Discrete Applied Mathematics 158 (2010) 1767–1776

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

'Strong'–'weak' precedence in scheduling: Extensions to series–parallel orders

Moshe Dror [a], George Steiner [b,*]

[a] MIS, Eller College of Management, The University of Arizona, Tucson, AZ 85721, United States

[b] Operations Management, DeGroote School of Business, McMaster University, Hamilton, ON, Canada L8S 4M4

ARTICLE INFO

Article history:
Received 25 June 2009
Received in revised form 25 May 2010
Accepted 28 June 2010
Available online 17 August 2010

Keywords:
Scheduling
Posets
Strong and weak precedence

ABSTRACT

We examine computational complexity implications for scheduling problems with job precedence relations with respect to strong precedence versus weak precedence. We propose a consistent definition of strong precedence for chains, trees, and series–parallel orders. Using modular decomposition for partially ordered sets (posets), we restate and extend past complexity results for chains and trees as summarized in Dror (1997) [5]. Moreover, for series–parallel posets we establish new computational complexity results for strong precedence constraints for single- and multi-machine problems.

1. Introduction

Job scheduling with precedence constraints has been at the forefront of scheduling research since its inception. Application of scheduling with precedence relations can be traced to PERT scheduling, early operating systems task scheduling with respect to its stability, and much more. In terms of solvability (e.g., finding an optimal schedule in polynomial time), adding precedence relations to otherwise independent jobs in general makes a problem harder. Given a set of jobs J together with a partial order $<_P$ defined on $J \times J$, the common interpretation of precedence constraints (prior to Dror et al., [6,7]) was the following: for any $i, j \in J$, $i <_P j$ means that job i has to be completed before the processing of job j can start, or in other words, ordering the jobs by their start times must yield a total order for J that is consistent with $<_P$ (a *linear extension*). This interpretation of the precedence constraints represented by partial order $<_P$ allows any amount of time delay between the completion of job i and the start of job j. A different interpretation of job precedence has been proposed in [6,7], recognizing that in some cases, further constraining the start time of job j with respect to the completion time of job i is justified in practice and implies a different problem and a more constrained solution. Simply, asking for a job j to *immediately* follow the completion of job i links the two jobs in a strong precedence. We will use the term *weak precedence relation* for the traditional definition of precedence. For chained jobs, strong precedence implies that the jobs must be processed consecutively without the option of inserting a job not in the chain between the chained jobs. This is exactly the way the algorithm described by Assad et al. [1] schedules chained jobs in a manufacturing of mattresses. As a further motivation for our pursuit of the strong/weak precedence distinction, we note that some scheduling problems are solvable in polynomial time with strong precedence even though the weak precedence version is NP-hard and vice versa [6].

The strong/weak precedence distinction for chained jobs was introduced in [6,7]. In [8] this distinction was introduced for in-tree and out-tree precedence graphs. However, the strong chain precedence as defined in [6,7] was not consistent with the strong precedence tree relation as defined in [8]. This definitional inconsistency is rectified in the present paper. In the past,

* Corresponding author.
E-mail addresses: mdror@eller.arizona.edu (M. Dror), steiner@mcmaster.ca (G. Steiner).

0166-218X/$ – see front matter
doi:10.1016/j.dam.2010.06.015

series–parallel precedence relations have not been examined from the prospective of strong versus weak job precedence. In this paper we define strong precedence for series–parallel orders consistent with strong precedence for chains and trees using modular decomposition of posets and develop new results for scheduling problems. For completeness, we also provide an abbreviated survey of the state-of-the-art for some scheduling problems by reviewing their complexity with respect to strong and weak precedence relations. The outline of the paper is as follows: In Section 2 we state the extended strong precedence constraints consistent for chains, trees, and series–parallel precedence graphs. Section 3 examines single-machine problems. Section 4 studies multi-machine problems. In Section 5 we summarize our findings.

2. Extended strong precedence constraints

Definition 2.1. Let $P = (J, <_P)$ be a partially ordered set (a poset). Let $\prec$ denote the immediate predecessor and $\|$ the incomparability relation in P. That is, $\prec$ represents the transitive reduction of $<_P$ and $\|$ holds for the incomparable pairs. For $j \in J$ denote by $Im\,Suc(j) \triangleq \{k | j \prec k\}$ and by $Im\,Pred(j) \triangleq \{k | k \prec j\}$. We say that $A \subseteq J$ is an in-module *in* P if $\forall i, j \in A$ we have $Im\,Suc(i) \cap (J \backslash A) = Im\,Suc(j) \cap (J \backslash A)$. A subset $B \subseteq J$ is an out-module in P if $\forall i, j \in B$ we have $Im\,Pred(i) \cap (J \backslash B) = Im\,Pred(j) \cap (J \backslash B)$. We say that $A \subseteq J$ is a bi-module, if it is both an in-module and an out-module.

Note that the empty set, any single point $i \in J$, or the whole set $A = J$ are *trivial* bi-modules. If we define $\min P \triangleq \{k | Im\,Pred(k) = \emptyset\}$, the set of minimal elements in P, and $\max P \triangleq \{k | Im\,Suc(k) = \emptyset\}$, the set of maximal elements in P, then it is easy to see that $\max P$ is always a *trivial* in-module and $\min P$ is always a *trivial* out-module. Furthermore, we note that if $A \subseteq J$ is an in-module in a general poset P, then $\forall i \in A$ we have either $Im\,Suc(i) \cap (J \backslash A) = \emptyset$ or $Im\,Suc(i) \cap (J \backslash A)$ is an antichain, i.e., a set of incomparable elements. Similarly, if $B \subseteq J$ is an out-module in a general poset P, then $\forall i \in B$ we have either $Im\,Pred(i) \cap (J \backslash B) = \emptyset$ or $Im\,Pred(i) \cap (J \backslash B)$ is an antichain. Therefore, any maximal in-module that is an antichain is simply a maximal set of elements with a common set of (immediate) successors. Similarly, any maximal out-module that is an antichain is simply a maximal set of elements with a common (immediate) predecessor set.

