A Series-Parallel digraph based relaxation for the COMPLETION constraint

Camille Bonnin, Thomas Dissaux, Arnaud Malapert, Nicolas Nisse, Marie-Laure Espinouse

A Series-Parallel digraph based relaxation for the COMPLETION constraint

Camille Bonnin
Univ. Grenoble Alpes, CNRS, Grenoble INP[1], G-SCOP, 38000 Grenoble, France

Thomas Dissaux
Université Côte d'Azur, CNRS, Inria, I3S, Sophia Antipolis, France

Arnaud Malapert
Université Côte d'Azur, CNRS, I3S, France

Nicolas Nisse
Université Côte d'Azur, Inria, CNRS, I3S, Sophia Antipolis, France

Marie-Laure Espinouse
Univ. Grenoble Alpes, CNRS, Grenoble INP[1], G-SCOP, 38000 Grenoble, France

Abstract

In 2011, Kovács and Beck defined the COMPLETION global constraint for minimizing the weighted flowtime for a single resource. This constraint is part of the cost-based domain filtering approach and as such, aims at removing values from the variables domains that cannot lead to a solution with a better cost than the best one found so far. The COMPLETION constraint uses a polynomial time relaxation that minimizes the weighted mean busy time on a single unary capacity resource with release dates and preemption. In this ongoing work, we consider another polynomial time relaxation that directly minimizes the weighted completion time without preemption where the precedence constraints form a Series-Parallel digraph. This is one of the first attempt to use a relaxation without preemption. We discuss how to build a Series-Parallel digraph from the time windows of the tasks or directly working on their interval graph.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling; Applied computing → Operations research

Keywords and phrases Constraint Programming, Global Constraint, Cost-Based Domain Filtering, Operation Research, Scheduling, Weighted Flowtime, Series-Parallel Digraphs

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

In our current society, scheduling problems are omnipresent with great diversity of constraints and objectives. Currently, the most famous and more studied objective function is the makespan (*i.e.*, the end time of the schedule). However, in real applications, other objective functions are also used. One of those objective function is the minimization of the weighted flowtime (*i.e.*, the weighted sum of the completion time of the tasks) which is used in a large variety of applications, from the industrial field, such as automobile gear manufacturing [8], to healthcare service sector [6] through multiprocessor scheduling [10].

Among the many existing methods used to solve scheduling problems, Constraint Programming (CP) is a well-established one. Still, it focuses more on solving satisfaction problems than optimization ones. Indeed, to solve an optimization problem in CP, the classical method is to solve a sequence of satisfaction problems where constraints bounding the domains of the objective variables are added [2]. Nevertheless, there are more and more

[1] Institute of Engineering Univ. Grenoble Alpes

studies on how to efficiently solve optimizations problems in CP. Such is the case of the cost-based domain filtering approach which uses relaxations in global constraints to prune combinations of values that cannot lead to better-costed solutions that the best one found so far [7]. The COMPLETION constraint defined by Kovács and Beck [11] is an example of this approach and aims at pruning combinations of values that cannot lead to solutions with a smaller weighted flowtime for a single-machine scheduling problem with time windows.

Our long therm objective is to develop another filtering algorithm for the COMPLETION constraint that uses a different relaxation than the one used by Kovács and Beck in order to compare their respective performances. The considered relaxation has precedence constraints in the form of a Series-Parallel digraph (SP-digraph) and can be solved by a polynomial time algorithm proposed by Lawler [12]. In order to use this relaxation for the COMPLETION constraint, "good" Series-Parallel digraph must be created from the time windows of the tasks. This ongoing work proposes two heuristics that take an interval graph (deduced from the time windows) and construct a precedences digraph in the form of a Series-Parallel digraph that is a sub-digraph of the precedences digraph corresponding to the interval graph. It also proposes a new class of digraphs that generalises the Series-Parallel digraphs and explain how to compute such a precedences digraph, but the complexity of the corresponding scheduling problem is currently unknown.

This paper is structured as follow. Section 2 gives the definition of the COMPLETION constraint and some notations from scheduling theory. Then, Section 3 describes the proposed approaches to compute a lower bound using Series-Parallel digraphs and graph theory. Section 4 concludes this paper and gives some perspectives.

