On the Representation of Resource Constraints in Project Scheduling

by

Frederik Stork Marc Uetz

No. 693/2000

On the Representation of Resource Constraints in Project Scheduling

Frederik Stork[*] [§] Marc Uetz[*] [‡]

February 9, 2001

Abstract

In project scheduling, resource constraints are usually defined via resource consumption and -availability. Many algorithmic approaches, however, are based on the concept of minimal forbidden sets to represent the resource constraints. Jobs of a forbidden set can be scheduled simultaneously with respect to the precedence constraints, however, they consume more resources than available. Forbidden sets are usually not given explicitly, and by definition even the number of inclusion-minimal forbidden sets may be exponential in the number of jobs. In this paper, we analyze some algorithmic questions related to these different respresentations, and we propose a simple backtracking algorithm to efficiently compute a minimal forbidden set representation. We evaluate the algorithm on well established test sets of the project scheduling problem library PSPLIB. In addition, we exhibit a close relation between the different representations of resource constraints and threshold hypergraphs.

Keywords: Project Scheduling, Resource Constraints, Forbidden Sets, Threshold Hypergraphs, Threshold Dimension, Minimal Cover Inequalities

1 Introduction

We consider scheduling problems where a set $V = \{1,2,\dots,n\}$ of jobs has to be executed subject to both precedence and resource constraints. Precedence constraints are given as an acyclic directed graph $D = (V,A)$, where $(i,j) \in A$ if j cannot be started before i has been completed. In addition, jobs need different (renewable) resource types $k \in K$ while being processed. A constant amount of $R_k \in \mathbb{N}$ units of each resource type is available throughout the project and each job j consumes $r_{jk} \leq R_k$ ($r_{jk} \in \mathbb{N}$) units of resource $k \in K$ while in process. A schedule is called feasible if it respects all precedence constraints and at any time t and for each resource type k, the sum of the resource consumption of all jobs which are in process at t does not exceed the availability R_k. This is the most common representation of resource constraints in project

[*]Author's addresses: Technische Universität Berlin, Fachbereich Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D-10623 Berlin, Germany. Email: {stork, uetz}@math.tu-berlin.de

[§]Supported by the Deutsche Forschungsgemeinschaft (DFG), grant Mo 446/3-3.

[‡]Supported by the German-Israeli Foundation for Scientific Research and Development (GIF), grant I 246-304.02/97.

scheduling. Let us call it the *threshold representation* (the motivation for this notation will become clear in Section 2). To give an example, scheduling problems on m (parallel) machines arise as the special case where exactly one resource is available in m units, and all jobs require exactly one unit of that resource.

The topic of this paper is an alternative representation of resource constraints, the so-called *(minimal) forbidden set representation*. A subset $F \subseteq V$ of jobs is called *forbidden* if the jobs in F are an anti-chain of the partial order defined by the precedence digraph D, and the total resource consumption $\sum_{j \in F} r_{jk}$ exceeds the resource availability R_k for some $k \in K$. F is called *minimal forbidden* if any proper subset $F' \subset F$ is *resource-feasible*, that is, $\sum_{j \in F'} r_{jk} \leq R_k$ for all $k \in K$. Let us denote by $\mathcal{F}$ the system of minimal forbidden sets, then $|\mathcal{F}|$ can obviously be exponential in n, the number of jobs. For a parallel machine scheduling problem with m machines, for example, the minimal forbidden sets are exactly the anti-chains of cardinality $m+1$.

Minimal forbidden sets are an important concept to represent resource constraints. In fact, they form the basis of numerous algorithmic approaches to resource-constrained project scheduling. Probably the most important field of application is stochastic scheduling, where job processing times are uncertain, e. g. in [15, 14, 19, 21, 26], but they also play a role in algorithmic approaches to deterministic project scheduling problems, e. g. in [24, 6]. In addition, forbidden sets are useful to derive cutting planes within integer programming approaches, e. g. in [1, 20]. Interestingly, Schäffter [25] derives inapproximability results for resource-constrained project scheduling problems by means of the forbidden set representation of resource-constraints: He proves that vertex coloring in graphs reduces to scheduling subject to forbidden sets, hence all inapproximability results for vertex coloring, e. g. in [10], also hold for resource-constrained project scheduling. Finally, forbidden sets can be seen as a generalization of the *disjunctive graph* concept known from job-shop scheduling, as pointed out by Radermacher [24]; see also [2].

The paper is organized as follows. In Section 2, we first discuss theoretical issues related to the threshold and minimal forbidden set representations of resource constraints, revealing a close relation to threshold (hyper-)graphs. In Section 3, we propose a backtracking algorithm which computes the system $\mathcal{F}$ of minimal forbidden sets for an instance which is given by the usual threshold representation. We show that, for instances with only one resource type ($|K| = 1$), the algorithm can be implemented to run in polynomial time with respect to the in- and output. A computational evaluation of the algorithm is presented in Section 4, based on the widely used instances from the project scheduling library PSPLIB [23]. The results exhibit the benefits of the proposed algorithm in comparison to an approach to compute $\mathcal{F}$ previously suggested by Bartusch [5]. Our results also provide further insights in the structure of the instances of the library. We conclude with some final remarks in Section 5.

2 Threshold and Forbidden Set Representations

In this section, we address several questions which are related to the transformation between the two above mentioned representations of resource constraints.

2.1 Relations to Threshold (Hyper-)Graphs

One can think of the system of minimal forbidden sets $(V,\mathcal{F})$ as an undirected hypergraph where jobs of the scheduling instance correspond to the vertices of the hypergraph and the minimal forbidden sets correspond to hyperedges. Let us first address the question if these hypergraphs have any particular property. To start with, consider the following problem: Given a problem instance with a minimal forbidden set representation of the resource constraints, what is the minimal number of resource types k required in a threshold representation? Obviously, $|\mathcal{F}|$ different resource types suffice, and it is easy to see that one resource type does not suffice in general, e. g. with $V=\{1,2,3,4\}$ and $\mathcal{F}=\{\{1,2\},\{3,4\}\}$. Moreover, as demonstrated by Example 1 in the Appendix, the number of resource types required in a threshold representation can be exponential in n, the number of jobs.