If $J = \mathbf{C}_1 \cup \mathbf{C}_2 \cdots \cup \mathbf{C}_K$ is the union of K parallel chains, then each $\mathbf{C}_i$ is *both* an in-module and an out-module (a bi-module) in P: if $A = \mathbf{C}_i$, then $\forall i, j \in A$ we have $Im\,Suc(i) \cap (J \backslash A) = Im\,Suc(j) \cap (J \backslash A) = \emptyset$; and if $B = \mathbf{C}_i$, then $\forall i, j \in B$ we have $Im\,Pred(i) \cap (J \backslash B) = Im\,Pred(j) \cap (J \backslash B) = \emptyset$. Furthermore, no proper nontrivial subset of a chain (i.e., not a single point) has both properties. In that case we say that each $\mathbf{C}_i$ is a *minimal nontrivial bi-module* in P. We will also refer to any $\mathbf{C}_i$ whose elements are incomparable to any other element of P as a parallel-chain bi-module. It is also easy to see that if the poset $P = (J, \prec_P)$ is an in-tree, then $Im\,Pred(i)$ is an in-module $\forall i \in J$; and if P is an out-tree, then $Im\,Suc(i)$ is an out-module $\forall i \in J$.

In general, all previous definitions of strong precedence constraints can be interpreted as follows: There are certain sub-posets $P_k \subset P$ specified for $k = 1, \ldots, K$ such that the jobs in each P_k must be scheduled *consecutively* and *as early as possible* after the first job in P_k. Intuitively, they imply that the jobs in each P_k can be replaced by appropriately defined composite (or compound) jobs for scheduling purposes. We introduce a more precise definition, which will allow us to unify and extend the concept of strong precedence constraints to general posets P.

Definition 2.2. Let S_i be the start time of job $i \in P$ in a schedule. We say that a subset $P' \subset P$ is scheduled in immediate and contiguous fashion if for every $i, j \in P'$ such that $j \in Im\,Suc(i)$, there is an $l \in Im\,Suc(i)$ such that $S_l = S_i + p_i$; and for every $i, j \in P'$ such that $i \| j$ and $S_i < S_j$ there is no $l \in P \backslash P'$ with $S_i < S_l < S_j$.

Using the above definitions, previously defined concepts of *strong precedence* relations in scheduling imply the following:

- for chains: the minimal nontrivial bi-modules (one chain from a set of parallel chains) must be scheduled in immediate and contiguous fashion (in the case of identical parallel processors, we can assume w.l.o.g. that the jobs from a given chain are processed on the same processor);
- for in-trees: the maximal in-modules, $Im\,Pred(i)$ for $i \in J$, must be scheduled in immediate and contiguous fashion
- for out-trees: the maximal out-modules, $Im\,Suc(i)$ for $i \in J$, must be scheduled in immediate and contiguous fashion.

There is a well-established theory (see e.g., [23]) of 'modular decomposition' for posets in general and series–parallel posets in particular. This is based on the following concept of modules:

Definition 2.3. Let $P = (J, <_P)$ be a partially ordered set. $A \subseteq J$ is a module *in* P if $\forall i, j \in A$ we have $Suc(i) \cap (J \backslash A) = Suc(j) \cap (J \backslash A)$, where $Suc(j) \triangleq \{k | j <_P k\}$, *and* $Pred(i) \cap (J \backslash A) = Pred(j) \cap (J \backslash A)$, where $Pred(j) \triangleq \{k | k <_P j\}$.

This means that a module is a subset that 'looks to be the same' from the outside world, i.e., every point in the module has exactly the same predecessors, successors and incomparable elements outside of it. We note that earlier we have defined in-modules and out-modules on the transitive reduction (Hasse diagram) of the poset. Modules are usually defined on the transitive closure of the poset, but it is also possible to define them, in a more complicated way, on the Hasse diagram. Furthermore, it is easy to see that every bi-module is also a module, but the reverse is not true: e.g. any proper subchain of a chain is a module but not a bi-module.

There are polynomial time algorithms for certain scheduling objectives on large classes of weak precedence constraints, including series–parallel posets (see [23] for details). They are all based on the following job-module property:

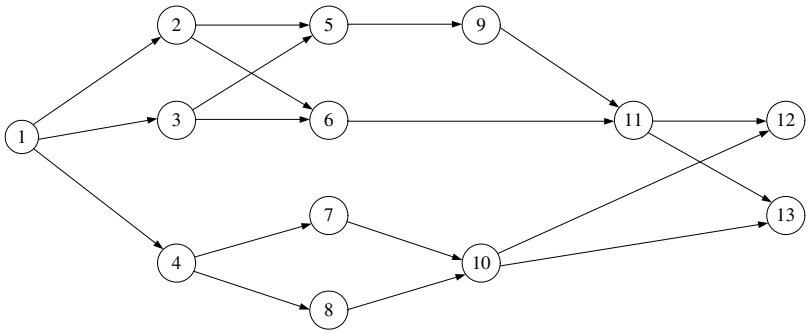

Fig. 1. A series–parallel poset.

Definition 2.4. Let $P = (J, <_P)$ be a partially ordered set representing the weak precedence constraints for a scheduling problem and let $A \subseteq J$ be a module *in* P. Consider an optimal schedule (sequence) σ_A of the jobs for the sub-problem induced on the set A. We say that the scheduling problem satisfies the job-module property if there is an optimal schedule (sequence) σ for the whole problem such that σ orders the jobs in A (i.e., their starting times) in the same sequence (relative to each other) as σ_A.

Definition 2.5. Series–parallel poset:

1. A poset with a single element is series–parallel;
2. If $P_1 = (J_1, <_{P_1})$ and $P_2 = (J_2, <_{P_2})$ are disjoint series–parallel posets, then
   (a) their series composition $P = P_1 * P_2$ is the poset on $J_1 \cup J_2$ with $i <_P j$ iff
   $\big((i <_{P_k} j \text{ for } i, j \in P_k \text{ and } k = 1, 2) \text{ or } (i \in P_1 \text{ and } j \in P_2)\big)$;
   (b) and their parallel composition $P = P_1 + P_2$ is the poset on $J_1 \cup J_2$ with $i <_P j$ iff
   $\big((i <_{P_k} j \text{ for } i, j \in P_k \text{ and } k = 1, 2)\big)$.

A poset is series–parallel if it can be built up from single elements by a sequence of series and parallel compositions. (Alternative characterizations of series–parallel posets can be found in [13,15,23,28].)

