doi:10.1016/j.ejor.2018.06.004 • www.stefancreemers.be • info@stefancreemers.be

Precedence theorems and dynamic programming for the single-machine weighted tardiness problem

Salim Rostami
Stefan Creemers
Roel Leus

Abstract - **We tackle precedence-constrained sequencing on a single machine in order to minimize total weighted tardiness. Classic dynamic programming (DP) methods for this problem are limited in performance due to excessive memory requirements, particularly when the precedence network is not sufficiently dense. Over the last decades, a number of precedence theorems have been proposed, which distinguish dominant precedence constraints for a job pool that is initially without precedence relation. In this paper, we connect and extend the findings of the foregoing two strands of literature. We develop a framework for applying the precedence theorems to the precedence-constrained problem to tighten the search space, and we propose an exact DP algorithm that utilizes a new efficient memory management technique. Our procedure outperforms the state-of-the-art algorithm for instances with medium to high network density. We also empirically verify the computational gain of using different sets of precedence theorems.**

Keywords - **scheduling, single machine, precedence constraints, weighted tardiness, dynamic programming**

1 Introduction

We consider a set $N = \{1, \ldots, n\}$ of jobs (activities) and a set E of precedence constraints: for any $i, j \in N$, if $(i, j) \in E$ then job i should be scheduled before job j. More specifically, E is a strict partial order on N, i.e., it is irreflexive (pairs $(j, j) \notin E$), asymmetric (if $(i, j) \in E$ then $(j, i) \notin E$), and transitive (if $(i, j), (j, k) \in E$ then $(i, k) \in E$). Associated with each job $i \in N$ is a processing time $p_i \in \mathbb{N}_0$, a due date $d_i \in \mathbb{N}$ and a tardiness weight $w_i \in \mathbb{N}_0$. All jobs are available at time 0 to be processed on a single continuously available machine. The problem is to find a sequence $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ of the jobs that minimizes the total weighted tardiness

$$T(\mathbf{s}) = \sum_{i \in N} w_i \max\{0, C_i - d_i\},$$

where $C_i = \sum_{j=1}^{\ell} p_{s_j}$ is the earliest completion time of job i, and $s_\ell = i$. We define $B_i^E = \{j \in N | (j, i) \in E\}$ and $A_i^E = \{j \in N | (i, j) \in E\}$ as the job sets that should be processed before and after i according to E, respectively. Using the notation of Graham et al. (1979), this problem is denoted by $1|\text{prec}|\sum w_j T_j$. The problem is strongly NP-hard (Lawler, 1977).

Two related problems have received quite some attention in the scheduling literature. The single-machine scheduling problem to minimize total weighted tardiness, $1||\sum w_jT_j$, has been surveyed by Abdul-Razaq et al. (1990), who describe various dynamic programming (DP) and branch-and-bound (B&B) algorithms. Potts and Van Wassenhove (1985) propose a B&B algorithm that solves instances with up to 50 jobs to optimality within practical time and memory limits. Tanaka et al. (2009) extend the Successive Sublimation DP (SSDP) of Ibaraki and Nakamura (1994) and solve relatively large instances with up to 300 jobs. The precedence-constrained single-machine scheduling problem to minimize total weighted completion time, $1|\text{prec}|\sum w_jC_j$, has been studied by, among others, Sidney (1975); Lawler (1978); Potts (1985); Hoogeveen and van de Velde (1995); van de Velde (1995); Margot et al. (2003); Correa and Schulz (2005); Schulz and Uhan (2011). Instances with up to 100 jobs were solved to optimality already 30 years ago (Potts, 1985).

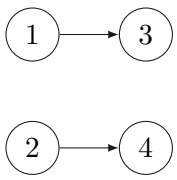
In contrast to the two aforementioned problems, the literature on $1|\text{prec}|\sum w_jT_j$, which is a generalization, is rather scarce. Schrage and Baker (1978) propose a DP method, the performance of which is very limited mainly due to memory insufficiency. Tanaka and Sato (2013) propose an extension of the algorithm of Tanaka et al. (2009) for the precedence-constrained problem that solves instances with up to 100 jobs (within practical time and memory limits) when the density of the precedence network is very low or very high. Davari et al. (2016) also report computational results for this problem, although their algorithm is developed for a generalized variant with release dates and deadlines; their algorithm solves instances with up to 50 activities.

2 Precedence theorems

Below, we will distinguish the set E of *technological* precedence constraints from the set D of all *dominant* precedence constraints, where a precedence constraint (i, j) is dominant iff there is at least one optimal solution in which i precedes j. Seeing that all feasible solutions respect E, we have $D \supseteq E$. In other words, D is the union of all optimal complete orders. A set of precedence constraints is called *acyclic* only if it is transitive and asymmetric. If there exist multiple optimal solutions then D is not acyclic. A *selection* $S \subseteq D$ is said to be dominant if there is at least one optimal solution that respects all its constraints. Consequently, acyclicity is a necessary but not sufficient condition for the dominance of sets of precedence constraints. Below, we describe precedence theorems and dominance rules to identify a dominant selection that extends E.

2.1 Precedence theorems for $E = \emptyset$

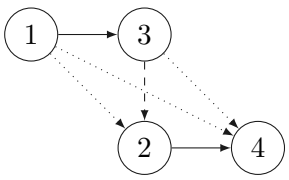
The three precedence theorems that Emmons (1969) proposes, are arguably some of the most fruitful results for $1||\sum T_j$; most of the exact approaches rely on these theorems. Later on, Rinnooy Kan et al. (1975) and Rachamadugu (1987) have extended Emmons' results to the weighted tardiness case $1||\sum w_jT_j$. These theorems distinguish dominant precedence constraints for a job pool with $E = \emptyset$. Starting from $S = \emptyset$ and using Emmons' theorems, one can add job pairs to S in an iterative fashion. Next, by solving the problem instance with precedence constraints S to optimality, an optimal solution to the original instance with



(a) Initial network

i	p_i	w_i	d_i
1	9	4	5
2	2	1	1
3	2	8	6
4	2	7	2

(b) Job parameters



(c) Transitive closure of $E \cup \{(3,2)\}$

Figure 1: An example instance

$E = \emptyset$ can be found. In line with Emmons (1969) and Rinnooy Kan et al. (1975), for any $X \subseteq N$, we define $P(X) = \sum_{i \in X} p_i$ and $\bar{X} = N \setminus X$. Similar to B_i^E and A_i^E, we define B_i^S and A_i^S based on S instead of E. Given a dominant S, and $i, j \in N$, Emmons' conditions are as follows:

E1. $p_i \leq p_j$ and $w_i \geq w_j$ and $d_i \leq \max\{d_j, P(B_j^S) + p_j\}$.