2 The COMPLETION constraint

The COMPLETION global constraint has been defined by Kovács and Beck in 2011 [11]. It aims to minimize the weighted sum of the completion times obtained when scheduling without preemption n tasks with time windows on a single resource. The corresponding problem is written $1|r_j;d_j|\sum w_jC_j$ in the Graham notation.

Each task T_j posses a weight, w_j, and a duration, p_j, which is the time taken to completely execute the task. For each task T_j, two variables are associated, S_j and C_j which represent respectively the start time and the end time of its execution. As each task must be executed without preemption, the constraint $C_j = S_j + p_j$ must be respected. Also, at each time, the current lower bound of S_j is called the release date of T_j and is denoted r_j. In the same way, the current upper bound of C_j is called the deadline of T_j and is denoted d_j. In addition to those notations, we write C the sum of the weighted completion times of all the tasks. We assume that all data are integers. The COMPLETION constraint takes the following form.

$$\begin{aligned} &\text{COMPLETION}([S_1, \ldots, S_n], [p_1, \ldots, p_n], [w_1, \ldots, w_n], C) \\ &\overset{\text{def}}{\iff} ((S_i + p_i \leq S_j \vee S_j + p_j \leq S_i) \wedge \textstyle\sum_i w_i(S_i + p_i) = C) \end{aligned}$$

This constraint is part of the cost-based domain filtering [7]. As such, it aims to reduce the time taken by the solver to find the optimal schedule and not only a feasible schedule. To achieve this result, the propagation algorithm proposed by Kovács and Beck [11] uses a lower bound computed by using a preemptive polynomial time relaxation, $1|r_j;pmtn|\sum w_jM_j$. This single machine preemptive relaxation enforces the respect of the release dates of the tasks and minimizes the weighted sum of the mean busy times, which gives a lower bound of the weighted flowtime. M_j, the mean busy time of a task j is the average of all the times in which j is executed. The goal of this ongoing work is to propose another propagation

algorithm for this constraint using another non-preemptive polynomial time relaxation, $1|sp\text{-}graph|\sum w_jC_j$. This scheduling problem is the single machine problem with precedences in the form of a SP-digraph minimizing the weighted sum of the completion times. This relaxation can be solved in polynomial time by an algorithm given by Lawler [12] which uses the tree decomposition of a SP-digraph to construct and manipulate sub-sequences of tasks. Using a polynomial non-preemptive relaxation is not the classical approach for this problem, so this may lead or not to different deductions that with a preemtive relaxation.

3 Computing a lower bound a using graph theory

To design a filtering algorithm for the COMPLETION constraint, computing a lower bound is an important step. This section describes the approaches studied to compute a lower bound using precedence constraints.

In order to deduce precedence relations from the time windows of n tasks, the first notion that comes to mind is the interval graph of the tasks. An interval graph, written here $\mathcal{I}$, is a graph where the vertices correspond to the tasks and where there is an edge between two vertices if and only if the time windows of the corresponding tasks intersect. For instance, Figure 1b represents the interval graphs $\mathcal{I}$ and $\mathcal{I}'$ of the sets of intervals of Figure 1a (respectively the set with only the black tasks and the set with also the red task).

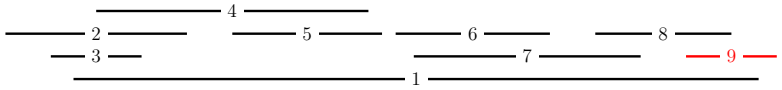
However, an interval graph indicates which combinations of tasks have no immediate precedence relation, but does not gives the precedence relations involving the other combinations. This information can be found by looking at the precedence digraph of $\mathcal{I}$. Such a digraph is written $\vec{\mathcal{I}}$ here and represents the interval order of the tasks. It has the same vertices as the corresponding $\mathcal{I}$, but there is an arc from two vertices v_i and v_j if and only if $d_i \leq r_j$ where d_i is the deadline of the task corresponding to v_i and r_j is the release date of the task corresponding to v_i. An illustration is given in Figure 1c where the precedences digraph $\vec{\mathcal{I}}$ of the digraph $\mathcal{I}$ of Figure 1b is depicted. The single machine scheduling problem with precedence constraints in the form of an interval order digraph minimizing the weighted flowtime, $1|interval\ order\ |\sum w_jC_j$, is then a relaxation of $1|r_j;d_j|\sum w_jC_j$. However, this problem is $\mathcal{NP}$-hard [1] for any interval order digraph of precedences. Therefore, our goal is to compute a subgraph of $\vec{\mathcal{I}}$ such that the corresponding single machine scheduling problem minimizing the weighted flowtime can be solved in polynomial time.