The Hasse diagram for a series–parallel poset is shown in Fig. 1.

In other words, series–parallel posets constitute the smallest class of partial orders that contains the one-element poset and is closed under series and parallel composition. As a consequence of the recursive definition, every series–parallel poset can be represented in a natural way by a binary tree, the *binary decomposition tree* (see Fig. 2. The leaves of this tree represent one-element posets, and each internal node corresponds to a partial order obtained by the series (S) or parallel (P) composition of the posets corresponding to the left and right son of the node.

It is clear that a chain on n elements can be built up by a sequence of $n - 1$ series compositions. If $P = \mathbf{C}_1 \cup \cdots \cup \mathbf{C}_k$ is a collection of disjoint chains, then it can also be considered as $P = \mathbf{C}_1 + \cdots + \mathbf{C}_k$, i.e., the parallel composition of these chains. Furthermore, it is easy to see that if P is an in-tree or an out-tree, then it can be built by a sequence of series and parallel compositions, such that one of the two sets in every series composition always has only a single element. This means that parallel chains, in-trees or out-trees are all special cases of series–parallel posets.

Note that P_1 and P_2 above are both modules in the above definition. They will be referred to as series and parallel modules in the respective cases. The scheduling problem $1|prec|\sum w_iC_i$ [26], jump number [27], and a number of other problems, including due date assignment problems and problems with learning effects, (see [24,16,14]) all satisfy the job-module property. Their solution with (weak) series–parallel precedence constraints is based on iteratively sequencing optimally the sub-problems on series or parallel modules in the binary decomposition tree until the whole problem becomes completely and optimally sequenced (see [23,24]).

Let P be the series (parallel) composition of the posets P_1 and P_2 and let σ_i be an optimal sequence for the sub-problem defined on P_i, $i = 1, 2$. If $P = P_1 * P_2$, then the job-module property guarantees that $\sigma = (\sigma_1, \sigma_2)$ is optimal for the problem defined on P. On the other hand, if $P = P_1 + P_2$, then the parallel chains corresponding to the sequences σ_1 and σ_2 have to be optimally merged into a sequence σ by some algorithm. For various implementations of this argument we refer the reader to [20,24].

Consider the poset Z defined on four elements by the following precedence relations: $a \prec c$, $b \prec c$ and $b \prec d$. The following is a useful alternative characterization of series–parallel posets (see e.g. [23]).

Theorem 2.6. *A poset P is series–parallel if and only if it does not contain any sub-poset isomorphic to the poset Z.*

Using our earlier interpretation for strong precedence constraints, we 'unify' the concept of strong precedence for both chains and trees and in addition, extend it to series–parallel and general posets.

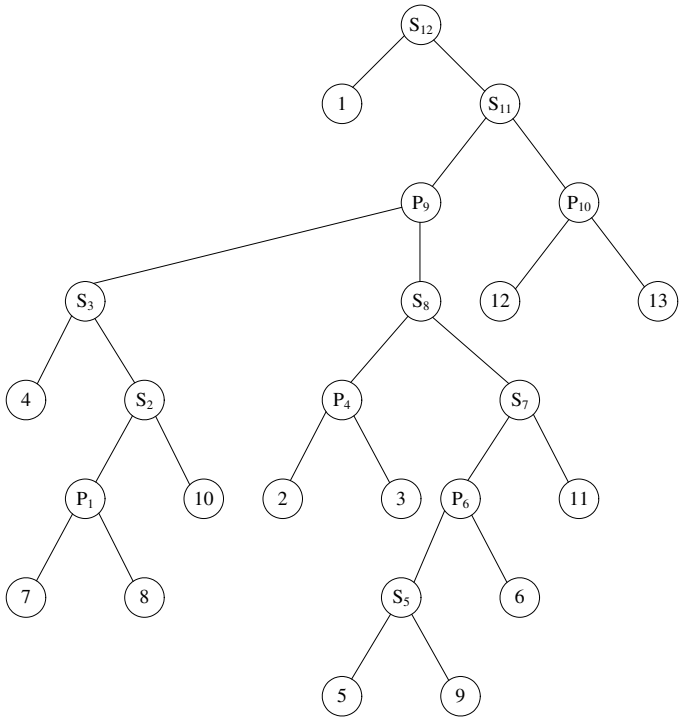

Fig. 2. The binary decomposition tree for the series–parallel poset of Fig. 1.

Definition 2.7. Consider a scheduling problem with poset $P = (J, \prec_P)$ with $\prec_P$ representing the transitive reduction of relations in P. We say that P represents strong precedence constraints, denoted by *strong prec*, for the problem if every schedule has to satisfy the following:

1. If $i \prec_P j$ then for the start times, we have $S_j \geq S_i + p_i$, i.e., every relation in $\prec_P$ is satisfied as a weak precedence constraint;
2. If P contains a parallel-chain bi-module (chain **C**), then the jobs in **C** must be scheduled in immediate and contiguous fashion.
3. Consider any nontrivial maximal in-module or out-module $A \neq \emptyset$ that is not a parallel-chain bi-module and which has a non-empty successor or predecessor set, respectively. Then A must be scheduled in immediate and contiguous fashion.

The definition implies that every feasible schedule satisfying the strong precedence constraints P would also have to be feasible for a version of the problem in which the poset P represents only weak precedence constraints. A feasible schedule for strong precedence constraints, however, also must satisfy the additional immediacy and contiguity constraints. We note that these constraints would be meaningless for the trivial modules and these were excluded from the definition in point 3. Furthermore, we require the immediacy and contiguity constraint only for nontrivial in-modules with a non-empty successor set to exclude the constraints for $A = \max P$. Similarly, requiring a non-empty predecessor set for out-modules excludes $A = \min P$. All modules which need to satisfy the contiguity constraint can be replaced by a *composite* (or *compound*) job composed of the jobs in the in-module or the out-module. We will denote this by putting these jobs between parentheses (in any sequence). If jobs in a composite job have to be processed in a prescribed sequence, then we will call such a composite job a *string* and will denote it by putting a $\rightarrow$ above this prescribed sequence of the jobs. We note that a composite job may contain other composite jobs among its parts. As we have discussed earlier, all the in-modules or out-modules considered in point 3 are antichains. Therefore, it is always possible to schedule the jobs in any such antichain in immediate and contiguous fashion, i.e., there always exists a feasible schedule satisfying the strong precedence constraints represented by any poset.