E2. $w_i \geq w_j$ and $d_j \geq \max\{d_i, P(\bar{A}_i^S) - p_j\}$.

E3. $d_j \geq P(\bar{A}_i^S)$.

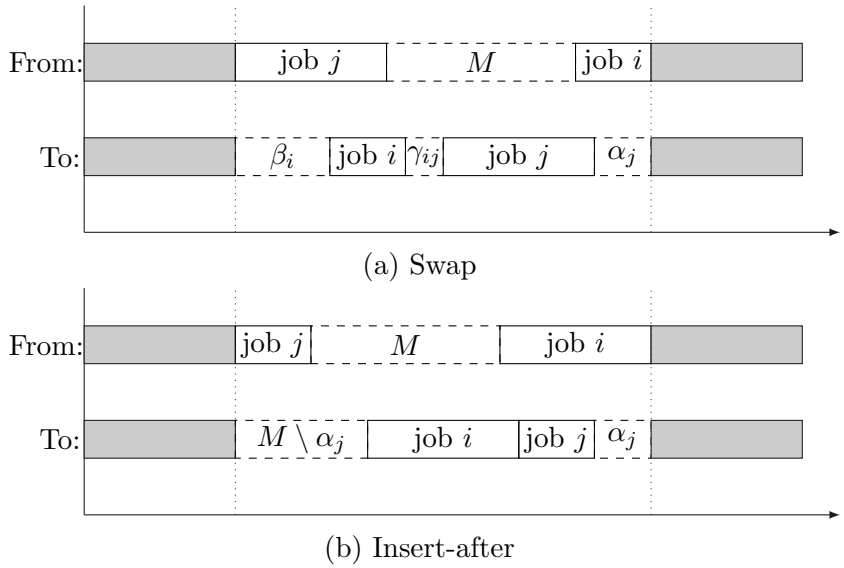
Emmons (1969) proves that when $E = \emptyset$, any of these conditions is sufficient to conclude $(i, j) \in D$. More recently, Kanet (2007) has generalized Emmons' results with seven new conditions (K1 to K7). These are stated in Appendix. Emmons (1969) and Kanet (2007) show that combining the dominant constraints that are identified by these theorems iteratively does not remove all optimal solutions, i.e., any thus-obtained S is dominant iff it is acyclic.

Given a dominant S and job pair (i, j), we define $I(S, i, j)$ as the indicator function of Emmons' and Kanet's theorems that returns 1 if (i, j) satisfies at least one condition, and 0 otherwise. Therefore, when $E = \emptyset$, $I(S, i, j) = 1$ implies $(i, j) \in D$. We also define $C(S) = E \cup \{(i, j) | I(S, i, j) = 1\}$. When $E = \emptyset$ then $C(S) \subseteq D$, but $C(S)$ is not necessarily acyclic. Moreover, extending S iteratively can only improve the theorem conditions for other job pairs to be identified as dominant. Hence, for given dominant S_1 and S_2 the following result is intuitive.

Proposition 1. *If $S_1 \subset S_2$ then $C(S_1) \subseteq C(S_2)$.*

2.2 Extended precedence theorems for general E

With a general set E and for any (i, j), the acyclicity of $E \cup \{(i, j)\}$ becomes a necessary condition for the dominance of (i, j). Furthermore, the precedence theorems that were discussed in Section 2.1 may not be applicable as is. Consider the example depicted in Figure 1, with $E = \{(1, 3), (2, 4)\}$ and $S = E$. We investigate an additional precedence constraint from job 3 to job 2. Since $I(E, 3, 2) = 1$ (based on K1, K4 and K5), we add the pair $(3, 2)$ to S. As depicted in Figure 1c, $(3, 2)$ implies the transitive edges $(1, 2)$, $(1, 4)$ and $(3, 4)$. Thus, we end up with the sequence $\mathbf{s}^1 = (1, 3, 2, 4)$ with $T(\mathbf{s}^1) = 159$, while for the optimal sequence $\mathbf{s}^* = (2, 4, 1, 3)$, $T(\mathbf{s}^*) = 119$. The two transitive edges $(1, 2)$ and $(1, 4)$ are not dominant, and consequently remove the optimal solutions. This counterexample



(a) Swap

(b) Insert-after

Figure 2: Swap and insert-after strategies when $E \neq \emptyset$

shows that with general E, Kanet's and Emmons' conditions cannot be directly invoked, i.e., $I(S, i, j) = 1$ is not sufficient to conclude $(i, j) \in D$. Hence, if $E \neq \emptyset$ then $C(S)$ is not necessarily a subset of D.

Kanet (2007) uses "swap" and "insert-after" strategies to prove his dominance theorems. Conditions E2–3 and K4–7 are obtained via the insert-after strategy, while Conditions E1 and K1–3 are derived using the swap strategy. Condition K1 generalizes E1, K4 and K5 generalize E2, K7 is the same as E3, and K2, K3 and K6 are entirely new in the sense that they can lead to the conclusion that a pair $(i, j) \in D$ even when $w_i < w_j$.

An illustration of the swap and insert-after strategies for general E and a given dominant S is provided in Figure 2, where $\beta_i = (M \cap B_i^E)$, $\alpha_j = (M \cap A_j^E)$ and $\gamma_{ij} = M \setminus (\alpha_j \cup \beta_i)$. The symbol M represents the set of intermediate jobs between i and j. "From" represents *any* sequence that respects S, and "To" is the resulting sequence after swapping j and i or inserting j after i. The latter sequence respects E but not necessarily S, i.e., a number of dominant precedence constraints in $S \setminus E$ might be violated. A sufficient condition for the dominance of (i, j) has the structure

$$\mathrm{LB}(\mathrm{TI}(i)) \geq \mathrm{UB}(\mathrm{TD}(j)) + \mathrm{UB}(\mathrm{TD}(\gamma_{ij})) + \mathrm{UB}(\mathrm{TD}(\alpha_j)), \tag{1}$$

where $\mathrm{LB}(\cdot)$ and $\mathrm{UB}(\cdot)$ are lower and upper bound functions, respectively, $\mathrm{TI}(i)$ is the tardiness improvement of job i, and $\mathrm{TD}(i)$ the tardiness degradation. Note that $\mathrm{TD}(\beta_i) = 0$.