Before describing the approaches considered to achieve this goal, some definitions need to be reminded. The transitive closure, $tc(\vec{D})$ of a digraph $\vec{D}$ is obtained by iteratively (while possible) adding the arc (u, v) for any two non-adjacent vertices u and v such that there exists a directed path from u to v. Note that, if $\mathcal{I}$ is an interval graph, then $tc(\vec{\mathcal{I}}) = \vec{\mathcal{I}}$. The "reverse" operation, is the transitive reduction $red(D)$ of a digraph $\vec{D}$ obtained by iteratively (while possible) removing the arc (u, v) for any two adjacent vertices u and v such that there exists a directed path of length at least 2 from u to v. As an example, Figure 1d represents the transitive reduction $red(\vec{\mathcal{I}})$ of the precedence digraph $\vec{\mathcal{I}}$ (Fig. 1c) of $\mathcal{I}$ (Fig. 1b).

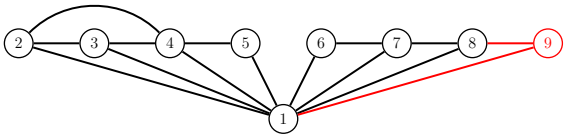
3.1 First approach: using the SP-digraphs

The relaxation $1|sp\text{-}graph|\sum w_jC_j$ can be solved in polynomial time [12]. This relation uses a precedences digraph in the form of a Series-Parallel digraph (SP-digraph). A digraph $\vec{D}$ is SP if it can be built recursively as follows. One single vertex is an SP-digraph. Given two SP-digraphs $\vec{D_1}$ and $\vec{D_2}$ with respective sources s_1 and s_2 and respective sinks k_1 and k_2:

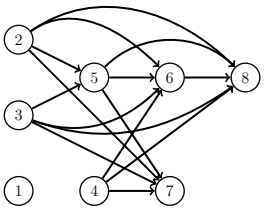
- $P(\vec{D_1}, \vec{D_2})$: the disjoint union of $\vec{D_1}$ and $\vec{D_2}$ is an SP-digraph (parallel operation);



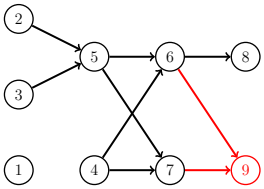
(a) The set of intervals in black and the set of intervals in black and red



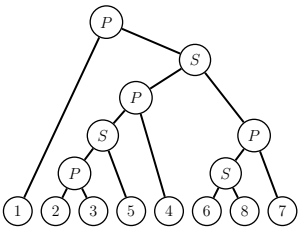
(b) $\mathcal{I}$ in black and $\mathcal{I}'$ in black and red



(c) Precedence digraph $\vec{\mathcal{I}}$ corresponding to $\mathcal{I}$



(d) $red(\vec{\mathcal{I}})$ in black and $red(\vec{\mathcal{I}}')$ in black and red



(e) Decomposition tree of the SP-digraph $red(\vec{\mathcal{I}})$

Figure 1 Two examples: a set of intervals (in black) corresponding to the digraph $\vec{\mathcal{I}}$ such that $red(\vec{\mathcal{I}})$ is an SP-digraph, and another set of intervals (in black and red) corresponding to the digraph $\vec{\mathcal{I}}'$ such that $red(\vec{\mathcal{I}}')$ is not an SP-digraph.

- $S(\vec{D_1}, \vec{D_2})$: the digraph obtained from the disjoint union of $\vec{D_1}$ and $\vec{D_2}$ by adding all arcs from k_1 to s_2 is a SP-digraph (series operation).

