SIAM J. COMPUT.
Vol. 33, No. 2, pp. 393–415


# SCHEDULING WITH AND/OR PRECEDENCE CONSTRAINTS*

ROLF H. MÖHRING†, MARTIN SKUTELLA‡, AND FREDERIK STORK§

**Abstract.** In many scheduling applications it is required that the processing of some job be postponed until some other job, which can be chosen from a pregiven set of alternatives, has been completed. The traditional concept of precedence constraints fails to model such restrictions. Therefore, the concept has been generalized to so-called AND/OR *precedence constraints* which can cope with this kind of requirement. In the context of traditional precedence constraints, feasibility, transitivity, and the computation of earliest start times for jobs are fundamental, well-studied problems. The purpose of this paper is to provide efficient algorithms for these tasks for the more general model of AND/OR precedence constraints. We show that feasibility as well as many questions related to transitivity can be solved by applying essentially the same linear-time algorithm. In order to compute earliest start times we propose two polynomial-time algorithms to cope with different classes of time distances between jobs.

**Key words.** project scheduling, AND/OR precedence constraints, earliest start schedule, mean payoff games

**AMS subject classifications.** 90B35, 05C65, 90C27, 91A43, 05C85

**DOI.** 10.1137/S009753970037727X

## 1. Introduction.

**Definition and motivation of AND/OR precedence constraints.** For a given set $V$ of jobs, a precedence constraint comprehends the requirement that a job $j$ cannot be started before another job $i$ has been completed. Precedence constraints are usually given by a set $A$ of ordered pairs $(i, j)$, $i \neq j \in V$, inducing an acyclic digraph $D = (V, A)$ where each node corresponds to a job and each arc represents a precedence constraint. In a feasible implementation of the project, the jobs have to be executed in accordance with the partial order defined by $D$. Since, in this setting, each job $j$ can only start after the completion of *all* its predecessors in $D$, we call these precedence constraints AND-*constraints*. However, there are many applications where jobs can be executed as soon as *any* of its predecessors has been completed; we refer to such temporal restrictions as OR-*constraints*. Traditional precedence constraints fail to model this requirement and consequently, the model has been generalized to so-called AND/OR *precedence constraints*. AND/OR precedence constraints can be represented by a set $\mathcal{W}$ of pairs $(X, j)$ with the meaning that job $j \in V$ cannot be executed before some job $i \in X \subseteq (V \setminus \{j\})$ has been completed. We

*Received by the editors August 23, 2000; accepted for publication (in revised form) January 18, 2003; published electronically February 18, 2004. This work was supported in part by the EU Thematic Networks APPOL I & II, Approximation and Online Algorithms (IST-1999-14084 and IST-2001-30012). Part of this research appeared as a two-page abstract [21] in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00).

http://www.siam.org/journals/sicomp/33-2/37727.html

†Technische Universität Berlin, Fakultät II, Institut für Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D-10623 Berlin, Germany (moehring@math.tu-berlin.de).

‡Max-Planck Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany (skutella@mpi-sb.mpg.de). This author was supported in part by DONET within the frame of the TMR Program (contract ERB FMRX-CT98-0202) while staying at C.O.R.E., Louvain-la-Neuve, Belgium, for the academic year 1998–1999.

§ILOG Deutschland GmbH, Ober-Eschbacher Straße 109, D-61352 Bad Homburg, Germany (fstork@ilog.de). This work was done while this author was at Technische Universität Berlin and received support from Deutsche Forschungsgemeinschaft (DFG), grant Mo 446/3-3.

call such $(X, j)$ pairs *waiting conditions* and job $j$ the *waiting job* for $(X, j)$. Notice that for a singleton $X = \{i\}$, the constraint $(X, j)$ is a traditional AND-constraint $(i, j)$.

An intuitive motivation for AND/OR precedence constraints is noted by Gillies and Liu [12]. An engine head has to be fixed by four bolts. However, one of the bolts may secure the engine head well enough to allow further work on it. If the set $X$ consists of the four jobs to secure the bolts and $j$ represents the further work on the engine head, then the waiting condition $(X, j)$ obviously models the desired temporal dependencies among the jobs. Another motivation is studied by Goldwasser and Motwani [13]. They consider the problem of partially disassembling a given product to reach a single part (or component). In order to remove a certain part, one previously may have to remove other parts which can be modeled by traditional (AND) precedence constraints. However, one may choose to remove that same part of the product from another geometric direction, in which case some other parts must be removed previously. This freedom of choice can be modeled by AND/OR precedence constraints. A third motivation is given by Dinic [6], who considers the setup of new technologies and products. In his model, a new technology requires certain products; on the other hand, a new product can be obtained as an output of one of several new technologies. The latter requirement leads to an OR-constraint, while the first requirement is a classical AND-constraint.

Our research is motivated by problems occurring in resource-constrained project scheduling. Resource constraints can be represented by so-called *minimal forbidden sets*, i.e., inclusion-minimal sets of jobs that cannot be scheduled simultaneously. In order to resolve the resource conflict that occurs due to a minimal forbidden set $F$, one may choose a job $j \in F$ to be a waiting job for $F \setminus \{j\}$ and introduce the corresponding waiting condition $(F \setminus \{j\}, j)$. Thus, we are able to represent solutions of project scheduling problems by a set $\mathcal{W}$ of waiting conditions. For details we refer to [26].

**Connections to other fields and related work.** The combinatorial structure of a system $\mathcal{W}$ of waiting conditions occurs in different fields of discrete mathematics and theoretical computer science. In the context of *directed hypergraphs* each $(X, j) \in \mathcal{W}$ represents a hyperarc with a set $X$ of source nodes and a single target node $j$. Ausiello, d'Atri, and Saccà [3] (see also [4]) generalize transitive closure and reduction algorithms from directed graphs to directed hypergraphs. Another related class of combinatorial objects are *antimatroids* (special greedoids) which can be defined via a set of waiting conditions; see, e.g., [17, page 22]. Furthermore, many problems stemming from artificial intelligence can be formulated by hierarchies of subproblems where different alternatives exist to solve these subproblems; see, e.g., [23]. There, a graphical representation of such hierarchies is called an AND/OR graph.

In the context of scheduling, Goldwasser and Motwani [13] derive inapproximability results for two single-machine scheduling problems with AND/OR precedence constraints. Gillies and Liu [12] consider single- and parallel-machine scheduling problems with different structures of AND/OR precedence constraints; they prove NP-completeness of finding feasible schedules in some settings that are polynomially solvable with traditional precedence constraints. Moreover, they give approximation algorithms for some makespan minimization problems.

A basic, important task in scheduling applications is the computation of earliest start times of jobs. The research on this topic will be discussed in more detail in section 7.1.

**Contribution of this paper.** For AND-constraints, fundamental problems such as deciding feasibility, finding transitive AND-constraints, and computing earliest start times of jobs can be solved efficiently by applying simple well-known graph algorithms. Most important for the algorithmic treatment is the fact that AND-constraints define acyclic structures on the set $V$ of jobs such that many problems can be solved by considering jobs in the order of a topological sort. Since this is not the case for AND/OR precedence constraints, the algorithms for AND-constraints cannot be applied in that setting.

In the first part of the paper we provide efficient algorithms and structural insights for the more general and complex model of AND/OR precedence constraints. We show that feasibility as well as questions related to generalized transitivity can be solved by applying essentially the same linear-time algorithm. Moreover, we discuss a natural generalization of AND/OR precedence constraints and prove that the same problems become NP-complete in this setting.

The second part of this paper is concerned with the computation of earliest start times if AND/OR precedence constraints are imposed among jobs. We consider different ranges of minimal *time distances* (or time lags) $d_{ij}$ between the start times of two jobs $i$ and $j$ that are coupled within a precedence constraint. For AND/OR precedence constraints, the problem then reduces to finding a solution to a system of *min-max-inequalities* which also has applications in many other fields. We discuss polynomial equivalence to finding optimal strategies for a class of two-person games played on directed graphs. For the case that such time lags are strictly positive (nonnegative) we devise polynomial-time algorithms.

**Outline.** The paper is organized as follows. After stating some basic requirements in section 2, we discuss feasibility and aspects of transitivity as well as an application thereof in sections 3 to 5. A generalization of AND/OR precedence constraints is considered in section 6. While we consider only the combinatorial structure of AND/OR precedence constraints in sections 3 to 6, we additionally deal with temporal data such as job processing times or time lags between jobs in section 7, where we provide algorithms for computing earliest job start times.

**2. Preliminaries.** In order to illustrate the presentation we use the following example throughout the paper.

*Example* 1. Let $V := \{j_1, \dots, j_7\}$ be the set of jobs and $\mathcal{W} := \{w_1 = (\{j_1, j_5\}, j_4)$, $w_2 = (\{j_2, j_6\}, j_4)$, $w_3 = (\{j_4, j_3\}, j_6)$, $w_4 = (\{j_4\}, j_5)$, $w_5 = (\{j_4, j_5, j_6\}, j_7)\}$ the set of waiting conditions.

**Graph representation.** We use a natural representation of AND/OR precedence constraints by a directed graph $D$ on the set $\mathcal{V} = V \cup \mathcal{W}$ of nodes. The set $A$ of arcs is constructed in the following way: For every waiting condition $w = (X, j) \in \mathcal{W}$, we introduce arcs $(i, w)$, for each $i \in X$, and one additional arc $(w, j)$. Notice that the size of the resulting digraph $D$ is linear in the input size of the problem. The sets $V$ and $\mathcal{W}$ form a bipartition of $D$. Similar digraphs are used to represent directed hypergraphs; see, e.g., [10] and [9]. For a node $j \in V \cup \mathcal{W}$, we use $in(j)$ and $out(j)$ to denote the sets $\{i \in V \cup \mathcal{W} : (i, j) \in A\}$ and $\{i \in V \cup \mathcal{W} : (j, i) \in A\}$, respectively. We also sometimes use the notation $in_D(j)$ and $out_D(j)$ to stress the underlying digraph $D$.

The digraph resulting from Example 1 is depicted in Figure 2.1. For the moment, the numbers associated with the arcs can be ignored; they come into play in section 7 when earliest job start times are computed. As usual, a cycle in $D$ is a sequence $(v_0, v_1, \dots, v_k, v_0)$, $v_\ell \in \mathcal{V}$, where $(v_0, v_1, \dots, v_k)$ is a directed path and there exists an

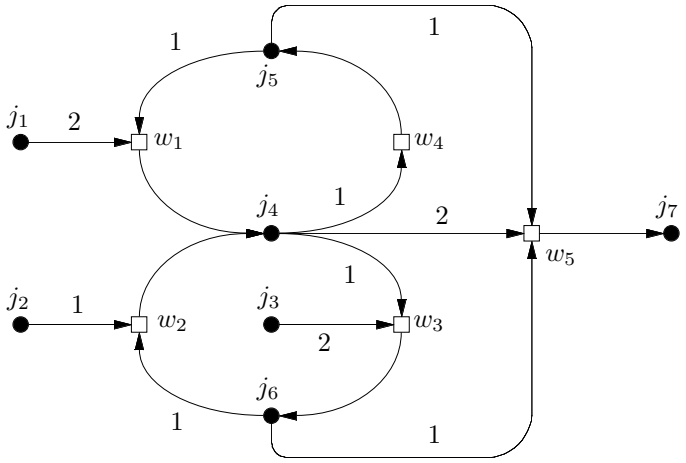


FIG. 2.1. *The digraph resulting from Example* 1. *Circular nodes correspond to jobs (*AND*-nodes), while square nodes represent waiting conditions (*OR*-nodes). Numbers associated with arcs define time lags used in section* 7 *(the time lags of arcs without a number are* 0*).*

arc from $v_k$ to $v_0$. We also consider *generalized cycles*, which are induced subgraphs $D'$ of $D$ that consist of node sets $V' \subseteq V$ and $\mathcal{W}' \subseteq \mathcal{W}$ such that $in_D(j) \cap \mathcal{W}' \neq \emptyset$ for each $j \in V'$ and $\emptyset \neq in_D(w) \subseteq V'$ for each $w \in \mathcal{W}'$.