We also observe that requiring the contiguity of schedules on the immediate predecessor set (i.e., an out-module) or successor set (i.e., an in-module) of a job is consistent with those applications where the strong precedence constraints express the requirement that the processing of these sets cannot be delayed (be interrupted) by other jobs not in the module. For example, this could be required in a process where the components of a subassembly cannot wait, i.e., their processing cannot be interrupted by other unrelated jobs, as they may deteriorate over time.

3. Single-machine problems

It is natural to examine how the complexity of a scheduling problem is affected if its weak precedence constraints are replaced by strong precedence. In the following discussion, we demonstrate that the relationship between the complexities

Table 1
The data for an instance of 1|*strong prec*| $\sum w_iC_i$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13
w_i	1	1	3	2	7	2	10	3	1	3	4	9	1
p_i	1	4	1	3	2	8	1	5	10	7	8	2	6

of the two problems is nontrivial. We will use the well-accepted $\alpha|\beta|\gamma$ notation for scheduling problems [21]. We put the qualifier *strong* in front of the precedence constraints in the β field if they represent strong precedence, otherwise they represent weak precedence only.

3.1. Hard problems

Dror et al. [8] proved the following theorem for regular cost functions f_j.

Theorem 3.1. *If the* 1|*chain*| $\sum f_j(C_j)$ *scheduling problem is NP-hard, then the* 1|*strong out-tree*| $\sum f_j(C_j)$ *problem is also NP-hard. If the* 1|*chain*| $\max f_j(C_j)$ *scheduling problem is NP-hard, then the* 1|*strong out-tree*| $\max f_j(C_j)$ *problem is also NP-hard.*

The proof is based on reducing the 1|*chain*| $\sum f_j(C_j)$ (or 1|*chain*| $\max f_j(C_j)$) problem to a 1|*strong out-tree*| $\sum f_j(C_j)$ (or 1|*strong out-tree*| $\max f_j(C_j)$) problem, where there is a one-to-one correspondence between the optimal schedules. Based on [22], it immediately yields the following corollary.

Corollary 3.2. *The* 1|*strong out-tree*, $p_j = 1$| $\sum U_j$ *problem is NP-hard.*

It is also shown in [8] that there are instances of 1|*strong out-tree*, $p_j = 1$| $\sum U_j$ and 1| out-tree , $p_j = 1$| $\sum U_j$ defined on the same job set and same out-tree poset representing the precedence constraints for which the ratio of the optimal objective function values of the two problems can be arbitrarily large. Thus adding the contiguity requirement of the strong precedence constraints can make the cost of an optimal schedule increase by an arbitrarily large amount.

3.2. Problems with strong series–parallel precedence constraints

Next we discuss how we can extend solution algorithms from weak to strong precedence constraints for scheduling problems satisfying the job-module property. For illustration, we use the instance of minimizing the total weighted completion time with strong series–parallel precedence constraints, 1|*strong series–parallel*| $\sum w_iC_i$, whose precedence constraints are represented by the poset of Fig. 1 and whose data is given in Table 1. (The data is the same as in the example solved by Lawler [20], allowing us to compare the optimum cost with strong and weak precedence constraints.)

Consider a poset $P = (J, <_P)$ and consider the maximal in-modules in P that are not parallel-chain bi-modules and have non-empty successor sets. As noted earlier, each of these sets must be a maximal antichain whose elements have a common set of immediate successors. No two of these maximal in-modules can overlap by definition. Thus, these maximal in-modules together with the set $\max P$ define a unique partition of J (which can be found in polynomial time) and is denoted by $\mathcal{IN}$. This partition contains the following sets for the example poset of Fig. 1: $\mathcal{IN} = \{\{1\}, \{2, 3\}, \{4\}, \{5\}, \{6, 9\}, \{7, 8\}, \{10, 11\}, \{12, 13\}\}$. Next we show that the contiguity constraints on in-modules and out-modules induce certain additional weak precedence relations between some of these modules that extend the original poset P.

Extensions of type 1 *(between in-modules)*

Consider two in-modules, $A_1, A_2 \in \mathcal{IN}\backslash\{\max P\}$ such that there is a $k \in A_1$ and $l \in A_2$ such that $k \prec l$ in P. The fact that A_1 and A_2 must be scheduled contiguously implies that *every* job from A_1 must precede *every* job in A_2 in every feasible schedule. This implies that the weak precedence constraints $u \prec v$ can be added to P for every $u \in A_1$ and $v \in A_2$ to get an extended poset P'. This however means that the jobs in A_2 have not only the same (immediate) successors, but also the same (immediate) predecessors in the resulting extended poset P'. In other words A_2 is a bi-module (and a job module) in the poset P'. We also observe that adding the relations $u \prec v$ for every $u \in A_1$ and $v \in A_2$ in fact creates a series composition between A_1 and A_2. This is important, because it implies that if the poset P is series–parallel, then so will be the extended poset P'. For example, consider $A_1 = \{5\}$ and $A_2 = \{6, 9\}$ in the poset of Fig. 1: Since $5 \prec 9$ and $\{6, 9\}$ must be scheduled contiguously, this implies that 5 must also precede 6 in any feasible schedule, so we can add the (weak precedence) relation $5 \prec 6$ to P. This results in the same predecessor set for 6 and 9, which makes $A_2 = \{6, 9\}$ a job module in the extended poset.

Extensions of type 2 *(between out-modules)*

Consider again a poset $P = (J, <_P)$ and consider the maximal out-modules that are not parallel-chain job modules and have non-empty predecessor sets. As noted earlier, each of these sets must be a maximal antichain whose elements have a common set of immediate predecessors. Again, no two of these maximal out-modules can overlap by definition. Thus, these maximal out-modules together with the set $\min P$ also define a unique partition of J (that can be found in polynomial time) and is denoted by $\mathcal{OUT}$. For the poset example in Fig. 1 this partition contains the following sets: $\mathcal{OUT} = \{\{1\}, \{2, 3, 4\}, \{5, 6\}, \{9\}, \{7, 8\}, \{10\}, \{11\}, \{12, 13\}\}$.