Any SP-digraph $\vec{D}$, can be assigned a binary rooted decomposition tree $tree(\vec{D})$ where there is a one-to-one mapping between the leaves of $tree(\vec{D})$ and the vertices of D and each internal node of $tree(\vec{D})$ either corresponds to a parallel (P) or series (S) operation. Several decomposition trees may be associated to a same SP-digraph. For instance, the Figure 1e represents a decomposition tree of the SP-digraph $red(\vec{\mathcal{I}})$ in Figure 1d.

In what follows, we extend the series composition to more than two digraphs by setting $S(\vec{D_1}, \ldots, \vec{D_p}) = S(\vec{D_1}, S(\vec{D_2}, \ldots, \vec{D_p}))$. Given a set of vertices X, we also set $P(X)$ to be the pairwise parallel composition of the vertices of X.

The name SP-digraphs [12] can be confusing. Indeed, on the one hand, Series-Parallel undirected graphs is a well studied graph class in graph theory, but underlying undirected graphs of SP-digraphs are not undirected Series-Parallel graphs [3, 14]. On the other hand SP-digraphs are actually more related to cographs (another graph class which is well studied in graph theory due to its relationship with clique-width, twin-width, modular decompositions, and graphs defined by induced subgraph obstructions since cographs are exactly P_4-free graphs) [3]. Precisely, it can be easily proved that, given a digraph D, $red(D)$ is SP if and only if the underlying graph of $tc(D)$ is a cograph [4, 13].

The relaxation using SP-digraphs is polynomial, however, there are interval graphs $\mathcal{I}$ such that $red(\vec{\mathcal{I}})$ is not SP. For instance, the digraph $\mathcal{I}'$ of Figure 1b (black and red) leads to the precedence digraph $red(\vec{\mathcal{I}}')$ of Figure 1d (black and red) which is not an SP-digraph. Indeed, the (undirected) path $(7, 9, 6, 8)$ is an induced P_4 in the underlying undirected graph

Algorithm 1 Heuristic 1

Data: the tasks T_j (corresponding to an interval graph $\mathcal{I}$) ordered by non decreasing release dates $r_i < r_j$ $(1 \leq i \leq j \leq n)$
Result: an SP-sub-digraph of $red(\vec{\mathcal{I}})$
$Sol = \{(path_1 = (T_1), deadline_1 = d_1)\}$
for $i = 2$ **to** n **do**
  Let $j \leq |Sol|$ be the smallest integer such that $deadline_j \leq r_i$
  if j *exists* **then** add T_i at then end of $path_j$ and replace $deadline_j$ by d_i
  else add $(path_{|Sol|+1} = (T_i), d_i)$ to Sol
return *the disjoint union of the directed paths* $path_j$, *for* $1 \leq j \leq |Sol|$.

Algorithm 2 Heuristic 2 $\mathcal{A}$

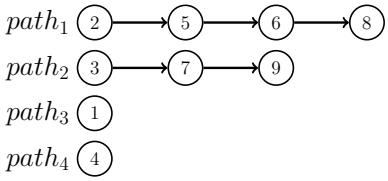
Data: an interval graph $\mathcal{I}$
Result: An SP sub-digraph of $red(\vec{\mathcal{I}})$
if $\mathcal{I}$ *is complete* **then** `// the tasks pairwise intersect`
  return $P(\mathcal{I})$; `// all the vertices of` $\mathcal{I}$ `are in parallel`
$K \leftarrow$ a minimal vertex separator; `// a selection criteria can be added`
Let $(C_1, \ldots, C_p)$ be the connected components of $\mathcal{I} \setminus K$ sorted by release dates
  `// for any` $k < k'$, $\max\{d_i \mid T_i \in C_k\} \leq \min\{r_j \mid T_j \in C_{k'}\}$
return $P(P(K), S(\mathcal{A}(C_1), \mathcal{A}(C_2), \ldots, \mathcal{A}(C_p)))$

G of $\vec{\mathcal{I}'}$ which implies that G is not a cograph, and so $red(\vec{\mathcal{I}'})$ is not an SP-digraph.