If an activity pair (i, j) satisfies Condition (1) for *every* feasible M and $G(N, E \cup \{(i, j)\})$ is acyclic, then if j precedes i in a given schedule, we can exchange the two jobs without increasing the tardiness function. Thus, for an acyclic set of activity pairs $\{(i, j), (k, l), \dots\}$ that each satisfy Condition (1), any optimal schedule that is not compatible with one or more of these pairs cannot be harmed by making as many interchanges as necessary to obtain an optimal schedule that respects all the pairs. Therefore, if the "To" sequence does not respect S, then by a finite number of swaps and insert-afters, it can be transformed into a sequence that respects S, such that the final sequence is at least as good as the intermediate

sequences. Given an instance $G(N, E)$, let $V \subseteq D$ be the set of all activity pairs that satisfy Condition (1). We conclude:

Proposition 2. *Any $S \supseteq E$ for which $(S \setminus E) \subseteq V$ is dominant iff S is acyclic.*

Hence, we search for an inclusion-maximal acyclic $S \supseteq E$ such that $(S \setminus E) \subseteq V$.

In Condition (1), the completion times of jobs i and j depend on $P(\alpha_j)$ and $P(\beta_i)$, so $\mathrm{TI}(i)$ and $\mathrm{TD}(j)$ depend on β_i and α_j. Also, $\mathrm{TD}(\alpha_j)$ can be positive in both strategies. Finally, even if $p_i \leq p_j$, the value $\mathrm{TD}(\gamma_{ij})$ can still be positive in the swap strategy. We therefore extend Emmons' and Kanet's theorems under the extra requirement that $\alpha_j = \beta_i = \emptyset$.

Proposition 3. *$A_j^E \subseteq A_i^S$ is a sufficient condition for $\alpha_j = \emptyset$.*

Proof. Proof. Remember that $\alpha_j = M \cap A_j^E$. The requirement that α_j is empty means all jobs in A_j^E are scheduled after job i. Intuitively, the condition $A_j^E \subseteq A_i^E$ is sufficient to ensure $\alpha_j = \emptyset$. Since the "From" sequence is feasible not only to E but also to S, the condition $A_j^E \subseteq A_i^S$ is also sufficient. □

Analogously, we can prove:

Proposition 4. *$B_i^E \subseteq B_j^S$ is a sufficient condition for $\beta_i = \emptyset$.*

The conditions $A_j^E \subseteq A_i^E$, $A_j^S \subseteq A_i^E$ and $A_j^S \subseteq A_i^S$ are also sufficient for $\alpha_j = \emptyset$ and $B_i^E \subseteq B_j^E$, $B_i^S \subseteq B_j^E$ and $B_i^S \subseteq B_j^S$ are also sufficient for $\beta_i = \emptyset$, but $A_j^E \subseteq A_i^S$ and $B_i^E \subseteq B_j^S$ are easier to fulfill compared to the other alternatives, since $E \subseteq S$.

When $\alpha_j = \beta_i = \emptyset$, a sufficient condition for the dominance of (i, j) takes the form

$$\mathrm{LB}(\mathrm{TI}(i)) \geq \mathrm{UB}(\mathrm{TD}(j)) + \mathrm{UB}(\mathrm{TD}(M)). \tag{2}$$

Kanet (2007) proves that given a dominant S and $(i, j) \in N \times N$, if $I(S, i, j) = 1$ then (i, j) satisfies Condition (2). We summarize our findings with the following formal statements.

Proposition 5. *When $E \neq \emptyset$ then each of Conditions E1 and K1–3 together with $A_j^E \subseteq A_i^S$ and $B_i^E \subseteq B_j^S$ imply $(i, j) \in D$.*

Proposition 6. *When $E \neq \emptyset$ then each of Conditions E2–3 and K4–7 together with $A_j^E \subseteq A_i^S$ imply $(i, j) \in D$.*

In the instance of Figure 1, $(3, 2) \notin D$ because $A_2^E \not\subseteq A_3^S$ and $B_3^E \not\subseteq B_2^S$.

Given a dominant $S \supseteq E$, and a pair (i, j) such that $S \cup \{(i, j)\}$ is acyclic, we define

$$\Gamma(S, i, j) = \{(k, l) \in N \times N | k \in (B_i^E \setminus B_j^S) \cup \{i\}, l \in (A_j^E \setminus A_i^S) \cup \{j\}\} \tag{3}$$

as the set of all transitive pairs associated with (i, j) that are not yet included in S.

Proposition 7. *If $\Gamma(S, i, j) \subseteq C(S)$ then $(i, j) \in D$.*

Proof. Proof. Given a set X of pairs, let X^+ be the transitive closure of X. Assume that we add the pairs in $\Gamma(S,i,j)$ to S, sequentially. We will do this in a specific order. If $\Gamma(S,i,j) \subseteq C(S)$ then in the first step, there exist $(k_1,l_1) \in \Gamma(S,i,j)$ and $S_1 = S \cup \{(k_1,l_1)\}$ such that $S_1^+ \cap (\Gamma(S,i,j) \setminus S_1) = \emptyset$. In other words, adding (k_1,l_1) to S does not imply any transitive pairs within $\Gamma(S,i,j)$. For such a pair (k_1,l_1) we have $A^E_{l_1} \subseteq A^S_{k_1}$ and $B^E_{k_1} \subseteq B^S_{l_1}$. Consequently, based on Proposition 5-6, $(k_1,l_1) \in D$. Analogously, in step $q > 1$, there exist $(k_q,l_q) \in \Gamma(S,i,j)$ and $S_q = S_{q-1} \cup \{(k_q,l_q)\}$ such that $S_q^+ \cap (\Gamma(S,i,j) \setminus S_q) = \emptyset$. Thus, $(k_q,l_q) \in D$. We conclude that if $\Gamma(S,i,j) \subseteq C(S)$ then all the pairs in $\Gamma(S,i,j)$ including (i,j) can be verified to be dominant and $S \cup \{(i,j)\}$ is a dominant set. □

2.3 Algorithmic application of the precedence theorems

In this section, we illustrate the algorithmic application of Proposition 7. Given a dominant partial order S that extends E, we propose a framework, Frame1, for evaluating the dominance of a given pair (i,j) without generating $C(S)$ explicitly.