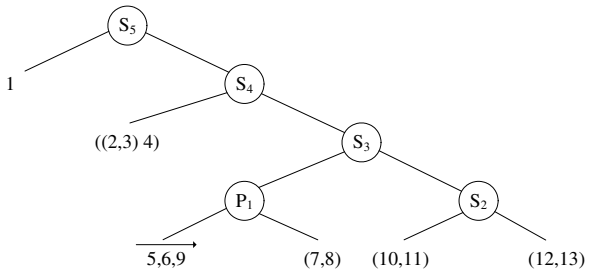

Fig. 3. The binary decomposition tree for the extended poset P'.

Consider now two out-modules, B_1 , $B_2 \in \mathcal{OUT}\backslash\{\min P\}$ such that there is a $k \in B_1$ and $l \in B_2$ and also $k \prec l$ in P. The fact that B_1 and B_2 must be scheduled contiguously implies that *every* job from B_2 must follow *every* job from B_1 in every feasible schedule. This implies that the weak precedence constraints $u \prec v$ can be added to P for every $u \in B_1$ and $v \in B_2$ to get an extended poset P'. This however means that the jobs in B_1 have not only the same (immediate) predecessors, but also the same (immediate) successors in the resulting extended poset P'. In other words, B_1 is a bi-module (and a job module) in the poset P'. We also observe that adding the relations $u \prec v$ for every $u \in B_1$ and $v \in B_2$ in fact creates a series composition between B_1 and B_2. This is important again because it implies that if the poset P is series–parallel, then so will be the extended poset P'. For example, consider $B_1 = \{2, 3, 4\}$ and $B_2 = \{5, 6\}$ in the poset of Fig. 1: Since $2 \prec 5$, $2 \prec 6$ and $\{2, 3, 4\}$ must be scheduled contiguously, this implies that 4 must also precede 5 and 6 in any feasible schedule, so we can add the (weak precedence) relations $4 \prec 5$ and $4 \prec 6$ to P. Similarly, considering $\{7, 8\}$ for B_2, we can add the relations $2 \prec 7$, $2 \prec 8$, $3 \prec 7$ and $3 \prec 8$ to P. This results in having every element of B_1 precede every element of $\{5, 6\} \cup \{7, 8\}$, which makes all successors for the elements of B_1 common, i.e., B_1 becomes a job module in the extended poset. Similarly, looking at $B_1 = \{5, 6\}$ and $B_2 = \{9\}$, the original relation $5 \prec 9$ and the contiguity requirement for B_1 imply that we can add the (weak precedence) relation $6 \prec 9$, which makes $B_1 = \{5, 6\}$ a job module in the extended poset.

Algorithm Strong SP
INPUT: A scheduling problem that satisfies the job-module property and whose strong precedence constraints are represented by the series–parallel poset $P = (J, \prec_P)$.

Step 1. Identify the partitions of J into in-modules and out-modules, respectively, denoted by $\mathcal{IN}$ and $\mathcal{OUT}$.

Step 2. Extend the poset P by adding the weak precedence constraints arising from the application of all extensions of types 1 and 2. Let the resulting poset be P'.

Step 3. Contract all nontrivial in-modules in $\mathcal{IN}\backslash\{\max P\}$ and all nontrivial out-modules in $\mathcal{OUT}\backslash\{\min P\}$ into composite jobs. If P has any parallel-chain bi-modules, then contract these into composite strings.

Step 4. Apply the original series–parallel algorithm for weak precedence constraints [20,24] to the scheduling problem defined by the composite jobs and weak precedence constraints P'.

OUTPUT: An optimal schedule.

Note that a scheduling problem on independent jobs (with empty precedence constraints) typically becomes polynomially solvable when the objective function satisfies the adjacent pairwise interchange property, which allows us to define a transitive and complete binary preference relation on the jobs. This relation determines an optimal sequence for the unconstrained jobs. For more details, we refer the reader to [24]. In the case of the $1 \parallel \sum w_iC_i$ problem, this preference relation is just the well-known weighted-shortest-processing-time (WSPT) order. This order can also be extended to subsets or sequences of jobs: For any $A \subseteq J$, we define $w(A) = \sum_{i\in A} w_i$ and $p(A) = \sum_{i\in A} p_i$. The preference order on these subsets is determined by the non-increasing WSPT ratios $w(A)/p(A)$. The polynomial time solution algorithm for the $1|series\text{–}parallel|\sum w_iC_i$ problem [20,24] repeatedly uses this preference order on sets and strings of jobs.

When applying Algorithm *Strong SP* to the example whose strong precedence constraints are represented by the poset depicted in Fig. 1, we add the extensions discussed above. The binary decomposition tree of the resulting extended weak precedence constraints P' is shown in Fig. 3.

Explanation of the schedule generated by Algorithm Strong SP:
Applying the original series–parallel algorithm in Step 4, we start sequencing the job modules with the leaves of the tree in Fig. 3. The elements of the composite job (7, 8) are scheduled in WSPT order resulting in the string $\overrightarrow{7, 8}$. Comparing the parallel strings $\overrightarrow{5, 6, 9}$ and $\overrightarrow{7, 8}$, $w(\overrightarrow{5, 6, 9})/p(\overrightarrow{5, 6, 9}) = 12/20$ and $w(\overrightarrow{7, 8})/p(\overrightarrow{7, 8}) = 13/6$, which means that $\overrightarrow{7, 8}$ should be scheduled before $\overrightarrow{5, 6, 9}$ in the job module corresponding to the node P_1 in the tree. The WSPT order on the elements of the composite job (10, 11) is 11, 10, so it gets replaced by the string $\overrightarrow{11, 10}$. The WSPT order on the elements of (12, 13) is 12, 13, so it gets replaced by the string $\overrightarrow{12, 13}$. The WSPT order on (2, 3) is 3, 2, so it gets replaced by the string $\overrightarrow{3, 2}$. This string comes before 4 in the WSPT order, so the composite job ((2, 3), 4) is replaced by the string

$\overrightarrow{3, 2, 4}$. Finally putting all strings in the order that satisfies the series compositions in the tree, we obtain the optimal schedule (1, 3, 2, 4, 7, 8, 5, 6, 9, 11, 10, 12, 13) with total weighted completion time of 1228. In comparison, when P is interpreted to represent only weak precedence constraints, the optimal sequence is (1, 4, 7, 3, 2, 5, 8, 10, 6, 9, 11, 12, 13) with $\sum w_i C_i = 1158$. This demonstrates that the strong precedence constraints can result in a substantial increase in the minimum cost.