It turns out that exactly the same problem has been studied in the context of *threshold (hyper-)graphs*: According to Golumbic [12], a threshold hypergraph is an undirected hypergraph $(V,\mathcal{F})$, $\mathcal{F}\subseteq 2^V$, with the following property: A non-negative integer value r_j can be assigned to each vertex $j\in V$ such that there is an integer *threshold* R with the property that a subset $B\subseteq V$ is *stable* if and only if $\sum_{j\in B} r_j \le R$. Here, a *stable set* of a hypergraph is a subset $B\subseteq V$ which does not contain any hyperedge, that is, $F\nsubseteq B$ for all $F\in\mathcal{F}$; see [9]. In other words, the system of stable sets of a threshold hypergraph can be represented by only one linear inequality, namely $\sum_{j\in V} r_j x_j \le R$. Here, $x=(x_1,\ldots,x_n)$ is the characteristic vector of a subset X of V, where $x_j=1$ if $j\in X$ and $x_j=0$ otherwise. Notice that the stable sets exactly correspond to the resource-feasible sets in our application, hence $(V,\mathcal{F})$ defines a threshold hypergraph exactly if one resource type suffices to represent the resource constraints. In analogy with the definitions for ordinary graphs, the *threshold dimension* t of a hypergraph $(V,\mathcal{F})$ can be defined as the minimum number of inequalities which are required to represent the system of stable sets; see Chvátal and Hammer [8]. More precisely, there exist t inequalities $\sum_{j\in V} r_{jk} x_j \le R_k$, $k=1,\ldots,t$, such that X is a stable set in $(V,\mathcal{F})$ if and only if all t inequalities are fulfilled. But even for ordinary graphs, the determination of the threshold dimension is NP-hard [8]. According to Yannakakis [27], already the decision problem if the threshold dimension of a graph is bounded by 3 is NP-complete (the decision problem if the threshold dimension of a graph is bounded by 2 can be solved in polynomial time). We refer to the surveys [17] and [7] for more details and references. Hence, we obtain the following theorem.

Theorem 1. *Given a project scheduling problem with minimal forbidden set representation $\mathcal{F}$ of resource constraints, and given that the number of minimal forbidden sets $\mathcal{F}$ is polynomial in n, it is NP-hard to determine the minimum number of resource types required in a threshold representation.*

Proof. The claim even holds if all minimal forbidden sets $F\in\mathcal{F}$ have cardinality 2. Then $(V,\mathcal{F})$ is an ordinary graph, and the problem corresponds to the determination of the threshold dimension of that graph, which is NP-hard. $\square$

2.2 From Thresholds to Minimal Forbidden Sets

We next discuss the complexity of the computation of $\mathcal{F}$, given the (usual) threshold representation of resource constraints. Clearly, since $\mathcal{F}$ can be exponential in n, the number of jobs, there is no algorithm with polynomial running time with respect to n. However, if only one resource type is present ($|K| = 1$), the following will be proved in Section 3.2.

Theorem 2. *Given a project scheduling problem with threshold representation of resource constraints, and given that the number of resource types* $|K|$ *equals* 1*, the minimal forbidden sets* $\mathcal{F}$ *can be computed in time polynomial in* $|\mathcal{F}|$ *and* n*.*

We finally consider the following three related problems that are important if an instance with threshold representation is given. For a given subset $W \subseteq V$ of jobs, which is an anti-chain of the partial order induced by the precedence constraints, we ask whether

(i) W is minimal forbidden,

(ii) W is contained in a (not necessarily minimal) forbidden set $F \supseteq W$, and

(iii) W is contained in a minimal forbidden set $F \supseteq W$.

It is trivial to decide Problem (i): W must be a forbidden set, that is, $\sum_{j \in W} r_{jk} \geq R_k + 1$ for some $k \in K$, and W is minimal forbidden if and only if $W \setminus \{j\}$ is resource-feasible for each $j \in W$ and all k. This can obviously be verified in $O(|K||W|)$ time. We can also decide Problem (ii) in polynomial time: Denote by $N \subseteq V$ all jobs in V which are unrelated to all jobs in W (with respect to the precedence constraints). Obviously, if there is a forbidden set F with $W \subseteq F$, then $F \subseteq W \cup N$, and there must be at least one resource type $k \in K$ such that the weight of the maximum weight anti-chain in $W \cup N$ exceeds R_k. A maximum weight anti-chain of a partially ordered set equals a maximum weight stable set in the underlying comparability graph. For each resource type k, this problem can be solved in time polynomial in n as a minimum flow problem; see [18]. Finally, Problem (iii) turns out to be NP-complete, since already the following, restricted problem is NP-complete.

Theorem 3. *Given a project scheduling problem (even without precedence constraints) with threshold representation of the resource constraints, and given that the number of resource types is polynomial in* n*, it is NP-complete to determine if a given job is contained in some minimal forbidden set* $F \in \mathcal{F}$ *or not.*

Proof. The problem is obviously in NP; according to the preceding remarks, a polynomially checkable proof is the set F itself. We will use a simple reduction of the NP-complete problem PARTITION (see, e. g., [11]). The problem PARTITION is the following: We are given n items of integral weight $r_j > 0$ with $\sum_{j=1}^{n} r_j$ even, and the question is if there exists a partition of the items into two subsets of equal total weight. Now define a project scheduling problem as follows. We have no precedence constraints and one job per item, each with resource requirement r_j. The resource availability is $R = \frac{1}{2}\sum_{j=1}^{n} r_j$. In addition, we have one more job, say i, with resource requirement $r_i = 1$. Now, if i is contained in a minimal forbidden set F, we have $\sum_{j \in F \setminus \{i\}} r_j = R$, since F is minimal forbidden and since i requires only one resource unit. On the other hand, if i is not contained in a minimal forbidden set, there is no subset F of the original items with total weight R. This completes the proof. □

2.3 Related Topics

Interestingly, threshold graphs and related questions have been considered also in the context of the so-called *PV-chunk synchronizing primitive*, which generalizes the classical *semaphore* concept for synchronization of parallel processing. In fact, apparently prior to Chvátal and Hammer [8], threshold graphs have been defined and characterized in this context by Henderson and Zalcstein [13]; see also [22].