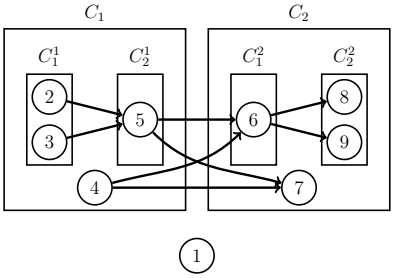
A first approach to compute a lower bound of $1|r_j; d_j|\sum w_j C_j$ is then to compute a SP-subgraph $\vec{\mathcal{I}'}$ of the precedence digraph $\vec{\mathcal{I}}$ of the interval graph $\mathcal{I}$ in order to use the algorithm to solve $1|sp\text{-}graph|\sum w_j C_j$. The goal is then to find the "best" SP-digraph possible in order to compute a "good" lower bound. A first criteria to look at is then to of course to minimize the number of lost arcs from $\mathcal{I}$. The following heuristics compute $\vec{\mathcal{I}'}$ from $\vec{\mathcal{I}}$.

Heuristic 1. (naive but very easy to implement and very fast algorithm) Recall that a stable set in a graph is a set of vertices that are pairwise non-adjacent. If X is a stable set of $V(\mathcal{I})$ (*i.e.*, the set of vertices of $\mathcal{I}$), then the tasks of X must be put in series (in order of the r_j). Partitioning the vertex-set of an interval graph $\mathcal{I}$ into a minimum number of stable sets can be done in linear time [9] (such a minimum number of stable sets equals the chromatic number of $\mathcal{I}$ and can be computed in polynomial time, *e.g.* , by a greedy colouring following the order of the vertices in the order of the r_j). Finally, the SP sub-digraph $\vec{\mathcal{I}'}$ is defined by putting in series the vertices in a same stable set and finally putting in parallel all stable sets. See Algorithm 1 and an example of its execution in Figure 2a.

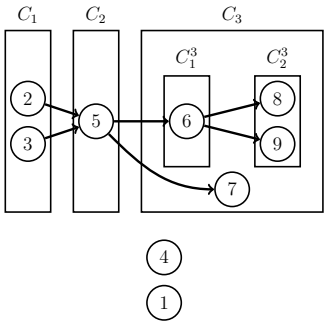
Heuristic 2. $\mathcal{A}$ (more evolved) It relies on the parallel structure of the minimal separators of $\mathcal{I}$. Recall that $K \subseteq V$ is a minimal separator if there exist $u, v \in V \setminus K$ such that all paths between u and v intersect K and K is minimal by inclusion. Note that, in any interval graph, any minimal separator induces a clique [5]. In an interval graph $\mathcal{I}$, let $(O_1 = L, O_2, \ldots, O_{p-1}, O_p = R)$ be the connected components of $\mathcal{I} \setminus K$. Roughly, the "left" part L is the connected component of $\mathcal{I} \setminus K$ containing the task with minimum release date and the "right" part R is the connected component of $\mathcal{I} \setminus K$ containing the task with maximum deadline. Finally, $(O_2, \ldots, O_{p-1})$ are connected components of simplicial vertices (their neighbourhoods induce a clique) adjacent to K, ordered in non-decreasing release date.



(a) Result of Algorithm 1 on the intervals described in Figure 1a.



(b) Result of Algorithm 2 on the intervals described in Figure 1a, with first separator $\{1\}$.



(c) Result of Algorithm 2 on the intervals described in Figure 1a, with first separator $\{1, 4\}$.

Figure 2 Examples of executions of Algorithm 1 and Algorithm 2.

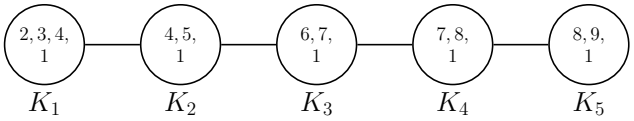
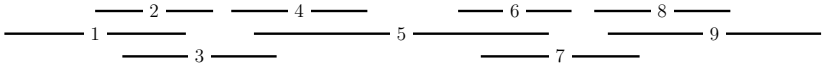


Figure 3 Clique path of the interval graph $\mathcal{I}'$ in Figure 1a.

