Technische Universität Berlin

FACHBEREICH 3
MATHEMATIK

Forcing Relations for AND/OR Precedence Constraints

by

Rolf H. Möhring Martin Skutella
Frederik Stork

No. 646/1998

Forcing Relations for AND/OR Precedence Constraints

Rolf H. Möhring* Martin Skutella* Frederik Stork*

July 31, 1999

Abstract

A natural generalization of ordinary precedence constraints are so-called AND/OR precedence constraints. In an AND constraint, a job must wait *for all* its predecessors while in an OR constraint, a job has to wait *for at least one* of its predecessors. We provide a linear-time algorithm for deducing additional AND/OR precedence constraints that are implied by the given ones. We show that this algorithm can also be used to verify feasibility of the given constraints. Besides their theoretical value, these results have significant impact in practical applications such as scheduling and assembly sequencing; we show how to use our algorithm to improve solution procedures for resource-constrained project scheduling problems. Finally, we prove that for a related, more general model, the problems under consideration are strongly NP-complete.

1 Introduction

For a set V of activities or jobs, precedence constraints are usually given by a set E of ordered pairs (i, j), $i \neq j \in V$, inducing an acyclic digraph (V, E) with the meaning that job j can only start after the completion of *all* jobs i with $(i, j) \in E$. In addition to these AND or *conjunctive* precedence constraints, one may quite naturally consider *disjunctive* (OR) precedence constraints. Such constraints are given by pairs (X, j), $X \subset V$, $j \in V \setminus X$, where job j may be started if *at least one* job $i \in X$ has been completed. We call j the *waiting job* for (X, j) and denote the set of all *waiting conditions* (X, j) by $\mathcal{W}$. Notice that for a singleton $X = \{i\}$, the waiting condition (X, j) is an ordinary AND precedence constraint (i, j).

An application of AND/OR precedence constraints in resource-constrained project scheduling is described below. Another interesting application has been given by Goldwasser and Motwani [1]. They consider the problem of partially

* TU Berlin, Department of Mathematics, Sekr. MA 6–1, Straße des 17. Juni 136, D-10623 Berlin, Germany, {moehring, skutella, stork}@math.tu–berlin.de. The second author is supported by DONET within the frame of the TMR Programme (contract number ERB FMRX-CT98-0202). The third author is supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Mo 446/3-3.

disassembling a given product to reach a single part (or component) of the product. In order to remove a certain part, one previously may have to remove other parts which can be modeled by ordinary (AND) precedence constraints. However, one may choose to remove that same part of the product from another (geometric) direction, in which case some different parts may have to be removed previously. This freedom of choice can be modeled by disjunctive (OR) precedence constraints.

2 Detection of Waiting Conditions

For a given set of jobs V with waiting conditions $\mathcal{W}$, we propose a linear-time algorithm to detect new waiting conditions that are forced by $\mathcal{W}$; that is, for given $U \subset V$, we want to identify all jobs that have to wait for at least one job in U. Interestingly, this problem is closely related to the problem of checking feasibility of $\mathcal{W}$, i. e., whether there exists a schedule (an assignment of start times to jobs) obeying $\mathcal{W}$. For the case of ordinary precedence constraints both problems can easily be solved: The feasibility problem can be solved by testing whether the underlying precedence digraph is acyclic and forced precedence constraints can be deduced by any transitive closure algorithm.

For the case of waiting conditions, the following algorithm solves both problems in linear time.

Algorithm 1.
Initialization. *Let V, $\mathcal{W}$, and $U \subset V$ be given and let L be an empty list.*
Recurrence. *While there exists a job $i \in V \setminus U$ that is not a waiting job of any of the waiting conditions in $\mathcal{W}$, insert it at the end of L. Whenever a waiting condition (X, j) becomes satisfied (which is the case when some $i \in X$ has already been added to L), delete (X, j) from $\mathcal{W}$.*

The following two theorems contain the main results of this extended abstract.

Theorem 1. *A set $\mathcal{W}$ of waiting conditions is feasible if and only if, for the input $U = \emptyset$, the list L obtained from Algorithm 1 contains all jobs of V.*

Theorem 2. *Let $\mathcal{W}$ be a set of feasible waiting conditions, $U \subset V$, and L the ordering obtained from Algorithm 1. Then, the set of waiting conditions (U, j) which are forced by $\mathcal{W}$ is precisely $\{(U, j) \mid j \notin (L \cup U)\}$.*