In the form of *minimal covers*, minimal forbidden sets also arise in the context of the *knapsack polytope*, or more generally knapsack inequalities in 0–1 integer programming. Given a 0–1–polytope $P = \{x \in \{0,1\}^{|V|} \mid \sum_{j\in V} r_j x_j \leq R\}$, a *cover* is a set $C \subseteq V$ with $\sum_{j\in C} r_j > R$, and C is called *minimal* if it is minimal with respect to this property. In other words, minimal covers exactly correspond to minimal forbidden sets in our application. In the context of integer programming, minimal covers play an important role, since they give rise to cover inequalities of the form $\sum_{j\in C} x_j \leq |C|-1$, which are valid for P, and also to lifted cover inequalities, which are even facet-inducing for P. We refer, e.g., to [3] for more details.

3 Computing Minimal Forbidden Sets

In this section, we propose an algorithm which computes the minimal forbidden set representation $\mathcal{F}$ for an instance which is given in its threshold representation. Notice that exponentially many minimal forbidden sets may exist, hence the output of such an algorithm may be exponential with respect to n, the number of jobs.

Bartusch [5, Section 7.1.2] has previously suggested an approach to compute all minimal forbidden sets $\mathcal{F}$. His algorithm is based on the following ‘divide-and-conquer’ approach. The given instance, say I, is partitioned into $|K|$ partial instances $I_1,\ldots,I_{|K|}$ where each I_k only consists of jobs which require a positive amount of resource k. Then, for each I_k, the set of minimal forbidden sets $\mathcal{F}_k$ is calculated with respect to resource k only. To this end, Bartusch first computes all maximal anti-chains of the corresponding partial instance I_k (see [4] for an algorithm which computes all maximal anti-chains of a partial order). He then determines all minimal forbidden sets contained in these anti-chains. For all k, the system of all so-computed sets then includes the minimal forbidden sets $\mathcal{F}_k$ of the partial instance I_k. Finally, the minimal forbidden sets $\mathcal{F}$ of the original instance are given by the inclusion-minimal sets of $\bigcup_k \mathcal{F}_k$. This approach, however, has the major drawback that within the individual subproblems many minimal forbidden sets of jobs are potentially computed that later turn out to be forbidden but not minimal forbidden. Even if there are comparatively few minimal forbidden sets $\mathcal{F}$ in total, already for one resource type k and one maximal anti-chain in the corresponding partial instance I_k, exponentially many minimal forbidden sets may exist.

3.1 Description of the Algorithm

The basic approach is to enumerate subsets of V in a tree T where each node w of T, except the root node, is associated to exactly one job $j \in V$ (however, the mapping of nodes to jobs is not an injection). If node w is associated to some job j, w has a child node for each job $i = j+1,\ldots,n$. The root node has a child node for each job $i \in V$.

Each node w of the tree defines a subset $W \subseteq V$ of jobs with $j \in W$ by traversing the tree from w to the root node, and collecting the associated jobs on that path. In fact, a node of the tree only consists of its associated job j, a pointer to its father, and, for technical reasons, the (current) number of child nodes. With these basic definitions, there is a one-to-one correspondence between the set of nodes of T and the power set 2^V of all subsets of V.

To build a tree $T(\mathcal{F})$ which exactly represents all minimal forbidden sets $\mathcal{F}$, the tree T is pruned during this generic process like in a branch-and-bound algorithm: A node w is discarded as soon as it can be proved that neither W nor any superset of W that is located in the subtree rooted at w is a minimal forbidden set. $T(\mathcal{F})$ is constructed in a DFS fashion. For each node w that is to be added within the construction of $T(\mathcal{F})$, it is tested whether W is a minimal forbidden set. This is done in two steps. First, we check whether the associated set W is an anti-chain with respect to the (transitively implicit) precedence constrains. If this is not the case, by definition of minimal forbidden sets, the subtree rooted at w can be discarded (including w itself). Otherwise, if W is forbidden, that is, $\sum_{j \in W} r_{jk} > R_k$ for some $k \in K$, we test if W is *minimal* forbidden. This is done by verifying whether each set $W \setminus \{j\}$, $j \in W$, is resource-feasible; see Problem (i) in Section 2.2. If this is the case then W is minimal forbidden and w is stored as a leaf of the tree $T(\mathcal{F})$. Otherwise, W is not minimal forbidden and the subtree rooted at w can be discarded (including w itself). If W is resource-feasible, there may exist minimal forbidden sets $F \supset W$ that are located in the subtree rooted at w; hence branching is required on w. Finally, if some node does not represent a minimal forbidden set, and does not have any further descendants, it is deleted from the tree. Notice that deletion of nodes is meant recursively, that is, if a deleted node w was the only child of its father u in $T(\mathcal{F})$ then u is deleted as well. Upon termination, the constructed tree $T(\mathcal{F})$ has precisely $|\mathcal{F}|$ leaves.

Let us give a simple example. Let $D = (V, A)$ be the precedence graph with $V = \{1,2,3,4\}$ and $A = \{(1,2)\}$. There is one resource type with availability $R_1 = 3$, and the resource requirement of jobs is $r_{1,1} = 3$, $r_{2,1} = 2$, $r_{3,1} = 1$, and $r_{4,1} = 1$. Then the minimal forbidden sets are $\{1,3\}$, $\{1,4\}$, and $\{2,3,4\}$. Figure 1 depicts the trees T and $T(\mathcal{F})$.

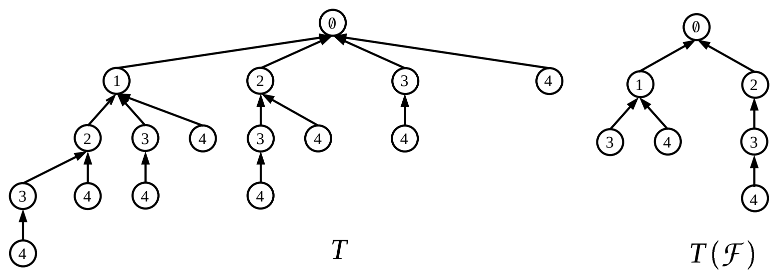

Figure 1: Example of the trees T and $T(\mathcal{F})$.