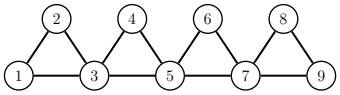
The recursive family of heuristics $\mathcal{A}$ consists first in well choosing a minimal separator K (for instance, balancing the sizes or weights of R and L) and then in proceeding recursively (on L and R). Precisely, $\mathcal{A}(\mathcal{I})$ returns $P(P(K), S(\mathcal{A}(L), \mathcal{A}(O_2), \ldots, \mathcal{A}(O_{p-1}), \mathcal{A}(R)))$, that is, $\mathcal{A}$ is applied recursively on each component of $\mathcal{I} \setminus K$, then composes their results in series "from left to right" and finally adds the tasks of K in parallel. See Algorithm 2 and an example of its execution in Figure 2b. More precisely, in the example, Algorithm 2 first considers the minimal separator $\{1\}$ (here, the separator is chosen in a balanced way, *i.e.*, , so that the size of R and L are as close as possible) and is applied recursively to both connected components $O_1 = R$ and $O_2 = L$ of $\mathcal{I}' \setminus \{1\}$. Then, in R (resp., in L), Algorithm 2 considers the separator $\{4\}$ (resp., $\{7\}$). Another example of execution is given in Figure 2c where the first considered separator is $\{1, 4\}$ and so $\mathcal{I}' \setminus \{1, 4\}$ has three connected components.

3.2 Second approach: toward Pseudo-SP-digraphs

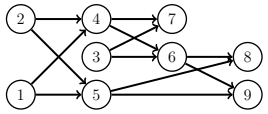
The advantage of previous heuristics is that they returned an SP-sub-digraph $\vec{\mathcal{I}}'$ of $\vec{\mathcal{I}}$ which allows to compute in polynomial time a solution of $1|sp\text{-}graph|\sum w_j C_j$ which provides a



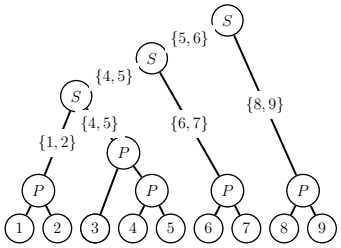
(a) A set of intervals



(b) The corresponding interval graph $\mathcal{I}$ and a clique path $CP = (\{1,2,3\},\{3,4,5\},\{5,6,7\},\{7,8,9\})$.



(c) The Pseudo-SP digraph $D(\mathcal{I}, CP)$



(d) Decomposition Tree of the Pseudo-SP digraph $D(\mathcal{I}, CP)$.

Figure 4 An example of a set of intervals $\mathcal{I}$ such that $\vec{\mathcal{I}} = D(\mathcal{I}, CP)$ is Pseudo-SP.

lower bound for $1|r_j;d_j|\sum w_jC_j$. On the other hand, the computed precedences digraph $\vec{\mathcal{I}'}$ may lose many arcs compared with $\vec{\mathcal{I}}$ and so could lead to a poor lower bound. This section presents another approach that might provide better lower bounds. This approach relies on the definition of a new class of sub-digraphs of precedence digraphs of interval graphs which is actually a superclass of the one of SP-digraphs (this is where we hope that using these new sub-digraphs, we may obtain better lower bounds).

What follows relies on representations of interval graphs as paths of maximal cliques. Recall that a vertex-set $X \subseteq V$ induces a maximal clique if the vertices of X are pairwise adjacent (*i.e.*, , X induces a complete graph) and no superset Y of X induces a complete graph. To any interval graph $\mathcal{I}$, it can be assigned a clique-path $(K_1, \ldots, K_\ell)$ such that K_i is a maximal clique of $\mathcal{I}$; $\mathcal{I} = \mathcal{I}[\bigcup_{i\leq\ell} K_i]$; and, for every $v \in V(\mathcal{I})$ and $i \leq k \leq j$, $v \in K_i \cap K_j$ implies that $v \in K_k$. Note that, in that case, $K_i \cap K_{i+1}$ induces a minimal separator for all $1 \leq i < \ell$. Moreover, note that an interval graph $\mathcal{I}$ may have several clique-paths and that a clique-path of $\mathcal{I}$ can be computed in linear time. An example is given in Figure 3.