**Realizations.** Given a set $V$ of jobs and a set $\mathcal{W}$ of waiting conditions, an implementation of the corresponding project requires a decision for each waiting condition $(X, j)$: One has to determine a job $i \in X$ that job $j$ should wait for. The entirety of these decisions must lead to a partial order $R = (V, \prec_R)$ on the set $V$ of jobs (the introduction of a cycle would lead to infeasibility) such that

$$\text{for each } (X, j) \in \mathcal{W}, \text{ there exists an } i \in X \text{ with } i \prec_R j. \tag{2.1}$$

Conversely, every partial order $R$ with property (2.1) defines an implementation of the project and is therefore called a *realization* for the set $\mathcal{W}$ of waiting conditions. In what follows, a set $\mathcal{W}$ of waiting conditions is called *feasible* if and only if there exists a realization for $\mathcal{W}$. Since any extension $R' = (V, \prec_{R'})$ of a realization $R$ (i.e., if $i \prec_R j$, then $i \prec_{R'} j$) fulfills property (2.1), $R'$ is also a realization. In particular, a set of AND/OR precedence constraints is feasible if and only if there exists a total order of the jobs which is a realization; we call such a realization *linear*.

Possible linear realizations of Example 1 are, for instance, $j_1 \prec \cdots \prec j_7$ and $j_3 \prec j_6 \prec j_7 \prec j_2 \prec j_1 \prec j_4 \prec j_5$.

**3. Feasibility.** In order to check whether a given set $\mathcal{W}$ of AND/OR precedence constraints is feasible, we try to construct a linear realization $L$ in a greedy way: While there exists a job $i \in V$ that is not a waiting job of any waiting condition in $\mathcal{W}$, it is inserted at the end of $L$. Whenever a waiting condition $(X, j)$ becomes satisfied (which is the case if some $i \in X$ is being added to $L$), $(X, j)$ is deleted from $\mathcal{W}$. Computational details are provided in Algorithm 1. We use a data structure $Q$ to temporarily store jobs from $V$. Implementing $Q$ as a stack or a queue leads to a linear-time algorithm.

THEOREM 3.1. *A set of* AND/OR *precedence constraints is feasible if and only if the list $L$ obtained from Algorithm* 1 *contains all jobs of $V$.*

**Algorithm 1:** Feasibility check of a set of waiting conditions.

**Input** : A set $V$ of jobs and waiting conditions $\mathcal{W}$.
**Output**: A list $L$ of jobs from $V$.

```
Q := ∅; L := ∅;
for jobs j ∈ V do
    a(j) := |{(X, j) ∈ W}|;
    if a(j) = 0 then add j to Q;
while Q ≠ ∅ do
    remove a job i from Q;
    insert i at the end of L;
1   for waiting conditions (X, j) ∈ W with i ∈ X do
        decrease a(j) by 1;
        if a(j) = 0 then add j to Q;
        remove (X, j) from W;
return L;
```

*Proof.* If $L$ contains all jobs, it follows from the construction of Algorithm 1 that, for each waiting condition $(X, j) \in \mathcal{W}$, there is at least one job $i \in X$ with $i \prec_L j$; therefore, according to (2.1), $L$ is a linear realization. Suppose now that the algorithm returns an incomplete list $L$ although the set of waiting conditions is feasible. Consider a linear realization $R$ and let $j \in V \setminus L$ be minimal with respect to the total order $\prec_R$. Since the algorithm was not able to add $j$ to $L$, there is a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus L$. Since $R$ is a realization, there exists a job $i \in X$ with $i \prec_R j$ which is a contradiction of the minimal choice of $j$. □

As a consequence of Theorem 3.1 we can formulate the following structural characterization of feasible waiting conditions. The lemma appears implicitly already in the work of Igelmund and Radermacher [14] within the context of stochastic resource-constrained project scheduling.

LEMMA 3.2. *A set of* AND/OR *precedence constraints is feasible if and only if there exists no generalized cycle in the associated digraph $D$.*

Note that Example 1 is feasible (recall that we already stated two linear realizations). However, if $w_1 = (\{j_1, j_5\}, j_4)$ is replaced by $(\{j_5\}, j_4)$, the instance becomes infeasible because $V' = \{j_4, j_5\}$ and $\mathcal{W}' = \{(\{j_5\}, j_4), (\{j_4\}, j_5)\}$ form a generalized cycle.

The following corollary states an algorithmic consequence of the structural insight of Lemma 3.2.

COROLLARY 3.3. *Job $j \in V$ is not contained in the list $L$ returned by Algorithm* 1 *if and only if $j$ is contained in a set $V' \subseteq V$ such that for all $i \in V'$ there is a waiting condition $(X, i) \in \mathcal{W}$ with $X \subseteq V'$.*

In particular, $L$ as a *set* does not depend on the individual jobs chosen from $Q$ in the while-loop of Algorithm 1.

In the proof of Theorem 3.1 we have shown that, for a feasible set of AND/OR precedence constraints $\mathcal{W}$, the list $L$ returned by Algorithm 1 is a linear realization of $\mathcal{W}$. In fact, it is an easy observation that Algorithm 1 can generate every linear realization of $\mathcal{W}$ through an appropriate choice of jobs from $Q$ in the while-loop.

*Remark.* The problem of checking feasibility of $\mathcal{W}$ can alternatively be solved by transforming it into a satisfiability problem (SAT) where each clause is of *Horn*

type. Such SAT instances are well known to be solvable in linear time; see [7]. The transformation and more details can be found in [26].

## 4. Detecting implicit AND/OR precedence constraints.

**4.1. Problem definition and related work.** We now focus on detecting "new" waiting conditions that can be deduced from the given set $\mathcal{W}$ of constraints. For $U \subset V$ and $j \in V \setminus U$, we say that the waiting condition $(U, j)$ is *implied* by $\mathcal{W}$ if and only if

(4.1) for every realization $R = (V, \prec_R)$ of $\mathcal{W}$, there exists some $i \in U$ with $i \prec_R j$.

By property (2.1), this is equivalent to the requirement that adding the waiting condition $(U, j)$ to $\mathcal{W}$ does not change the set of realizations for $\mathcal{W}$. Notice that it is sufficient to claim property (4.1) for every *linear* realization of $\mathcal{W}$.

For traditional precedence constraints the detection of implied waiting conditions is an easy task because the transitive closure which represents all such implicit constraints can be efficiently computed by standard graph algorithms. However, in general, the total number of implicit AND/OR precedence constraints is exponential in the input size of $V$ and $\mathcal{W}$. In particular, it is not possible to compute all implicit constraints efficiently. For the restricted case of AND/OR precedence constraints where the associated digraph $D$ is acyclic, Gillies [11] proposes an algorithm to determine jobs that have to wait for a single job $i$.

In the context of directed hypergraphs, Ausiello, d'Atri, and Saccà [3] (see also [4]) consider problems similar to those discussed in this section and in section 5 below. However, the results we present are not contained in their work because their definition of implicit hyperarcs differs from our definition of implicit waiting conditions. Their definition is based on three rules which are known as *Armstrong's axioms* within the context of functional dependencies in relational databases (see, e.g., [27]). In particular, [3, Definition 4] does not cover implications that can be deduced from the requirement of feasibility. For instance, in Example 1, the waiting condition $(\{j_1\}, j_4)$ is implied by $\mathcal{W}$, but it is not implied according to [3, Definition 4].

**4.2. Result.** For a given set $U \subseteq V$ we show that Algorithm 1 can be used to detect all implicit waiting conditions of the form $(U, j)$. For an arbitrary subset $Y \subseteq V$ the set $\mathcal{W}_Y$ of *induced* waiting conditions is given by $\mathcal{W}_Y := \{(X \cap Y, j) \mid (X, j) \in \mathcal{W}, j \in Y\}$. For $(X, j) \in \mathcal{W}$ with $j \in Y$ and $X \cap Y = \emptyset$, the resulting waiting condition $(\emptyset, j) \in \mathcal{W}_Y$ means that job $j$ cannot be planned at all with respect to $\mathcal{W}_Y$; in particular, $\mathcal{W}_Y$ is infeasible in this case.

THEOREM 4.1. *For given $U \subset V$ let $L$ be the output of Algorithm 1 with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. The set of waiting conditions of the form $(U, j)$ which are implied by $\mathcal{W}$ is precisely $\{(U, j) \mid j \in V \setminus (L \cup U)\}$.*

The proof is deferred to section 4.3 below. For Example 1 and $U := \{j_2, j_3\}$, the algorithm computes $L = \{j_1\}$, while for $U := \{j_1, j_2\}$ we obtain $L = \{j_3, j_6, j_7\}$. Thus, the waiting condition $(\{j_2, j_3\}, j_7)$ is implied by $\mathcal{W}$, while $(\{j_1, j_2\}, j_7)$ is not.

We can directly deduce the following corollary.

