a local due date with the same sink index. Note that this is the case in the initial state of the shifting bottleneck procedure, where none of the machine schedules has been fixed yet.

Since \(1|r_j, DPC|\sum w_j T^2\) are both NP-hard (in the strong sense) (Lenstra et al., 1977), we can conclude that \(1|r_j, DPC|\sum w_j \max T^2_{ik}\) is also strongly NP-hard.

3. Strengthening the problem formulation using constraint programming

In constraint programming (CP) based scheduling, each operation is assigned a time window during which it may feasibly be processed. The time window of an operation \(o_{ik}\) is specified by release time \(r_{ik}\), reflecting its earliest possible starting time, and a deadline \(d_{ik}\) by which it must be completed. Based on the capacity (disjunctive) constraints, several techniques have been proposed in order to tighten release times and deadlines or to even deduce new precedence relations between operations. We refer to Baptiste et al. (2001) for an overview on these methods.

Considering the structure of the single machine problem formulated in Section 2.2, the delayed precedence constraints can be used for tightening the operation release times of the problem. On the other hand, we present a concept of inferring operation deadlines in the most direct way via the operation specific cost functions.

3.1. Tightening release times

Sourd and Nuijten (2000) have introduced the concept of multi-machine lower bounds for job shops. Their central observation is the fact that, in a partial schedule, two or more unscheduled operations on the same machine may share a common (unscheduled) successor operation linked by delayed precedence constraints. The release time of the latter operation is computed as \(\sum w_j T^2_j\) on the total weighted maximum tardiness resulting from a single, isolated operation \(o_{ik}\) can be described by a piecewise linear cost function (Pinedo, 2002), as exemplarily shown in Fig. 1 for due dates \(d^d_{ik} = 5\), \(d^d_{ik} = 7\), \(d^d_{ik} = 10\) and weights \(w_1 = 2\), \(w_2 = 4\) and \(w_3 = 1\). In general, let \(g_i\) \(\colon N \rightarrow R\) denote the piecewise linear cost function for a given operation \(o_{ik}\). The breakpoints of \(g(x)\) coincide with the local due dates of \(o_{ik}\) and the slopes equal the accumulated weights. Propagating the constraint \(\sum w_j T^2_j \leq UB(TWT_{\text{max}})\) triggers \(g(x) \leq UB(TWT_{\text{max}})\) for each operation. The deadlines \(d_{ik}\) are now given by

\[d_{ik} = g^{-1}(UB(TWT_{\text{max}})).\quad (3.2)\]

In fact, the computation of the right hand side of Eq. (3.2) can be achieved by finding the largest value \(x\) for which \(g(x) \leq UB(TWT_{\text{max}})\) still holds.

In a straightforward CP model of the \(TWT_{\text{max}}\) single machine problem, each \(T^2_{ik}\) could be modeled as a separate variable in order to appropriately construct the objective function. In such a case, there is, however, no direct and immediate link between tardiness variables which belong to the same operation, because the cost values are computed with respect to sink indices \(\max T^2_{ik}\) for all \(z\) instead of operations. Hence the piecewise linear cost structure is not directly “visible” to a standard CP engine. As experiments with IBM ILOG Scheduler revealed, immediate propagation of \(UB(TWT_{\text{max}})\)
values to operation deadlines is not achieved by simple propagation (IlosSolver.propagate()) and it usually takes several iterations until consistency is reached.

Clearly, the effectiveness of propagation based on operation specific piecewise linear cost functions increases with the number of local due dates defined for a particular operation. A large number of due dates leads to steeper slopes of the function $g(x)$ and consequently to tighter deadlines.

4. Comparing sequences

In this section, we present an approach for comparing two sequences of operations for the same single machine problem with respect to their total cost. We restrict our scope to the comparison of sequences which involve exactly the same set of operations (in different order) and are thus of the same length. The sequences do not have to be complete; in fact we are primarily interested in the comparison of partial sequences with regard to dominance relations as discussed in Section 5. Note that the considered (partial) sequences are assumed to be scheduled from the beginning of the time horizon, hence according to a forward sequencing principle.

Embedded in an enumerative method, the purpose of dominance relations is to reduce the solution space of a combinatorial optimization problem (Jouglet and Carlier, 2011). In the context of the problem under consideration, it is thus desirable to expand only promising partial sequences and to discard the remaining ones. However, before we can define necessary conditions for a sequence to dominate another one, it is necessary to establish basic comparison concepts.

Let $\sigma$ and $\sigma'$ be two (partial) sequences to be compared and $S$ and $S'$ the associated schedules. Let further denote $WT_{\text{max}}(\sigma)$ and $C_{\text{max}}(\sigma)$ the weighted maximum tardiness and the maximum completion time associated with a (partial) sequence $\sigma$. Symbols $\sigma_i$ and $\sigma(x)$ refer to the operation at the $x$th position of the sequence. We further introduce subsequences of $\sigma$, which can be bounded at one side, e.g. $\sigma_{[x,y)}$, denoting the subsequence with indices $1 \leq x < y$ or represent an interval between two positions $x$ and $y$, i.e. $\sigma_{[x,y)}$. The concatenation of a subsequence $\sigma'$ to an existing partial sequence $\sigma$ is denoted by $\sigma|\sigma'$.

