‘Strong’-‘Weak’ Precedence in Scheduling: Extended Order Implications

Moshe Dror · George Steiner

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 graphs. Using modular decomposition for partially ordered sets (posets), we restate and extend past complexity results for chains and trees as summarized in [5]. Moreover, for series parallel graphs we establish new computational complexity results for strong precedence constraints for single and multiple 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 a precedence relation to otherwise independent jobs in general results in a harder problem than the original one. Given a set of jobs $J$ together with a partial order $\prec_p$ defined on $J \times J$, the common interpretation of precedence constraints (prior to Dror et al., [6,7]) was the following: for any $i, j \in J$, $i \prec_p j$ means that job $i$ has to be completed before the processing of job $j$ can start. This interpretation of the precedence constraints represented by partial order $\prec_p$ 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] and [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 different solution. Simply put, asking for a job $j$ to immediately follow the completion of job $i$ links the two jobs in a strong precedence. A weak precedence relation reflects
the ‘old’ 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 in-between the chained jobs. This is exactly the way the QMS algorithm sched-
ules chained jobs in a manufacturing of mattresses as described by Assad et al. (1995)
[1]. 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 has been introduced in [6]
and [7]. In [8] this distinction has been introduced for in-tree and out-tree precedence
graphs. However, the strong chain precedence as defined in [6] and [7] was not consistent
with the strong precedence tree relation as defined in [8]. This definitional inconsistinc
has been rectified in the present paper. In the past, 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 graphs 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 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 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 1 Let \( P = (J, \prec) \) be a partially ordered set (a poset). Let \( \prec \) denote
the immediate predecessor relation in \( P \). That is, \( \prec \) represents the transitive reduction
of \( \prec \). For \( j \in J \) denote by \( ImSuc(j) = \{ k | j \prec k \} \) and by \( ImPred(j) = \{ k | k \prec j \} \).
We say that \( A \subseteq J \) is an in-module in \( P \) if \( \forall i, j \in A \) we have \( ImSuc(i) \cap (J \setminus A) = ImSuc(j) \cap (J \setminus A) \).
A subset \( B \subseteq J \) is an out-module in \( P \) if \( \forall i, j \in B \) we have
\( ImPred(i) \cap (J \setminus B) = ImPred(j) \cap (J \setminus B) \). 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. 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 \( ImSuc(i) \cap (J \setminus A) = \emptyset \) or \( ImSuc(i) \cap (J \setminus A) \) is an antichain.
Similarly, if \( B \subseteq J \) is an out-module in a general poset \( P \), then \( \forall i \in B \) we have either
\( ImPred(i) \cap (J \setminus B) = \emptyset \) or \( ImPred(i) \cap (J \setminus B) \) 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 \( J = C_1 \cup C_2 \ldots \cup C_K \) is the union of \( K \) parallel chains, then each \( C_i \) is both
an in-module and an out-module (a bi-module) in \( P \) : if \( A = C_i \), then \( \forall i, j \in A \) we
have \( ImSuc(i) \cap (J \setminus A) = ImSuc(j) \cap (J \setminus A) = \emptyset \); and if \( B = C_i \), then \( \forall i, j \in B \) we
have \( ImPred(i) \cap (J \setminus B) = ImPred(j) \cap (J \setminus B) = \emptyset \). Furthermore, no proper nontrivial
subset of a chain (i.e., not a single point) has both properties. In that case we say that
each \( C_i \) is a minimal non-trivial bi-module in \( P \). It is also easy to see that if the poset
\( P = (J, \prec_P) \) is an in-tree, then \( \text{ImPred}(i) \) is an in-module \( \forall i \in J \); and if \( P \) is an out-tree, then \( \text{ImSuc}(i) \) is an out-module \( \forall i \in J \).