Theorem 3.3. *Consider a scheduling problem* $1|$*strong series–parallel*$|f$ *that satisfies the job-module property. Then Algorithm Strong SP finds an optimal schedule for the problem in* $O(n^2)$ *time.*

Proof. We note that the cost function for the scheduling problem can be either of the type $f = \sum f_j(C_j)$ or $f = f_{\max} = \max_j f_j(C_j)$. Consider now a scheduling problem with strong series–parallel precedence constraints P and assume that the problem also satisfies the job-module property. Algorithm *Strong SP* contracts the nontrivial in-modules and out-modules into composite jobs and extends the original poset P by adding all the implied weak precedence relations between these modules to P. The partitions into in- and out-modules and the construction of the extended poset can clearly be obtained in $O(n^2)$ time. As discussed above, every feasible schedule must satisfy these extended weak precedence relations. The fact that composite jobs cannot be broken up automatically ensures that all contiguity constraints prescribed by the strong precedence constraints will be satisfied. The original series–parallel algorithm always finds an optimal schedule for the problem with the extended weak precedence constraints in $O(n \log n)$ time. □

We mention the following immediate consequence of the preceding theorem.

Corollary 3.4. *The total weighted completion time problem with strong series–parallel precedence constraints,* $1|$*strong series–parallel*$|\sum w_j C_j$*, is solvable in polynomial time.*

Remark. We note that the complexity of Algorithm *Strong SP* could also be reduced to $O(n \log n)$ by a more careful implementation of the partitioning and extension operations using the binary decomposition tree. We decided to omit this here for the sake of brevity, since explaining the operations on the binary decomposition tree would require extensive technical details.

4. Multi-machine models

4.1. Mean flow time on parallel machines

4.1.1. The $Pm|$*strong chains*$|\sum C_j$ problem

We start our discussion by extending to any fixed number (m) of identical parallel machines a lemma that was proved in [6] for the two-machine ($m = 2$) case.

Lemma 4.1. *Consider a* $Pm|$*strong chains*$|\sum C_j$ *problem on* K *chains,* $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_K$*, where* $|\mathbf{C}_i| = n_i,\ i = 1, 2, \ldots, K$*, and let* $p_{k,i}$ *denote the processing time of the* i*th job in* $\mathbf{C}_k,\ i = 1, 2, \ldots, n_k$*. This problem reduces to a* $Pm \parallel \sum w_k C_k$ *problem on* K *compound jobs, with no precedence constraints between them, where compound job* k *has processing time* $p_k = \sum_{j=1}^{n_k} p_{k,j}$ *and weight* $w_k = n_k$ *for* $k = 1, 2, \ldots, K$*.*

Proof. Let $C_{k,n_k}(k = 1, 2, \ldots, K)$ be the completion time of the last job from chain $\mathbf{C}_k$ in a schedule. Since $\mathbf{C}_k$ is a bi-module, its jobs must be processed contiguously. Thus if $p_{k,i}$ denotes the processing time of the ith job in $\mathbf{C}_k$ for $i = 1, 2, \ldots, n_k$, then its completion time must be $C_{k,i} = C_{k,n_k} - (p_{k,i+1} + \cdots + p_{k,n_k})$. Thus, the total completion time of chain $\mathbf{C}_k$, denoted by TFT_k, is

$$TFT_k = n_k C_{k,n_k} - [(n_k - 1)p_{k,n_k} + (n_k - 2)p_{k,n_k-1} + \cdots + p_{k,2}].$$

If we let $mft_k = [(n_k - 1)p_{k,n_k} + (n_k - 2)p_{k,n_k-1} + \cdots + p_{k,2}]$, then this term is schedule-independent, and only the $n_k C_{k,n_k}$ part of TFT_k varies with the schedule. Therefore, the total completion time of a schedule can be written as

$$\sum_{k=1}^{K}\sum_{i=1}^{n_k} C_{k,i} = \sum_{k=1}^{K} TFT_k = \sum_{k=1}^{K} n_k C_{k,n_k} - \sum_{k=1}^{K} mft_k,$$

where the second sum is a schedule-independent constant. Thus the original problem is indeed equivalent to scheduling K unconstrained compound jobs with processing times $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = n_k$ for $k = 1, 2, \ldots, K$. □

The $Pm \parallel \sum w_k C_k$ problem is known to be NP-hard even for $m = 2$ [2,22], but the following easy-to-prove lemma is in the 'folklore'.

Lemma 4.2. *The* $Pm \parallel \sum w_k C_k$ *problem always has an optimal schedule which orders the jobs on each machine in shortest-weighted-processing-time (WSPT) order.*

Dror et al. [6] showed how to partition optimally a WSPT sequence of (compound) jobs between two machines by formulating this problem as a 0–1 program. They cited Kubiak's dynamic programming algorithm [19], which solves this 0–1 program on K jobs in $O(K\sum w_k)$, i.e., pseudo-polynomial time. Notice, however, that Lemma 4.1 reduces $P2|strong\ chains|\sum C_j$ to a $P2\parallel\sum w_kC_k$ problem on K compound jobs with weights $w_k = n_k$ for $k = 1, 2, \ldots, K$. Thus, $\sum w_k = \sum n_k = n$ in this case. Combining these results, they obtain the following:

Theorem 4.3. *An optimal solution for the $P2|strong\ chains|\sum C_j$ problem on K chains can be found in $O(Kn)$ time.*

In contrast, we mention the following result of Du et al. [9].

Theorem 4.4. *The $P2|chains|\sum C_j$ problem is NP-hard in the strong sense.*

Next we show how we can generalize Theorem 4.3 to the case of $m > 2$ machines. We use the following dynamic programming (DP) algorithm, which essentially follows the recursive scheme originally suggested by Rothkopf [25], where it is assumed that it is possible to index (order) the jobs in such a way that the jobs assigned to a given machine can be assumed to be processed in the order of their indices.

Algorithm DP

Input: N jobs indexed in the (WSPT) order $1, 2, \ldots, N$.