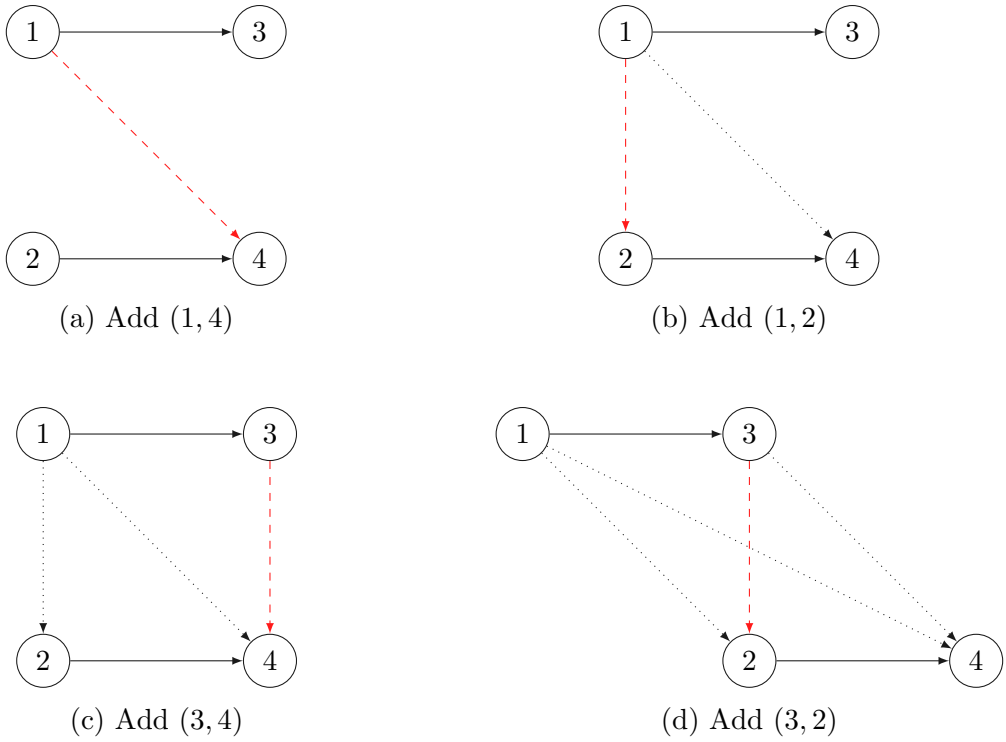
The idea of Frame1 is to sequentially add the pairs in $\Gamma(S,i,j)$ to S such that each addition entails no transitive pair within $\Gamma(S,i,j)$ (as in the proof of Proposition 7). To this end, for each $(k,l) \in \Gamma(S,i,j)$ we determine the longest path between k and l in the transitive reduction of the graph $G(N, S \cup \{(i,j)\})$. We assume unit length (weight) for all the edges (activity pairs). The Floyd-Warshall algorithm, for instance, can be used to calculate these longest paths efficiently. We define $L(S,i,j)$ as a sequence of the pairs in $\Gamma(S,i,j)$ in non-increasing order of their corresponding longest path length.

Next, Frame1 checks the pairs in $L(S,i,j)$ sequentially. Let (k_q,l_q) be the q^{th} element of $L(S,i,j)$. In the first step, we check (k_1,l_1): if $I(S,k_1,l_1) = 1$ then we define $S_1 = S \cup \{(k_1,l_1)\}$ and we proceed to the next step; otherwise, we terminate the framework by concluding that $(i,j) \notin D$. Analogously, in any step $q > 1$, if $I(S_{q-1},k_q,l_q) = 1$ then we construct $S_q = S_{q-1} \cup \{(k_q,l_q)\}$. Note that by considering the pairs in $\Gamma(S,i,j)$ in the order of $L(S,i,j)$, based on Proposition 1, we benefit most from possible improvements in the theorem conditions for identifying the dominance of (k_q,l_q). If all the activity pairs in $L(S,i,j)$ are successfully added to S, then the framework ends with a dominant $S' \supset E$ that includes (i,j).

Figure 3 provides an illustration of the application of Frame1 to the instance depicted in Figure 1a. For this example we have $L(E,3,2) = ((1,4),(1,2),(3,4),(3,2))$ for the corresponding longest path lengths $3,2,2$ and 1 in the transitive reduction of $G(N, E \cup \{(3,2)\})$. Given the parameters in Figure 1b, the framework terminates in Figure 3a as $I(E,1,4) = 0$, by concluding that $(3,2) \notin D$. For other parameter values, if $(3,2) \in D$ then the framework adds further pairs as depicted in Figures 3b-3d.

3 DP algorithm

In this section, we propose a DP algorithm for solving $1|\text{prec}|\sum w_jT_j$. The recursion (Section 3.1) is the same as in Schrage and Baker (1978), but our DP utilizes a more efficient memory management technique (Section 3.2) that enables us to solve larger instances with



(a) Add $(1, 4)$ (b) Add $(1, 2)$

(c) Add $(3, 4)$ (d) Add $(3, 2)$

Figure 3: An example of the application of Frame1

the same memory limit. In order to decrease the size of the state space, we replace E by the inclusion-maximal dominant selection $S \supseteq E$ that was discussed in Section 2.

3.1 DP recursion

Each DP state $Y \subseteq N$ represents a subproblem with $|Y|$ to-be-scheduled jobs, where the first $n - |Y|$ positions of the sequence are filled and we decide the job in the $(n - |Y| + 1)^{\text{th}}$ position. The state space Φ contains all feasible states. A state Y is feasible if it respects the precedence constraints, that is, $\forall i \in Y : A_i^S \subset Y$.

Given a state $Y \in \Phi$, let $Q(Y) = \{i \in Y \mid B_i^S \cap Y = \emptyset\}$ be the set of tasks that are eligible to be scheduled. Hence, $Q(Y)$ contains all the possible decisions to be made in Y. Selecting job $i \in Q(Y)$ results in a transition to state $Y \setminus \{i\}$. The value function F computes the minimum total cost for the subproblem corresponding to Y. Defining $C_i = P(\bar{Y}) + p_i$, the value function can be computed via the backward recursion

$$F(Y) = \min_{i \in Q(Y)} \{w_i \max\{0, C_i - d_i\} + F(Y \setminus \{i\})\}. \tag{4}$$

Starting from the unique final state $Y = \emptyset$ with $F(\emptyset) = 0$, the recursion iteratively calculates the objective value for the preceding states by (4).