Using the above definitions, strong precedence relation in scheduling implies the following:

- for chains: the minimal non-trivial bi-modules (one chain from a set of parallel chains) must be scheduled in a contiguous fashion, i.e., every job in the chain must be scheduled as early as possible after the first job on the chain. If \( i \prec j \) in a chain, then this implies \( S_j = S_i + p_i \), where \( S_i \) is the start time for any job \( l \) and \( p_i \) is its processing time (in the case of identical parallel processors we assume w.l.o.g. that the jobs from a given bi-module are processed on the same processor);
- for in-trees: the maximal in-modules, \( \text{ImPred}(i) \) for some \( i \), must be scheduled in a contiguous fashion (successively), i.e., every job in the in-module must be scheduled as early as possible after the first job in it;
- for out-trees: the maximal out-modules, \( \text{ImSuc}(i) \) for some \( i \), must be scheduled in a contiguous fashion (successively), i.e., every job in the out-module must be scheduled as early as possible after the first job in it.

There is a well-established theory of ‘modular decomposition’ for posets in general and series-parallel posets in particular. This is based on the following concept of modules:

**Definition 2** Let \( P = (J, \prec_P) \) be a partially ordered set. \( A \subseteq J \) is a module in \( P \) if \( \forall i, j \in A \) we have \( \text{Suc}(i) \cap (J \setminus A) = \text{Suc}(j) \cap (J \setminus A) \), where \( \text{Suc}(j) = \{ k \mid j <_P k \} \), and \( \text{Pred}(i) \cap (J \setminus A) = \text{Pred}(j) \cap (J \setminus A) \), where \( \text{Pred}(j) = \{ k \mid k <_P j \} \).

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 [19] for details). They are all based on the following job-module property:

**Definition 3** Let \( P = (J, \prec_P) \) 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) \( \sigma_A \) 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) \( \sigma \) for the whole problem such that \( \sigma \) orders the jobs in \( A \) (i.e., their starting times) in the same sequence (relative to each other) as \( \sigma_A \).

**Definition 4** Series-parallel poset:
1. A poset with a single element is series-parallel;
2. If \( P_1 = (J_1, \prec_{P_1}) \) and \( P_2 = (J_2, \prec_{P_2}) \) are disjoint series parallel posets, then
   (a) their series composition \( P = P_1 \ast P_2 \) is the poset on \( J_1 \cup J_2 \) with \( i \prec_P j \) iff \( (i <_{P_k} j \text{ for } i, j \in P_k \text{ and } k = 1, 2) \) or \( (i \in P_1 \text{ and } j \in P_2) \);
A series-parallel poset is one that can be built up from single elements by a sequence of series and parallel compositions. The Hasse diagram for a series-parallel poset is shown in Figure 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 \( P = C_1 \cup ... \cup C_k \) is a collection of disjoint chains, then it can also be considered as \( P = C_1 + ... + C_k \), 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 compositions. They will be referred to as *series* and *parallel modules*, respectively. The scheduling problem \( 1|\text{prec}|\sum w_iC_i \) ([22]), jump number ([23]), and a number of other problems (see [20]) 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 [19,20]).

Let \( P \) be the series (parallel) composition of the posets \( P_1 \) and \( P_2 \) and let \( \sigma_i \) be an optimal sequence for the subproblem defined on \( P_i, i = 1, 2 \). If \( P = P_1 \ast P_2 \), then the job-module property guarantees that \( \sigma = (\sigma_1, \sigma_2) \) is optimal for the problem defined

![Fig. 1 A series-parallel poset](image-url)
on $P$. On the other hand, if $P = P_1 + P_2$, then the parallel chains corresponding to the sequences $\sigma_1$ and $\sigma_2$ have to be optimally merged into a sequence $\sigma$ by some algorithm. For various implementations of this argument we refer the reader to [17,20].

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. [19]).

**Theorem 1** 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 posets.

**Definition 5** Consider a scheduling problem with $P = (J, \prec_P)$ a poset on the job set $J$ with $\prec_P$ representing the transitive reduction of relations in $P$. We say that $P$ represents strong precedence constraints, denoted by $\text{strong prec}$, 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 a contiguous fashion, i.e., every job in \( C \) must be scheduled as early as possible after scheduling the first job in \( C \).