Before we describe our comparison approach, we first introduce the notion of “relevant” sink indices. In the context of total weighted maximum cost scheduling, it is not possible to ultimately assess the quality of partial sequences without considering potential costs resulting from unscheduled operations. In order to illustrate this important observation we give the following example:

**Example 4.1.** Consider two operations at the beginning of a partial sequence $\sigma$ and let $\sigma(1)|\sigma(2)$ and $\sigma(2)|\sigma(1)$ be two alternative orders of these operations. Furthermore let $\sigma'_{[x,2]}$ denote the final optimal sequence of operations after the second position of this partial sequence which we assume to be independent of the first two operations. The maximum cost values for the subsequences based on three sink indices are listed in the table below:

|       | $\sigma(1)|\sigma(2)$ | $\sigma(2)|\sigma(1)$ | $\sigma'_{[x,2]}$ |
|-------|------------------------|------------------------|-------------------|
| $T_{\text{max}}^1$ | 6                      | 0                      | 6                  |
| $T_{\text{max}}^2$ | 0                      | 4                      | 0                  |
| $T_{\text{max}}^3$ | 2                      | 0                      | 3                  |

Suppose now that we try to evaluate the two alternatives $\sigma(1)|\sigma(2)$ and $\sigma(2)|\sigma(1)$ while the remaining operations are not yet scheduled, i.e. without knowing $\sigma'_{[x,2]}$. Given the weights $W_1 = 2$, $W_2 = 1$, and $W_3 = 1$, we compute $WT_{\text{max}}(\sigma(1)|\sigma(2)) = 2 \cdot 6 + 1 \cdot 2 = 14$ and $WT_{\text{max}}(\sigma(2)|\sigma(1)) = 1 \cdot 4 + 1 \cdot 3 = 7$. Hence $\sigma(2)|\sigma(1)$ appears to be the better sequence and we would schedule the operations in this order. However, if we continue scheduling, we will find out that finally $T_{\text{max}}^1(\sigma(2)|\sigma(1)) = 6$ and $T_{\text{max}}^3(\sigma(1)|\sigma(2)) = 3$. This means that $\sigma(2)|\sigma(1)$ finally results in a higher total cost:

$$WT_{\text{max}}(\sigma(2)|\sigma(1)) = 2 \cdot 6 + 1 \cdot 1 + 1 \cdot 4 + 1 \cdot 3 = 19 > WT_{\text{max}}(\sigma(1)|\sigma(2)) = 2 \cdot 6 + 0 \cdot 4 + 1 \cdot 3 = 15,$$

which is clearly not what we intended.

As a consequence, it is not legitimate to draw conclusions based on a simple, straightforward comparison of cost values for alternative subsequences as long as there are unscheduled operations. However, if we were able to determine that a set of unscheduled operations will not affect the maximum cost for some sink index $z$, we could identify a relevant set of indices for cost comparison. Let $NS$ denote the set of unscheduled operations with regard to a given partial sequence $\sigma$, that is $NS = \{i | i \in f^h(\sigma)\}$. The new concept of relevant sink indices is introduced formally in the following definition:

**Definition 4.1.** A sink index $z$ is called relevant for total cost comparison if and only if there is no unscheduled operation which has a longest path to sink $y_z$ defined. Formally, we can define a predicate

$$\text{IsRelevant}(z) : \iff \forall j \in NS : L(\log_j, y_z) \text{ is undefined}.$$ (4.1)

It remains to define a way to compare different partial sequences with respect to their relevant maximum costs. For this purpose, we first define a function $g^*(\sigma, \sigma', x, y, NS)$, which computes the gain in the maximum cost w.r.t. the sink index $z$ when sequence $\sigma'$ is scheduled instead of $\sigma$ between positions $x$ and $y$:

$$g^*(\sigma, \sigma', x, y, NS) = \max(T_{\text{max}}(\sigma,x), T_{\text{max}}(\sigma,y), T_{\text{max}}(NS)) - \min(T_{\text{max}}(\sigma',x), T_{\text{max}}(\sigma',y), T_{\text{max}}(NS)).$$ (4.2)

In a typical application scenario, a sequence $\sigma'$ is obtained by altering the order of operations of $\sigma$ only between two given positions $x$ and $y$. If this is the case we therefore have $\sigma_{x,y} = \sigma'_{x,y}$ and $\sigma_{x,y} = \sigma_{x,y}$. If $g^*(\sigma, \sigma', x, y, NS)$ is positive, this means that the maximum tardiness of $\sigma'$ between positions $x$ and $y$ is smaller than that of the original $\sigma$. The last component of the maximum term in each line, $T_{\text{max}}(NS)$, reflects the inherent tardiness resulting from unscheduled operations. It is important to note that changing a scheduled sequence may have an impact on the release time of unscheduled operations via delayed precedence constraints. As a consequence, the inherent tardiness of the set of unscheduled operations cannot be considered as a fixed value. Hence, in the strict sense, $T_{\text{max}}(NS)$ has to be computed separately for $\sigma$ and $\sigma'$. If on the other hand, a retarding influence of $\sigma'$ on $NS$ can be excluded, it is basically possible to compute $T_{\text{max}}(NS)$ only once for $\sigma$ and use the same value in the second line of Eq. (4.2). It is even possible to neglect this part of tardiness at all for the computation of $g^*(\sigma, \sigma', x, y, NS)$.

