ELSEVIER

European Journal of Operational Research 142 (2002) 435–443

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Invited Review

# Review of properties of different precedence graphs for scheduling problems

Jacek Blazewicz [a,*], Daniel Kobler [b]

[a] *Instytut Informatyki, Politechnika Poznanska and Instytut Chemii Bioorganicznej, PAN, Poznan, Poland*
[b] *Department of Mathematics, EPFL, CH-1015 Lausanne, Switzerland*

Received 17 September 2001; accepted 17 October 2001

**Abstract**

Precedence constraints are a part of a definition of any scheduling problem. After recalling, in precise graph-theoretical terms, the relations between task-on-arc and task-on-node representations, we show the equivalence of two distinct results for scheduling problems. Furthermore, again using these links between representations, we exhibit several new polynomial cases for various problems of scheduling preemptable tasks on unrelated parallel machines under arbitrary resource constraints.


*Keywords:* Scheduling; Precedence constraints; Graph theory

## 1. Introduction

Scheduling problems are formulated in different contexts, computer systems, manufacturing and project scheduling being the most representative [2,4,6,7,18]. An important part of a definition of any scheduling problem consists in the precedence constraints among tasks to be processed by processors (machines). Arising in different areas, these constraints are used to represent a technological order (manufacturing), activity precedences (project scheduling) or parallel and sequential parts in computer programs. It is known that two basic directed graph representations may be used to depict precedence constraints. They are, respectively, task-on-arc graphs, where arcs correspond to tasks and nodes represent time events in a schedule, and task-on-node graphs, where nodes correspond to tasks and arcs reflect precedence constraints. The first, task-on-arc representation, is mainly used in the project scheduling context [24,26], while the second, task-on-node representation, is used to a wider extent in manufacturing and computer scheduling problems [4,6,7,18].

The aim of the paper is to point out relationships between results for scheduling problems that turn out to be equivalent, and give some new results for scheduling preemptable tasks on parallel machines. In order to achieve this, we recall in precise graph-theoretical terms the relationship

* Corresponding author. Tel.: +48-61-8790-790; fax: +48-61-771-525.
*E-mail address:* blazewic@put.poznan.pl (J. Blazewicz).

0377-2217/02/$ - see front matter 
PII: S0377-2217(01)00379-4

between the two standard representations of task precedence constraints. Although this relationship has been studied for a long time, it is, in our opinion, not taken enough into account, as we will show in this paper. As a result of these studies we will also prove that problem $Rm|pmtn, res\ldots, interval\ order|C_{\max}$ (as denoted with the use of the well-known notation of scheduling problems [5,10]) is solvable in polynomial time.

Before doing this let us set up the subject more precisely. Usually in a scheduling problem we are given a set of (parallel) processors $\mathscr{P}$ and a set of tasks $\mathscr{T}$ together with precedence constraints $\prec$ (a partial order). Sometimes a set of additional resources $\mathscr{R}$, tasks are competing for, is also given. Tasks may be preemptable or not and are to be processed on processors in a certain order optimizing a given criterion. This order must obey relation $\prec$, where $T_i \prec T_j$ means that task $T_j$ may be assigned to a processor only after task $T_i$ is finished. In a standard way, the task precedence constraints $\prec$ of a scheduling problem are represented by a directed graph. A directed graph (or digraph) is said to be acyclic if it does not contain any directed cycle (circuit). In particular, loops on vertices are also forbidden. We will consider set $\mathscr{G}$ of acyclic digraphs (several parallel arcs from a vertex to another one being allowed) and one of its subsets, both used to represent precedence constraints. To simplify the reading, we will use the term 'graph' to mean 'directed graph'. The term 'path' denotes a 'directed path'.

The paper is organized as follows. In Section 2 we consider in a formal way the two main methods for representing task precedence constraints by a graph. In Section 3, links between the two representations are discussed. A thorough analysis allows one to show that problem $Rm|pmtn, res\ldots, interval\ order|C_{\max}$ is solvable in polynomial time. We conclude in Section 4.