3.2 DP memory management

Schrage and Baker (1978) generate Φ in increasing (lexicographic) order of the binary representation $\tau(Y) = \sum_{i \in Y} 2^{i-1}$ of the states Y. The main drawback of this approach is that

it requires the storage of the entire state space. We propose a novel memory management technique that drastically reduces the number of states that are stored in memory simultaneously. In the recursion, any state that immediately follows an arbitrary Y consists of $|Y|-1$ jobs. For any $f \leq n$, let $\Upsilon^f \subset \Phi$ denote the set of all feasible states Y with $|Y| = f$. Also, for a given $Y \in \Phi$, let $H(Y) = \{i \in \bar{Y} \mid A_i^S \subseteq Y\}$ be the set of jobs in $\bar{Y}$ with all successors in Y. Hence, for any $i \in H(Y)$, $Y \cup \{i\}$ is a feasible state. For a given f $(0 < f \leq n)$ we have

$$\Upsilon^f = \bigcup_{Y \in \Upsilon^{f-1}} \{Y \cup \{i\} \mid i \in H(Y)\} \tag{5}$$

and $\Upsilon^0 = \{\emptyset\}$. Thus, for any $f \leq n$, the generation of Υ^f depends only on Υ^{f-1}. In the example of Figure 1a, from $\Upsilon^0 = \{Y_0\}$ with $Y_0 = \emptyset$ we have $H(Y_0) = \{3, 4\}$, and $\Upsilon^1 = \{Y_1, Y_2\}$ with $Y_1 = \{4\}$ and $Y_2 = \{3\}$. Moreover, according to (4), for any state $Y \in \Upsilon^f$ the objective value $F(Y)$ is determined by the objective value of states in Υ^{f-1}. We conclude that, once Υ^f is created (Equation (5)) and the objective value of its members is computed (Equation (4)), the set Υ^{f-1} can be discarded from memory. This means that at any time during the execution of the DP we need to store at most two sets of states (Υ^{f-1} and Υ^f). This enables us to solve larger instances (with respect to the number of jobs) that were not solvable (within practical memory limits) before.

Next, we elaborate an algorithm that generates Υ^f from Υ^{f-1}. The elements of the resulting Υ^f are generated in decreasing order of their binary representation if the states in Υ^{f-1} are processed in decreasing binary order, and for each state $Y \in \Upsilon^{f-1}$, the activities in $H(Y)$ are scanned in decreasing index order (i precedes j if $i > j$). In the previous example, we consider the states in Υ^1 (with $\tau(Y_1) = 8$ and $\tau(Y_2) = 4$) in decreasing binary order (Y_1 precedes Y_2), and the elements of $H(Y_1) = \{2, 3\}$ and $H(Y_2) = \{1, 4\}$ in decreasing index order (e.g., for Y_1, 3 precedes 2). Consequently, the first generated state is $Y_3 = \{3, 4\}$, which is obtained by adding job 3 to Y_1. The next generated states are $Y_4 = \{2, 4\}$, $Y_5 = \{3, 4\}$ and $Y_6 = \{1, 3\}$, respectively. Since $Y_5 = Y_3$, the resulting Υ^2 is generated in decreasing binary order.

To check whether a newly generated state Y' already exists in Υ^f i.e., $Y' \in \Upsilon^f$, one straight-forward way is to employ *binary search* in Υ^f, with time complexity $\mathcal{O}(\log |\Upsilon^f|)$. However, the binary-ordered generation of the states in Υ^f allows us to perform this check in $\mathcal{O}(1)$. The algorithmic description of this subroutine is provided in Algorithm 1, where $H(Y)[i]$ is defined as the i^{th} element in $H(Y)$. Moreover, let Υ_i^f be the i^{th} element of Υ^f, and u be counter of the states of Υ^f that are already generated by Algorithm 1. Assuming that the job indices form a topological order of $G(N, S)$, we prove:

Theorem 1. *In Algorithm 1, Any newly generated state Y' has not been generated before iff $\tau(Y') < \tau(\Upsilon_u^f)$.*

Proof. Proof. Given Υ^{f-1}, we intend to obtain Υ^f. We assume that the states in Υ^{f-1} are in decreasing binary order, i.e., for any $2 \leq i \leq |\Upsilon^{f-1}|$ we have $\tau(\Upsilon_{i-1}^{f-1}) > \tau(\Upsilon_i^{f-1})$. We also assume that for each $Y \in \Upsilon^{f-1}$ the jobs in $H(Y)$ are in decreasing index order, i.e., for $2 \leq i \leq |H(Y)|$ we have $H(Y)[i-1] > H(Y)[i]$. In the first step, Υ_1^f is produced by adding $H(\Upsilon_1^{f-1})[1]$ to Υ_1^{f-1}. This state is obviously unique. In any step $q > 1$, a state Y' is generated. If Y' is created from the same state in Υ^{f-1} as for Υ_u^f then we have

Algorithm 1 Υ^f generation subroutine

Input: Υ^{f-1} in decreasing binary order **and** $\forall Y \in \Upsilon^{f-1} : H(Y)$ in decreasing index order
Output: Υ^f in decreasing binary order
$u = 1$
$\Upsilon_1^f = \Upsilon_1^{f-1} \cup H(\Upsilon_1^{f-1})[1]$
for $i = 1$ **to** $|\Upsilon^{f-1}|$ **do**
  for $j = 1$ **to** $|H(\Upsilon_i^{f-1})|$ **do**
    $Y' = \Upsilon_i^{f-1} \cup H(\Upsilon_i^{f-1})[j]$
    if $\tau(Y') < \tau(\Upsilon_u^f)$ **then**
      $u = u + 1$
      $\Upsilon_u^f = Y'$
    end if
  end for
end for
return Υ^f

$\tau(Y') < \tau(\Upsilon_u^f)$, because a job with smaller index is added to the same state. Next, we assume that Υ_u^f and Y' are produced from Υ_a^{f-1} and Υ_b^{f-1}, respectively, with $a < b$. Let $g = \min\{i \in Y_b^{f-1}\}$ and $h = \max\{i \in \Upsilon_a^{f-1} \setminus \Upsilon_b^{f-1}\}$. Since $g \in Y_b^{f-1}$ while $h \notin Y_b^{f-1}$, we have $h \neq g$. If $h < g$ then $\Upsilon_a^{f-1} < \Upsilon_b^{f-1}$, which contradicts the decreasing order of the states in Υ^{f-1}. Therefore, $h > g$. We have $\tau(Y') \geq \tau(\Upsilon_u^f)$ only if $H(\Upsilon_b^{f-1})[1] \geq h$. In this case, since the job indices form a topological order of $G(N, S)$, we know that $B_g^S \cap \Upsilon_b^{f-1} = \emptyset$ and $g \notin A^S_{H(\Upsilon_b^{f-1})[1]}$. Consequently, $Y^* = \Upsilon_b^{f-1} \setminus \{g\} \cup \{H(\Upsilon_b^{f-1})[1]\}$ is a feasible state with cardinality $f - 1$. Since $\tau(Y^*) > \tau(\Upsilon_b^{f-1})$, there exists $c \leq a$ such that the state $\Upsilon_c^{f-1} = Y^*$. Knowing that $Y' = Y^* \cup \{g\}$, we conclude that Y' has already been generated. If $\tau(Y') < \tau(\Upsilon_u^f)$, on the other hand, then Y' is new. In conclusion, Y' has already been generated iff $\tau(Y') \geq \tau(\Upsilon_u^f)$. □

In the previous example we have $\tau(Y_5) > \tau(Y_4)$, so Y_5 is not stored.