The function $g^*$ computes the gain in maximum tardiness without considering the relevance of the given sink index. When deciding whether to replace $\sigma$ by $\sigma'$, this is an important aspect as already mentioned above. Using **Definition 4.1** we define a new function $g^*_r$, which takes the same arguments as $g^*$, but considers positive gain values only if index $z$ is relevant:

$$g^*_r(\sigma, \sigma', x, y, NS) = \begin{cases} g^*(\sigma, \sigma', x, y, NS) & \text{if index } z \text{ is relevant,} \\ \min(0, g^*(\sigma, \sigma', x, y, NS)) & \text{otherwise.} \end{cases}$$ (4.3)
The total weighted relevant gain, i.e. the relevant gain in total weighted maximum tardiness, \( G_{rel} \) is then simply given by

\[
G_{rel}(\sigma, \sigma', x, y, NS) = \sum_{i=1}^{n} w_i G_{rel}^i(\sigma, \sigma', x, y, NS).
\]  

(4.4)

Under the presence of delayed precedence constraints, the gain in total weighted maximum tardiness as defined above is not sufficient to determine whether a sequence \( \sigma \) can be replaced by an alternative sequence \( \sigma' \). Let us consider the typical scenario of creating a sequence \( \sigma \) by modifying the order of operations of a sequence \( \sigma \) between arbitrary positions \( x \) and \( y \). In the modified sequence, some operations may start later than originally in \( \sigma \), hence successors of such operations with respect to the DPCs may be delayed as well. If all operations are scheduled, it is possible to compute the impact of such a delay. In case \( NS \neq \emptyset \), however, the effect on total weighted maximum cost cannot be assessed in advance since increased release times of unscheduled operations lead to a different (sub-)problem setting. As a consequence, it is only feasible to consider modified sequences which do not delay any operation from \( NS \).

In order to simplify the comparison process, we check whether any successor operation, scheduled or unscheduled, is delayed. We can assume that the modified sequence between \( x \) and \( y \) and the operation completion times in the associated schedule satisfy the DPCs. Hence we are only interested in successors which either occur in the sequence after position \( y \), i.e. in \( \sigma_{-y} = \sigma_{-y} \) or are still unscheduled. Suppose that release times of unscheduled operations have been adjusted during scheduling such that \( \forall j \in NS: r_j \equiv C_{\text{max}}(\sigma) \) and the DPCs are satisfied, we define a state dependent operation release date \( \bar{r}_j \) as

\[
\bar{r}_j(\sigma) = \begin{cases} 
\bar{r}_j & \text{if } j \neq \sigma, \\
C_{\text{c}}(\sigma) - p_j & \text{otherwise.}
\end{cases}
\]  

(4.5)

**Definition 4.2.** A sequence \( \sigma' \) obtained by altering the order of operations of a sequence \( \sigma \) between arbitrary positions \( x \) and \( y \) is called DPC-neutral if and only if no successor of any operation between \( x \) and \( y \) is delayed (with respect to state dependent release times computed according to \( \sigma \)):

\[
\text{DPC-neutral}(\sigma', \sigma, x, y) : \iff \forall j \in \sigma_{-y}, \forall s \in \text{Succ}(j) : C_{\text{c}}(\sigma') + p(j, s) \leq \bar{r}_j(\sigma).
\]  

(4.6)

In the remainder of this paper, this notion is used in a more informal manner. When referring to a modified sequence \( \sigma' \) as DPC-neutral, this means that it is DPC-neutral with respect to the original sequence \( \sigma \) over its total length, that is, when DPC-neutral \((\sigma', \sigma, 1, |\sigma'|)\) is true.

Following the terminology of Jouglé (2002), we can now define under which conditions a sequence \( \sigma' \) is at least “as good as” a sequence \( \sigma \). In this context, we assume that the range of compared positions may start at an arbitrary position \( x \) whereas its upper bound \( y \) is always the last position of sequence \( \sigma \) and thus \( \sigma' \). This means that \( \sigma_{-y} = \emptyset \). Let further \( r_{\text{min}}(NS) \) denote the minimum release time among all unscheduled operations (as determined by \( \sigma \)).

**Definition 4.3.** A sequence \( \sigma' \) is at least as good as another sequence \( \sigma \) if the following two conditions are satisfied:

1. \( \sigma' \) is DPC-neutral.
2. \( C_{\text{max}}(\sigma') \leq r_{\text{min}}(NS) \) and \( G_{rel}(\sigma, \sigma', x, |\sigma'|, NS) \geq 0 \).

The first condition can be checked easily according to **Definition 4.2.** As for the second condition, the first part indicates that \( \sigma' \) is not allowed to directly delay the earliest unscheduled operation, whereas the second part simply requires \( \sigma' \) not to produce any additional (relevant) cost compared to \( \sigma \).

Based on **Definition 4.3**, we can define an even more rigorous comparison concept, providing information which of two sequences to be compared is actually “better” than the other. Slightly differing from Jouglé (2002) we added an additional condition for the case each of the two sequences are at least as good as the other.

**Definition 4.4.** A sequence \( \sigma' \) is better than a sequence \( \sigma \) if sequence \( \sigma' \) is at least as good as \( \sigma \) and if either (1) sequence \( \sigma \) is not at least as good as \( \sigma' \) or (2) \( C_{\text{max}}(\sigma') < C_{\text{max}}(\sigma) \) or (3) sequence \( \sigma' \) is lexicographically smaller than \( \sigma \).

Note that both definitions are slightly different to their counterparts described by Jouglé (2002). The latter are more general and allow \( C_{\text{max}}(\sigma') \) to exceed \( r_{\text{min}}(NS) \) under certain conditions. This extension, however, turned out to be too time-consuming in the \( TWT_{\text{max}} \) context in relation to the resulting benefits.

**5. A dominance rule based on alternative sequences**

Dominance rules are a popular and usually effective means of reducing the number of solution alternatives for algorithms which work in an enumerative fashion (Jouglet and Carlier, 2011). In a branch-and-bound algorithm for example, dominance rules may allow branches which do not lead to better solutions to be discarded. As for the problem at hand, such rules are based on the comparison of partial sequences of operations.