3. Consider any non-trivial 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 a contiguous fashion, i.e., every job in \( A \) must be scheduled as early as possible after scheduling the first job in \( A \).

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. Of course, a feasible schedule for strong precedence constraints also must satisfy the additional contiguity constraints. We note that the contiguity constraints would be meaningless for the trivial modules; these were excluded from the definition in point 3. Furthermore, we require the contiguity constraint only for non-trivial in-modules with a non-empty successor set to exclude the contiguity constraint for \( A = \max P \), where \( \max P \) denotes the maximal elements of \( P \). Similarly, requiring a non-empty predecessor set for out-modules excludes \( A = \min P \), where \( \min P \) denotes the minimal elements in \( P \). All modules which need to satisfy the contiguity constraint can be replaced by a composite 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.

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 of the two problems is nontrivial.

3.1 Hard problems

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

**Theorem 2** If the \( 1|\text{chain}| \sum f_j(C_j) \) scheduling problem is NP-hard, then the \( 1|\text{strong out}−\text{tree}| \sum f_j(C_j) \) problem is also NP-hard. If the \( 1|\text{chain}| \max f_j(C_j) \) scheduling problem is NP-hard, then the \( 1|\text{strong out}−\text{tree}| \max f_j(C_j) \) problem is also NP-hard.

The proof is based on reducing the \( 1|\text{chain}| \sum f_j(C_j) \) (or \( 1|\text{chain}| \max f_j(C_j) \)) problem to a \( 1|\text{strong out}−\text{tree}| \sum f_j(C_j) \) (or \( 1|\text{strong out}−\text{tree}| \max f_j(C_j) \)) problem, where there is a one-to-one correspondence between the optimal schedules. It immediately yields the following corollary.

**Corollary 1** The \( 1|\text{strong out}−\text{tree}, p_j = 1| \sum U_j \) problem is NP-hard.
Table 1 The data for an instance of $\min w_iC_i$:

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>w_i</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>p_i</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

It is also shown in [8] that there are instances of $\min \sum w_iC_i$ and $\min \sum U_j$ 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 for scheduling problems satisfying the job-module property from weak to strong precedence constraints. For illustration we use the instance of $\min \sum w_iC_i$ whose precedence constraints are represented by the poset of Figure 1 and whose data is given in Table 1. (The data is the same as in the example solved by Lawler [17], allowing us to compare the optimum cost with and without strong precedence constraints.)

Consider a poset $P = (J, \prec_P)$ and consider the maximal in-modules in $P$ that are not parallel-chain job 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 $\IN$. This partition contains the following sets for the example poset of Figure 1: $\IN = \{\{1\}, \{2,3\}, \{4\}, \{5\}, \{6,9\}, \{7,8\}, \{10,11\}, \{12,13\}\}$. 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, $A_1, A_2 \in \IN \setminus \{\max P\}$ such that there is a $k \in A_1$ and $l \in A_2$ such that $k \prec_P l$ 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 $u \prec_P v$ 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 job module in the poset of $P'$. For example, consider $A_1 = \{5\}$ and $A_2 = \{6,9\}$ in the poset of Figure 1: Since $5 \prec_P 9$ 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 $5 \prec_P 6$ 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.

Consider again a poset $P = (J, \prec_P)$ 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 $\mathit{OUT}$. For the poset example in Figure 1 this partition contains the following sets: $\mathit{OUT} = \{\{1\}, \{2,3,4\}, \{5,6\}, \{9\}, \{7,8\}, \{10\}, \{11\}, \{12,13\}\}$.