Proposition 8. *For each f ($0 \leq f \leq n$), Υ^f generated by Algorithm 1 includes all the states $Y \in \Phi$ with $|Y| = f$, and consequently, $\bigcup_{0 \leq f \leq n} \Upsilon^f = \Phi$.*

Proof. Proof. Given Υ^{f-1}, and at each iteration corresponding to the loop counter i, Algorithm 1 considers all the jobs in $H(\Upsilon_i^{f-1})$. Therefore, it generates all the feasible states that can be obtained from Υ_i^{f-1}. Although the algorithm stores only the states Y' with $\tau(Y') < \tau(\Upsilon_u^f)$, Theorem 1 proves that all the discarded states were already generated and stored by the algorithm in the previous iterations. Hence, the resulting Υ^f includes all the feasible states Y with $|Y| = f$, and consequently $\bigcup_{0 \leq f \leq n} \Upsilon^f = \Phi$. □

ρ	n					
	40		50		100	
	min	avg	min	avg	min	avg
0.2	0.43	0.63	0.56	0.68	0.77	0.82
0.1	0.15	0.28	0.21	0.35	0.49	0.57
0.05	0.04	0.10	0.06	0.11	0.14	0.23
0.02	0.01	0.03	0.01	0.03	0.02	0.04
0.01	< 0.01	0.01	< 0.01	0.01	< 0.01	0.01
0.005	0.00	0.01	< 0.01	0.01	< 0.01	0.01
0	0.00	0.00	0.00	0.00	0.00	0.00

Table 1: OS1 of the Tanaka dataset

4 Computational results

All the experiments are performed on an Intel Core i5-4590, 3.3 GHz computer with 32 GB RAM. The memory consumption and CPU times are limited to 8 GB and 5400 seconds, respectively.

4.1 Instances

We consider two sets of instances to evaluate the performance of our DP algorithm. The first dataset is obtained from the OR-Library instances of $1||\sum w_j T_j$ with 40, 50 and 100 jobs[1]. Tanaka and Sato (2013) then add precedence constraints to the instances as in Potts and Van Wassenhove (1985) and Hoogeveen and van de Velde (1995): for any $i, j \in N, i < j$, the precedence constraint (i, j) is imposed with a specified probability ρ, which is chosen from $\{0.005, 0.01, 0.02, 0.05, 0.1, 0.2\}$. We refer to this dataset as the *Tanaka* dataset[2].

In our experiments we have observed that the probability ρ is not a suitable predictor for the difficulty of an instance (required runtime and memory), and that the the *order strength* (OS) is far more informative. The value OS measures the density of the precedence network, and is defined as the number of precedence-related activity pairs divided by the maximum possible number of such pairs. Demeulemeester et al. (2003) propose the random network generator RanGen, which takes OS as input parameter for the generation. Below, we denote the OS of the generated instance before and after applying precedence theorems by OS1 and OS2, respectively.

In Table 1 we report the minimum (min) and average (avg) OS1 of Tanaka's instances. The table shows that very low OS values are over-represented, i.e., 57% of the instances ($\rho \in \{0, 0.005, 0.01, 0.02\}$) have very low densities (OS1 ≤ 0.04), and consequently they will have very similar runtime and memory requirements. Moreover, each ρ value results in instances with various OS values, and so instances of different difficulty levels would be categorized within the same setting (n and ρ). We therefore use RanGen to generate a second set of instances where OS1 $\in \{0, 0.2, 0.4, 0.6, 0.8\}$ and $n \in \{20, 40, 60, 80, 100, 120\}$. We will

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html`

[2] `https://sites.google.com/site/shunjitanaka/prec-single`

OS1	n											
	20		40		60		80		100		120	
	#	Δ	#	Δ	#	Δ	#	Δ	#	Δ	#	Δ
0.8	125	21.24	125	16.55	125	13.11	125	10.87	125	10.45	125	8.93
0.6	125	25.00	125	17.01	125	14.29	125	11.37	125	10.40	125	10.32
0.4	125	26.02	125	16.66	125	13.26	125	13.01	112	9.92	33	10.33
0.2	125	26.43	125	17.05	125	12.61	37	10.49	20	12.85	11	9.61
0	125	30.66	125	21.04	125	16.13	98	12.72	72	13.35	56	9.47

Table 2: Partitioning Φ: improvements in memory requirement

n	20	40	60	80	100	120
Δ	33.64	24.48	20.18	17.57	15.76	14.42

Table 3: Partitioning Φ when $S = E = \emptyset$

refer to this dataset as the *standard* dataset. The same method and parameters are used as in Potts and Van Wassenhove (1985) to generate job durations, due dates and tardiness weights. Hence, for each setting (each n and OS1), we generate 125 instances, which adds up to 3750 instances in total.

4.2 Memory gain of partitioning the state space

Table 2 provides details on the efficiency of the memory management technique proposed in Section 3.2. We define

$$\Delta = \frac{\max_{0<f\leq n}\{\Upsilon^{f-1} + \Upsilon^{f}\}}{|\Phi|} \times 100$$

as the maximum percentage of Φ that is simultaneously stored in memory. The results pertain to the standard dataset; the average Δ is computed over the instances that are successfully solved to optimality (#). Based on the table, our technique for partitioning the state space is between 3.26 and 11.20 times more efficient than storing the entire state space (as done by Schrage and Baker (1978)). The difference is higher for larger instances, for which memory management is of higher importance.

For the instances without precedence constraints ($S = E = \emptyset$), we can express the memory gain of our proposed technique in closed form. Given an instance of size n, we have $|\Phi| = \sum_{0\leq f\leq n} \binom{n}{f}$, and $\max_{0<f\leq n}\{\Upsilon^{f-1} + \Upsilon^{f}\} = \binom{n}{n/2-1} + \binom{n}{n/2}$. The results are provided in Table 3. When $S = E = \emptyset$, partitioning the state space decreases the memory requirement up to 85.58% (for $n = 120$).