A dominance rule which takes into account a sequence \( \sigma \) of already scheduled operations has been proposed by Jouglé (2002) for conventional total cost one machine scheduling. In this section we describe how the underlying concept can be transferred to the \( TWT_{\text{max}} \) problem with all necessary adaptations regarding the objective function’s structure and the delayed precedence constraints. The dominance principle itself is based on the notion of “better” sequences according to **Definition 4.4**. If we can find a better permutation of an existing sequence, we know that this sequence is dominated and can be replaced. Let \( PF \) denote the set of unscheduled operations which can be immediately appended to the current partial sequence. This property holds for operations which are (1) schedulable with respect to the DPCs and (2) meet the active schedule criterion. Hence, \( PF \subseteq NS \) and is called the set of possible first operations. The dominance rule can be used to “filter” set \( PF \) in the following way:

**Theorem 5.1** Jouglé (2002). Let \( i \) be the job index of an operation from the set \( PF \) and \( \sigma \) the current partial sequence. If there exists a permutation \( \pi \) of \( \sigma|i \) which is better than \( \sigma|i \), then \( \sigma|i \) is dominated by \( \pi \).

**Definition 4.4** implies that we can safely replace a dominated sequence. In the above case this means that operation \( o_k \) can be removed from the set of possible first operations if **Theorem 5.1** applies.

The most straightforward way of searching for a better permutation would be to enumerate all possible permutations. Clearly this is impractical from a computational perspective. Instead we adopt the scheme proposed by Jouglé (2002), who enumerates permutations which are obtainable by either

- inserting \( i \) at some position in \( \sigma \) or
- interchanging \( i \) with another operation in \( \sigma \).

The insertions and interchanges are limited to the last \( k \) positions while the first part of the sequence is held fixed. The parameter setting \( k = 6 \) as determined by Jouglé (2002) for the \( TWT \) problem also proves successful in the \( TWT_{\text{max}} \) case.
Algorithm 1 describes the processing scheme of this dominance rule: The first two lines of the outer loop initialize the positional variables \( x \) and \( y \). The variable \( x \) represents the current insertion/interchange position and is varied within the range of the last \( k \) positions of \( \sigma \). The set \( OF \) represents the set of all unscheduled operations except the one which is currently being tested. In the inner loop, one interchange sequence \( \sigma^{\text{XCH}} \) and one insertion sequence \( \sigma^{\text{INS}} \) are created for each position \( x \). The decision whether to remove the current operation \( i \) from the set \( PF \) depends on the total weighted relevant gain achieved by the respective sequence. Due to efficiency considerations, we deviate in the following points from Jouglé (2002):

1. We use an even more restrictive version of Definition 4.4: if the gain equals zero, we do not verify whether the original sequence is not at least as good as the modified one by computing the gain resulting from changing the modified sequence back to the original one (which may be lower than zero). Rather, we only check the sequence completion times or perform a lexicographical comparison.
2. The sequences obtained by insertions and interchanges are not further improved with respect to local optimality.

The frequency of occurrence of better permutations of \( \sigma[i] \) could be increased if we covered all these aspects in the dominance rule. However, experimental studies have shown that the gain in effectiveness does not justify the increased computational effort in time-critical settings. Especially trying to improve the permuted sequences by applying the concept of local optimality and idle time removal turned out to be much less effective than in the total weighted tardiness case. This is mainly due to the notion of relevant gain, which prevents positive gain values to a large extent as long as sequences are not complete.

6. Lower bounding

The specific structure of the \( TWT_{\text{max}} \) objective function makes it difficult to develop appropriate and effective lower bounding schemes. Most lower bounds in the single machine literature have been designed for simple (weighted) sum objectives. The first approach explicitly considering problem 1 was the arc-fixing lower bound by Pinedo and Singer (1999). However, this concept is primarily intended for simply deciding whether the current branch can be fathomed and does not yield a bound value in the classic sense. Apart from that, excessive computation times are encountered for problems involving more than 15 operations.

In a recent paper (Braune et al., 2011), several dedicated lower bounding schemes for the problem at hand have been proposed and compared in a computational study. However, the embedding into a branch-and-bound method has not yet been investigated. Preliminary experiments revealed that only the lower bound based on due date aggregation \( LB_{\text{TWT}} \) is able to effectively prune the branching tree and to finally achieve a reduction in the overall computational effort. While its average relative gaps are loose for the initial problems, it is able to consistently outperform the other lower bounds when partial sequences already exist and the remaining subproblems get smaller. Furthermore, the computational overhead is low due to the limitation to one single due date per operation.

For the sake of completeness we briefly sketch the main ideas behind the due date aggregation based lower bound \( LB_{\text{TWT}} \). The basic principle consists in aggregating multiple local due dates to one single due date per operation and choosing corresponding weight values. The resulting condensed problem is then of type 1\( |r_j| \sum w_j T_j \), assuming that DPCs are relaxed. If it could be shown for any schedule of operations that the objective function value of the new problem is lower or equal than that of the original problem, the same applies for any lower bound of the new problem. Hence, the central goal is to find a transformation from a \( TWT_{\text{max}} \) problem to a \( TWT \) problem which is able to guarantee this property.