Given an interval graph $\mathcal{I}$ and a clique path $CP = (K_1, \ldots, K_\ell)$ of it, let define the graph $D(\mathcal{I}, CP)$ on the same vertex-set and such that, for every $1 \leq i < \ell$, there is an arc from every vertex $u \in K_i \setminus K_{i+1}$ to any vertex $v \in K_{i+1} \setminus K_i$. An example is given in Figure 4c.

Note that, for any interval graph $\mathcal{I}$ with clique-path CP, $D(\mathcal{I}, CP)$ is a subgraph of $red(\vec{\mathcal{I}})$ but there are examples where $red(\vec{\mathcal{I}})$ and $D(\mathcal{I}, CP)$ may be different even if $D(\mathcal{I}, CP)$ is an SP digraph (see $\mathcal{I}'$ in Figure 1d, $D(\mathcal{I}', CP)$ (where CP is the clique-path of Figure 3) can be obtained from $red(\vec{\mathcal{I}'})$ by removing the arc from the task 6 to the task 9). To go further in that direction, let us define the new following digraph class.

A digraph D is a Pseudo-Serie-Parallel digraph (PSP-digraph) if it can be built recursively as follows. A vertex is a PSP-digraph. Given two PSP-digraphs D_1 and D_2 with respective sources s_1 and s_2 and respective sinks k_1 and k_2:

- the disjoint union of D_1 and D_2 is a PSP digraph (parallel operation);

- the digraph obtained from the disjoint union of D_1 and D_2 by adding all arcs from $X \subseteq W_1$ to $Y \subseteq S_2$ is a PSP-digraph (pseudo-series operation).

There is a connection between interval graphs and PSP-digraphs. We can prove that:

▶ **Lemma 1.** *For any interval graph $\mathcal{I}$ with clique-path CP, $D(\mathcal{I}, CP)$ is a PSP digraph.*

It is easy to prove that $D(\mathcal{I}, CP)$ is a sub-digraph of $\vec{\mathcal{I}}$ and so, an optimal solution of $1|PSP - diagraph|\sum w_jC_j$ is a lower bound of $1|r_j; d_j|\sum w_jC_j$.

To any PSP-digraph D, we can assign some binary rooted tree $tree(D)$ (a decomposition tree) where there is a one-to-one mapping between the leaves of $tree(D)$ and the vertices of D and each internal node of $tree(D)$ either corresponds to a parallel or to a pseudo-series operation (labelled with the corresponding sets X and Y). An example is given in Figure 4d where, for the series-node (labeled with S), the label of the edge going to its left (resp., right) child represents the set X (resp., the set Y) such that all arcs are added from X to Y. A PSP-digraph may have several decomposition trees.

Let $tree(H)$ be a decomposition tree of a PSP-digraph $H = (V, A)$. We define a new transformation in order to strengthen the relationship between PSP-digraphs and interval graphs. Let $F(H)$ be the result of the following recursive algorithm (which is simply a bottom up dynamic programming on the nodes of $tree(H)$).

- if $tree(H)$ is a single leaf, which corresponds to $v \in V = \{v\}$, then let $F(tree(H)) = (\{v\})$;
- otherwise, let ro be the root of $tree(H)$ and let l and r be the left and right children of ro. Let $(D_l, S_l, tree(H)_l)$ and $(D_r, S_r, tree(H)_r)$ be the Pseudo-SP obtained from the subtrees $tree(H)_l$, $tree(H)_r$ of $tree(H) \setminus ro$ rooted respectively in l and r. Let $F(tree(H)_l) = (K^l_1, \ldots, K^l_{m^l})$ and $F(tree(H)_r) = (K^r_1, \ldots, K^r_{m^r})$. Let $tree(H)'_l \subseteq tree(H)_l$ and $S'_r \subseteq S_r$ be the labels of the edges between ro and respectively l and r. Let $F(tree(H)) = (K^l_1 \cup (S_r \setminus S'_r), \ldots, K^l_{m^l} \cup (S_r \setminus S'_r), K^r_1 \cup (tree(H)_l \setminus tree(H)'_l), \ldots, K^r_{m^r} \cup (tree(H)_l \setminus tree(H)'_l))$. If $tree(H)'_l \cup S'_r = \emptyset$ (parallel composition), then let us remove the clique $K^r_1 \cup (tree(H)_l \setminus tree(H)'_l)$.