## 2. Precedence constraints representations

### 2.1. The task-on-node representation

The most common representation of precedence constraints used in the literature is the *task-on-node graph representation* (or *vertex diagram*). The corresponding graph $H$ has a vertex for each task, and an arc (directed) from $T_i$ to $T_j$ if and only if $T_i$ has to precede $T_j$ according to $\prec$. If graph $H$ has a circuit, the corresponding scheduling problem has clearly no feasible solution; therefore we may suppose that $H$ is acyclic. In order to reduce redundant information, transitivity arcs in $H$ are not represented; this is the *transitive reduced* representation.

Let us denote by $\mathscr{G}_n$ the subset of $\mathscr{G}$ of graphs without parallel arcs, without loops and without transitivity arcs. The graphs used for the task-on-node representation are members of $\mathscr{G}_n$. Moreover, every $H$ in $\mathscr{G}_n$ can be seen as a task-on-node representation of precedence constraints of some scheduling problem. If dummy tasks are not used, then precedence constraints $\prec$ correspond exactly to one graph $H$ in $\mathscr{G}_n$ (up to isomorphism).

If introducing dummy tasks (with processing time 0) is allowed, then several graphs can be associated to $\prec$. Let us define the following operations on graphs in $\mathscr{G}_n$:

*task-splitting*: replace task $T_i$ by two tasks $T_i'$ and $T_i''$ and an arc $(T_i', T_i'')$. One of these two tasks is the original task $T_i$, and the other is a dummy task. Each arc that was entering (resp. leaving) $T_i$ is replaced by an arc entering $T_i'$ (resp. leaving $T_i''$).

*fusion*: if there are two sets of tasks $V_1$ and $V_2$ such that there is an arc from each task in $V_1$ to each task in $V_2$, add a vertex $v$ (corresponding to a dummy task) and replace all arcs from $V_1$ to $V_2$ by all possible arcs from $V_1$ to $v$ and from $v$ to $V_2$.

The fusion operation will be called *arc-splitting* when $|V_1| = |V_2| = 1$. To these three operations correspond natural reverse operations.

All these operations, and others we did not mention here, transform graph $H \in \mathscr{G}_n$ into another graph $H' \in \mathscr{G}_n$. The two graphs are equivalent from a scheduling point of view: they represent (essentially) the same task precedence constraints. In this case we will say that they are *s-equivalent*. But from a graph theoretical point of view, they are not equivalent (that is, isomorphic). Considering some property of the task-on-node graphs that allows a polynomial resolution of some scheduling problems, these operations may be useful. If the

task-on-node graph $H$ does not have this property, an s-equivalent graph having it can perhaps be obtained through these operations.

Notice that there exist other operations that allow to obtain s-equivalent graphs. Some of these operations change the number of non-dummy tasks, for example by replacing a task by two shorter ones.

### 2.2. The task-on-arc representation

Another representation of task precedence constraints that can be used is the *task-on-arc graph representation* (or *arc diagram*). The corresponding graph $G$ has an arc for each task, the vertices being time events. If task $T_i$ precedes task $T_j$, there must be a path from the terminal endpoint of arc $T_i$ to the initial endpoint of arc $T_j$. For the same reason as in the task-on-node representation, we may suppose that graph $G$ has no circuit. But here parallel arcs are allowed. Therefore graphs used for task-on-arc representation are members of $\mathscr{G}$. And every $G$ in $\mathscr{G}$ can be seen as a task-on-arc representation of the precedence constraints of some scheduling problem. On the contrary to the task-on-node representation, several task-on-arc graphs may correspond to a set of precedence constraints $\prec$, even if no dummy tasks are allowed. A simple such example is given in Fig. 1 where the two graphs represent the same precedence constraints.

A vertex is said to be a *source* (resp. a *sink*) if it has no incoming (resp. outgoing) arc. Let us define the following operations:

*source-* (*sink-*) *splitting*: replace a source (resp. sink) $v$ by two sources (resp. sinks) $v'$ and $v''$. Each arc that was leaving (resp. entering) $v$ is replaced by an arc leaving (resp. entering) either $v'$ or $v''$, the other endpoint remaining unchanged.

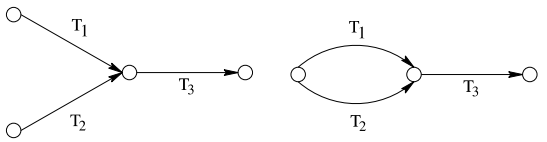


Fig. 1. Two different representations of the constraints $\{T_1 \prec T_3, T_2 \prec T_3\}$.