COROLLARY 4.2. *Given $U \subset V$, the set of waiting conditions of the form $(U, j)$ that are implied by $\mathcal{W}$ can be computed in linear time.*

**4.3. Correctness.** We next state some rather technical lemmas which directly show the validity of Theorem 4.1. The theorem can alternatively be proved by a simpler argumentation (similar to the proof of Theorem 3.1), but we need the lemmas

to establish other results in section 5 below. In addition, with the extended argumentation, we are able to strengthen Theorem 4.1 slightly (see Corollary 4.6). The following definition will be useful throughout the discussion: For a given feasible set $\mathcal{W}$ of waiting conditions and a set $U \subseteq V$ let

$$\begin{aligned} Y_U &:= \{j \in V \setminus U \mid (U,j) \text{ is not implied by } \mathcal{W}\}, \\ Z_U &:= \{j \in V \setminus U \mid (U,j) \text{ is implied by } \mathcal{W}\} = V \setminus (U \cup Y_U) \ . \end{aligned}$$

LEMMA 4.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions and let $U \subseteq V$. Then there exists a (linear) realization $R = (V, \prec_R)$ of $\mathcal{W}$ such that $Y_U$ is an order ideal of $R$; i.e., $R$ "starts" with the jobs in $Y_U$.*

*Proof.* Let $R' = (V, \prec_{R'})$ be a linear realization of $\mathcal{W}$ that maximizes the cardinality of the largest order ideal $J$ of $R'$ with $J \subseteq Y_U$. To show that $J = Y_U$, by contradiction, we assume that there is a job $j' \in Y_U \setminus J$. Since, by definition of $Y_U$, the waiting condition $(U, j')$ is not implied by $\mathcal{W}$, there is a linear realization $R = (V, \prec_R)$ of $\mathcal{W}$ with $j' \prec_R U$ ($j'$ precedes all elements in $U$). Let $j \in Y_U \setminus J$ be minimal with respect to $R$. By maximality of $J$, job $j$ cannot be moved to the position directly after $J$ in $R'$ without violating a waiting condition. Thus, there exists $(X, j) \in \mathcal{W}$ with $X \subseteq V \setminus J$. By (2.1), there exists an $i \in X$ with $i \prec_R j$. Notice that we have $i \notin U$ because $i \prec_R j \preceq_R j' \prec_R U$. Moreover, due to the minimal choice of $j$ and the fact that $i \notin J$ it follows that $i \notin Y_U$. As a consequence we obtain $i \in X \setminus (U \cup Y_U)$. By definition of $Z_U$, this yields $i \in Z_U$. Thus, the waiting condition $(U, i)$ is implied by $\mathcal{W}$, which is a contradiction of $i \prec_R j \preceq_R j' \prec_R U$. □

Let us call a set $Y \subseteq V$ *feasible with respect to* $\mathcal{W}$ if and only if the induced set $\mathcal{W}_Y$ of waiting conditions is feasible. The result in Corollary 3.3 can then be restated as follows.

COROLLARY 4.4. *Algorithm* 1 *returns the unique maximal feasible subset of $V$ with respect to $\mathcal{W}$.*

In conjunction with the following lemma, Corollary 4.4 provides an efficient way of detecting waiting conditions implied by $\mathcal{W}$. This concludes the proof of Theorem 4.1 (in fact, the lemma essentially is a reformulation of Theorem 4.1).

LEMMA 4.5. *Let $\mathcal{W}$ be a feasible set of* AND/OR *precedence constraints, $U \subset V$, and $j \in V \setminus U$. Then the waiting condition $(U, j)$ is implied by $\mathcal{W}$ if and only if $j$ is not contained in the unique maximal feasible subset of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$.*

*Proof.* We have to show that $Y_U$ is the unique maximal feasible subset $F$ of $V \setminus U$ with respect to $\mathcal{W}_{V \setminus U}$. By Lemma 4.3, there exists a linear realization $R$ of $\mathcal{W}$ starting with the jobs in $Y_U$. This induces a linear realization of $\mathcal{W}_{Y_U}$ and consequently, by definition, $Y_U$ is feasible with respect to $\mathcal{W}$. Moreover, since $\mathcal{W}_{Y_U} = (\mathcal{W}_{V \setminus U})_{Y_U}$, the subset $Y_U$ is feasible with respect to $\mathcal{W}_{V \setminus U}$, which yields $Y_U \subseteq F$.

To show that $F \subseteq Y_U$, by contradiction, assume that $F \setminus Y_U \neq \emptyset$. Since $F \cap U = \emptyset$ we then have $F \cap Z_U \neq \emptyset$. Let $R' = (F, \prec_{R'})$ be a linear realization of $\mathcal{W}_F$ and choose $i \in F \cap Z_U$ minimal with respect to $R'$. Since $i \in Z_U$, by definition of $Z_U$, the waiting condition $(U, i)$ is implied by $\mathcal{W}$. Therefore, moving job $i$ to the position directly after $Y_U$ in $R$ violates a waiting condition $(X, i) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Let us consider the induced waiting condition $(X \cap F, i) \in \mathcal{W}_F$. It follows from the minimal choice of $i$ that $i' \notin F \cap Z_U$ for all $i'$ with $i' \prec_{R'} i$. With $F \cap U = \emptyset$, this implies $i' \notin X \cap F$, which is a contradiction of the fact that $R'$ is a realization for $\mathcal{W}_F$. □

Finally, notice that there may exist (implicit) waiting conditions $(X, j)$ inside the considered set $U$, i.e., $X \subset U$ and $j \in U \setminus X$. Theorem 4.1 can be strengthened in the following way. Consider the situation after the execution of Algorithm 1 with input

$V \setminus U$ and $\mathcal{W}_{V\setminus U}$ and let $L$ be the resulting list of jobs. Furthermore, let $U' \subseteq U$ denote the set of jobs from $U$ that can be added to $L$ without violating any waiting condition of $\mathcal{W}$.

COROLLARY 4.6. *For given $U \subseteq V$ the set of waiting conditions $(U', j)$ which is implied by $\mathcal{W}$ is precisely $\{(U', j) \mid j \in V \setminus (L \cup U')\}$.*

*Proof.* We show that the maximal feasible subsets $F$ and $F'$ of $V \setminus U$ and $V \setminus U'$, respectively, coincide. The corollary then follows from Lemma 4.5. It is clear that $F \subseteq F'$ since $U' \subseteq U$. Conversely, suppose by contradiction that $F' \setminus F \neq \emptyset$. Denote by $R = (V, \prec_R)$ and $R' = (V, \prec_{R'})$ linear realizations of $\mathcal{W}$ where $F$ and $F'$ are order ideals, respectively (the existence of $R$ and $R'$ follows from Lemmas 4.3 and 4.5). Let $j \in F' \setminus F$ be the smallest job in $F' \setminus F$ with respect to $R'$. Since $j \notin F$, moving job $j$ in $R$ to the position directly after $F$ violates a waiting condition $(X, j)$ with $X \cap F = \emptyset$. Since $R'$ is a realization there must exist some $i \in X$ with $i \prec_{R'} j$. But $i \in F' \setminus F$, which contradicts the minimal choice of $j$. ∎

The results presented in this section turn out to be useful in the context of minimal representations of AND/OR precedence constraints discussed in section 5. Besides this, Corollary 4.6 led to a considerable speedup of computation time within branch-and-bound procedures for stochastic resource-constrained project scheduling; see [26] for details.

**5. Minimal representations of AND/OR precedence constraints.** While for traditional precedence constraints a minimal representation without redundancies is given by the transitive reduction and can be computed by simply removing redundant (i.e., transitive) constraints, the situation is slightly more complicated for AND/OR precedence constraints. In order to obtain a *unique* minimal representation, it is not sufficient to iteratively remove redundant waiting conditions that are implied by the others.

DEFINITION 5.1. *A set $\mathcal{W}$ of waiting conditions is called* minimal *if*

(i) *no waiting condition $(X, j) \in \mathcal{W}$ is implied by $\mathcal{W} \setminus \{(X, j)\}$, and*

(ii) *for each waiting condition $(X, j) \in \mathcal{W}$, the set $X$ is minimal with respect to inclusion; i.e., for all $i \in X$, the waiting condition $(X \setminus \{i\}, j)$ is not implied by $\mathcal{W}$.*

*Two sets $\mathcal{W}$ and $\mathcal{W}'$ of waiting conditions are called* equivalent *if their sets of (linear) realizations coincide. Moreover, if $\mathcal{W}'$ is minimal, then $\mathcal{W}'$ is called a* minimal reduction *of $\mathcal{W}$.*

The set $\mathcal{W}$ from Example 1 is not minimal: If the waiting condition $(\{j_4, j_5, j_6\}, j_7)$ is replaced by $(\{j_4, j_6\}, j_7)$, the resulting instance is equivalent to Example 1. This follows from waiting condition $(\{j_4\}, j_5)$, which ensures that whenever $j_5 \prec_R j_7$ in some realization $R = (V, \prec_R)$ we also have $j_4 \prec_R j_5$ and $j_4 \prec_R j_7$. Note that if we additionally replace $(\{j_1, j_5\}, j_4)$ by $(\{j_1\}, j_4)$, the resulting set of waiting conditions is minimal (and still equivalent to Example 1).

THEOREM 5.2. *Each feasible set of waiting conditions has a unique minimal reduction.*

To prove the theorem we need the following technical lemma.

LEMMA 5.3. *Let $\mathcal{W}$ be a feasible set of waiting conditions with $(U, j) \in \mathcal{W}$. The waiting condition $(U, j)$ is implied by $\mathcal{W}' := \mathcal{W} \setminus \{(U, j)\}$ if and only if there exists some $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$.*

*Proof.* If $(U, j)$ is implied by $\mathcal{W}'$, then any ordering of $V$ where $Y_U$ is an ideal and $j$ is placed directly after $Y_U$ is not a realization of $\mathcal{W}'$. Thus, there exists some $(X, j) \in \mathcal{W}$ with $X \subseteq U \cup Z_U$. Contrarily, suppose that there exists some $(X, j) \in \mathcal{W}'$

**Algorithm 2:** Computation of a minimal reduction.

**Input** : A set $V$ of jobs and waiting conditions $\mathcal{W}$.
**Output**: A minimal reduction of $\mathcal{W}$

```
for each (U, j) ∈ W do
    L := call Algorithm 1 with input V \ U and W_{V\U} and
         compute a(i) for each i ∈ V;
    if a(j) > 1 then
        delete (U, j) from W;
    else for i ∈ U do
        if a(i) > 0 then delete i from U;
return W;
```

with $X \subseteq U \cup Z_U$ but $(U, j)$ is not implied by $\mathcal{W}'$. Then there exists some $h \in X \setminus U$ and a linear realization $R' = (V, \prec_{R'})$ of $\mathcal{W}'$ with $Y_U \prec_{R'} h \prec_{R'} j \prec_{R'} U$. Since $\mathcal{W}$ is feasible, there exists some $i \in U$ that can be moved to the position directly after $h$ without violating a waiting condition of $\mathcal{W}'$ (this follows from Corollary 4.6). In addition, the resulting linear realization $R = (V, \prec_R)$ satisfies the waiting condition $(U, j)$ and thus should be a realization with respect to $\mathcal{W}$. However, we have $h \prec_R i \prec_R j \prec_R U \setminus \{i\}$, which is a contradiction of $h \in Z_U$. □

*Proof of Theorem* 5.2. Let $\mathcal{W}$ and $\mathcal{W}'$ be equivalent and both minimal. It suffices to show that $(U, j) \in \mathcal{W}$ implies $(U, j) \in \mathcal{W}'$. By Lemma 4.3, there exists a linear realization $R$ of $\mathcal{W}$ starting with $Y_U$. Since the order obtained by moving $j$ to the position directly after $Y_U$ in $R$ is not a realization, there exists a waiting condition $(X, j) \in \mathcal{W}'$ with $X \subseteq U \cup Z_U$. We show next that $U \subseteq X$. By minimality of $\mathcal{W}'$ this implies $X = U$, which concludes the proof.