3.2 Analysis of the Algorithm

Let us now discuss the computational complexity of the proposed algorithm. We prove that the algorithm can be implemented to run polynomial in n and $|\mathcal{F}|$, the size of the in- and output, if there is only one resource type (see Theorem 2 above). Note that, for practical purposes, our implementation differs from the algorithm described next; it will be discussed in Section 3.3 below.

Let us first assume that $|K| = 1$, let r_j be the resource consumption of jobs $j \in V$, and let R be the resource availability. To make the above generic procedure polynomial in the size of the in- and output, we consider the jobs in a non-increasing order of their resource consumption r_j; so assume w.l.o.g. that $r_1 \geq r_2 \geq \cdots \geq r_n$. Then the following observation is immediate.

Lemma 1. *If $|K| = 1$ and if the jobs are considered in non-increasing order of r_j, i. e., in the order $1 \prec 2 \prec \cdots \prec n$ in $T(\mathcal{F})$, then each forbidden set F found by the generic procedure described in Section 3.1 is already minimal forbidden.*

Proof. Say a forbidden set $F = \{j_1, j_2 \ldots, j_t\}$ is found, where $j_1 < j_2 < \cdots < j_t$. Then, by construction, the set $F \setminus \{j_t\}$ is resource feasible, and since $r_{j_1} \geq r_{j_2} \geq \cdots \geq r_{j_t}$, also all sets $F \setminus \{j_i\}$ are resource feasible for all $i = 1, \ldots, t-1$. □

Hence, the above described procedure only generates nodes w which correspond to anti-chains W which are either resource feasible or minimal forbidden. Now recall that for any given feasible subset of jobs $W \subseteq V$, which is an anti-chain with respect to the precedence constraints, one can decide in time polynomial in n if W is contained in some (not necessarily minimal) forbidden set or not; this is Problem (ii) mentioned in Section 2.2. In particular, at any node w considered in the generic procedure described in Section 3.1, associated to some job j, one can decide in time polynomial in n if the corresponding anti-chain W is contained in some forbidden set F with $F \subseteq W \cup \{j+1, \ldots, n\}$ or not. In other words, one can decide in time polynomial in n if node w will eventually lead to some forbidden set or not. Combined with Lemma 1, we now obtain the following.

Lemma 2. *If $|K| = 1$ and if the jobs are considered in non-increasing order of r_j, i. e., in the order $1 \prec 2 \prec \cdots \prec n$ in $T(\mathcal{F})$, at any node w considered in the generic procedure described in Section 3.1, one can decide in time polynomial in n if w will eventually lead to some minimal forbidden set or not.*

Since the number of nodes in $T(\mathcal{F})$ is obviously polynomial in $|\mathcal{F}|$, this shows that for $|K| = 1$, the time required to compute the tree $T(\mathcal{F})$ is in fact polynomial in n and $|\mathcal{F}|$, which concludes the proof for Theorem 2.

For $|K| > 1$, however, the described algorithm is not polynomial in $\mathcal{F}$. The reason is that, given a node w considered in the generic procedure described in Section 3.1, we can no longer decide in polynomial time if the associated anti-chain W is contained in a *minimal* forbidden set or not (recall Theorem 3). This was possible for the case $|K| = 1$ only due to Lemma 1, which does no longer hold if $|K| > 1$. Consequently, if $|K| > 1$, one possibly ends at nodes w such that the associated set of jobs W is forbidden, but not minimal forbidden. In fact, the number of such nodes may be exponential in $|\mathcal{F}|$ for our algorithm, as is demonstrated by Example 2 given in the Appendix.

3.3 Implementation and Fast Reduction Tests

Contrary to what was described in Section 3.2, in our actual implementation we only considered heuristic but very efficient 'reduction tests' in order to decide if a node of the tree potentially leads to a minimal forbidden set or not. These simple tests greatly improved the performance of the simple generic procedure described in Section 3.1; they will be described in this section. To simplify notation, we omit the resource index k. Resource requirements r_j and supply R are treated as vectors and any inequality involving r_j or R is meant component-wise. Moreover, r_W denotes the vector of total resource consumption of W.

First, motivated by the results of Section 3.2, also for instances with more than one resource type it showed to be computationally more effective to consider the jobs in a suitable ordering: Therefore we identify a resource $k^* \in K$ that is scarcest, defined as a resource with smallest ratio $R_k / \sum_{j \in V} r_{jk}$, and assume that jobs are numbered in non-increasing order of their consumption of this resource type k^*. Although this does not help theoretically, it helps to heuristically close the gap between r_W and R in as many nodes w as soon as possible. Notice that this is particularly important since for any node w of the generic tree T, the subtree rooted at w is extremely unbalanced: If $s(j)$ denotes the size of a subtree rooted at some node w associated to job j, then $s(j) = 1 + \sum_{k=j+1}^{n} s(k)$, hence $s(j) = 2^{n-j}$.

Next, two jobs i and j cannot be in a common forbidden set if there is a (transitively implicit) precedence constraint between i and j. In addition, we implemented two other heuristic tests to determine if no minimal forbidden set contains both i and j. First, if the resources required by i and j are *disjoint* in the sense that $r_{ik} \cdot r_{jk} = 0$ for each $k \in K$, then i and j together do not belong to any minimal forbidden set. Second, let U be the set of jobs that are unrelated to both i and j with respect to the (transitively implicit) precedence constraints. Then, if $r_j + r_i + r_U \leq R$, then i and j are not contained in a common minimal forbidden set, either. All above tests are performed as preprocessing, and the resulting information is stored in a Boolean matrix M of size $n \times n$ in order to provide access in $\mathrm{O}(1)$ time.

Finally, we implemented another heuristic test which is particularly useful to keep the tree small if resource constraints are weak; it is a heuristic test in order to detect nodes w which cannot lead to any forbidden set: For a given node w, associated to some job j and a set W of jobs, $j \in W$, we simply sum up the resource requirements of all jobs out of $\{j+1, \dots, n\}$ that are not precedence-related to any of the jobs of W; denote this set by N. Then, if $r_W + r_N \leq R$, the subtree rooted at w can be discarded because each of the subsets of jobs in that subtree is resource-feasible. We also experimented with the exact method which decides if a given node of the tree leads to a forbidden set or not; see Problem (ii) of Section 2.2 for details. However, the computational overhead was too large due to the time required to solve the associated minimum-flow problems.