Let $F_j(t_1, \ldots, t_m)$ be the minimum cost of a schedule for jobs $1, 2, \ldots, j$ subject to the constraint that the last job on machine i is completed at t_i for $i = 1, 2, \ldots, m$. Then we have the following recursion

$$F_j(t_1, \ldots, t_m) = \min_{1\le i\le m}(F_{j-1}(t_1, \ldots, t_i - p_j, \ldots, t_m) + w_jt_i),$$

with the initial conditions

$$F_0(t_1, \ldots, t_m) = \begin{cases} 0 & \text{if } t_i = 0 \text{ for all } i = 1, \ldots, m \\ \infty & \text{otherwise.} \end{cases}$$

It is well known that these recursive equations can be solved in $O(mnT^{m-1})$ time, where $T = \sum_{j=1}^{N} p_j$. Thus combining this with Lemma 4.1, we obtain the following:

Theorem 4.5. *The $Pm|strong\ chains|\sum C_j$ problem can be solved in $O(mnT^{m-1})$ time, where $T = \sum_{j=1}^{N} p_j$.*

Theorems 4.3, 4.5 and 4.4 suggest that mean flow time minimization on parallel machines may be somewhat easier with *strong chains* precedence constraints than just with *chains*. When the number of machines is also considered to be a part of the input for the problem, however, then the situation reverses.

Theorem 4.6 (*[5]*). *The $P|strong\ chains, p_j = 1|\sum C_j$ problem is NP-hard in the strong sense.*

Theorem 4.7 (*[7]*). *The $P|chains, p_j = 1|\sum C_j$ problem can be solved in polynomial time.*

4.1.2. The $Pm|strong\ chains|\sum w_jC_j$ problem

It can be easily seen that Lemma 4.1 can also be extended to the weighted mean flow time problem:

Lemma 4.8. *Consider a $Pm|strong\ chains|\sum w_jC_j$ problem on K chains, $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_K$, where $|\mathbf{C}_i| = n_i$ for $i = 1, 2, \ldots, K$, and let $p_{k,i}$ and $w_{k,i}$ denote the processing time and weight, respectively, of the ith job in $\mathbf{C}_k$ for $i = 1, 2, \ldots, n_k$. Then this problem reduces to a $Pm\parallel\sum w_kC_k$ problem on K compound jobs, with no precedence constraints between them, where compound job k has processing time $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, \ldots, K$.*

Proof. Let C_{k,n_k} $(k = 1, 2, \ldots, K)$ be the completion time of the last job from chain $\mathbf{C}_k$ in a schedule. Since $\mathbf{C}_k$ is a bi-module, its jobs must be processed contiguously. Therefore, the completion time of the ith job in $\mathbf{C}_k$ can be written as $C_{k,i} = C_{k,n_k} - (p_{k,i+1} + \cdots + p_{k,n_k})$. Thus, the total weighted completion time of chain $\mathbf{C}_k$, denoted by TWT_k, is

$$TWT_k = w_kC_{k,n_k} - \left[(w_k - w_{k,n_k})p_{k,n_k} + (w_k - w_{k,n_k} - w_{k,n_k-1})p_{k,n_k-1} + \cdots + \left(w_k - \sum_{j=2}^{n_k} w_{k,j}p_{k,2}\right)\right].$$

If we let $mwt_k = [(w_k - w_{k,n_k})p_{k,n_k} + (w_k - w_{k,n_k} - w_{k,n_k-1})p_{k,n_k-1} + \cdots + (w_k - \sum_{j=2}^{n_k} w_{k,j}p_{k,2})]$, then this term is schedule-independent, and only the w_kC_{k,n_k} part of TWT_k varies with the schedule. Therefore, the total weighted completion time of a schedule can be written as

$$\sum_{k=1}^{K}\sum_{i=1}^{n_k} w_{k,i}C_{k,i} = \sum_{k=1}^{K} TWT_k = \sum_{k=1}^{K} w_kC_{k,n_k} - \sum_{k=1}^{K} mwt_k,$$

This problem reduces to a $Pm\parallel\sum w_kC_k$ problem on K compound jobs, with no precedence constraints between them, where compound job k has processing time $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, \ldots, K$. □

Lemma 4.8 means that we can apply the Algorithm DP even to the $Pm \parallel \sum w_k C_k$ problem that the $Pm|strong\ chains|\sum w_j C_j$ problem was shown to reduce to. Thus we obtain the following theorem.

Theorem 4.9. *The $Pm|strong\ chains|\sum w_j C_j$ problem can be solved in $O(mnT^{m-1})$ time, where T is the sum of all job processing times.*

4.2. Makespan problems on parallel machines

Minimizing the makespan on parallel machines is one of the most extensively studied problems in the literature. Even $P2 \parallel C_{\max}$ is known to be NP-hard and when the number of machines is considered to be a part of the input, i.e., we consider $P \parallel C_{\max}$, the problem becomes strongly NP-hard [11]. This indicates that there is not much hope for obtaining polynomial time solutions for these problems with or without precedence constraints. The situation is somewhat better for the case of unit processing times. One of the earliest algorithms for precedence-constrained scheduling is due to Hu [17], which always schedules from the set of available jobs the one that is at the beginning of the longest current chain of unexecuted jobs (the critical path algorithm).

Theorem 4.10 (*[17]*). *$P|$ in-tree, $p_j = 1|C_{\max}$ is solved in polynomial time by the critical path algorithm.*

An alternative polynomial time algorithm for the case of out-trees was given by Davida and Linton [4]. Garey et al. [12] studied the case when the precedence graph is the union of disjoint in-trees and out-trees (*opposing forest*). They showed that $P|opposing\ forest, p_j = 1|C_{\max}$ is NP-hard, but $Pm|opposing\ forest, p_j = 1|C_{\max}$ is polynomially solvable for fixed m. The first polynomial time solution for the two-machine problem with general precedence constraints, $P2|prec, p_j = 1|C_{\max}$, was presented by Fujii et al. [10]. Its complexity was improved by the well-known Coffman–Graham algorithm [3] in 1972. The complexity of $P3|prec, p_j = 1|C_{\max}$ is a long-standing open question in scheduling theory.

Although relatively little is known for the above problems with strong precedence constraints, it seems that these problems may be harder than their counter-parts with (weak) precedence constraints. In contrast with Theorem 4.10, we mention the following result:

Theorem 4.11 (*[8,5]*). *The $P|strong\ in\text{-}tree, p_j = 1|C_{\max}$ problem is NP-hard in the strong sense.*

This can be further strengthened as follows.