Extensions of type 2 (between out-modules)

Consider now two out-modules, $B_1, B_2 \in \mathit{OUT} \setminus \{\min P\}$ such that there is a $k \in B_1$ and $l \in B_2$ and also $k < l$ 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 $u < v$ 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 job module in the poset of $P'$. For example, consider $B_1 = \{2,3,4\}$ and $B_2 = \{5,6\}$ in the poset of Figure 1: Since $2 < 5, 2 < 6$ 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 $4 < 5$ and $4 < 6$ to $P$. Similarly, considering $\{7,8\}$ for $B_2$, we can add the relations $2 < 7, 2 < 8, 3 < 7$ and $3 < 8$ 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 $5 < 9$ and the contiguity requirement for $B_1$ imply that we can add the (weak precedence) relation $6 < 9$, which makes $B_1 = \{5,6\}$ a job module in the extended poset.

Extensions of type 3 (between overlapping out- and in-modules)

Consider now an $A \in \mathcal{IN} \setminus \{\max P\}$ and a $B \in \mathit{OUT} \setminus \{\min P\}$ such that $U = A \cap B \neq \emptyset$, but $A \neq U$ and $B \neq U$. By definition both $A$ and $B$ are constrained to be scheduled contiguously. Moreover, every job in $A \setminus U$ must be scheduled contiguously with the jobs in $U$, and every job in $B \setminus U$ must also be scheduled contiguously with $U$. This implies that the common elements $U$ "glue" the overlapping modules together into composite jobs, and all feasible sequences are of the form $(A \setminus U), (B \setminus U)$ or $(B \setminus U), (A \setminus U)$. Now take a job $u \in U$. We claim that if $P$ is a series-parallel poset, then there must be an $a \in (A \setminus U)$ and $b \in (B \setminus U)$ with $b <_P a$ : There cannot be an $a \in (A \setminus U)$ and $b \in (B \setminus U)$ with $a <_P b$ since this would imply $a <_P u$ (as $u \in B$ and $B$ is an out-module), which is not possible since $u \in A$ and $A$ is an in-module. If we had $a|b$, then for any common immediate successor $x$ of $a$ and $u$, we would have $a < x$ and $u < x$, while for any common immediate predecessor $y$ of $B$, we would have $y < b$ and $y < u$, and together these would imply that $\{a,y,x,b\}$ form a subposet $Z$, that is forbidden in a series-parallel precedence setting by Theorem 1. Thus, we must have $b <_P a$ for $a \in (A \setminus U)$ and $b \in (B \setminus U)$. This implies that any feasible sequence for the composite jobs must be of the form $(B \setminus U), (A \setminus U)$, and this in turn implies that every predecessor of any job in $A$ must also precede all the jobs in $B$. Also any successor of any job in $B$ must succeed every job in $A$. In other words, if we add these induced weak precedence constraints to $P$ then composite job $A \cup B$ will be a job module in the extended poset. As an illustration of this point, consider in the poset of Figure 1 the sets $B = \{5,6\}$ and $A = \{6,9\}$: We have $U = A \cap B = \{6\}$, and since $5 < 9$, the only feasible sequence is $(B \setminus U), (A \setminus U) = (5,6,9)$. This makes the string $(5,6,9) a
job module in the poset $P$ extended by the relation $5 \prec 6$.

Another example for overlapping in-modules and out-modules in the poset of Figure 1 is $A = \{10, 11\}$ and $B = \{10\}$. Note that $U = A \cap B = B$ in this case, thus the gluing operation adds the “new” composite job $(U, (A \setminus U))$, which of course is just $A$. In this case, by applying extension 3, the weak precedence relations $9 \prec 10, 7 \prec 11$ and $8 \prec 11$ are added, thus making $A$ a job module in the extended poset.