A simple transformation can be obtained according to the following scheme: For a given operation \( j \), an “aggregated” due date \( d_j \) is defined by simply selecting one of its local due dates. Assumed that \( z \) is the index of the selected local due date, the operation’s (aggregated) weight \( w_j \) is simply set to \( w_{j, z} \). If it is ensured that no other operation is assigned a local due date with the same index \( z \), it is easy to see that for any schedule \( S \), the sum of the weighted tardiness values \( \sum w_j T_j \) cannot exceed the \( TWT_{\text{max}} \) cost of the original problem. In fact, each operation has at least one local due date originating from the job it belongs to, hence it is possible to find at least one feasible assignment of due date indices to operations.

However, a more effective transformation leading to sharper bounds can be obtained as suggested by Braune et al. (2011): Tighter due dates \( d_j \) can be expected to lead to increased tardiness values in any particular schedule and thus to higher total cost. Intuitively, the same fact potentially applies for a lower bound on the optimal cost. A straightforward approach would therefore be to choose for each operation the earliest local due date for obtaining \( d_j \). However, operations which only have one single local due date defined may occupy indices and force other operations to fall back on later due dates.

An appropriate choice of weight values \( w_{j, z} \) yet allows to select local due dates with the same index for more than one operation. Let \( J \) be the set of operations for which a local due date with the same index \( z \) has been selected. Then \( w_j \) can safely be set to

\[
\begin{align*}
\bar{w}_j &= \frac{w_{j, z}}{|J|}, & \forall j \in J, \\
\end{align*}
\]

without violating the lower bound property of \( \sum w_j T_j \) as in Equation (6.1).

A lower bound on the simplified problem may be obtained now by applying any lower bounding scheme available for 1\( |r_j| \sum w_j T_j \). In accordance with Braune et al. (2011), we basically rely on the minimum cost of assigning operations to completion times of a preemptive SRPT schedule. Even more efficiently, a tight lower
bound on this cost can be computed using an approach proposed by Baptiste and Le Pape (2005).

7. The branch-and-bound scheme

One of our main goals in the context of the TWT\(_{\text{max}}\) single machine scheduling problem has been to develop an exact solution approach which is more efficient than existing (dedicated) approaches and conventional methodologies such as mixed integer programming or constraint programming. To the best of our knowledge, the only dedicated exact solution method for this specific problem is the enumeration algorithm as proposed by Pinedo and Singer (1999). If the aperture size of this algorithm equals the number of operations on the machine to be scheduled, it is actually equivalent to a branch-and-bound scheme. However, the effectiveness of the underlying arc-fixing lower bound drastically decreases with the problem size, in particular for \( n \gg 15 \), mainly due to its computational complexity.

As an alternative, we propose a branch-and-bound algorithm which relies on the dominance rule and the lower bounding scheme introduced in Sections 5 and 6 respectively. Our algorithm works according to the forward sequencing principle as a depth-first search procedure and is in its basic framework very similar to the algorithm described by Jouglet (2002) for standard total cost one machine scheduling. The incorporation of constraint programming techniques, both generic ones such as for example edge-finding constraint propagation and problem-specific approaches like the ones presented in Section 3, has been a major design aspect.

The basic flow structure of our branch-and-bound method is outlined in Algorithm 2. The described method Branch represents just the most fundamental framework, leaving several aspects open. In particular, the branching order is not clearly specified. In our final computational experiments we use a modified version of the ATC scheme introduced in Sections 5 and 6 respectively. Our algorithm works according to the forward sequencing principle as a depth-first search procedure and is in its basic framework very similar to the algorithm described by Jouglet (2002) for standard total cost one machine scheduling. The incorporation of constraint programming techniques, both generic ones such as for example edge-finding constraint propagation and problem-specific approaches like the ones presented in Section 3, has been a major design aspect.

The basic flow structure of our branch-and-bound method is outlined in Algorithm 2. The described method Branch represents just the most fundamental framework, leaving several aspects open. In particular, the branching order is not clearly specified. In our final computational experiments we use a modified version of the ATC scheme introduced in Sections 5 and 6 respectively. Our algorithm works according to the forward sequencing principle as a depth-first search procedure and is in its basic framework very similar to the algorithm described by Jouglet (2002) for standard total cost one machine scheduling. The incorporation of constraint programming techniques, both generic ones such as for example edge-finding constraint propagation and problem-specific approaches like the ones presented in Section 3, has been a major design aspect.

The basic flow structure of our branch-and-bound method is outlined in Algorithm 2. The described method Branch represents just the most fundamental framework, leaving several aspects open. In particular, the branching order is not clearly specified. In our final computational experiments we use a modified version of the ATC scheme introduced in Sections 5 and 6 respectively. Our algorithm works according to the forward sequencing principle as a depth-first search procedure and is in its basic framework very similar to the algorithm described by Jouglet (2002) for standard total cost one machine scheduling. The incorporation of constraint programming techniques, both generic ones such as for example edge-finding constraint propagation and problem-specific approaches like the ones presented in Section 3, has been a major design aspect.