Algorithm 1 shows further details of the proposed procedure. For a given node w of the tree, associated to some job j, and some job $i > j$, Algorithm 1 calls the subroutines *EvaluateNode(i, w)* and possibly also *CreateNode(i, w)*. *EvaluateNode* computes the status of the set $W \cup \{i\}$, i. e., it decides whether $W \cup \{i\}$ is (minimal) forbidden or resource-feasible. Algorithmic details are given in Algorithm 2. *CreateNode(i, w)* generates a new node of the tree which is a child of w and associated to job i.

```
Algorithm 1: Compute all minimal forbidden sets
    Input   : Jobs $V$, precedence constraints $A$, resource constraints $r_j, R$.
    Output  : The set of minimal forbidden sets represented by the tree $T(\mathcal{F})$.

    Find suitable $k \in K$ and create ordering $L$ of jobs with non-increasing $r_{jk}$;
    Compute Boolean matrix $M$ which indicates whether $\exists F$ with $i, j \in F$;
    $\mathcal{F} := \emptyset$;    // stores the forbidden sets
    $root$ := root node of tree $T$; $Stack := \emptyset$;
    for all jobs $j \in V$ do
        $w$ := CreateNode($j, root$);
        push $w$ on $Stack$;
    while $Stack \neq \emptyset$ do
        remove node $w$ from $Stack$;
        $j$ := job associated to $w$; $W$ := set of jobs associated to $w$;
        for all jobs $i >_L j$ do
            EvaluateNode($i, w$);
            if $W \cup \{i\}$ is a minimal forbidden set then
                $u$ := CreateNode($i, w$);
                Add $u$ to $\mathcal{F}$;
            if $W \cup \{i\}$ is feasible then
                $u$ := CreateNode($i, w$);
                Add $u$ to $Stack$;
        (Recursively) delete $w$ if it is not minimal forbidden and has no children;
    return $\mathcal{F}$;
```

```
Algorithm 2: EvaluateNode
    Input   : A feasible set $W$ of jobs (represented by node $w$) and a new job $i$.
    Output  : Status of the set $W \cup \{i\}$
              (feasible / minimal forbidden / can be discarded).

    if $i$ and some $j \in W$ cannot be in a minimal forbidden set w. r. t. $M$ then
        return (can be discarded);
    if $r_{Wk} + r_{ik} > R_k$ for some $k$ then
        for $j \in W$ do
            if $r_{Wk} + r_{ik} - r_{jk} > R_k$ for some $k$ then
                return (can be discarded);
        return (minimal forbidden);
    $N$ := jobs of $\{j | V \ni j >_L i\}$ that potentially can be in some minimal forbidden
        set $F$ with $(W \cup \{i\}) \subset F$ (according to Matrix $M$);
    if $r_W + r_N \leq R$ then return (can be discarded);
    else return (resource-feasible);
```

3.4 Compact Representation of Forbidden Sets

Algorithm 1 immediately suggests to store the minimal forbidden sets in a data structure given by the tree $T(\mathcal{F})$. The jobs of the forbidden sets are represented as nodes in the tree, and upon building the tree as described before, a vector of pointers to the leaves of $T(\mathcal{F})$ is generated. To access (or loop) all jobs of a forbidden set F, one simply traverses $T(\mathcal{F})$ from the leaf which corresponds to F to the root node, obviously in $\mathrm{O}(|F|)$ time. In comparison to a representation as a vector of lists of the corresponding job numbers (which would certainly be the simplest data structure that provides fast access to traverse all minimal forbidden sets), this reduces memory requirement considerably (empirically analyzed in Section 4 below).

4 Computational Evaluation

We first describe the computational setup and the benchmark instances, and then analyze the performance of the proposed algorithm in dependence on different parameters which have been used to generate the instances.

4.1 Setup and Benchmark Instances

Our experiments were conducted on a Sun Ultra 1 with 143 MHz clock pulse operating under Solaris 2.7. The code is written in C++ and has been compiled with the GNU g++ compiler version 2.91.66 using the -O3 optimization option. The memory limit was set to 50 MB.

We have tested the proposed algorithm on instances of the library PSPLIB [23] that was generated by Kolisch and Sprecher with the help of the instance generator ProGen [16]. The library contains instances with 30, 60, 90, and 120 jobs, respectively. The instances have been generated by modifying three parameters, (i) the *network complexity* (*NC*) which is the average number of direct successors of a job, (ii) the *resource factor* (*RF*) which describes the average number of different resource types required in order to process a job divided by the total number of resource types, and (iii) the *resource strength* (*RS*), which is a measure of the scarcity of the resources (see [16] for details). The parameters have been chosen out of the sets $NC \in \{1.5, 1.8, 2.1\}$ and $RF \in \{0.25, 0.5, 0.75, 1.0\}$. For the benchmark sets with 30, 60 and 90 jobs the resource strength *RS* has been chosen from the values $\{0.2, 0.5, 0.7, 1.0\}$, while for instances with 120 jobs it was chosen from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. The smaller the parameter *RS*, the scarcer are the resources; hence, on average, the resource capacities are scarcer for the instances with 120 jobs. For each combination of the parameters, 10 instances have been generated at random. This results in 480 instances for each of instance sizes 30, 60, and 90, and 600 instances with 120 jobs.