The reverse operations are called *source-* (*sink-*) *merging*. Notice that adding and removing isolated vertices can be seen as source-splitting and source-merging (or even sink-splitting and sink-merging).

These operations transform a graph $G \in \mathscr{G}$ into another graph $G' \in \mathscr{G}$. These graphs are s-equivalent, but not isomorphic. In fact, if two graphs $G$ and $G'$ are s-equivalent, and no dummy tasks are allowed, we can transform $G$ into $G'$ (and $G'$ into $G$) with these operations. In other words, the task-on-arc representation is unique up to the number of sources, sinks and isolated vertices (and up to isomorphism) [1,16]. Among all the graphs that can be obtained from a graph $G$ by using these operations, we will distinguish one of them. The graph $G_m$ is obtained from $G$ by removing all isolated vertices and merging all sources (sinks) into one source (sink). The graph $G_m$ can be seen as the result of iterative source- (sink-) merging applied to $G$.

If dummy tasks are allowed, there exist further operations that give s-equivalent graphs. For example:

*vertex-splitting*: replace vertex $v$ by two vertices $v'$ and $v''$, and an arc $(v', v'')$ corresponding to a dummy task. Each arc that was entering (resp. leaving) $v$ is replaced by an arc entering $v'$ (resp. leaving $v''$).

Here also, all the operations might allow to transform a graph into an s-equivalent graph that has a desired graph theoretical property allowing a polynomial resolution of a scheduling problem.

### 2.3. Links between the two representations

It is well known that every precedence constraints $\prec$ has a task-on-node representation, and that this is not true for task-on-arc representation if dummy tasks are not allowed. Indeed, the following constraints on the four tasks $T_1, \ldots, T_4$ cannot be represented with a task-on-arc graph (without dummy tasks):

$$T_1 \prec T_3, \quad T_1 \prec T_4, \quad T_2 \prec T_3.$$

But when dummy tasks are allowed, every set of precedence constraints can be represented with a task-on-arc graph. The relationship between the two representations can be explained through the

notion of adjoint (also called line digraph or directed line-graph) [3,11,22].

For graph $H = (A, U)$ and vertex $a \in A$, we denote by $N^+(a)$ (resp. $S(a)$) the set of immediate successors (resp. successors) of $a$:

$$N^+(a) = \{b \in A \,|\, (a,b) \in U\}$$

and

$$S(a) = \{b \in A \,|\, \text{there is a path from } a \text{ to } b \text{ in } H\}.$$

The following theorems give characterizations of the adjoints.

**Theorem 1** [3]. *A graph is an adjoint if and only if for every pair of vertices a and b the following is true*:

$$N^+(a) \cap N^+(b) \neq \emptyset \;\Rightarrow\; N^+(a) = N^+(b).$$

**Theorem 2** [12]. *An acyclic graph G in $\mathscr{G}_n$ is an adjoint if and only if G has no induced subgraph isomorphic to N* (*see Fig. 2*).

In Theorem 2, the fact that $G$ is in $\mathscr{G}_n$ is important. If $G$ is not transitively reduced, this result does not hold, as shown by the graph $G = (\{a, b, c\}, \{(a,b),(b,c),(a,c)\})$. For an adjoint $H$, finding a graph $G$ such that $H$ is the adjoint of $G$ can easily be done in polynomial time [3,21].

It is well known that a task-on-node representation of precedence constraints given by a task-on-arc graph can be obtained by taking the adjoint of this graph. In other words, if $G \in \mathscr{G}$ and $H \in \mathscr{G}_n$ is the adjoint of $G$, then $G$ and $H$ are s-equivalent. According to Theorem 2, every task-on-node graph without an induced subgraph isomorphic to $N$ (a so-called *N-free graph*) is an adjoint. Therefore, every partial order $\prec$ whose task-on-node graph is $N$-free (and only such a $\prec$) has a task-on-arc representation without dummy tasks. It is easy to determine a graph whose adjoint is a given graph [3,21].