▶ **Lemma 2.** *For any decomposition tree $tree(H)$ of a PSP digraph H, then $F(tree(H))$ is a path of cliques of an interval graph $\mathcal{I}_H$ defined on the same set of vertices as H. Moreover, $D(\mathcal{I}_H, F(tree(H))) = H$.*

The natural next question is to know whether $1|PSP-diagraph|\sum w_jC_j$ can be computed in polynomial time. The strong relationship between interval graphs and PSP-digraphs, and the fact that SP-digraphs is a subclass of PSP-digraphs where our scheduling problem is polynomial-time solvable let us hope that the algorithm of Lawler [12] (for SP-digraphs) may possibly be extended for PSP-digraphs. Our future work will consist in designing a polynomial-time exact algorithm (or to prove that this problem is $\mathcal{NP}$-complete).

4 Conclusion and prospects

To conclude, we have identified a second polynomial relaxation for the COMPLETION constraint. This relaxation is not preemptive and requires a SP precedences digraph to compute a lower bound on the weighted flowtime. As, it is $\mathcal{NP}$-hard to keep all the information from the time windows, we proposed two heuristics for building a SP precedences digraph from the time windows. We also proposed an extension of the SP-digraphs that can capture more information from the time windows, but the complexity of the scheduling problem is unknown.

The short-term prospect is to integrate the two heuristics into the COMPLETION constraint and to compare their performances with the original relaxation used in the COMPLETION

constraint. The second short-term prospect is to answer the open question about the complexity of our extension of the PSP-digraphs. Last, the long-term prospect is to design filtering algorithms for the time windows of the tasks.

References

1. Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson. On the approximability of single-machine scheduling with precedence constraints. *Math. Oper. Res.*, 36(4):653–669, 2011. doi:10.1287/moor.1110.0512.
2. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2001. doi:10.1007/978-1-4615-1479-4.
3. Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999. URL: https://books.google.ca/books?id=vt5lhYqCYLMC&l.
4. Christophe Crespelle and Christophe Paul. Fully dynamic recognition algorithm and certificate for directed cographs. *Discret. Appl. Math.*, 154(12):1722–1741, 2006. doi:10.1016/j.dam.2006.03.005.
5. G. A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25:71–76, 1961.
6. Christian Fiegl and Carsten Pontow. Online scheduling of pick-up and delivery tasks in hospitals. *Journal of Biomedical Informatics*, 42(4):624–632, 2009. URL: https://www.sciencedirect.com/science/article/pii/S1532046409000239, doi:10.1016/j.jbi.2009.02.003.
7. Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming – CP'99*, volume 1713, pages 189–203, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. doi:10.1007/978-3-540-48085-3_14.
8. Ravindra Gokhale and M. Mathirajan. Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *The International Journal of Advanced Manufacturing Technology*, 60(9):1099–1110, June 2012. doi:10.1007/s00170-011-3653-3.
9. Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.
10. Gero Greiner, Tim Nonner, and Alexander Souza. The bell is ringing in speed-scaled multiprocessor scheduling. *Theory of Computing Systems*, 54(1):24–44, January 2014. doi:10.1007/s00224-013-9477-9.
11. András Kovács and J. Christopher Beck. A global constraint for total weighted completion time for unary resources. *Constraints*, 16(1):100–123, 2011. doi:10.1007/s10601-009-9088-x.
12. E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 75–90. Elsevier, 1978. URL: https://www.sciencedirect.com/science/article/pii/S0167506008703236, doi:10.1016/S0167-5060(08)70323-6.
13. L. Stewart and University of Toronto. Dept. of Computer Science. *Cographs : a Class of Tree Representable Graphs*. Technical report (University of Toronto. Department of Computer Science). University of Toronto, Department of Computer Science, 1978. URL: https://books.google.fr/books?id=8sL_MwEACAAJ.
14. Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, page 1–12, New York, NY, USA, 1979. Association for Computing Machinery. doi:10.1145/800135.804393.