To give a basic intuition of the proofs, we introduce the concept of *realizations.* An acyclic digraph $R = (V, E_R)$ is called a realization of V and $\mathcal{W}$ if for every waiting condition $(X, j) \in \mathcal{W}$, there is some $i \in X$ with $(i, j) \in E_R$. Obviously, all waiting conditions are fulfilled when scheduling jobs in the (partial) order defined by R, i. e., starting each job as soon as all its predecessors in R are completed. Conversely, every feasible schedule defines such a realization by choosing the arcs of R according to the schedule. Notice that any extension of a realization $R = (V, E_R)$ to a total order $R' = (V, E_{R'})$, $E_R \subseteq E_{R'}$, is also a realization. Therefore, a collection $\mathcal{W}$ of waiting conditions is feasible if and only if there exists a total order of the set of jobs which is a realization. For the input $U = \emptyset$, Algorithm 1 tries to compute such a realization. The proof of Theorem 1 is based on the following insight.

Lemma 1. *A set of waiting conditions is infeasible if and only if there exists a non-empty set $V' \subseteq V$ such that for all $j \in V'$ there is a waiting condition (X, j) with $X \subseteq V'$.*

Notice that the set of jobs $V' = V \setminus L$ remaining after the execution of Algorithm 1 (with input $U = \emptyset$) always fulfills this condition.

Lemma 1 is already implicitly contained in the work of Igelmund and Radermacher [2] who considered the following variant of Algorithm 1 (for the special case $U = \emptyset$): For given non-negative processing times of jobs, their algorithm tries to compute an earliest start schedule by planning jobs one after another, always choosing the one with the currently smallest possible starting time with respect to the waiting conditions $\mathcal{W}$. Thus, a difference to Algorithm 1 is the special choice of the job added to L in each iteration. In particular, jobs are (implicitly) sorted by their starting times which implies that the running time of this algorithm is super-linear in general.

The proof of Theorem 2 is based on the observation that a waiting condition (U, j) is forced by $\mathcal{W}$ if and only if, for every realization $R = (V, E_R)$, there exists a predecessor i of j with $i \in U$. One can show that, after the execution of Algorithm 1, the remaining jobs $V \setminus L$ are exactly those which cannot occur before the entire set U in any realization.

3 An Application in Project Scheduling

We give an application of Algorithm 1 that has its origin in resource-constrained project scheduling. Here, both (ordinary) precedence constraints and resource constraints have to be taken into consideration. Resource constraints can be represented by so-called *forbidden sets*, i. e., inclusion-minimal sets of pairwise not related jobs that cannot be scheduled simultaneously because they share some common limited resources. In order to resolve the resource conflict that occurs due to a 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 (or partial solutions) of project scheduling problems by a set $\mathcal{W}$ of waiting conditions.

Quite some solution procedures for such resource-constrained project scheduling problems iteratively select waiting jobs for the given forbidden sets. It is then of great importance to decide whether, for a forbidden set F, the set of waiting conditions arising from the given precedence constraints and the choices of waiting jobs for previously considered forbidden sets already force a waiting condition $(F \setminus \{j\}, j)$ for some job $j \in F$. Algorithm 1 gives the answer to this question. Stork [3] reports on a considerable speedup when using Algorithm 1 within a branch and bound scheme for resource-constrained project scheduling.

4 An NP-complete generalization

We conclude with the following related result. The AND/OR precedence constraints considered above have a natural generalization: For two sets $X, X' \subseteq V$, the *generalized waiting condition* (X, X') is fulfilled if at least one job $j \in X'$ is waiting for at least one job $i \in X$.

Theorem 3. *Given a set of jobs V with a set $\mathcal{W}$ of generalized waiting conditions, it is already strongly NP-complete to decide whether an ordinary precedence constraint (i, j), $i, j \in V$, is forced by $\mathcal{W}$. In particular, it is strongly NP-complete to decide feasibility of $\mathcal{W}$.*

Proof. The proof uses a reduction from SAT. For each Boolean variable x we introduce two jobs which correspond to the two literals x and $\bar{x}$ (negation of x), and for each clause C we introduce a corresponding job and a waiting condition $(X_C, \{C\})$ (X_C denotes the set of literals in C). Finally, we introduce two additional jobs s and t together with the waiting conditions $(\{s\}, \{x, \bar{x}\})$ for all x and $(\{C\}, \{t\})$ for all C. Then, job t has to wait for job s if and only if the underlying instance of SAT does not have a satisfying truth assignment. □

References

[1] M. H. Goldwasser and R. Motwani. Complexity measures for assembly sequences. *International Journal of Computational Geometry and Applications*, 9:371–418, 1999.

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

[3] F. Stork. A branch-and-bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints. Technical Report 613/1998, Technical University of Berlin, Department of Mathematics, Germany, 1998.