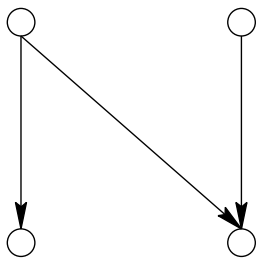

Fig. 2. Graph $N$.

Obviously, not every task-on-node graph $H$ is an adjoint. But, as also explained in [16] for example, it is easy to find a graph $H'$ which is s-equivalent to $H$ and is an adjoint; such a $H'$ can be obtained by performing arc-splitting on each arc of $H$.

Knowing this relationship between the two representations, one can consider either type of a graph. In particular, when polynomial cases of scheduling problems are investigated, properties about the precedence constraints can be expressed with respect to either two representations. But despite the fact that many researchers consider this relationship as very well known, this knowledge is still not always applied, as we will now show.

## 3. Polynomially solvable scheduling problems

In this section we will present a polynomial time approach to precedence constrained scheduling problems under arbitrary resource constraints. Although some elements of this approach are known for several years [4,25], no paper explored it yet to a full extent. This is especially true for the *task-on-node representation* which will be considered as the second one.

Assume now that tasks $T_1, T_2, \ldots, T_m$ are to be processed preemptively on $m$ parallel processors $P_1, P_2, \ldots, P_m$ in order to minimize schedule length. Assume first that these processors are identical, i.e. each task $T_j,\ j = 1, 2, \ldots, n$ is characterized by its processing time $p_j$. There is also given a set of additional resources $\mathscr{R} = \{R_1, R_2, \ldots, R_s\}$ with respective resource limits $m_1, m_2, \ldots, m_s$. Resource requirements of task $T_j,\ j = 1, 2, \ldots, n$, are specified by resource requirement vector $\mathbf{R}(T_j) = (R_1(T_j), R_2(T_j), \ldots, R_s(T_j))$, each component of which denotes a requirement for a particular resource type, $R_k(T_j) \leqslant m_k,\ k = 1, 2, \ldots, s$. Let the set of tasks $\mathscr{T}$ be partially ordered by precedence constraints relation $\prec$ expressed in a form of a *task-on-arc graph* $G \in \mathscr{G}$ (cf. Fig. 3).

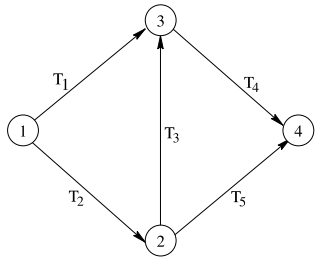


Fig. 3. An example task-on-arc graph $G$.

Let us assume now, that nodes (being in fact time events in a schedule) of graph $G$ are ordered in such a way that an occurrence of node $i$ is not later than an occurrence of node $j$, if $i < j$. Now, by *main set* $\mathscr{M}_i$, $i = 1, 2, \ldots, n-1$, we will denote a set of tasks which could be processed, from the viewpoint of precedence constraints only, between the occurrence of node $i$ and $i+1$. (In case of graph $G$ in Fig. 3, three such main sets can be distinguished: $\mathscr{M}_1 = \{T_1, T_2\}$, $\mathscr{M}_2 = \{T_1, T_3, T_5\}$, $\mathscr{M}_3 = \{T_4, T_5\}$.) By a *resource feasible set* we will mean here such a subset of any main set, for which total processor and resource requirements of tasks comprising it do not exceed processor and resource limits (respectively for any resource component). Let $K$ be a number of different resource feasible sets. By variable $y_i$ the processing time of the $i$th resource feasible set, and by $\mathscr{Q}_j$ the set of indices of only those resource feasible sets that contain task $T_j \in \mathscr{T}$, will be denoted. Now, our scheduling problem may be formulated as the following linear programming (LP) one:

$$\text{Minimize} \quad \sum_{i=1}^{k} y_i \tag{1}$$

$$\text{subject to} \quad \sum_{i \in \mathscr{Q}_j} y_i = p_j, \quad j = 1, 2, \ldots, n, \tag{2}$$

$$y_i \geqslant 0, \quad i = 1, 2, \ldots, k. \tag{3}$$

As a solution of the above problem one gets optimal values $y_i^*$ of interval lengths in an optimal schedule. The tasks processed in the intervals are members of the corresponding resource feasible subsets.