Assume that $X \not\supseteq U$ and let $i \in U \setminus X$. We obtain a linear realization $R'$ by moving $i$ to the position directly after $Y_U$ in $R$; otherwise, there exists a waiting condition $(Z, i) \in \mathcal{W}$ with $Z \subseteq U \cup Z_U$. Since all jobs in $Z_U$ have to wait for a job in $U$, the waiting condition $(U \setminus \{i\}, i)$, and thus $(U \setminus \{i\}, j)$, is implied by $\mathcal{W}$, which is a contradiction of the minimality of $\mathcal{W}$.

Since moving $j$ to the position directly after $Y_U \cup \{i\}$ in $R'$ violates the waiting condition $(X, j)$, there exists a waiting condition $(Z, j) \in \mathcal{W}$ with $Z \subseteq (U \setminus \{i\}) \cup Z_U$. However, by Lemma 5.3, the set $\mathcal{W} \setminus \{(U, j)\}$ implies the waiting condition $(U, j)$, which is a contradiction of the minimality of $\mathcal{W}$. □

Let us next consider the following straightforward polynomial-time algorithm to compute a minimal reduction of a set $\mathcal{W}$ of waiting conditions. For each $(X, j) \in \mathcal{W}$, apply Algorithm 1 with input $V \setminus X$ and $\mathcal{W}_{V \setminus X}$. If, besides $(X, j)$, some other waiting condition prevents $j$ from being added to $L$, then remove $(X, j)$ from $\mathcal{W}$. Otherwise, remove all $i$ from $X$, which cannot be added to $L$ because some waiting condition of $\mathcal{W}$ is violated. Finally, output the resulting set of waiting conditions. An implementation of this rough scheme is given in Algorithm 2. There, $a(j)$, $j \in V$, denotes the number of waiting conditions of the form $(X, j)$ that are left in $\mathcal{W}_{V \setminus U}$ after Algorithm 1 was called with input $V \setminus U$ and $\mathcal{W}_{V \setminus U}$. Notice that $a(j)$ is computed within the execution of Algorithm 1. In the following theorem we prove the correctness of the algorithm (as defined earlier, $A$ is the set of arcs in the digraph induced by $\mathcal{W}$).

THEOREM 5.4. *Algorithm* 2 *computes the minimal reduction of a set* $\mathcal{W}$ *of waiting conditions in* $O(|\mathcal{W}| \cdot |A|)$ *time.*

*Proof.* We first show that, through the procedure, the transformed set of waiting conditions is equivalent to $\mathcal{W}$ given as input. We then argue that, once the algorithm has finished, the obtained set of waiting conditions is minimal.

Denote by $\mathcal{W}^k$, $k \in \{1, \dots, |\mathcal{W}|\}$, the set of waiting conditions after the $k$th iteration of the outer for-loop of Algorithm 2. Furthermore, let $\mathcal{W}^0 := \mathcal{W}$. Suppose that some $(U, j)$ is removed from $\mathcal{W}^{k-1}$ in the $k$th iteration of the algorithm. Since $a(j) > 1$ in the $k$th iteration, there exists a waiting condition $(X, j) \in \mathcal{W}^{k-1}$ with $X \neq U$ and $X \subset U \cup Z_U$. With Lemma 5.3, $(U, j)$ is implied by $\mathcal{W}^k = \mathcal{W}^{k-1} \setminus \{(U, j)\}$ and can thus be deleted from $\mathcal{W}^{k-1}$. Now assume that, in $(U, j)$, some job $i$ was deleted from $U$ in the $k$th iteration. Then $a(i) > 0$ and with Corollary 4.6 it follows that $(U \setminus \{i\}, i)$ is implied by $\mathcal{W}^{k-1}$. Together with $(U, j)$ this shows that $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}^{k-1}$. Thus, $\mathcal{W}^{k-1}$ is equivalent to $\mathcal{W}^k$ for all $k \in \{1, \dots, |\mathcal{W}|\}$, which directly implies that $\mathcal{W}$ and $\mathcal{W}' := \mathcal{W}^{|\mathcal{W}|}$ are equivalent.

We now show that $\mathcal{W}'$ is minimal. Let us first suppose that some $(U, j) \in \mathcal{W}'$ is implied by $\mathcal{W}' \setminus \{(U, j)\}$. Then, by Lemma 5.3, there exists another waiting condition $(X, j) \in \mathcal{W}' \setminus \{(U, j)\}$ with $X \subseteq U \cup Z_U$. Notice that $Z_U$ in dependence of $\mathcal{W}' \setminus \{(U, j)\}$ and all $\mathcal{W}^k$, $k \in \{0, \dots, |\mathcal{W}|\}$, is constant because the associated sets of realizations coincide. The waiting conditions $(U, j)$ and $(X, j)$ have been constructed in some iterations $k$ and $k'$, respectively, in which waiting conditions $(U', j) \in \mathcal{W}$ with $U \subseteq U'$ and $(X', j) \in \mathcal{W}$ with $X \subseteq X'$ have been treated by the algorithm. If $(X, j) \in \mathcal{W}^{k-1}$, then, by Lemma 5.3, $(U', j)$ would have been removed from $\mathcal{W}^{k-1}$. Consequently, $k' > k$. Since $U$ was obtained from $U'$ (in the $k$th iteration of the algorithm), by Corollary 4.6, there exists a linear realization $R$ which starts with $Y_U$ and is followed first by an *arbitrary* job $i \in U$ and then by job $j$. Since, by assumption, $\mathcal{W}^{k-1}$ and $\mathcal{W}' \setminus \{(U, j)\}$ are equivalent, $(X, j)$ must be respected by $R$; hence $U \subseteq X$. But then $(X', j)$ is deleted in iteration $k' > k$, a contradiction. Next, suppose that $\mathcal{W}'$ contains a waiting condition $(U, j)$ such that, for some $i \in U$, the waiting condition $(U \setminus \{i\}, j)$ is implied by $\mathcal{W}'$. Since $i$ was not removed from $U'$ in the $k$th iteration of the algorithm, it follows from Corollary 4.6 that $(U' \setminus \{i\}, j)$ is not implied by $\mathcal{W}^{k-1}$. Thus there exists a linear realization $R = (V, \prec_R)$ of $\mathcal{W}^{k-1}$ with $j \prec_R (U' \setminus \{i\})$ and in particular $j \prec_R (U \setminus \{i\})$. Since $R$ is also a realization for $\mathcal{W}'$, the waiting condition $(U \setminus \{i\}, j)$ is not implied by $\mathcal{W}'$—a contradiction.

The above argumentation shows that $\mathcal{W}'$ is minimal and thus Algorithm 2 computes a minimal reduction of $\mathcal{W}$. Finally, the running time follows from the fact that Algorithm 1 is called $|\mathcal{W}|$ times. ∎

Notice that the cardinality of a minimal set of waiting conditions might still be exponential in the number of jobs $|V|$: Let $V = \{1, 2, \dots, 2\ell + 1\}$; in order to model the constraint that job $2\ell + 1$ can be planned only after at least $\ell$ other jobs, we need exactly $\binom{2\ell}{\ell}$ waiting conditions.

**6. An NP-complete generalization.** Suppose that we generalize the definition of waiting conditions from $(X, j)$, $X \subset V$, $j \in V \setminus X$ to $(X, X')$ with $X, X' \subset V$ and $X \cap X' = \emptyset$. The generalized waiting condition $(X, X')$ is fulfilled if at least one job $j \in X'$ is waiting for at least one job $i \in X$. We show in the theorem below that the problems considered in sections 3 and 4 become NP-complete in this generalized setting.

THEOREM 6.1. *Given a set of jobs with generalized waiting conditions, it is NP-complete to decide whether or not a waiting condition* $(\{i\}, \{j\})$ *is implied for two jobs* $i$ *and* $j$.

*Proof.* We construct a reduction from the satisfiability problem SAT. Given an instance of SAT, we introduce for each Boolean variable $x$ two jobs which correspond to the two literals $x$ and $\bar{x}$ (negation of $x$); to keep notation simple, we denote these jobs also by $x$ and $\bar{x}$. Moreover, for each clause $C$ we introduce a corresponding job (also denoted by $C$) and a waiting condition $(X_C, \{C\})$, where $X_C$ denotes the set of literals in clause $C$; in other words, job $C$ may not be started before at least one job corresponding to a literal of clause $C$ has been completed. Finally, we introduce two additional jobs $s$ and $t$ together with the following waiting conditions: For each variable $x$, at least one of the jobs $x$ and $\bar{x}$ has to wait for $s$; i.e., we have the waiting condition $(\{s\}, \{x, \bar{x}\})$. For each clause $C$, job $t$ has to wait for the corresponding job, which is given by the waiting condition $(\{C\}, \{t\})$.

It is easy to check that in the constructed scheduling instance job $t$ has to wait for job $s$ if and only if the underlying instance of SAT does not have a satisfying truth assignment. If there is a satisfying truth assignment, then we can construct a linear realization where $t$ precedes $s$ in the following way: First we take all jobs corresponding to literals with value "true" in an arbitrary order; next we append all jobs corresponding to clauses in some order; afterwards we add $t$, then $s$, and finally all remaining jobs corresponding to literals with the value "false." On the other hand, if there is a linear realization where $t$ precedes $s$, we can define a corresponding satisfying truth assignment in the following way: For each variable $x$, assign $x$ the value true (false) if the job corresponding to $x$ ($\bar{x}$) precedes $s$; notice that at most one of the two cases can happen: if neither job $x$ nor $\bar{x}$ precedes $s$, we assign an arbitrary value to the variable $x$. □

As a consequence of Theorem 6.1, we obtain that the problem of deciding feasibility for a set of generalized waiting conditions is also NP-complete. To see this, add to the construction in the proof the waiting condition $(\{t\}, \{s\})$. Then the given instance of SAT is feasible if and only if the constructed instance of the scheduling problem with generalized waiting conditions is feasible.

**7. Computing earliest job start times.** This section is concerned with the computation of earliest job start times subject to AND/OR precedence constraints. The underlying problem is to find a solution to a system of *min-max-inequalities*. There are several other applications of such systems of inequalities; we will mention some of them below.

**7.1. Problem definition and related work.** For the remainder of the paper we assume that together with each waiting condition $w = (X, j) \in \mathcal{W}$ and each job $i \in X$ we are given an integral time lag $-M < d_{iw} < M$, $M \geqslant 0$. We aim at finding a vector of earliest start times $S = (S_1, \ldots, S_n)$ such that for each waiting condition $(X, j) \in \mathcal{W}$ the constraint

$$S_j \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \tag{7.1}$$

is satisfied. Job processing times $p_i$ can be modeled by setting $d_{iw} := p_i$ for all $w = (X, j)$ with $i \in X$. Negative values $d_{iw}$ represent so-called *maximal time lags* that define latest possible start times of jobs $i \in X$ relative to $j$.