Before we turn to our computational experiences with these instances, let us briefly comment on the relationship between the systems of minimal forbidden sets and the above mentioned parameters. According to Radermacher [24, p. 237], instances are *essentially equal* if both precedence constraints and the systems of minimal forbidden sets coincide. In this respect, the variation of the resource factor *RF* does not necessarily lead to essentially different instances: Although two instances have a different resource factor, they can be identical in the sense that they have identical precedence constraints and systems of minimal forbidden sets. For example, if the threshold representation of the resource constraints defines a threshold hypergraph without isolated nodes, the resource factor is obviously 1. However, the same system of minimal forbidden sets can be represented by $|\mathcal{F}|$ different resource types, which generally leads to a resource factor strictly smaller than 1. Another remark addresses the above definition of network complexity. Since the definition as the average number of direct successors disregards transitive precedence constraints, instances with identical network complexity may have an essentially different topology, hence also essentially different systems of minimal forbidden sets. For example, for $V = \{1,\dots,4\}$, the precedence constraints $A_1 := \{(1,2),(1,4),(3,4)\}$ and $A_2 := \{(1,2),(2,3),(3,4)\}$ both have a network complexity $NC = 3/4$. While (V,A_2) is a chain, and hence has no non-trivial anti-chain, (V,A_1) has three non-trivial anti-chains. However, our computational results with the PSPLIB instances show that, on average, there is a meaningful correlation between all three parameters and the system of minimal forbidden sets.

4.2 Computational Results

Table 1 shows for each test set the number of solved instances (#solved), that is, all minimal forbidden sets could be computed within the memory restriction of 50 MB, as well as the average and maximum number of forbidden sets (∅ #FS and max. #FS) and required computation times (∅ CPU and max. CPU). As the table suggests, the algorithm easily computes all minimal forbidden sets for the instances with 30 jobs; the computation time is negligible. Most of the instances with 60 jobs can also be solved in short time, however, there already exist few (17) instances for which not all minimal forbidden sets could be determined within the memory restriction of 50 MB (even with a limit 500 MB, 7 instances remain unsolved).

#jobs	#inst.	#solved	∅ #FS	max. #FS	∅ CPU	max. CPU
30	480	480	326	4,411	0.01	0.2
60	480	463	101,773	2,163,692	7	167
90	480	309	255,476	1,867,239	23	490
120	600	340	243,871	1,996,505	13	200

Table 1: For each set of instances the table displays the number of instances in the test set (#inst.), the number of solved instances (#solved), the average and the maximum number of minimal forbidden sets (∅ #FS and max. #FS), and the average and the maximum computation time in seconds (∅ CPU and max. CPU).

Although for larger instances the average memory requirement strongly increases, the algorithm still solves more than a half of the instances with 90 and 120 jobs with no more than 50 MB memory requirement. Note that, even for instances with 120 jobs, for all instances with scarce resources ($RS = 0.1$) or small resource factor ($RF = 0.25$, that is, each job requires only one resource type on average), the algorithm computes all minimal forbidden sets at an average running time of less than 5 seconds. Instances with scarce resources are known to be particularly hard with respect to makespan minimization and lower bound computations.

Forbidden set statistics. Figures 2 and 3 show how the average number and cardinality of minimal forbidden sets depend on the instance parameters *RS*, *RF*, and *NC*. Since we did not observe that these parameters were significantly correlated, all figures are based on average values with respect to the whole set of instances (with 30 jobs). As expected, both the number and cardinality of minimal forbidden sets heavily depend on the instance parameters *RS*, *RF*, and *NC*; let us briefly analyze the outcome of this evaluation.

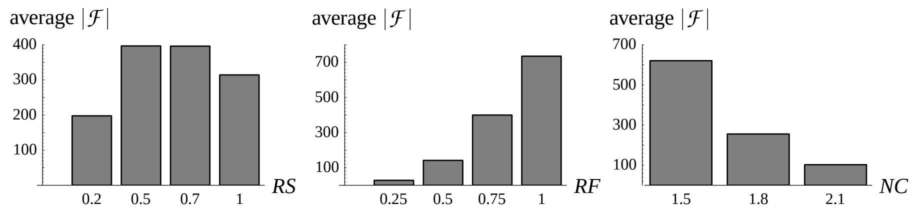

Figure 2: The plots display the average number of minimal forbidden sets depending on the instance parameters *RS* (left), *RF* (middle), and *NC* (right). The data is based on the test set with 30 jobs per instance.

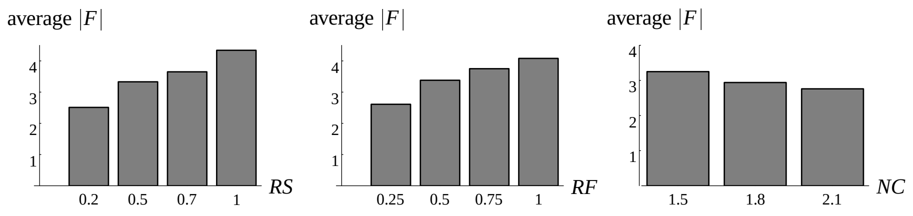

Figure 3: The plots display the average cardinality of minimal forbidden sets depending on the instance parameters *RS* (left), *RF* (middle), and *NC* (right). The data is based on the test set with 30 jobs per instance.

The dependence of the average cardinality of minimal forbidden sets on the resource strength *RS* as shown in Figure 3 is intuitive. With respect to the average number of minimal forbidden sets in dependence of the resource strength *RS*, it is noticeable that this figure is small either if the resource strength *RS* is very low (0.2; scarce resource capacity) or very high (1.0; loose resource capacity). For scarce resources, this is due to the fact that the minimal forbidden sets tend to be of small cardinality, as also suggested by Figure 3. For loose resources this is due to the fact that if there are hardly any resource constraints, already many anti-chains tend to be resource-feasible, hence there are fewer forbidden sets at all (with larger cardinality on average, as can be seen in Figure 3).

The behavior of the average cardinality of minimal forbidden sets in dependence of the resource factor *RF* in Figure 3 can be explained as follows. If each job requires only one or few resource types on average, that is, the resource factor *RF* is small, it is very likely that in a given anti-chain, there are pairs (i, j) of jobs with disjoint resource requirements ($r_{ik} \cdot r_{jk} = 0$ for all resource types k), hence minimal forbidden sets tend to be smaller on average the smaller the resource factor *RF*. (Consequently, there are also fewer of them, as can be seen in Figure 2.)

With respect to the network complexity *NC*, our results show that both number and cardinality of minimal forbidden sets trends down when *NC* increases. The reason is that, for the considered instances, the total number of precedence constraints (including transitive ones) increases with the network complexity. Recall, however, that the network complexity is not a measure for the total number of precedence constraints in general (see Section 4.1).