Algorithm $\text{Strong SP}$

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

1. **Step 1.** Identify the partitions of $J$ into in-modules and out-modules, respectively, denoted by $IN$ and $OUT$.
2. **Step 2.** Extend the poset $P$ by adding the weak precedence constraints arising from the application of all extensions of type 1 and 2. Let the resulting poset be $P'$.
3. **Step 3.** Contract all nontrivial in-modules in $IN \setminus \{\max P\}$ and all nontrivial out-modules in $OUT \setminus \{\min P\}$ into composite jobs. If $P$ has any parallel-chain bi-modules, then contract these into composite strings.
4. **Step 4.** Extend $P'$ further by adding the weak precedence constraints arising from the application of all extensions of type 3 and apply the gluing operation to form composite jobs on the overlapping modules. Let the resulting extended poset be $P''$.
5. **Step 5.** Apply the original series-parallel algorithm for weak precedence constraints [17, 20] 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 detail, we refer the reader to [20]. In the case of the $1\|\sum w_i C_i$ 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 $1\|\sum w_i C_i$ problem [17, 20] repeatedly uses this preference order on sets and strings of jobs.

When applying Algorithm $\text{Strong SP}$ to the example whose strong precedence constraints are represented by the poset depicted in Figure 1, we add the extensions 1-3 as discussed above. The binary decomposition tree of the resulting extended weak precedence constraints $P''$ is shown in Figure 3.

Explanation of the schedule generated by Algorithm $\text{Strong SP}$:

Applying the original series-parallel algorithm in Step 5, we start sequencing the job-modules with the leaves of the tree in Figure 3. The elements of the composite job
Fig. 3 The binary decomposition tree for the extended poset $P^c$.

(7, 8) are scheduled in WSPT order resulting in the string $7, 8$. Comparing the parallel strings $5, 6, 9$ and $7, 8$, $w(5, 6, 9)/p(5, 6, 9) = 12/20$ and $w(7, 8)/p(7, 8) = 13/6$, which means that $7, 8$ should be scheduled before $5, 6, 9$ 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 $11, 10$. The WSPT order on the elements of $(12, 13)$ is $12, 13$, so it gets replaced by the string $12, 13$. The WSPT order on $(2, 3)$ is $3, 2$, so it gets replaced by the string $3, 2$. This string comes before $4$ in the WSPT order, so the composite job $(2, 3)$ is replaced by the string $3, 2, 4$. Finally putting all strings in the order that satisfies the series compositions in the tree, we obtain the optimal schedule $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$ 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 $\sum w_iC_i = 1158$. This demonstrates that the strong precedence constraints can result in a substantial increase in the minimum cost.

**Theorem 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 polynomial 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 $f = f_{max} = \max_j f_j(C_j)$. The problem only needs to satisfy the job-module property.

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 $P$. 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.
We mention the following immediate consequence of the preceding theorem.

**Corollary 2** The total weighted completion time problem with strong series-parallel precedence constraints, \(1|\text{strong series-parallel}| \sum w_j C_j\), is solvable in polynomial time.

4 Multi-machine Models

4.1 Mean Flow Time on Parallel Machines

4.1.1 The \(Pm|\text{strong chains}| \sum C_j\) problem

We start our discussion by extending to any number of machines a lemma that was proved in [6] for the two-machine case.

**Lemma 1** Consider a \(Pm|\text{strong chains}| \sum C_j\) problem on \(K\) chains, \(C_1, C_2, \ldots, C_K\), where \(|C_i| = n_i\), \(i = 1, 2, \ldots, K\), and let \(p_{k,i}\) denote the processing time of the \(i\)th job in \(C_k\), \(i = 1, 2, \ldots, n_k\). This problem reduces to a \(Pm|\sum w_k C_k\) 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 \(C_k\) in a schedule. Since \(C_k\) is a bi-module, its jobs must be processed continuously. Thus if \(p_{k,i}\) denotes the processing time of the \(i\)th job in \(C_k\) 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 \(C_k\), 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 \(Pm|\sum w_k C_k\) problem is known to be NP-hard even for \(m = 2\) [2,18], but the following easy-to-prove lemma is in the “folklore”.

**Lemma 2** The \(Pm|\sum w_k C_k\) 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 Kubrak’s dynamic programming algorithm [15], which solves this 0-1 program on \(K\) jobs in \(O(K \sum w_k)\), i.e., pseudo-polynomial time. Notice, however, that Lemma 1 reduces \(P2|\text{strong chains}| \sum C_j\) to a \(P2|\sum w_k C_k\) 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 An optimal solution for the $P^2|strong\ chains|\sum C_j$ problem on $K$ chains can be found in $O(Kn)$ time.