4.3 Computational gain of Kanet's theorems

Since Kanet's theorems extend Emmons' results, the former will be at least as effective as the latter in tightening the precedence constraints. We compare the performance of the DP for the standard dataset with three settings: (1) when none of the dominance rules are

n	OS1	Kanet			Emmons			None	
		OS2	#	CPU	OS2	#	CPU	#	CPU
40	0.8	0.91	**125**	**0.00**	0.90	**125**	**0.00**	**125**	**0.00**
	0.6	0.78	**125**	**0.00**	0.77	**125**	**0.00**	**125**	**0.00**
	0.4	0.63	**125**	**0.01**	0.62	**125**	**0.01**	125	0.09
	0.2	0.48	**125**	**0.26**	0.46	125	0.28	125	7.77
	0	0.65	**125**	**0.35**	0.60	125	0.50	0	-
60	0.8	0.90	**125**	**0.00**	0.89	**125**	**0.00**	**125**	**0.00**
	0.6	0.77	**125**	**0.01**	0.76	**125**	**0.01**	125	0.08
	0.4	0.61	**125**	**0.46**	0.60	125	0.51	125	10.82
	0.2	0.45	**125**	**90.36**	0.44	125	96.64	80	3,654.03
	0	0.63	**125**	**59.13**	0.61	125	78.04	0	-
80	0.8	0.90	**125**	**0.00**	0.89	**125**	**0.00**	**125**	**0.00**
	0.6	0.76	**125**	**0.08**	0.75	125	0.10	125	1.33
	0.4	0.60	**125**	**44.00**	0.59	125	48.52	123	735.57
	0.2	0.43	**37**	**469.77**	0.43	36	412.28	0	-
	0	0.63	**98**	**225.63**	0.61	97	234.97	0	-
100	0.8	0.89	**125**	**0.00**	0.89	**125**	**0.00**	125	0.03
	0.6	0.75	**125**	**1.09**	0.75	125	1.42	125	23.18
	0.4	0.59	**112**	**970.52**	0.59	111	894.18	0	-
	0.2	0.42	**20**	**332.99**	0.41	20	403.56	0	-
	0	0.64	**72**	**233.10**	0.62	71	256.95	0	-

Table 4: Computational gain of precedence theorems

applied, (2) when Emmons' conditions are used for pre-processing and (3) after employing Kanet's theorems. Table 4 presents the results, where the average OS2 is calculated over all instances of each setting (125), while the average CPU is computed over the solved instances (#). Not surprisingly, increasing the density of the precedence graphs via the dominance rules significantly improves the performance of the DP. The improvement of Kanet's theorems over Emmons' theorems, on the other hand, is less significant. With these observations, we address Kanet's concluding suggestion in his 2007 article to conduct a full-scale computational study to test the utility of his theorems.

4.4 Comparing algorithms

Tables 5 and 6 compares the performance of our DP algorithm with the state-of-the-art algorithm proposed by Tanaka and Sato (2013) (denoted by *Tan* in the tables). We apply Kanet's theorems to each instance before solving it via DP. For Tan, on the other hand, we have observed that tightening the precedence graphs consistently increases the CPU requirements, and we therefore apply Tan to the instances with the original (untightened) precedence graph. As before, CPU is the average runtime of the instances solved to optimality, and # denotes the number of solved instances (out of 125). For each setting, the best algorithm is indicated in bold, where the best performance is decided first based on # and

n	OS1	OS2	DP		Tan	
			#	CPU	#	CPU
20	0.8	0.94	**125**	**0.00**	125	0.05
	0.6	0.82	**125**	**0.00**	125	0.03
	0.4	0.68	**125**	**0.00**	125	0.06
	0.2	0.54	**125**	**0.00**	125	0.09
	0	0.68	**125**	**0.00**	125	0.02
40	0.8	0.91	**125**	**0.00**	125	0.17
	0.6	0.78	**125**	**0.00**	125	2.20
	0.4	0.63	**125**	**0.01**	125	7.37
	0.2	0.48	**125**	**0.26**	125	10.93
	0	0.65	125	0.35	**125**	**0.15**
60	0.8	0.9	**125**	**0.00**	125	2.28
	0.6	0.77	**125**	**0.01**	125	18.57
	0.4	0.61	**125**	**0.46**	125	116.21
	0.2	0.45	**125**	**90.36**	120	360.75
	0	0.63	125	59.13	**125**	**0.47**

n	OS1	OS2	DP		Tan	
			#	CPU	#	CPU
80	0.8	0.9	**125**	**0.00**	125	8.26
	0.6	0.76	**125**	**0.08**	125	142.31
	0.4	0.6	**125**	**44.00**	83	1,014.55
	0.2	0.43	37	469.77	**51**	**770.89**
	0	0.63	98	225.63	**125**	**1.10**
100	0.8	0.89	**125**	**0.00**	125	22.81
	0.6	0.75	**125**	**1.09**	118	1,005.92
	0.4	0.59	**112**	**970.52**	25	835.34
	0.2	0.42	20	332.99	**23**	**299.45**
	0	0.64	72	233.10	**125**	**2.85**
120	0.8	0.89	**125**	**0.01**	125	70.08
	0.6	0.75	**125**	**14.13**	44	1,272.65
	0.4	0.59	**33**	**713.29**	15	665.01
	0.2	0.41	11	658.58	**17**	**583.52**
	0	0.62	56	418.16	**125**	**5.98**

Table 5: CPU times – standard dataset

then on CPU. In search for a second benchmark algorithm, we have also tested the use of ILOG CP Optimizer 12.8 (after tightening the networks with the precedence theorems), but this led to results that were consistently (far) inferior to DP and Tan, and so the results are not reported.

Table 5 shows that our DP outperforms Tan in most of the settings for the standard dataset. Our algorithm is superior in all instances with $n = 20$, in all instances with OS1> 0 when $40 \leq n \leq 80$ and in the instances with medium to high density ($0.4 \leq$ OS ≤ 0.8) when $n \geq 80$. This is reasonable, since Tan is an extension of the algorithm of Tanaka et al. (2009) for the scheduling problem without precedence constraints and hence, can be expected to perform well especially for low OS. In total, our DP outperforms Tan in 22 out of the 30 settings. Note that in some settings, our DP is faster than Tan by orders of magnitude. When $n = 100$ and OS $= 0.6$, for instance, our DP is at least 922 times faster than Tan. In all the instances that our DP cannot solve, the algorithm is interrupted due to insufficient memory; in other words, memory is the main bottleneck of our algorithm. For the unsolved instances of Tanaka and Sato's algorithm, either memory or CPU limits are reached first, depending on the instance.

Table 6 compares our DP and Tan on the Tanaka dataset. As discussed in Section 4.1, most of Tanaka's instances have very low network densities, where Tan performs better. Hence, this dataset is not a suitable choice for a fair comparison between the two algorithms. Nevertheless, for the instances with medium to high density, our DP outperforms Tan both on CPU time and in number of solved instances. In the setting with $n = 100$ and $\rho = 0.1$, for instance, our DP is on average at least 557 times faster than Tan. Again, for all the instances that our DP cannot solve, the memory limit is reached, while both memory and time limits were attained for the instances that were not solved by Tanaka and Sato's algorithm.