Let us analyze now, the conditions under which a solution of the above LP problem gives an optimal schedule. We see that it depends on an order of nodes of graph $G$ and an optimal schedule can be constructed in the above way if this order is *unique*. Graph $G$ with a unique ordering of nodes is called *a uniconnected activity network* [25], *uan* for short. (In fact graph $G$ in Fig. 3 is a uniconnected activity network.) To analyze a complexity of this approach, one should calculate a number of variables in the above LP problem. We see that it depends polynomially on the input length (of the scheduling problem), if the number of processors $m$ is fixed. (Note that additional resources can only decrease a number of LP variables.) In such a case one may use a non-simplex algorithm [15] or [14] which solves LP problem in time polynomial in the number of variables and constraints. Thus, using a notation of scheduling problems (cf. [5,10]), one may conclude that problem $Pm|pmtn, res \ldots, uan|C_{\max}$ is solvable in polynomial time.

Let us now consider the case of parallel, unrelated processors. In such a model each task $T_j$ is characterized by a processing time vector $\mathbf{p_j} = [p_{j1}, p_{j2}, \ldots, p_{jm}]^{\mathrm{T}}$, where $p_{ji}$ denotes a processing time of tasks $T_j$ on processor $P_i$ (provided that all the required additional resources are granted). In this case one may formulate a similar LP problem to (1)–(3), but now for each resource feasible set one must consider several *processing modes* (each task being assigned to different processors). Eq. (2) also changes a form and the summation is done over a normalized time, so that the right-hand side is equal to 1 (or 100%) instead of $p_j$. Again, we see that a number of LP variables (different processing modes) is bounded from above by a polynomial in the input length of the scheduling problem if a number of processors $m$ is fixed. Thus, in this case, one may conclude that problem $Rm|pmtn, res \ldots, uan|C_{\max}$ is solvable in polynomial time.

We see that a property of precedence constraints allowing for polynomial solvability is called *uniconnectedness* and can be defined equivalently as follows: an activity network (task-on-arc graph) is said to be a uniconnected activity network (*uan* for short) if for every pair of vertices $v$ and $w$, there is a path from $v$ to $w$ or from $w$ to $v$ (but not both since we deal with acyclic graphs only).

We will first show that being uniconnected is equivalent to having a Hamiltonian path.

**Theorem 3.** *Let G be an activity network (task-on-arc graph). G is uniconnected if and only if G has a Hamiltonian path.*

**Proof.** We first show the 'if' part. Let $v$ and $w$ be two vertices of $G$. It is clear that if $G$ has a Hamiltonian path, then there is a path from $v$ to $w$ or from $w$ to $v$ (simply follow the Hamiltonian path). The graph, being acyclic, cannot have both paths.

We now prove the 'only if' part. Consider a longest path $P$ (in the sense of a number of nodes) in $G$. Let $v_1$ be its first vertex and $v_p$ its last. Notice that since the graph is acyclic, such a path is simple and can be found in polynomial time. Assign to each vertex $w$ value $r(w)$ of the length of a longest path from $v_1$ to $w$. If there is no path from $v_1$ to $w$, set $r(w) = \infty$. Again, all longest paths are simple and the values $r(w)$ can be determined in polynomial time. Notice that $r(v_1) = 0$.

We first show that vertex $w$ with $r(w) = \infty$ cannot exist. If the graph is uniconnected, there is a path from $v_1$ to $w$ or from $w$ to $v_1$. But $v_1$ is the first vertex of $P$ and therefore has in-degree 0. Hence the path between $v_1$ and $w$ must be from $v_1$ to $w$. Therefore $r(w)$ is finite. Second, we notice that for each arc $(v, w)$ we have $r(v) < r(w)$, by definition of the values $r(\cdot)$ and by the absence of circuits.