We finally observed that the average cardinality of the minimal forbidden sets increases with the number of jobs. The respective average values (based on the number of solved instances as given in Table 1) are 3.5 (maximum 10) for 30 jobs, 4.9 (maximum 16) for 60 jobs, 5.1 (maximum 13) for 90 jobs, and 4.5 (maximum 12) for 120 jobs. Notice that the average and maximum cardinality is comparatively small for the test set with 120 jobs, which is due to the fact that the resource strength parameters are smaller for these instances (and perhaps also since quite some (260) of the 600 instances could not be solved within the 50 MB memory limitation).

Computational performance. Let us next analyze the computational performance with respect to running times, and compare the proposed algorithm (with and without reduction tests) to a variation of the earlier mentioned divide-and-conquer approach by Bartusch [5]; see Section 3. Table 2 first shows the average and maximal computation times for the algorithm proposed in this paper, both with and without reduction tests.

	#jobs	#solved	∅ CPU	max. CPU
with reduction tests	30	480	0.01	0.2
no reduction tests	30	480	0.04	0.5
with reduction tests	60	463	7	167
no reduction tests	60	446	145	6,280

Table 2: For both variations of the algorithm, the table displays the number of solved instances (#solved) and the respective average and the maximum computation time in seconds (∅ CPU and max. CPU); based on instances with 30 and 60 jobs, respectively.

The results obviously confirm that the additional reduction tests proposed in Section 3.3 are worthwhile, reducing the average required computation time by a factor of almost 4 for the instances with 30 jobs. The importance of the additional reduction tests is even more apparent for the instances with 60 jobs. There, the average computation time decreases from 145 seconds (without reduction tests) to 7 seconds (with reduction tests); a factor of more than 20.

Figure 4 shows more details with respect to the computation times, based on the test set with 30 jobs. It is intuitive that the computation times are small whenever there are only few, and small minimal forbidden sets, and large if there are many and large minimal forbidden sets. This is indeed validated by Figure 4. There is however, one more remark on the computation times which concerns the reduction tests proposed in Section 3.3 (see also Table 2): If the reduction tests are not performed, the dependence of computation time on the resource factor *RF* gives a picture which is exactly reverse, showing that these tests are extremely effective particularly for instances where the resource factor *RF* is small. In fact, for $RF = 0.25$, the computation time increases from 1.4 ms (Figure 4; with reduction tests) to 41 ms (without reduction tests).

We have also experimented with a divide-and-conquer approach, based on the paradigm of the previously described approach proposed by Bartusch [5] (see the brief

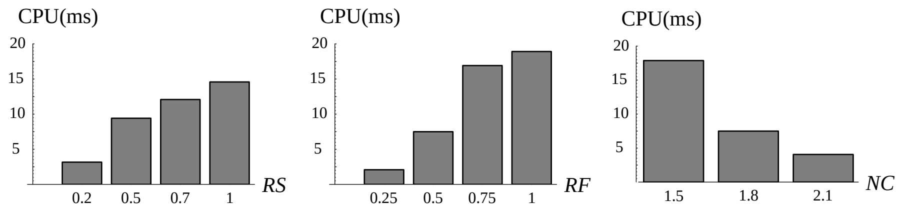

Figure 4: The plots display the average running time (in milliseconds) depending on the instance parameters *RS* (left), *RF* (middle), and *NC* (right). The data is based on the test set with 30 jobs per instance.

description of this approach in Section 3). The basic idea is to partition the given instance, say I, into $|K|$ partial instances $I_1, \ldots, I_{|K|}$ where each I_k only consists of jobs which require a positive amount of resource k. Then, for each I_k, the set of minimal forbidden sets $\mathcal{F}_k$ is calculated with respect to resource k only. This has the major advantage that each of the subproblems can be solved in polynomial time with respect to its in- and output, which is n and $|\mathcal{F}_k|$, respectively (see Theorem 2 and Section 3.2). In comparison to the algorithm proposed in this paper, this can in fact reduce the computation time by orders of magnitude for particular instances, as demonstrated by Example 2 in the Appendix. On the other hand, it is obvious that the systems of minimal forbidden sets $\mathcal{F}_k$ for the subproblems I_k may be exponential with respect to $\mathcal{F}$ itself. Apart from that, even if the minimal forbidden sets $\mathcal{F}_k$ are given, in order to obtain the minimal forbidden sets $\mathcal{F}$ for the original instance, an efficient computation of the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$ is necessary, which constitutes a non-trivial problem in its own.

Based on the instances from the PSPLIB, we have compared the time required to compute $\mathcal{F}$ using the algorithm proposed in this paper with the overall time required time to compute all minimal forbidden sets $\mathcal{F}_k$ for all partial instances I_k, $k = 1, \ldots, |K|$. It turned out that these computation times are in fact comparable on average, however, for only few instances the divide-and-conquer approach was more efficient. In particular, notice that this comparison does not even take into account the additional overhead required to compute the inclusion-minimal subsets of $\bigcup_k \mathcal{F}_k$. In fact, using a straightforward implementation, this overhead turned out to be a major bottleneck of the divide-and-conquer approach; it required by far more computation time than the computation of the minimal forbidden sets $\bigcup_k \mathcal{F}_k$ itself. Hence, a divide-and-conquer approach seems to be beneficial only for instances with very particular structure (e. g. Example 2 in the Appendix).

Memory requirements. Finally, we analyze the memory required to store the minimal forbidden sets in the data structure given by the tree $T(\mathcal{F})$, in comparison to the ordinary list representation, where each minimal forbidden set is stored as a list of job numbers. For the instances with 30 jobs, the average sum $\sum_{F \in \mathcal{F}} |F|$ is roughly 1400, while the average number of nodes in T is only 600. Despite of the fact that we have to maintain some overhead in order to generate (and delete) the tree $T(\mathcal{F})$, namely an integer which counts the number of children of each node in the tree, the memory requirement is reduced by a factor of roughly 1.5 in comparison to the list representation. For instances with 60, 90, and 120 jobs, the memory requirements are reduced by a factor of roughly 2.5. (This value only refers to instances for which all minimal forbidden sets could be computed within the given memory limitation of 50 MB.) Consequently, compared to the ordinary list representation, the proposed data structure given by $T(\mathcal{F})$ allows a much more efficient representation of minimal forbidden sets.