In order to simplify the presentation, we sometimes interpret nodes of the digraph $D$ that represent waiting conditions as dummy jobs. We then assume that the vector $S$ also contains start times of these dummy jobs and constraint (7.1) is replaced by

$$S_w \;\geqslant\; \min_{i \in X}(S_i + d_{iw}) \qquad \text{and} \qquad S_j \;\geqslant\; S_w.$$

We call the jobs in $V$ AND-*nodes* and the jobs in $\mathcal{W}$ OR-*nodes* of the digraph $D$. We assume that a dummy AND-node $s$ precedes all other AND-nodes; i.e., we introduce a waiting condition $(\{s\}, j)$ for all $j \in V$. In $D$, time lags can easily be integrated by associating each $d_{iw}$ as a weight to the arc $(i, w)$; see Figure 2.1.

The problem of finding earliest start times can then be formulated on $D$ as follows: Find a componentwise minimal schedule $S \in \mathbb{Z}^{|\mathcal{V}|}$ fulfilling $S_s \geqslant 0$ and

$$\text{(ES)} \qquad \begin{aligned} S_j &\geqslant \max_{(w,j)\in A} (S_w + d_{wj}), \qquad j \in V, \\ S_w &\geqslant \min_{(j,w)\in A} (S_j + d_{jw}), \qquad w \in \mathcal{W}. \end{aligned}$$

In the above formula we added the term $d_{wj}$ for symmetry reasons. Without loss of generality we assume that $d_{wj} = 0$. The case $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and $d_{iw}$ by $d_{iw} + d_{wj}$ for all $i \in in(w)$. Besides schedules $S \in \mathbb{Z}^{|\mathcal{V}|}$ we also consider *partial schedules* $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$ where the start time of a job may be infinite, meaning that the job is not planned. As usual, a (partial) schedule fulfilling the constraints of (ES) is called *feasible*. In particular, the partial schedule $S = (\infty, \ldots, \infty)$ fulfills all inequalities of (ES) and is thus feasible. Moreover, it is easy to see that if $S'$ and $S''$ are feasible partial schedules, then their componentwise minimum $S := \min\{S', S''\}$ is also feasible. In particular, there always exists a (unique) componentwise minimal partial schedule $S^*$, called the *optimal* partial schedule (notice that $S^* \geqslant 0$ for all AND-nodes). It follows that, instead of considering the above system of inequalities, we alternatively may consider the corresponding system of equations (which is obtained from (ES) by replacing each "$\geqslant$" by "$=$").

Presuming different restrictions on the range of arc weights, several algorithms have been suggested to solve (ES). Note that all restrictions on arc weights are meant to refer to arcs $(j, w)$ between AND-nodes $j$ and OR-nodes $w$ only. For the case of nonnegative arc weights without cycles of zero length in $D$, a modification of Dijkstra's shortest path algorithm can be applied. An algorithm suggested by Knuth [16] has running time $O(|\mathcal{V}| \log |\mathcal{V}| + |A|)$. Other approaches are proposed in [6], [10], and, in the context of resource-constrained project scheduling, [14] (see also [22]). Levner, Sung, and Vlach [18] consider a generalized model of AND/OR precedence constraints where a so-called *threshold value* $1 \leqslant \ell_w \leqslant |X|$ is associated with each waiting condition $w = (X, j)$, indicating that $j$ may start if at least $\ell_w$ jobs from $X$ have been completed. They show that Dijkstra's shortest path algorithm can also be generalized to solve their model (with positive arc weights). For a discussion of the case with nonnegative arc weights *and* cycles of zero length we refer to subsection 7.4. The general case $-M < d_{jw} < M$ is a frequently studied problem with applications in many different areas, e.g., *game theory* [29] and *interface timing verification* (see [24] and [20]). Moreover, there are applications stemming from online optimization; see [29, section 7] for a collection of examples. Interestingly, although a pseudopolynomial algorithm to solve this case of (ES) is easily obtained, no algorithm polynomial in $|\mathcal{V}|$ and $\log(M)$ is currently known.

**7.2. Arbitrary arc weights.** In this section we study the case of arbitrary arc weights $-M < d_{jw} < M$.

**7.2.1. Feasibility.** For arbitrary arc weights the feasibility results stated in section 3 are no longer valid. They are based on the requirement that all $d_{jw} = p_j > 0$, and, consequently, an AND-node $j$ can start if and only if for all $(X, j) \in \mathcal{W}$ at least one $i \in X$ has previously been started (compare with condition (2.1)). However, if we

allow $d_{jw} \leqslant 0$, this is no longer the case. In what follows we derive a necessary and sufficient feasibility criterion for (ES) with arbitrary arc weights which generalizes the feasibility criterion given in Lemma 3.2. For the remainder of the section we call a set $\mathcal{W}$ of waiting conditions *feasible* if and only if there exists a feasible schedule for (ES).

Before we derive the criterion, we discuss how a given instance can be simplified without changing the optimal partial schedule $S^*$. First, we make the problem more restrictive by removing all but one incoming arc of each OR-node $w$. If the remaining arc $(j, w)$ fulfills $S_j^* + d_{jw} \leqslant S_w^*$, clearly, all inequalities of (ES) are still satisfied. Consequently, $S^*$ and the optimal (partial) schedule of the more restrictive instance coincide. In a similar fashion we can remove all but one incoming arc $(w, j)$ of each AND-node $j$ without changing the earliest start times. However, removing such arcs means relaxing the problem, and some more work has to be done in order to obtain the desired result.

LEMMA 7.1. *For each digraph $D$ representing a set of* AND/OR *precedence constraints, there exists a subdigraph $\bar{D}$ on the same set $\mathcal{V}$ of nodes with $|in_{\bar{D}}(j)| \leqslant 1$ for all* AND*-nodes $j \in V$ such that $S^* = \bar{S}^*$, where $\bar{S}^*$ denotes the optimal (partial) schedule of $\bar{D}$.*

*Proof.* We construct $\bar{D}$ by iteratively removing arcs $(w, j)$ from $D$ that do not affect the earliest start time of the AND-node $j$. By contradiction, assume that once all such arcs have been removed, there is some AND-node $j$ with $|in_{\bar{D}}(j)| > 1$. Thus, removing any incoming arc of $j$ reduces the earliest start time of $j$. Denote by $S^1$ and $S^2$ the optimal (partial) schedules obtained if two different incoming arcs $(w_1, j)$ and $(w_2, j)$ are removed from $\bar{D}$; then $S_j^* > S_j^1$ and $S_j^* > S_j^2$. Without loss of generality let $S_j^1 \leqslant S_j^2$; we define a new (partial) schedule $S$ through

$$S_i \;:=\; \min\{S_i^1 + S_j^2 - S_j^1, S_i^2\} \qquad \text{for all } i \in \mathcal{V}.$$

By definition, $S \leqslant S^2$ and $S_j = S_j^2$. For each arc $(w, j)$ with $w \neq w_2$ we have $S_j^2 \geqslant S_w^2$, which yields $S_j = S_j^2 \geqslant S_w^2 \geqslant S_w$. For $w = w_2$ we get $S_j = S_j^1 + S_j^2 - S_j^1 \geqslant S_w^1 + S_j^2 - S_j^1 \geqslant S_w$. We obtain $S_j \geqslant \max_{(w,j)\in A}(S_w)$. Furthermore, $S$ also fulfills all other inequalities of (ES), because both $S^1 + S_j^2 - S_j^1$ and $S^2$ fulfill the inequalities and so does its minimum $S$. Consequently, $S$ is a feasible (partial) schedule, which is a contradiction of the minimality of $S^*$ since $S_j < S_j^*$. ∎

For the subsequent presentation, recall the definition of a (generalized) cycle from section 2. Note that we assume all cycles to be directed cycles.

COROLLARY 7.2. *Let $\bar{D}$ be as in Lemma* 7.1*. Then all cycles in $\bar{D}$ have strictly positive length.*

*Proof.* Assume that there is a cycle $(w_1, j_1, w_2, j_2, \dots, w_k, j_k, w_1)$ of nonpositive length in $\bar{D}$. By definition of $\bar{D}$, $(w_\ell, j_\ell)$ is the only incoming arc for node $j_\ell$, $\ell = 1, \dots, k$. Thus, one can construct a feasible partial schedule for $\bar{D}$ satisfying

$$S_{w_1} = S_{j_1} = -1 \qquad \text{and} \qquad S_{w_\ell} = S_{j_\ell} = -1 + \sum_{q=2}^{\ell} d_{j_{q-1}w_q} \quad \text{for } \ell = 2, \dots, k.$$

With Lemma 7.1, this is also possible for the original digraph $D$, which yields a contradiction of the requirement $S_{j_1}^* \geqslant 0$. ∎

LEMMA 7.3. *A set of* AND/OR *precedence constraints with arbitrary arc weights is feasible if and only if each generalized cycle in $D$ contains a cycle of nonpositive length.*

*Proof.* Let $C$ be a generalized cycle which contains only cycles of positive length and suppose that some node $v \in C$ can be scheduled at $S_v < \infty$. Since $v$ has at least one incoming arc, there must exist a node $u \in in(v)$ with $S_v \geqslant S_u + d_{uv}$. Iterating this argument, since $|C|$ is finite, we obtain a cycle in $C$ with nonpositive length—a contradiction.

Conversely, suppose that the given instance is infeasible. Let $Z \neq \emptyset$ denote the set of nodes whose earliest start times are $\infty$. By Lemma 7.1, we can relax the problem by removing all but one incoming arc of each AND-node such that the earliest start times remain unchanged for the resulting digraph $\bar{D}$. Remove all OR-nodes from $Z$ whose out-degree is 0 in $\bar{D}$ and denote the resulting set of nodes by $Z'$. Then $Z'$ induces a generalized cycle $C$ in $D$. Moreover, by definition of $Z'$, every cycle in $C$ is also contained in $\bar{D}$ and therefore has positive length by Corollary 7.2. ∎

Lemma 7.3 reduces to Lemma 3.2 if all the arc weights $d_{jw}$ are strictly positive. Lemma 7.3 enables us to show that the decision problem of (ES) is in both NP and co-NP. The decision problem corresponding to (ES) is to decide whether or not a feasible schedule $S < \infty$ for (ES) exists.

LEMMA 7.4. *The decision problem corresponding to* (ES) *is in NP ∩ co-NP.*

*Proof.* It is clear that the decision problem corresponding to (ES) is in NP because, for a given feasible schedule $S$ of some instance $I$, it is easy to verify all constraints of $\mathcal{W}$. Moreover, it follows from Lemma 7.3 that the decision problem corresponding to (ES) is in co-NP. We can guess a generalized cycle violating the condition in Lemma 7.3, which can be verified in polynomial time by, for example, some standard minimum mean weight cycle algorithm. ∎

**7.2.2. A simple pseudopolynomial time algorithm.** For the case of (ES) with arbitrary arc weights several pseudopolynomial algorithms (partly independent of each other) have been proposed; see, e.g., [5] and [25] as well as [24] and [29]. A very simple (pseudopolynomial) algorithm is as follows: First, initialize $S_j := 0$ for all $j \in V$. Then, while $S$ violates some waiting condition $w = (X, j) \in \mathcal{W}$, set $S_j := \min_{i \in X}(S_i + d_{iw})$. If $S_j$ becomes larger than a given time horizon $T$, then stop and return that the given instance is infeasible. The time horizon $T$ can be chosen as $T := \sum_{j \in V}(\max_{w \in out(j)} |d_{jw}|)$. One can show straightforwardly by induction that $S \leqslant S^*$ in each iteration of the algorithm. If the recurrence stops with $S_j \leqslant T$ for all $j \in V$, then all constraints are obviously fulfilled. Hence $S \geqslant S^*$ and thus we have $S = S^*$. Moreover, at least one start time of a job is increased by 1 in each iteration. Thus the number of iterations is $O(|V| \cdot T)$. Finding a violated waiting condition obviously requires at most $O(|A|)$ time and thus the total complexity is $O(|V| \cdot |A| \cdot T)$. Note that for the special case that $D$ is acyclic, earliest job start times can easily be computed in linear time along a topological sort. Moreover, if each AND-node (OR-node) has at most one incoming arc, node start times can be computed by, for example, a slight modification of the Bellman–Ford shortest (longest) path algorithm in time $O(|V| \cdot |A|)$.