Finally, we show that no vertex is outside of $P$. For, if such a vertex, say $x$, exists, consider a longest path $P_x$ from $v_1$ to $x$ and let $y$ be the last vertex that is both on $P$ and $P_x$. By definition of $P$, $y$ cannot be $v_p$. Let $v$ be the successor of $y$ in $P$ and $w$ the successor of $y$ in $P_x$. Then we have $r(v) = r(w)$. But this is not possible: there must be a path from $v$ to $w$ or from $w$ to $v$. As noticed before, the value of $r(\cdot)$ can only increase when following a path, hence the equality $r(v) = r(w)$ is a contradiction, and the vertex $x$ cannot exist. Therefore $P$ is a Hamiltonian path. □

Now let us turn our attention to the *task-on-node graphs*. To define an interesting class of graphs let us consider a finite set $V$ and a collection $(I_v)_{v\in V}$ of intervals $I_v$ on the reals. This collection defines a partial order $\prec$ on $V$ as follows:

$v \prec w \iff I_v$ is entirely before $I_w$.

Such a partial order is called an *interval order*. Without loss of generality, we may assume that the intervals have the form $[n_1, n_2)$ with $n_1$ and $n_2$ integral. It can be shown that $\prec$ is an interval order if and only if the transitive closure of the task-on-node representation of this order does not contain $2K_2$ (see Fig. 4) as an induced subgraph [8].

Interval orders are useful in scheduling, since several problems become polynomial when the precedence constraints have that form [13,17,19, 20]. In particular.

**Theorem 4** [19]. *Problem $Pm|pmtn, interval\ order|C_{\max}$ can be solved in polynomial time.*

The proof of Theorem 4 is based on the polynomial resolution of a linear programming program. To find more about interval orders, see for example [16].

Below we will show that the result of Theorem 4 is in fact a special case of the already presented polynomial-time approach for solving problem $Rm|pmtn, res\ldots, uan|C_{\max}$. Let us consider the following theorem.

**Theorem 5.** *If G is a uan, then G is a task-on-arc representation of an interval order.*

**Proof.** By Theorem 3, $G = (V, A)$ is composed of a Hamiltonian path $P = (v_1, \ldots, v_n)$ with possibly some additional arcs of the form $(v_i, v_j)$ with $i < j$. The interval order we are looking for is defined by the following collection of intervals $(I_a)_{a\in A}$. For every arc $a = (v_i, v_j)$ of $A$, we put the interval $[i, j)$ into the collection.

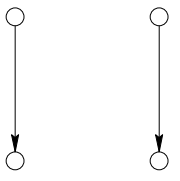

Fig. 4. Graph $2K_2$.

We have now to show that $I_a = [i, j)$ is entirely to the left of $I_{a'} = [i', j')$ if and only if $a$ has to precede $a'$ in the task precedence constraints represented by $G$. This is easy to show, since:

$I_a = [i, j)$ is entirely to the left of $I_{a'} = [i', j')$
$\iff$ $j \leqslant i'$
$\iff$ there is a path from $v_j$ to $v_{i'}$ in $G$ (along $P$)
$\iff$ $a$ with head $j$ has to precede $a'$ with tail $i'$. □

If dummy tasks are not allowed, an interval order does not necessarily have a task-on-arc representation. Indeed, if we consider the collection of intervals $\{[1,2),[1,3),[2,4),[3,4)\}$, its task-on-node representation is graph $N$ in Fig. 2. According to the remark following Theorem 2, it implies that this partial order does not have a task-on-arc representation without dummy tasks. But the equivalence of task-on-node and task-on-arc representations can be obtained through the use of dummy tasks. Since we allow them also here the following result can be proved.

**Theorem 6.** *Any interval order has a task-on-arc representation with a Hamiltonian path* (*and therefore corresponds to a uan*).

**Proof.** Consider any collection of intervals $(I_a)_{a\in A}$ with $I_a = [b_a, e_a)$. We define the following graph $G = (V, E)$. Set

$$V = \{b_a \mid a \in A\} \cup \{e_a \mid a \in A\}.$$

For any $v$ in $V$, let $next(v)$ be the vertex $w > v$ such that there is no $x$ in $V$ with $v < x < w$ ($next(v)$ is not defined for the largest $e_a$). Set

$$A' = \{(v, next(v)) \mid v \in V \text{ and } next(v) \text{ defined}\}$$

and

$$E = A' \cup \{(b_a, e_a) \mid a \in A\}.$$

The arcs in $A'$ represent dummy tasks. This graph $G$ has indeed a Hamiltonian path, starting with the smallest $b_a$ ($\min_{a\in A} b_a$), following the arcs in $A'$ and ending at the largest $e_a$ ($\max_{a\in A} e_a$). It remains to show that $I_a = [b_a, e_a)$ is entirely to the left of $I_{a'} = [b_{a'}, e_{a'})$ if and only if arc $(b_a, e_a)$ has to precede arc $(b_{a'}, e_{a'})$ in the task precedence constraints represented by $G$. We do not have to deal with arcs in $A'$ since they represent dummy tasks:

$I_a = [b_a, e_a)$ is entirely to the left of $I_{a'} = [b_{a'}, e_{a'})$
$\iff$ $e_a \leqslant b_{a'}$
$\iff$ there is a path from $e_a$ to $b_{a'}$ in $G$ (using the arcs in $A'$)
$\iff$ $(b_a, e_a)$ with head $e_a$ has to precede $(b_{a'}, e_{a'})$ with tail $b_{a'}$. □

The following corollary is a direct consequence of Theorems 5 and 6:

**Corollary 1.** *Let $\mathcal{O}$ be a partial order. If dummy tasks are allowed, $\mathcal{O}$ is an interval order if and only if $\mathcal{O}$ can be represented as a uan.*

Syslo proved a similar result in that direction in [23], but not in relation to uan. Although the statement of his Theorem 2 is a little bit misleading, he showed that a partial order $\mathcal{O}$ is an interval order if and only if its canonical arc diagram (a particular arc diagram representing $\mathcal{O}$) has a Hamiltonian path (of a specific form). [1]

Using the above corollary, we now have:

**Theorem 7.** *Problem $Rm|pmtn, res \ldots, interval\ order|C_{\max}$ is solvable in polynomial time.*

This result shows the importance of taking into account the relationship between task-on-node and task-on-arc representations of precedence constraints, and its impact on previous results.

As mentioned in Section 2.1, the task-on-node graph is usually given by its transitive reduced form. But the characterization of interval orders ($2K_2$-free) applies to the transitive closure of the task-on-node graph. Therefore, a description of transitive reduced form of the task-on-node graph of an interval order is interesting. This problem has been considered in [9], based on results of [17].

[1] Let us notice at this point that the definition of canonical representation/arc diagram in [23] is based on two functions $l(p)$ and $r(p)$ whose definitions suffered from typos: it should be $N_p^- = M_{l(p)}$ and $N_p^+ = N_{r(p)}$ (instead of $N_p^- = N_{l(p)}$ and $N_p^+ = M_{r(p)}$).

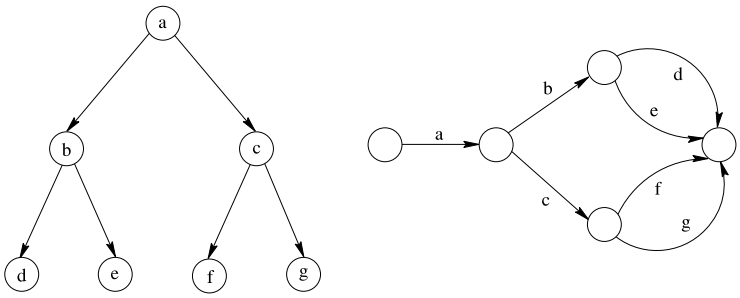


Fig. 5. An example of an out-tree (a) being not an interval order nor a uniconnected activity network (b).

**Definition.** An acyclic digraph $G = (V, A)$ is an *interval directed acyclic graph* (*dag*) if and only if for any pair of vertices $u$ and $v$, we have either $S(u) \subseteq S(v)$ or $S(v) \subseteq S(u)$.

**Proposition 1** [9]. *The transitively closed interval dags are exactly the task-on-node graphs of interval orders.*

Therefore, the adjoint of a uan is an interval dag (by Theorem 5). Moreover, it is easy to check that the adjoint of a uan is transitively reduced (by the properties of an adjoint). We could have used the characterization given in Proposition 1 to prove Theorems 5 and 6, but this would not simplify the proofs.