### Algorithm 2. Outline of method Branch as the core of the proposed branch-and-bound algorithm

**Input:** A sequence \( \sigma \) representing the current partial sequence, a set \( NS \) of unscheduled operations and an initial upper bound \( UB \)

**Output:** An operation sequence \( \sigma^* \) which is optimal for the given problem

1. \( PF \leftarrow NS \);
2. Filter set \( PF \) according to dominance rules;
3. Create permutation \( \pi \) of operations in \( PF \) sorted according to a particular priority index;
4. for \( i \leftarrow 1 \) to \( |\pi| \) do
5. \( \sigma^* \leftarrow \sigma(\pi) \);
6. \( NS \leftarrow NS(\pi) \);
7. if \( NS = 0 \) then
8. Apply CP to tighten \( r_j \) and \( d_j \) for all \( j \in NS \);
9. Compute lower bound \( LB \) w.r.t. given bounding scheme;
10. if \( LB < UB \) then
11. \( \text{Branch}(\sigma^*, NS) \);
12. end
13. else
14. if \( \text{TWT}_{\text{max}}(\sigma^*) < UB \) then
15. \( UB \leftarrow \text{TWT}_{\text{max}}(\sigma^*) \);
16. \( \sigma^* \leftarrow \sigma^* \);
17. end
18. end
19. end
20. end

8. Computational results

The branch-and-bound algorithm outlined in Section 7 incorporating the customized CP techniques from Section 3 and the dominance rule based on alternative sequences (cf. Section 5) has been applied to benchmark problem instances in order to assess its performance. We analyze the impact of the presented concepts on the required computation time by successively removing them from the branch-and-bound algorithm and compare the results to those obtained by the exact approach of Pinedo and Singer (1999), mixed integer and constraint programming formulations solved by IBM ILOG CPLEX and IBM ILOG Scheduler respectively.

#### 8.1. Experimental setup

Since the primary application area of this method is the machine-based decomposition of job shops, we aimed at generating benchmark instances which typically arise in such a scenario. We obtained our single machine benchmarks through application of a shifting bottleneck procedure to total weighted tardiness job shops. The latter are created following the scheme of Pinedo and Singer (1999): Well known job shop instances initially intended for the makespan case are modified by adding a due date and a weight for each of the jobs. Precisely speaking, the job weights \( w_i \) are chosen from the set \( \{4, 2, 1\} \) such that for the first 20% of jobs \( w_i = 4 \), for the second 60% \( w_i = 2 \) and the remaining 20% \( w_i = 1 \). The due dates \( d_i \) are computed according to the formula

\[
d_i = r_i + \beta \cdot \sum_{k=1}^{m} P_k,
\]

where \( \beta \) denotes a due date tightness factor ranging between 1.3 and 1.6.

A main goal of our research has been the applicability of the branch-and-bound algorithm to one-machine problems involving considerably more than 10 operations. However, as can still be observed in weighted tardiness single machine scheduling, the complexity drastically increases with each additional operation and instances with 30 operations are already at the edge of solvability. Since the problem considered in this paper is a generalization of the total weighted tardiness problem, it can be considered at least as “difficult” (cf. Section 2.3). We therefore limit our experiments to instances with \( n \in \{15, 20, 30\} \), \( m = 10 \) and \( \beta \in \{1.3, 1.5, 1.6\} \). Actually, we rely on instances la21-25, la26-30 and la31-35 (Lawrence, 1984) for \( n = 15, 20 \) or 30, respectively. Corresponding data has been retrieved from the OR-Library1 (Beasley, 1990).

The single machine instances to which we finally apply our solution approach have been collected during the run of a shifting bottleneck procedure. It is important to remark that we are only interested in subproblems with delayed precedence constraints and thus multiple local due dates, hence the subproblems occurring at the initial stage, i.e. when none of the machine schedules has been fixed yet, are omitted. Due to the size limitations of the dedicated enumerative method (Pinedo and Singer, 1999), we relied on IBM ILOG CPLEX for solving the mixed integer formulation of the subproblems. As could be expected, this approach is not able to provide optimal subproblem solutions in many cases but it is easy to implement and allows for a better reproducibility of the experiments. The configuration of the applied shifting bottleneck procedure is summarized in Table 1.

The final number of single machine problem instances (with at least one DPC) obtained for each combination of initial problem size and due date tightness setting ranges between 700 and 2000.

---

1 Available online at http://www.people.brunel.ac.uk/~mastribjeb/info.html.
All the methods described in this paper have been implemented in C# under the Microsoft.NET framework. The experiments have been run on a Windows XP x64 workstation equipped with an Intel Core2 Duo CPU (E6850) @ 3 GHz and 4 GB RAM.

### 8.2. Main results

In this section, we present computational results obtained by our branch-and-bound method for $TWT_{\text{max}}$ problem instances with different sizes and due date tightness factors. We compare our approach to “conventional” exact solution methodologies such as mixed integer programming (MIP) and constraint programming (CP) and to the existing enumeration algorithm developed by Pinedo and Singer (1999). The MIP solver (IBM ILOG CPLEX 11.2.1) relies on the mathematical formulation provided in Section 2.2 and is run in its default configuration. As for the CP approach, IBM ILOG Scheduler 6.7 is used to establish a model of the $TWT_{\text{max}}$ problem in terms of constraint programming and to solve this model given a standard goal ($\text{IloMediumHigh}$) and to the existing enumeration algorithm developed by Pinedo and Singer (1999). The MIP solver (IBM ILOG CPLEX 11.2.1) relies on the mathematical formulation provided in Section 2.2 and is run in its default configuration. As for the CP approach, IBM ILOG Scheduler 6.7 is used to establish a model of the $TWT_{\text{max}}$ problem in terms of constraint programming and to solve this model given a standard goal ($\text{IloMediumHigh}$) and to the existing enumeration algorithm developed by Pinedo and Singer (1999).

The actual configuration of our branch-and-bound algorithm is summarized in Table 2. A custom CP procedure is applied at each node of the branch-and-bound tree in order to strengthen the subproblem formulations. This procedure relies on standard constraint based scheduling concepts (precedence graph, time-table, disjunctive and edge-finding constraint propagation) and also integrates the dedicated CP concepts presented in Section 3.