**7.2.3. A game-theoretic application.** We next consider a class of two-player games played on bipartite directed graphs which are directly related to the problem (ES). There exists substantial literature on different variations of this game; see, e.g., [29], [8], [15], [28], and references therein. Each player is identified with one of the node partitions of the graph. The game starts at a fixed node $j_0$ and the player associated with that node chooses an incoming arc $(w_0, j_0)$. Then, at node $w_0$, the other player chooses an incoming arc $(j_1, w_0)$ and so on. The objective and the stopping criterion depend on the considered variation of the game.

One variant is the so-called *mean payoff game* (MPG), where an integer weight is associated to each arc of the digraph. Furthermore, it is assumed that each node has at least one incoming arc. The MPG is finished as soon as the path $P$ resulting from the game contains a cycle and the *outcome* $\nu$ of the game is the mean weight of the arcs of that cycle. One player wants to maximize the outcome, while the other player wants to minimize it. It has been shown by Ehrenfeucht and Mycielski [8] that both players have positional optimal strategies; that is, the decisions of both players depend on neither previous choices nor the start node $j_0$. In the following we always assume that $j_0$ is associated with the maximization player.

The decision problem corresponding to MPG is to decide whether the outcome of the game is positive. Zwick and Paterson [29] have noted that this problem is in NP $\cap$ co-NP. Even more, Jurdziński [15] showed that the problem is in UP $\cap$ co-UP. It seems to be intuitively clear that MPG and (ES) are closely related. We next show that this is indeed the case.

LEMMA 7.5. *The decision problems corresponding to MPG and* (ES) *are polynomially equivalent.*

*Proof.* Given an instance of (ES), we construct an instance of MPG in the following way. First, we add an additional job $t$ and a waiting condition $w_j = (\{j\}, t)$ with $d_{jw_j} = 0$ for every job $j \in V$; moreover, we add a waiting condition $w = (\{t\}, s)$ with $d_{tw} = -T$, where $T$ is the time horizon discussed in section 7.2.2. Notice that there exists a feasible schedule for the original instance of (ES) if and only if the earliest start time of the new job $t$ is finite. The game digraph $D$ is now the digraph representing the new scheduling instance. The starting node is $j_0 := t$ and the maximization player starts. We show that the set of AND/OR precedence constraints is feasible if and only if $\nu \leqslant 0$.

*Only if*: Based on an optimal schedule $S^* < \infty$, we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node $w$, choose an incoming arc $(j, w)$ with $S^*_j + d_{jw} = S^*_w$. Then, for two vertices $v_1$ and $v_2$ on the path formed by the game, the weight of the (directed) subpath from $v_1$ to $v_2$ is at most $S^*_{v_2} - S^*_{v_1}$ (for each arc $(w, j)$ on the path we have $S^*_j \geqslant S^*_w$ and for each arc $(j, w)$ on the path we have $S^*_w = S^*_j + d_{jw}$). In particular, the length of the cycle terminating the game is at most 0 (choose $v_1 = v_2$).

*If*: For an infeasible scheduling instance it follows from Lemma 7.3 that there exists a generalized cycle $C$ in $D$ which contains only cycles of positive length. Without loss of generality, $C$ contains the node $t$ (if $t$ is not in $C$, then consider the generalized cycle where all waiting conditions $(\{j\}, t)$ with $j$ in $C$ are added to $C$). We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in $C$. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave $C$. This yields $\nu > 0$.

Given a digraph $D$ representing an instance of MPG, we construct an instance $I$ of (ES) in the following way. First, we assume without loss of generality that every node in $D$ associated to the minimization player has out-degree one—it is an easy observation that a node with out-degree $q > 1$ can be replaced by $q$ copies with out-degree one without changing the outcome of the game. Moreover, we assume that the weight of the only arc $(w, j)$ leaving a node $w$ associated to the minimization player is 0. The case of $d_{wj} \neq 0$ can be handled by replacing $d_{wj} \neq 0$ by 0 and $d_{iw}$ by $d_{iw} + d_{wj}$ for all $i \in in(w)$; recall the transformation in the second paragraph of section 7.1.

The set $V$ of jobs in the instance $I$ of (ES) is the set of nodes associated to the maximization player. For each node $w$ of the minimization player, we introduce a waiting condition $w = (in_D(w), j)$ where $out_D(w) = \{j\}$. The time lag $d_{iw}$ for $i \in in_D(w)$ is given by the corresponding arc weight in $D$. Moreover, we add a dummy start node $a$ preceding all other AND-nodes. We refer to this scheduling instance as $I'$. Finally, in order to obtain the instance $I$, we modify every waiting condition $w = (X, j)$ with $j \neq j_0$ and $j_0 \notin X$ by adding $j_0$ to $X$ and setting $d_{j_0 w} = T + 1$; in other words, if job $j_0$ can be planned, then all other jobs can be planned, too. In particular, instance $I$ is feasible if and only if the earliest start time $S^*_{j_0}$ of job $j_0$ in instance $I'$ is finite. Thus, it remains to show that $S^*_{j_0} < \infty$ if and only if $\nu \leqslant 0$.

*Only if*: Consider instance $I'$. Based on the optimal partial schedule $S^*$ of instance $I'$ ($S_{j_0} < \infty$), we give a strategy for the minimization player which ensures $\nu \leqslant 0$: In each OR-node $w$ with $S^*_w < \infty$, choose an incoming arc $(j, w)$ with $S^*_j + d_{jw} = S^*_w$. As a consequence, $S^*_i < \infty$ for all nodes $i$ visited during the game. Moreover, for two vertices $v_1$ and $v_2$ on the path formed by the game, the weight of the (directed) subpath from $v_1$ to $v_2$ is at most $S^*_{v_2} - S^*_{v_1}$. In particular, the length of the cycle terminating the game is at most 0.

*If*: For an infeasible scheduling instance $I$, by Lemma 7.3, there exists a generalized cycle $C$ in the corresponding digraph $D_I$ which contains only cycles of positive length. Without loss of generality, $C$ contains the node $j_0$. Notice that $C$ also forms a generalized cycle for the digraph $D$. We give a strategy for the maximization player which ensures $\nu > 0$: In each step, choose an arc which starts at a node in $C$. Such an arc always exists by the definition of generalized cycles. Moreover, again by the definition of generalized cycles, the minimization player is not able to leave $C$. This yields $\nu > 0$. □

With Lemma 7.5, it follows from [15] that the decision problem corresponding to the scheduling problem (ES) is in UP ∩ co-UP. Moreover, MPG and hence also (ES) can be computed in subexponential time: Zwick and Paterson [29] have shown that so-called *simple stochastic games* are at least as hard as MPGs. The outcome of simple stochastic games can be computed in subexponential time, as has been shown by Ludwig [19]. Despite these observations, there is no polynomial-time algorithm for (ES) with arbitrary arc weights known.

**7.3. Positive arc weights.** In this section we restrict ourselves to the case of positive arc weights or, more generally, nonnegative arc weights without cycles of length 0 in $D$. As in [16] and [6] we basically obtain a slight generalization of Dijkstra's shortest path algorithm. During the course of the algorithm we call a job *planned* as soon as its start time has been fixed.

The algorithm maintains a partial schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$ where initially $S_w = \infty$ for all OR-nodes $w$. All OR-nodes which are not yet planned are maintained in a heap where the sorting key for node $w$ is its tentative start time $S_w$ (initially $S_w = \infty$).

Having set $S_s = 0$ (and also $S_w = 0$ for all $w \in out(s)$) we proceed over time by always choosing an OR-node $w = (X, j)$ with minimum start time from the heap and plan $w$ at its tentative start time $S_w$. If all other OR-nodes $(X', j)$ preceding $j$ have already been planned, we also plan $j$ at the current time. In this case, the start times of all OR-nodes $w'$ with $w' \in out(j)$ are updated to $S_{w'} := \min\{S_{w'}, S_j + d_{jw'}\}$. If after termination some OR-node $w$ is started at $S_w = \infty$, the considered instance is infeasible. Implementational details are given in Algorithm 3.

If we apply Algorithm 3 to Example 1 (arc weights are given in Figure 2.1), we

**Algorithm 3:** Computation of earliest job start times for digraphs without cycles of length 0.

**Input** : A directed graph $D$ representing a set $V$ of jobs and waiting conditions $\mathcal{W}$ with positive arc weights on the arcs in $V \times \mathcal{W}$.

**Output**: A feasible (partial) schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$.

```
Heap := ∅;
for AND-nodes j ∈ V do a(j) := |in(j)|;
1 S_s := 0;    // AND-node s is planned at time 0
for OR-nodes w ∈ W do
    if w ∈ out(s) then insert w in Heap with key S_w := 0;
    else insert w in Heap with key S_w := ∞;

while Heap ≠ ∅ do
2   remove next OR-node w_0 = (X, j) from Heap;  // OR-node is planned
    reduce a(j) by 1;
    if a(j) = 0 then
3       S_j := max_{w∈in(j)} S_w; // AND-node is planned
        for OR-nodes w ∈ out(j) do
            S_w := min{S_w, S_j + d_jw};
            decrease key of w in Heap to S_w;
    delete node w_0 and all incident arcs from D;
return S;
```

obtain the start times $(0, 0, 0, 2, 3, 2, 3)$ for AND-nodes and $(2, 1, 2, 3, 3)$ for OR-nodes. One possible order in which start times get fixed is $j_1 \prec j_2 \prec j_3 \prec w_2 \prec w_1 \prec j_4 \prec w_3 \prec j_6 \prec w_5 \prec j_7 \prec w_4 \prec j_5$.

THEOREM 7.6. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with nonnegative arc weights and without cycles of length* 0*, Algorithm* 3 *computes an optimal partial schedule $S$. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node $w$.*

*Proof.* In this proof we say that an AND-node is *planned* if its start time is fixed (lines 1 and 3), while an OR-node is planned if it is removed from the heap (line 2).