Theorem 4.12 (*[7]*). *The $P|strong\ chains, p_j = 1|C_{\max}$ problem is NP-hard in the strong sense.*

The last problem becomes somewhat easier to solve on a fixed number of machines by using the above-mentioned dynamic programming algorithm first proposed by Rothkopf [25].

Theorem 4.13 (*[7]*). *The $Pm|strong\ chains, p_j = 1|C_{\max}$ problem is solvable in $O(mn^{m+1})$ time.*

5. Summary

The notion of strong versus weak precedence in job scheduling is interesting even beyond its connection to modular decomposition for chains, trees, and series–parallel posets. For instance, as demonstrated by Krings and Dror [18], it can be used for addressing the classical problem of scheduling instability in non-preemptive static list scheduling: Defining strong precedence relation in terms of precedence sequence of predetermined nested subgraphs results in a schedule that is inherently stable without the introduction of new precedence edges contrary to what was proposed prior to [18].

The main contribution of this paper lies primarily in its consistent extension of the notion of strong precedence with respect to chains, trees, and series–parallel posets. The construction of in-module, out-module, and bi-module structures, defined with respect to the transitive reduction of a poset P, allows a consistent definition of strong precedence based on modular decomposition for posets. The majority of new computational complexity results presented in this paper is the consequence of the strong precedence extension for series–parallel graphs. For instance, we show, among a number of other results, that the problem $1|strong\ series\text{–}parallel|\sum w_j C_j$ is solvable in polynomial time. In addition, we verified the complexity results for the 'old' definition of strong precedence for chains and trees [6–8] with respect to the present 'modular' definition of the weak/strong precedence distinction.

Acknowledgement

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant No. OG1798-08.

References

[1] A.A. Assad, M.O. Ball, R.W. Dahl, Sleeping beauties: scheeduling the production of mattresses with sequence-dependent set ups, in: Proceeding, 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, Paris, France, 1995, pp. 229–239.
[2] J.L. Bruno, E.G. Coffman, J.R. Sethi, Scheduling independent tasks to reduce mean finishing time, Commun. ACM 17 (1974) 382–387.

[3] E.G. Coffman, R.L. Graham, Optimal scheduling of two-processor systems, Acta Inform. 1 (1972) 200–213.
[4] G.I. Davida, D.J. Linton, A new algorithm for the scheduling of tree structured tasks, in: Proc. Conf. Inform. Sci. and Syst. Baltimore, MD, USA, 1976, pp. 543–548.
[5] M. Dror, Chains and trees: 'strong'–'weak' order in job scheduling, Order 14 (1997) 211–228.
[6] M. Dror, W. Kubiak, P. Dell'Ollmo, Scheduling chains to minimize mean flow time, Inform. Process. Lett 61 (1997) 297–301.
[7] M. Dror, W. Kubiak, P. Dell'Ollmo, 'Strong' 'weak' chain constrained scheduling, Ric. Oper. 83 (1997) 35–49.
[8] M. Dror, W. Kubiak, J.Y.-T. Leung, Tree precedence in scheduling: the strong–weak condition, Inform. Process. Lett. 71 (1999) 127–134.
[9] J. Du, J.Y.-T. Leung, C.-H. Lin, Scheduling chain-structured tasks to minimize makespan and mean flow time, Inform. and Comput. 92 (1991) 219–236.
[10] M. Fujii, T. Kasami, K. Ninomiya, Optimal sequencing of two equivalent processors, SIAM J. Appl. Math. 17 (1969) 784–789. (erratum); SIAM J. Appl. Math. 20 (141) (1971).
[11] M.R. Garey, D.S. Johnson, Strong NP-completeness results: motivation, examples and implications, J. Assoc. Comput. Mach. 24 (1978) 499–508.
[12] M.R. Garey, D.S. Johnson, R.E. Tarjan, M. Yannakakis, Scheduling opposing forests, SIAM J. Algebr. Discrete Methods 4 (1983) 72–93.
[13] V.S. Gordon, Some properties of series–parallel graphs, Izv. Akad. Nauk. BSSR 1 (1981) 18–23. (in Russian).
[14] V.S. Gordon, C.N. Potts, V.A. Strusevich, J.D. Whitehead, Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation, J. Sched. 11 (2008) 357–370.
[15] V.S. Gordon, J.-M. Proth, V.A. Strusevich, Single machine scheduling and due date assignment under series–parallel precedence constraints, Cent. Eur. J. Oper. Res. 13 (2005) 15–35.
[16] V.S. Gordon, V.A. Strusevich, Earliness penalties on a single machine subject to precedence constraints: SLK due date assignment, Comput. Oper. Res. 26 (1999) 157–177.
[17] T.C. Hu, Paralllel sequencing and assembly line problems, Oper. Res. 9 (1961) 841–848.
[18] A.W. Krings, M. Dror, Real-time dispatching: scheduling stability and precedence, Internat. J. Found. Comput. Sci. 10 (1999) 313–327.
[19] W. Kubiak, New results on the completion time variance minimization, Discrete Appl. Math. 58 (1995) 157–168.
[20] E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, Ann. Discrete Math. 2 (1978) 75–90.
[21] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: S. Graves (Ed.), Handbooks in OR & MS, vol. 4, Elsevier Publ., 1993.
[22] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.
[23] R.H. Mőhring, Computationally tractable classes of ordered sets, in: I. Rival (Ed.), Algorithms and Order, Kluwer Publ., Dordrecht, The Netherlands, 1989.
[24] E.L. Monma, J.B. Sidney, Sequencing with series–parallel precedence constraints, Math. Oper. Res. 3 (1979) 215–224.
[25] M.H. Rothkopf, Scheduling independent tasks on parallel processors, Manage. Sci. 12 (1966) 437–447.
[26] J.B. Sidney, Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs, Oper. Res. 23 (1975) 283–298.
[27] G. Steiner, On finding the jump number by substitution decomposition, Order 2 (1985) 9–23.
[28] J.R. Valdes, E. Tarjan, E.L. Lawler, The recognition of series–parallel digraphs, SIAM J. Comput. 11 (1982) 361–370.