Initial upper bounds for all methods have been generated using an X-RM priority dispatching rule (Morton and Ramnath, 1995) which has been adapted to consider multiple due dates. The priority index $I_{jk}(t)$ of an operation $j$ is computed as

$$I_{jk}(t) = \left( 1 - \frac{B \max(0, r_k - t)}{\bar{p}} \right) \sum_{z=t}^{\infty} w_z \exp \left( - \frac{d_{zjk} - b_{jk} - t}{K \bar{p}} \right),$$

where $B = 0.8$, $\bar{p}$ has been set to the average processing time of unscheduled operations and the best parameter $K \in \{0.2, 0.5, 0.7, 1, 2, 3, 4, 5, 8, 10\}$ is determined experimentally for each instance such that the objective function value is as low as possible.

Table 3 provides a comparison of average and maximum computation times required for solving $TWT_{\text{max}}$ instances across different exact solution approaches, including the existing enumeration approach and our new branch-and-bound method, denoted as $\text{B&BPAS}$ and $\text{B&Bnew}$ respectively. A summary of the number of generated nodes is given in Table 4 for the dedicated approaches. Similar statistics may also be retrieved for the generic methods (MIP and CP), but they are not immediately comparable due to different branching (or backtracking) schemes and are thus omitted here.

As expected, the computational effort drastically increases with the number of operations. Also the due date tightness factor impacts the problem difficulty. Less tight due date settings, especially $h = 1.5$, lead to a higher degree of intractability, except for $n = 30$, where instances resulting from $h = 1.6$ appear to be easier to solve. Anyway, in all cases, the proposed branch-and-bound method consistently outperforms the other methods on average.

### Tables

**Table 1**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBP type</td>
<td>Single pass (no backtracking)</td>
</tr>
<tr>
<td>Subproblem opt. method</td>
<td>MIP (IBM ILOG CPLEX)</td>
</tr>
<tr>
<td>Subproblem time limit</td>
<td>300 seconds</td>
</tr>
<tr>
<td>Machine prioritization</td>
<td>Infeasibility (Braune, 2008)</td>
</tr>
<tr>
<td>Re-optimization</td>
<td>Full</td>
</tr>
</tbody>
</table>

**Table 2**

<table>
<thead>
<tr>
<th>Component</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint programming level</td>
<td>full (standard + dedicated)</td>
</tr>
<tr>
<td>Dominance rule(s)</td>
<td>alternative sequences</td>
</tr>
<tr>
<td>Lower bound</td>
<td>$L_{\text{BWT}}$</td>
</tr>
<tr>
<td>Initial upper bound</td>
<td>X-RM rule</td>
</tr>
</tbody>
</table>

**Table 3**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\beta$</th>
<th>$\text{MIP}$</th>
<th>$\text{CP}$</th>
<th>$\text{B&amp;BPAS}$</th>
<th>$\text{B&amp;Bnew}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Mean</td>
<td>Max</td>
<td>Mean</td>
<td>Max</td>
</tr>
<tr>
<td>15</td>
<td>1.3</td>
<td>1.022</td>
<td>304.7</td>
<td>0.043</td>
<td>10.5</td>
</tr>
<tr>
<td>15</td>
<td>1.5</td>
<td>1.154</td>
<td>427.5</td>
<td>0.027</td>
<td>7.5</td>
</tr>
<tr>
<td>15</td>
<td>1.6</td>
<td>0.656</td>
<td>127.5</td>
<td>0.032</td>
<td>8.8</td>
</tr>
<tr>
<td>20</td>
<td>1.3</td>
<td>9.173</td>
<td>3600</td>
<td>0.881</td>
<td>424.8</td>
</tr>
<tr>
<td>20</td>
<td>1.5</td>
<td>20.145</td>
<td>3600</td>
<td>5.875</td>
<td>2681</td>
</tr>
<tr>
<td>20</td>
<td>1.6</td>
<td>26.715</td>
<td>3600</td>
<td>10.783</td>
<td>3600</td>
</tr>
<tr>
<td>30</td>
<td>1.3</td>
<td>197.860</td>
<td>3600</td>
<td>184.150</td>
<td>3600</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>322.990</td>
<td>3600</td>
<td>265.730</td>
<td>3600</td>
</tr>
<tr>
<td>30</td>
<td>1.6</td>
<td>267.390</td>
<td>3600</td>
<td>174.080</td>
<td>3600</td>
</tr>
</tbody>
</table>

**Table 4**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\beta$</th>
<th>$\text{B&amp;BPAS}$</th>
<th>$\text{B&amp;Bnew}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Mean</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>Max</td>
<td>Mean</td>
</tr>
<tr>
<td>15</td>
<td>1.3</td>
<td>2</td>
<td>366</td>
</tr>
<tr>
<td>15</td>
<td>1.5</td>
<td>0</td>
<td>467</td>
</tr>
<tr>
<td>15</td>
<td>1.6</td>
<td>0</td>
<td>651</td>
</tr>
<tr>
<td>20</td>
<td>1.3</td>
<td>4</td>
<td>5235</td>
</tr>
<tr>
<td>20</td>
<td>1.5</td>
<td>0</td>
<td>47,105</td>
</tr>
<tr>
<td>20</td>
<td>1.6</td>
<td>0</td>
<td>100,593</td>
</tr>
<tr>
<td>30</td>
<td>1.3</td>
<td>32</td>
<td>965,111</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>45</td>
<td>1,664,454</td>
</tr>
<tr>
<td>30</td>
<td>1.6</td>
<td>0</td>
<td>1,770,672</td>
</tr>
</tbody>
</table>