By construction of Algorithm 3, $S$ is a feasible partial schedule. Assume that $S$ is not optimal and let $v$ be a node with $S_v > S^*_v$ and $S^*_v$ minimal. If $v$ is an AND-node, then there must exist an OR-node $w = (X, v)$ with $S_w = S_v > S^*_v \geqslant S^*_w$ and we set $v := w$. Otherwise, if $v$ is an OR-node $(X, j)$, then there must exist an AND-node $i \in X$ with $S^*_v = S^*_i + d_{iv}$ (otherwise $S^*$ is not minimal). Moreover, $S_i > S^*_i$. To see this suppose that $S_i = S^*_i$. Then, at the stage of Algorithm 3 where $S_i$ is planned, $v$ has already been removed from $D$. Since start times (in the order in which nodes are planned) are nondecreasing we have $S_v \leqslant S^*_i$, and hence $S_v \leqslant S_v + d_{iv} \leqslant S^*_i + d_{iv} = S^*_v$, a contradiction of $S_v > S^*_v$.

Since $v$ was chosen such that $S^*_v$ is minimal, we have $S^*_i \geqslant S^*_v = S^*_i + d_{iv}$. Thus $d_{iv} = 0$ and we set $v := i$. Iterating this argument, we can construct a cycle (since there are only finitely many nodes) of length 0—a contradiction. □

LEMMA 7.7. *Algorithm* 3 *can be implemented to run in* $O(|\mathcal{W}| \log |\mathcal{W}| + |A| + |V|)$ *time.*

*Proof.* Since each OR-node enters the heap precisely once, the while-loop is exe-

cuted $|\mathcal{W}|$ times. Each AND-node is planned only once and therefore the inner for-loop is executed at most $|A|$ times. If we choose a Fibonacci-heap for maintaining the OR-nodes, the cost of line 2 is $\log|\mathcal{W}|$ and we obtain the claimed running time. □

In contrast to previously proposed algorithms, the heap data structure maintains only OR-nodes, which leads to the improved running time $O(|\mathcal{W}|\log|\mathcal{W}|+|A|+|V|)$ instead of $O((|V|+|\mathcal{W}|)\log(|V|+|\mathcal{W}|)+|A|)$.

**7.4. Nonnegative arc weights.** As an extension of the case discussed in section 7.3 we present an $O(|V|+|A|\cdot|\mathcal{W}|)$ algorithm that is capable of dealing with arbitrary arc weights $d_{jw} \geqslant 0$ and thus with cycles of length 0 in $D$. Levner, Sung, and Vlach [18] observed that the algorithm proposed by Knuth [16] fails to compute earliest job start times when cycles of length 0 occur.

After submission of this paper for publication, we learned that Adelson-Velsky and Levner [1] also discovered a polynomial-time algorithm for the problem. Their algorithm proceeds over time starting at time $t = 0$. For each $t$, the algorithm determines all jobs to be started at $t$ by an appropriate labeling procedure which runs in $O(|A|)$ time. Thereafter, $t$ is increased to the next tentative start time of some job. It may happen, however, that for some $t$ considered, no job is started. Adelson-Velsky and Levner [1], [2] show that the number of times $t$ is increased is bounded by $|A|$. Consequently, they obtain an $O(|A|^2)$ algorithm. Due to a more careful update of (tentative) start times of jobs, our algorithm's worst-case running time is $O(|V|+|A|\cdot|\mathcal{W}|)$.

Adelson-Velsky and Levner [2] misleadingly claim that our algorithm solves only a special case of the problem (in which only a single arc leaves each OR-node). A digraph $D$ with multiple arcs $(w,j)$ leaving an OR-node $w$ can obviously be polynomially transformed into a digraph where only a single arc leaves each OR-node as follows: Add a new AND-node $i$, a new arc $(w,i)$, and add a waiting condition $(i,\{j\})$ for each arc $(w,j)$. Then remove all arcs $(w,j)$. With this transformation our algorithm's worst-case complexity is also $O(|A|^2)$ in the original input size. However, in Lemma 7.9 below we argue that our algorithm can also handle the case of multiple arcs leaving an OR-node directly without a transformation.

A rough scheme of the algorithm is as follows. Analogously to Algorithm 3 we maintain all OR-nodes $w$ in a heap where the sorting key is its tentative start time $S_w$ (initially $S_w = \infty$). Furthermore, whenever an AND-node $j$ is planned, the start times of all OR-nodes $w \in out(j)$ are updated to $S_w = \min\{S_w, S_j + d_{jw}\}$. We proceed over time starting at $t = 0$. For the current time $t$ we compute a set $U$ of (nonstarted) nodes that can be started at $t$. The set $U$ is computed by maintaining the induced subgraph $D^0$ of $D$ where all planned nodes and all arcs of positive weight have been deleted. In $D^0$, the set $U$ is computed as a set of nodes such that for each AND-node $j$, all predecessors $w \in in_{D^0}(j)$ are also in $U$, and for each OR-node $w$, at least one predecessor $j \in in_{D^0}(w)$ is also in $U$. Then, as we will prove in Theorem 7.8 below, all nodes of $U$ can be started at the current time $t$. Next we remove a new OR-node $w$ from the heap and increase $t$ to $S_w$. If $t = \infty$, the algorithm stops. Then either no OR-node was left in the heap (and we have computed a feasible schedule) or all OR-nodes $w$ in the heap fulfill $S_w = \infty$ (indicating that the given instance is infeasible). Details are provided in Algorithm 4.

If we apply Algorithm 4 to Example 1 with arc weights as in Figure 2.1 except $d_{5w_1} = 0$ and $d_{4w_4} = 0$, we get the following:

Iteration 1: $U = \{j_1, j_2, j_3\}$, $w = w_2$, $t := 1$.

Iteration 2: $U = \{j_4, j_5, w_1, w_4\}$, $w = w_3$, $t := 2$.

**Algorithm 4:** Computation of earliest job start times for nonnegative time lags.

**Input** : A directed graph $D$ representing a set $V$ of jobs and waiting conditions $\mathcal{W}$ with nonnegative arc weights on the arcs in $V \times \mathcal{W}$.

**Output**: A feasible (partial) schedule $S \in (\mathbb{Z} \cup \{\infty\})^{|\mathcal{V}|}$.

```
set D^0 := D and remove all arcs with positive weight from D^0;
t := 0;
Heap := ∅;
for OR-nodes w ∈ W do
    S_w := ∞;
    insert w in Heap with key S_w;
while t < ∞ do
    compute U ⊆ V(D^0) maximal with
1       (in_{D^0}(j) ⊆ U   ∀ j ∈ U ∩ V) and (in_{D^0}(w) ∩ U ≠ ∅   ∀ w ∈ U ∩ W);
    for AND-nodes j ∈ U  (j ∈ U ∩ V)  do
        S_j := t;        // node j is planned at time t
        for OR-nodes w ∈ out_D(j) do
2           S_w := min{S_w, S_j + d_jw};
3           decrease key of w in Heap to S_w;
    for OR-nodes w ∈ U  (w ∈ U ∩ W)  do
4       S_w := t;        // node w is planned at time t
        remove w from Heap;
    Delete all nodes from U in D and D^0;
    if Heap ≠ ∅ then
5       remove the next OR-node w from Heap;
6       t := S_w;
7       remove w from D and D^0;        // node w is planned at time t
    else  t := ∞;
return S;
```

Iteration 3: $U = \{j_6\}$, $w = w_5$, $t := 2$.
Iteration 4: $U = \{j_7\}$, $Heap = \emptyset$, $t := \infty$.

Thus, we obtain start times $(0, 0, 0, 1, 1, 2, 2)$ for AND-nodes and $(1, 1, 2, 1, 2)$ for OR-nodes.

THEOREM 7.8. *For a given set of* AND/OR *precedence constraints represented by a digraph $D = (V \cup \mathcal{W}, A)$ with nonnegative weights on the arcs, Algorithm* 4 *computes an optimal partial schedule $S$. In particular, the instance is infeasible if and only if $S_w = \infty$ for some* OR*-node $w$.*

*Proof.* We first prove that the variable $t$ never decreases; i.e., the algorithm proceeds over time and tries to plan the jobs (and remove them from $D$ and $D^0$) as early as possible in order of nondecreasing start times. Assume that $t$ decreases in line 6 of the algorithm and let $t_0$ denote its value before the decrease. Since the OR-node $w$ determining $t$ was not chosen in line 5 during the last iteration of the while-loop (when $t$ was set to $t_0$), its tentative start time $S_w$ has decreased during the current iteration in lines 2 and 3. This is a contradiction of $S_j = t_0$ and $d_{jw} \geqslant 0$.

Observe that the start time $S_i$ of any node $i \in \mathcal{V}$ is never changed after the node is planned (and thus deleted from the graphs $D$ and $D^0$).

We can now prove that the partial schedule $S$ returned by Algorithm 4 is feasible by verifying all constraints of (ES). By construction of the algorithm, for an AND-node $j \in V$, every OR-node $w \in in(j)$ either has been planned before or is planned together with $j$ in the same iteration of the while-loop; this follows from the first property of $U$ in line 1. Thus, the constraint in (ES) corresponding to $j$ is fulfilled.

Consider now an arbitrary OR-node $w \in \mathcal{W}$. If $w$ is planned as part of a subset $U$ in line 4, it follows from the second property of $U$ in line 1 that there is a job $j \in in(w)$ with $d_{jw} = 0$, and $j$ is planned at the same time as $w$. Otherwise, if $w$ is planned in line 7 and $S_w < \infty$, the start time $S_w$ of $w$ must have been decreased in some iteration of the while-loop in line 2; since the start time $S_j$ of the node $j \in V$ causing the last decrease of $S_w$ has not changed since then, $S_w = S_j + d_{jw}$ in the final partial schedule $S$. Thus, the constraint in (ES) corresponding to $w$ is fulfilled.

Next we prove that the partial schedule $S$ returned by Algorithm 4 is optimal. Let $S^*$ be the optimal partial schedule and assume that there are nodes $i \in \mathcal{V}$ with $S_i^* < S_i$; we choose such an $i'$ with minimum $S_{i'}^*$ and set $t_0 := S_{i'}^*$; let $U_0 = \{i \in \mathcal{V} \mid t_0 = S_i^* < S_i\}$. We distinguish two cases.

*First case.* In some iteration of Algorithm 4, $t$ adopts the value $t_0$. We consider the iteration of the while-loop in which $t$ is increased above $t_0$ in line 6. Let $D^0$ be the digraph at the beginning of the iteration and $U$ the set computed at the start of this iteration. Then $U \cap U_0 = \emptyset$ and, by maximality of $U$, the set $U \cup U_0$ cannot satisfy the conditions in line 1. Since $S^*$ is a feasible partial schedule, the first condition of line 1 is valid for $U \cup U_0$, i.e., $in_{D^0}(j) \subseteq U \cup U_0$ for all $j \in (U \cup U_0) \cap V$. Thus, the second condition is violated: there exists a node $w \in U_0 \cap \mathcal{W}$ with $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$. Moreover, by optimality of $S^*$, there exists a node $j \in in_D(w)$ with $S_w^* = S_j^* + d_{jw}$, in particular $S_j^* \leqslant t_0$. We next show that $S_j = S_j^*$. If $S_j^* < t_0$, the claim follows from the minimality of $t_0$. Otherwise, observe that $j \notin U_0$ (if $j \in U_0$, we have $j \in in_{D^0}(w)$, which contradicts $in_{D^0}(w) \cap (U \cup U_0) = \emptyset$). Then with $S_j^* = t_0$ and $j \notin U_0$ it follows from the definition of $U_0$ that $S_j = S_j^*$. In particular, $S_w$ has been set to $S_j + d_{jw} = S_w^*$ in line 2 after $j$ was planned. Since $S_w$ is never increased in Algorithm 4, we get a contradiction of $S_w > S_w^*$.