To this end let us also comment on some other classes of precedence graphs, well known in the literature. For example *directed caterpillars* are special cases of interval orders, and thus, such a structure results in polynomial time algorithm. On the other hand, *out-trees* (as well as *in-trees* and *series–parallel* precedence constraints) although for some simple cases equivalent to interval orders (and uan) in general differ from the latter, and thus, cannot be solved in polynomial time by the described approach (cf. Fig. 5).

## 4. Conclusion

Although the links between task-on-node and task-on-arc representations of precedence constraints of scheduling problems are considered as well known, it is useful to remember them. Indeed, in this paper we showed that the notions of a uniconnected activity network and an interval order are equivalent from the scheduling point of view. This allowed us in a first step to show that two distinct results from the literature are in fact equivalent. Moreover, we were also able to exhibit further polynomially solvable cases, especially problem $Rm|pmtn, res \ldots, interval\ order|C_{\max}$. We hope that this will illustrate the importance of a good understanding of the various classes of vertex and arc diagrams.

## Acknowledgements

The paper has been written while the second author stayed at Instytut Informatyki, Politechnika Poznanska whose support under KBN grant 8T11A01618 is gratefully acknowledged.

## References

[1] M. Aigner, On the linegraph of a directed graph, Mathematische Zeitschrift 102 (1967) 56–61.
[2] K. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[3] C. Berge, Graphs and Hypergraphs, North-Holland, Amsterdam, 1973.
[4] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer, Berlin, 1996.
[5] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Applied Mathematics 5 (1983) 11–24.
[6] P. Brucker, Scheduling Algorithms, Springer, Berlin, 1997.

[7] E.G. Coffman Jr. (Ed.), Scheduling in Computer and Job Shop Systems, Wiley, New York, 1976.
[8] P.C. Fishburn, Intransitive indifference in preference theory: A survey, Operations Research 18 (1970) 207–228.
[9] H.N. Gabow, A linear-time recognition algorithm for interval dags, Information Processing Letters 12 (1981) 20–22.
[10] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: A survey, Annals of Discrete Mathematics 5 (1979) 287–326.
[11] R.L. Hemmiger, L.W. Beineke, Line graphs and line digraphs, in: L.W. Beineke, R.J. Wilson (Eds.), Selected Topics in Graph Theory, Academic Press, London, 1978, pp. 271–305.
[12] C. Heuchenne, Sur une certaine correspondance entre graphes, Bulletin de la Sociéte Royal des Sciences de Liège 33 (1964) 743–753.
[13] K. Jansen, Analysis of scheduling problems with typed task systems, Discrete Applied Mathematics 52 (1994) 223–232.
[14] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373–395.
[15] L.G. Khachiyan, A polynomial algorithm for linear programming, Doklady Akademii Nauk SSSR 244 (1979) 1093–1096 (in Russian).
[16] R.H. Möhring, Computationally tractable classes of ordered sets, in: I. Rival (Ed.), Algorithms and Order, Kluwer Academic Publishers, Dordrecht, 1989, pp. 105–193.
[17] C.H. Papadimitriou, M. Yannakakis, Scheduling interval-ordered tasks, SIAM Journal of Computing 8 (1979) 405–409.
[18] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[19] N.W. Sauer, M.G. Stone, Preemptive scheduling of interval orders is polynomial, Order 5 (1989) 345–348.
[20] G. Steiner, Minimizing the number of tardy jobs with precedence constraints and agreeable due dates, Discrete Applied Mathematics 72 (1997) 167–177.
[21] M.M. Syslo, A labeling algorithm to recognize a line digraph and output its rootgraph, Information Processing Letters 15 (1982) 28–30.
[22] M.M. Syslo, A graph-theoretic approach to the jump-number problem, in: I. Rival (Ed.), Graphs and Orders, Reidel, Dordrecht, 1985, pp. 185–215.
[23] M.M. Syslo, The jump number problem on interval orders: A 3/2 approximation algorithm, Discrete Mathematics 144 (1995) 119–130.
[24] F.B. Talbot, J.H. Patterson, An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems, Management Science 24 (1978) 1163–1174.
[25] J. Weglarz, J. Blazewicz, W. Cellary, R. Slowinski, An automatic revised simplex method for constrained resource network scheduling, ACM Transactions of the Mathematical Software 3 (1977) 295–300.
[26] J. Weglarz (Ed.), Project Scheduling, Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Dordrecht, 1999.