Tables 5 and 6 give a more detailed overview on problem difficulty for instances with $h = 1.5$ and $n \in \{20, 30\}$. Our experiments reveal that the relative problem difficulty is primarily determined by the delayed precedence constraints. Besides the number of DPCs in a problem instance we consider the height of the precedence graph $h_{\text{DPC}}$, denoted as $h_{\text{DPC}}$ or simply $h$ in the tables. It is obvious from the result tables that problems with $h = 1$ cause the highest computational effort across all exact methods. A higher number of DPCs and/or levels in the precedence graph usually leads to a more restricted solution space and thus moderate combinatorics because many decisions concerning the relative sequence of operations are already anticipated. On the other hand, it has to be remarked that the incorporation of multiple local due dates and DPCs in general exacerbates problem solving when compared to the standard TWT case (cf. Section 2.3), even if only few of them are actually defined.

Overall, our branch-and-bound algorithm is able to solve problem instances from all difficulty levels in a very efficient manner and clearly outperforms the other methods on average. Only for some highly constrained instances, one of the existing approaches yields an optimal solution in shorter time. As for the hard instances, the computation time can be reduced by a factor of up to 20.

Apart from the average results, it has to be pointed out that some instances with $n = 30$ could not be solved to optimality within the given time frame of 3600 seconds. Table 7 lists the absolute number of unsolved instances for $n = 30$ and $h = 1.5$. Our
The branch-and-bound method is able to solve all instances to optimality except three among the hardest ones whereas the other methods fail considerably more often even for lower difficulty levels. In particular, the performance of Pinedo and Singer’s enumeration algorithm is twofold: On the one hand it is sometimes faster than the MIP or CP solvers for “easy” or small instances while it considerably deteriorates when solving the hard or large ones (cf. Tables 3 and 6).

As a last aspect of our computational study, we analyzed the performance of our branch-and-bound method under different configuration settings. Table 8 shows the corresponding comparison in terms of the average number of nodes and the required computation time instances with \( n \in \{15, 20\} \) and \( \beta = 1.5 \). The first column refers to the applied level of constraint programming. In the “medium” setting, only standard CP scheduling techniques (precedence graph, time-table, disjunctive and edge-finding constraint propagation) are applied, while the “full” setting additionally includes release time and deadline tightening according to Sections 3.1 and 3.2. The second column and the third column indicate which combination of dominance rule and lower bound has been applied.

Clearly, the incorporation of the dominance rule presented in Section 5 dramatically decreases the computation times in any of the examined configurations. CP techniques on their own, i.e. with no dominance rule currently active, are able to definitely reduce the number of nodes, whereas the computation times only decrease marginally due to the additional effort at each node of the branch-and-bound tree. The combination of CP with the proposed dominance rule yields more promising results, as can be observed for \( n = 20 \). In this context, the impact of the dedicated CP techniques as described in Section 3 has to be stressed. Standard CP scheduling concepts (level “medium”) are not sufficient to consistently improve the runtime behavior. An improvement of at least 10% compared to the version with no CP at all is only obtainable by incorporating the problem-specific CP concepts.

The impact of lower bounding on the other hand is very limited. LBTWT is only able to slightly improve the efficiency of the algorithm in combination with CP and without the dominance rule for \( n = 20 \). Without constraint programming, too many nodes remain to be bounded and LBTWT is not effective enough to fathom the required amount for reducing the overall computation time. The dominance rule on the other hand is responsible for discarding most of the branches. LBTWT is not able to successfully filter the remaining ones and thus just produces additional computational effort.

9. Conclusion

In this paper, we have presented an exact approach of branch-and-bound type for the single machine TWTmax problem. The main
focus has been put on a dominance rule based on alternative sequences involving already scheduled operations. Although the basic idea of such a rule is not new, the structure of the objective function and the existence of the delayed precedence constraints required a substantial re-design of the underlying concepts.

The dedicated CP techniques can be used to tighten release times and deadlines of the problem formulation at each stage of the branch-and-bound algorithm. In combination with generic disjunctive constraint propagation, this usually leads to a considerable reduction of combinatorics and thus to an acceleration of the algorithm.

Contrariwise, the positive impact of the lower bound is rather limited and depends on which of the other techniques are actually applied. Only in particular configurations it is able to reduce the computation time in a significant manner.

In essence, the performance of the branch-and-bound algorithm itself inherently relies on the proposed dominance rule. The latter turned out to be a very effective means of filtering the set of possible first operations. However, the combination of all the described techniques results in the smallest number of nodes and the lowest computation time on average. The computational study shows that our method is able to clearly outperform both conventional and existing dedicated exact solution approaches. The latter fail to solve a considerable proportion of hard instances involving 30 operations while our branch-and-bound algorithm is able to solve almost all instances to optimality.

The next step from a methodological point of view is of course the embedding of the subproblem solution method into a shifting bottleneck procedure for medium and large total weighted tardiness job shops and the analysis of runtime behavior and effectiveness. Since subproblem instances with a small number of delayed precedence constraints are generally hard to solve, even for our approach, computation time restrictions may have to be imposed to allow the encompassing bottleneck procedure to finish within a reasonable time frame. In such a context, the application of heuristic approaches which are able to quickly provide near-optimal solutions may be a promising alternative. Hence the development of problem-specific heuristics, in particular with the incorporation of dominance principles as outlined for conventional total cost one-machine scheduling by Jouglet et al. (2008), represents a further direction of our future research.

References