*Second case.* The variable $t$ never adopts the value $t_0$ in Algorithm 4; in particular, $t_0 > 0$ and $U_0 = \{i \in \mathcal{V} \mid S_i^* = t_0\}$. Since $S^*$ is optimal, decreasing all start times $S_j^*$ for $j \in U_0$ to $t_0 - 1$ violates a constraint of (ES). Thus, there exists a node $w \in U_0 \cap \mathcal{W}$ such that $S_w^* = S_j^* + d_{jw}$ for some $j \in V$ with $d_{jw} > 0$, i.e., $S_j^* < t_0$. Therefore, $S_j = S_j^*$ and $S_w$ has been set to $S_j + d_{jw} = S_w^*$ in line 2 after $j$ was planned. Since $S_w$ is never increased in Algorithm 4, we get a contradiction of $S_w > S_w^*$. ∎

The bottleneck for the running time of Algorithm 4 is the computation of the set $U$ in each iteration of the while-loop. In fact, it turns out that the linear-time algorithm for checking feasibility of a set of AND/OR precedence constraints (for the case of positive arc weights) provides an elegant and fast solution for this problem.

LEMMA 7.9. *Given a bipartite digraph $D$ with node set $N \cup M$ and arc set $A$, the (unique) maximal set $U \subseteq N \cup M$ with $in_D(w) \subseteq U$ for all $w \in U \cap N$ and $in_D(j) \cap U \neq \emptyset$ for all $j \in U \cap M$ can be computed in linear time.*

*Proof.* First, for $U$ and $U'$ fulfilling the conditions given in the lemma, their union $U \cup U'$ also fulfills those conditions. Therefore, such a unique maximal subset $U$ exists.

We show that $U$ can be computed by applying essentially Algorithm 1 to an appropriately constructed instance. Define the set $V = M$ of jobs and the following set $\mathcal{W}$ of waiting conditions: For each $w \in N$ and each $j \in out_D(w)$, introduce a waiting condition $(in_D(w), j)$. Notice that the input size of this instance is not necessarily

linear in the input size of the given digraph $D$ since the set $in_D(w)$ is stored once for every $j \in out_D(w)$. We can avoid this undesired increase in the input size by storing, for each $w \in N$, the corresponding waiting conditions as $(in_D(w), out_D(w))$ with the interpretation that every job in the second set is a waiting job for the first set. Algorithm 1 can easily be adapted to handle this compactified input in linear time by replacing the for-loop starting in line 1 with

**for** *waiting conditions* $(X, Y) \in \mathcal{W}$ *with* $i \in X$ **do**
  **for** $j \in Y$ **do**
    decrease $a(j)$ by 1;
    **if** $a(j) = 0$ **then** add $j$ to $Q$;
  remove $(X, Y)$ from $\mathcal{W}$;

By Corollary 3.3, Algorithm 1 computes a set $L \subseteq V$ such that $V' := V \setminus L$ is a maximal subset of $V$ with the following property: For all $j \in V'$ there exists a waiting condition $(X, j) \in \mathcal{W}$ with $X \subseteq V'$. Thus, the set

$$U = \big(\{w \in N \mid in(w) \subseteq V'\} \cup V'\big) \subseteq N \cup M$$

fulfills the conditions given in the lemma, i.e., $in_D(w) \subseteq U$ for all $w \in U \cap N$ and $in_D(j) \cap U \neq \emptyset$ for all $j \in U \cap M$. Assume that there is a bigger set $U^* \supset U$ that also fulfills these conditions. By construction of $U$, there exists a node $j \in M \cap (U^* \setminus U)$. Since the set $U^* \cap M$ of jobs has the property described in Corollary 3.3, we get a contradiction of the maximality of $V'$. □

With the help of this lemma, we can now give a bound on the running time of Algorithm 4.

COROLLARY 7.10. *Algorithm* 4 *can be implemented to run in* $\mathrm{O}(\,|\mathcal{W}| \cdot |A| + |V|\,)$ *time.*

*Proof.* First, all isolated AND-nodes are planned and thus removed from $D^0$ in the first iteration of the while-loop. Moreover, in each iteration, at least one OR-node is removed from $D^0$ and the number of iterations is thus bounded by $|\mathcal{W}|$. Finally, the running time of each iteration is dominated by the computation of $U$, which can be done in $\mathrm{O}(\,|A|\,)$ time. □

Notice that, in the sense of Lemma 7.9, Algorithm 4 and its worst-case complexity (Corollary 7.10) are both valid for digraphs $D$ where OR-jobs have multiple outgoing arcs (of length 0).

**8. Concluding remarks.** The contribution of the paper is twofold. On the one hand we have provided efficient algorithms for various basic problems that occur when scheduling jobs subject to AND/OR precedence constraints (i.e., generalized feasibility, transitivity, and the computation of earliest start times of jobs for nonnegative arc weights). On the other hand we have provided further insights for solving the problem (ES) with arbitrary arc weights ($-M \leq d_{jw} \leq M$) that may help to design a polynomial-time algorithm for this important problem. In particular, the feasibility criterion (Lemma 7.3) and the algorithm for nonnegative arc weights (Algorithm 4) may be helpful.

**Acknowledgments.** We would like to thank Gerhard Woeginger for helpful discussions and for bringing the references [29], [15], and [24] to our attention. Furthermore, we thank Martin Zachariasen for pointing us to the work of Adelson-Velsky and Levner [1].

## REFERENCES


[1] G. M. Adelson-Velsky and E. Levner, *Project Scheduling in AND/OR Graphs: A Generalization of Dijkstra's Algorithm*, Technical report, Department of Computer Science, Holon Academic Institute of Technology, Holon, Israel, 1999.

[2] G. M. Adelson-Velsky and E. Levner, *Project scheduling in AND/OR graphs: A generalization of Dijkstra's algorithm*, Math. Oper. Res., 27 (2002), pp. 504–517.

[3] G. Ausiello, A. d'Atri, and D. Saccà, *Graph algorithms for functional dependency manipulation*, J. ACM, 30 (1983), pp. 752–766.

[4] G. Ausiello, A. d'Atri, and D. Saccà, *Minimal representation of directed hypergraphs*, SIAM J. Comput., 15 (1986), pp. 418–431.

[5] F. Chauvet and J.-M. Proth, *The PERT-Problem with Alternatives: Modelisation and Optimisation*, Technical report, Institut National de Recherche en Informatique et en Automatique (INRIA), France, 1999.

[6] E. A. Dinic, *The fastest algorithm for the PERT problem with AND- and OR-nodes (the new-product-new technology problem)*, in Proceedings of the International Conference on Integer Programming and Combinatorial Optimization, R. Kannan and W. R. Pulleyblank, eds., University of Waterloo Press, Waterloo, Canada, 1990, pp. 185–187.

[7] W. F. Dowling and J. H. Gallier, *Linear-time algorithms for testing satisfiability of propositional Horn formulae*, J. Logic Program., 1 (1984), pp. 267–284.

[8] A. Ehrenfeucht and J. Mycielski, *Positional strategies for mean payoff games*, Internat. J. Game Theory, 8 (1979), pp. 109–113.

[9] G. Gallo, C. Gentile, D. Pretolani, and G. Rago, *Max Horn SAT and the minimum cut problem in directed hypergraphs*, Math. Programming, 80 (1998), pp. 213–237.

[10] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, *Directed hypergraphs and applications*, Discrete Appl. Math., 42 (1993), pp. 177–201.

[11] D. W. Gillies, *Algorithms to Schedule Tasks with AND/OR Precedence Constraints*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1993.

[12] D. W. Gillies and J. W.-S. Liu, *Scheduling tasks with AND/OR precedence constraints*, SIAM J. Comput., 24 (1995), pp. 797–810.

[13] M. H. Goldwasser and R. Motwani, *Complexity measures for assembly sequences*, Internat. J. Comput. Geom. Appl., 9 (1999), pp. 371–418.

[14] G. Igelmund and F. J. Radermacher, *Algorithmic approaches to preselective strategies for stochastic scheduling problems*, Networks, 13 (1983), pp. 29–48.

[15] M. Jurdziński, *Deciding the winner in parity games is in $UP \cap co\text{-}UP$*, Inform. Process. Lett., 68 (1998), pp. 119–124.

[16] D. E. Knuth, *A generalization of Dijkstra's algorithm*, Inform. Process. Lett., 6 (1977), pp. 1–5.

[17] B. Korte, L. Lovász, and R. Schrader, *Greedoids*, Springer-Verlag, Berlin, 1991.

[18] E. Levner, S. C. Sung, and M. Vlach, *Makespan minimization in projects with threshold activities*, Asia-Pacific J. Oper. Res., 19 (2002), pp. 195–204.

[19] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Inform. and Comput., 117 (1995), pp. 151–155.

[20] K. L. McMillan and D. L. Dill, *Algorithms for interface timing verification*, in Proceedings of the IEEE International Conference on Computer Design, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 48–51.

[21] R. H. Möhring, M. Skutella, and F. Stork, *Forcing relations for AND/OR precedence constraints*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00), ACM, New York, SIAM, Philadelphia, 2000, pp. 235–236.

[22] R. H. Möhring and F. Stork, *Linear preselective policies for stochastic project scheduling*, Math. Methods Oper. Res., 52 (2000), pp. 501–515.

[23] N. J. Nilsson, *Principals of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

[24] U. Schwiegelshohn and L. Thiele, *Dynamic min-max problems*, Discrete Event Dyn. Syst., 9 (1999), pp. 111–134.

[25] C. Schwindt, *A Branch-and-Bound Algorithm for the Resource-Constrained Project Duration Problem Subject to Temporal Constraints*, Technical report 544, WIOR, University of Karlsruhe, Germany, 1998.

[26] F. Stork, *Stochastic Resource-Constrained Project Scheduling*, Ph.D. thesis, Department of Mathematics, Technische Universität Berlin, 2001.

[27] J. D. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, Rockville, MD, 1982.

[28] J. Vöge and M. Jurdziński, *A discrete strategy improvement algorithm for solving parity games*, in Proceedings of the 12th International Conference of Computer Aided Verification, CAV 2000, E. A. Emerson and A. P. Sistla, eds., Lecture Notes in Comput. Sci. 1855, Springer-Verlag, Chicago, IL, 2000, pp. 202–215.

[29] U. Zwick and M. Paterson, *The complexity of mean payoff games on graphs*, Theoret. Comput. Sci., 158 (1996), pp. 343–359.