ρ	n															
	40					50					100					
	OS1	DP		Tan		OS1	DP		Tan		OS1	DP		Tan	
		#	CPU	#	CPU		#	CPU	#	CPU		#	CPU	#	CPU
0.2	0.63	**125**	**0.00**	125	1.22	0.68	**125**	**0.00**	125	3.06	0.82	**125**	**0.00**	125	4.36
0.1	0.28	**125**	**0.06**	125	8.69	0.35	**125**	**0.23**	125	27.49	0.57	**125**	**0.78**	124	434.70
0.05	0.10	**125**	**1.69**	125	4.60	0.11	125	42.07	**125**	**28.56**	0.23	64	620.33	**123**	**1,173.00**
0.02	0.03	125	2.19	**125**	**1.41**	0.03	121	128.43	**125**	**5.38**	0.04	29	199.69	**125**	**218.62**
0.01	0.01	125	1.22	**125**	**0.38**	0.01	124	58.92	**125**	**1.54**	0.01	41	386.22	**125**	**44.68**
0.005	0.01	125	0.56	**125**	**0.32**	0.01	125	27.56	**125**	**0.63**	0.01	52	140.44	**125**	**35.59**
0	0.00	125	0.22	**125**	**0.02**	0.00	125	3.38	**125**	**0.05**	0.00	74	327.01	**125**	**0.57**

Table 6: CPU times – the Tanaka dataset

5 Conclusions

In this article, we have extended Emmons (1969) and Kanet (2007) results for $1||\sum w_jT_j$ to allow for precedence constraints. We have also developed an efficient memory management technique for the DP algorithm of Schrage and Baker (1978). Combining these contributions, we have proposed an exact method for solving $1|\text{prec}|\sum w_jT_j$. We have empirically shown that our method outperforms the state-of-the-art algorithm of Tanaka and Sato (2013) for instances with medium to high density. Thus, our DP complements Tanaka and Sato's algorithm, which performs better for instances without precedence constraints or with very low network density. Finally, we have also addressed Kanet's concluding suggestion in his 2007 article to conduct a full-scale computational study to test the utility of his theorems.

Appendix

Kanet's Conditions K1-K7 follow. For a dominant S and a pair $(i,j) \in N \times N$:

K1. $p_i \leq p_j$ and $w_i \geq w_j$ and $(d_i \leq \max\{d_j, (w_i - w_j)(P(B_i^S \cup B_j^S) + p_i + p_j)/w_i + w_j d_j/w_i\}$ or $d_i \leq (w_i - w_j)(P(B_i^S \cup B_j^S) + p_i + p_j)/w_i + w_j(P(B_j^S) + p_j)/w_i)$.

K2. $p_i \leq p_j$ and $w_i < w_j$ and $d_j \geq (w_j - w_i)P(\bar{A}_i^S)/w_j + w_i d_i/w_j$ and $d_j \geq (w_j - w_i)P(\bar{A}_i^S)/w_j + w_i(P(\bar{A}_i^S \cap \bar{A}_j^S) - p_j)/w_j$.

K3. $p_i \leq p_j$ and $w_i < w_j$ and $d_i \leq (w_i - w_j)P(\bar{A}_i^S)/w_i + w_j(P(B_j^S) + p_j)/w_i$ and $p_i \leq (w_i - w_j)(P(\bar{A}_i^S) - P(B_j^S))/w_i + w_j p_j/w_i$.

K4. $w_i \geq w_j$ and $d_j \geq \min\{d_i, (w_j - w_i)(P(B_i^S \cup B_j^S) + p_i + p_j)/w_j + w_i d_i/w_j$ and $d_j \geq P(\bar{A}_i^S) - w_i p_j/w_j$.

K5. $w_i \geq w_j$ and $d_i \leq (w_i - w_j)(P(B_i^S \cup B_j^S) + p_i + p_j)/w_i + w_j(P(B_j^S) + p_j)/w_i$ and $p_j \geq w_j(P(\bar{A}_i^S) - P(B_j^S) - p_j)/w_i$.

K6. $w_i < w_j$ and $d_j \geq (w_j - w_i)P(\bar{A}_i^S)/w_j + w_i d_i/w_j$ and $d_j \geq P(\bar{A}_i^S) - w_i p_j/w_j$.

K7. $d_j \geq P(\bar{A}_i^S)$.

References

Abdul-Razaq, T., Potts, C., and Van Wassenhove, L. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26(2-3):235–253.

Correa, J. and Schulz, A. (2005). Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30(4):1005–1021.

Davari, M., Demeulemeester, E., Leus, R., and Talla Nobibon, F. (2016). Exact algorithms for single-machine scheduling with time windows and precedence constraints. *Journal of Scheduling*, 19(3):309–334.

Demeulemeester, E., Vanhoucke, M., and Herroelen, W. (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1):17–38.

Emmons, H. (1969). One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17(4):701–715.

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Hoogeveen, J. and van de Velde, S. (1995). Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems. *Mathematical Programming*, 70(1-3):173–190.

Ibaraki, T. and Nakamura, Y. (1994). A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76(1):72–82.

Kanet, J. (2007). New precedence theorems for one-machine weighted tardiness. *Mathematics of Operations Research*, 32(3):579–588.

Lawler, E. (1977). A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342.

Lawler, E. (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90.

Margot, F., Queyranne, M., and Wang, Y. (2003). Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992.

Potts, C. (1985). A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*, 31(10):1300–1311.

Potts, C. and Van Wassenhove, L. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377.

Rachamadugu, R. (1987). Technical note–A note on the weighted tardiness problem. *Operations Research*, 35(3):450–452.

Rinnooy Kan, A., Lageweg, B., and Lenstra, J. (1975). Minimizing total costs in one-machine scheduling. *Operations Research*, 23(5):908–927.

Schrage, L. and Baker, K. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26(3):444–449.

Schulz, A. and Uhan, N. (2011). Near-optimal solutions and large integrality gaps for almost all instances of single-machine precedence-constrained scheduling. *Mathematics of Operations Research*, 36(1):14–23.

Sidney, J. (1975). Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23(2):283–298.

Tanaka, S., Fujikuma, S., and Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12(6):575–593.

Tanaka, S. and Sato, S. (2013). An exact algorithm for the precedence-constrained single-machine scheduling problem. *European Journal of Operational Research*, 229(2):345–352.

van de Velde, S. (1995). Dual decomposition of a single-machine scheduling problem. *Mathematical Programming*, 69(1-3):413–428.