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

Theorem 5 The $P^2|chains|\sum C_j$ problem is NP-hard in the strong sense.

Next we show how we can generalize Theorem 4 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 [21], 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 order of their indices.

Algorithm DP

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

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

$$F_j(t_1, ..., t_m) = \min_{1 \leq i \leq m} (F_{j-1}(t_1, ..., t_i - p_j, ..., t_m) + w_j t_i),$$

with the initial conditions

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

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 1, we obtain the following:

Theorem 6 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, 6, and 5 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 part of the input for the problem, however, then the situation reverses.

Theorem 7 [5] The $P|strong\ chains,p_j = 1|\sum C_j$ problem is NP-hard in the strong sense.

Theorem 8 [7] The $P|chains,p_j = 1|\sum C_j$ problem can be solved in polynomial time.

4.1.2 The $Pm|strong\ chains|\sum w_j C_j$ problem

It can be easily seen that Lemma 1 can also be extended to the weighted mean flow time problem:

Lemma 3 Consider a $Pm|strong\ chains|\sum w_j C_j$ problem on $K$ chains, $C_1, C_2, ..., C_K$, where $|C_i| = n_i$ for $i = 1, 2, ..., K$, and let $p_{k,i}$ and $w_{k,i}$ denote the processing time and weight, respectively, of the $i$th job in $C_k$ for $i = 1, 2, ..., n_k$. Then this problem reduces to a $Pm||\sum w_k C_k$ 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 $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, ..., K$. 


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

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

If we let $\mu w_{k} = [(w_k - w_{k,n_k})p_{k,n_k} + (w_k - w_{k,n_k} - w_{k,n_k-1})p_{k,n_k-1} + \cdots + (w_k - \sum_{j=2}^{n_k} w_{k,j}p_{k,2})]$, 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

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

This problem reduces to a $P_m|\sum w_k C_k$ 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 $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, ..., K$.

Lemma 3 means that we can apply the Algorithm DP even to the $P_m|\sum w_k C_k$ problem that the $P_m|\text{strong chains}|\sum w_j C_j$ problem was shown to reduce to. Thus we obtain the following theorem.

Theorem 9 The $P_m|\text{strong chains}|\sum w_j C_j$ 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 $P2|\text{Cmax}$ is known to be NP-hard and when the number of machines is considered to be part of the input, i.e., we consider $P||\text{Cmax}$, 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 T.C. Hu [13], 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 10 $P|\text{in - tree}, p_j = 1|\text{Cmax}$ 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 $P|\text{opposing forest}, p_j = 1|\text{Cmax}$ is NP-hard, but $Pm|\text{opposing forest}, p_j = 1|\text{Cmax}$ is polynomially solvable for fixed $m$. The first polynomial time solution for $P2|\text{prec}, p_j =
1|C_{\text{max}} \text{ 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|\text{prec}, p_j = 1|C_{\text{max}} \text{ 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 10, we mention the following result:}

**Theorem 11** [8, 5] The $P|\text{strong in - tree}, p_j = 1|C_{\text{max}}$ problem is NP-hard in the strong sense.

This can be further strengthened as follows.

**Theorem 12** [7] $P|\text{strong chains}, p_j = 1|C_{\text{max}}$ 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 [21].

**Theorem 13** [7] $Pm|\text{strong chains}, p_j = 1|C_{\text{max}}$ 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 graphs. For instance, as demonstrated by Krings and Dror ([14]), 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 [14].

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 graphs. 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 parallel $|C_j$ is solvable in polynomial time. In addition, we verified the complexity results for the ‘old’ definition of strong precedence for chains and trees ([6],[7],[8]) with respect to the present ‘modular’ definition of the weak/strong precedence distinction.

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