5 Concluding Remarks

We addressed several questions related to the transformation between two different representations of resource constraints in project scheduling, the threshold and the minimal forbidden set representation. Moreover, we presented and analyzed a simple algorithm which computes all minimal forbidden sets for an instance which is given in a threshold representation. There are some open questions which remain for future research: The question if some given job *i* is contained in some minimal forbidden set or not (given a threshold representation of the resource constraints) was proved to be NP-complete, even for the case without precedence constraints using a reduction from PARTITION (Theorem 3). In fact, it is not hard to see that this problem can be solved in pseudo-polynomial time by iteratively solving SUBSETSUM problems. However, it is open whether the problem becomes strongly NP-hard if also precedence constraints are present. Another interesting open problem is the question whether minimal forbidden sets can be computed in time polynomial in the in- and the output size of the problem if the number of resource types is greater than one, which would generalize Theorem 2.

References

[1] R. Alvarez-Valdés Olaguíbel and J. M. Tamarit Goerlich. The project scheduling polyhedron: Dimension, facets, and lifting theorems. *European Journal of Operational Research*, 67:204–220, 1993.

[2] E. Balas. Project scheduling with resource constraints. In E. M. L. Beale, editor, *Applications of Mathematical Programming Techniques*. The English University Press, London, 1970.

[3] E. Balas and E. Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics,* 34:119–148, 1978.

[4] M. Bartusch. An algorithm for generating all maximal independent subsets of a poset. *Computing,* 26:343–354, 1981.

[5] M. Bartusch. *Optimierung von Netzplänen mit Anordnungsbeziehungen bei knappen Betriebsmitteln*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1984.

[6] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.

[7] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999.

[8] V. Chvátal and P. L. Hammer. Aggregation of inequalities in integer programming. *Annals of Discrete Mathematics*, 1:145–162, 1977.

[9] P. Duchet. Hypergraphs. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics,* chapter 7, pages 381–432. Elsevier Science, Amsterdam, 1995.

[10] U. Feige and J. Kilian. Zero-knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[12] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[13] P. B. Henderson and Y. Zalcstein. A graph-theoretic characterization of the $\mathrm{PV}_{\mathrm{chunk}}$ class of synchronizing primitives. *SIAM Journal on Computing*, 6:88–108, 1977.

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

[15] G. Igelmund and F. J. Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13:1–28, 1983.

[16] R. Kolisch and A. Sprecher. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[17] N. V. R. Mahadev and U. N. Peled. *Threshold Graphs and Related Topics*, volume 56 of *Annals of Discrete Mathematics*. North-Holland, 1995.

[18] R. H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Order*, NATO Advanced Science Institute Series, pages 41–101. D. Reidel Publishing Company, Dordrecht, 1985.

[19] R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems I: General strategies. *ZOR - Zeitschrift für Operations Research*, 28:193–260, 1984.

[20] R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. Technical Report 680, Fachbereich Mathematik, Technische Universität Berlin, 2000. Submitted.

[21] R. H. Möhring and F. Stork. Linear preselective policies for stochastic project scheduling. *Mathematical Methods of Operations Research*, 2000. to appear.

[22] E. T. Ordman. Dining philosophers and graph covering problems. *The Journal of Combinatorial Mathematics and Combinatorial Computing*, 1:181–190, 1987.

[23] `ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/data.html`, January 2000.

[24] F. J. Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4:227–252, 1985.

[25] M. Schäffter. Scheduling with respect to forbidden sets. *Discrete Applied Mathematics*, 72:141–154, 1997.

[26] F. Stork. A branch-and-bound algorithm for minimizing expected makespan in stochastic project networks with resource constraints. Technical Report 613/1998, Technische Universität Berlin, Department of Mathematics, Germany, 1998. Revised July 2000.

[27] M. Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic and Discrete Methods*, 3:351–358, 1982.

Appendix

Example 1. *Let* $V = \{1,\ldots,4n\}$*, and let* $V_1 = \{1,\ldots,2n\}$ *and* $V_2 = \{2n+1,\ldots,4n\}$ *be a partition of* V*. Now, for each* $U_1 \subseteq V_1$ *define a corresponding* $U_2 := \{u+2n \mid u \in U_1\}$*, and let* $\mathcal{F} := \{U_1 \cup U_2 \mid U_1 \subset V_1, |U_1| = n\}$ *be the minimal forbidden sets.*

Here, the number of minimal forbidden sets is $\binom{2n}{n} \in \Omega(2^n)$. Now for any two distinct minimal forbidden sets, say $U_1 \cup U_2$ and $W_1 \cup W_2$, where $U_1, W_1 \in V_1$ and $U_2, W_2 \in V_2$ according to the above definition, two different resource types are required, since otherwise at least one of the sets $U_1 \cup W_1$, $U_2 \cup W_2$, $U_1 \cup W_2$, or $U_2 \cup W_1$ would not be resource feasible. Hence, $\Omega(2^n)$ resource types are required to represent $\mathcal{F}$. In other words, the threshold dimension of $(V,\mathcal{F})$ is $\Omega(2^n)$. $\square$

Example 2. *Let* $V = \{1,\ldots,n\}$*,* $|K| = 2$*,* $R_1 = R_2 = 2n-4$*,* $r_{1,1} = r_{2,1} = r_{n-1,2} = r_{n,2} = n$*, and* $r_{jk} = 1$ *otherwise.*

Then $\mathcal{F}$ consists of exactly six sets, namely $\{1,2\}$, $\{n-1,n\}$, and $\{i,3,4,\ldots,n-3,n-2,j\}$ for $i = 1,2$ and $j = n-1,n$. Hence $|\mathcal{F}| \in O(1)$ for any $n \in \mathbb{N}$, but the number of nodes which are examined within our algorithm is exponential in n. Notice that this is an example where the previously described divide-and-conquer approach by Bartusch (see Section 3) is beneficial, since it runs polynomial in n. $\square$

Reports from the group

"Combinatorial Optimization and Graph Algorithms"

of the Department of Mathematics, TU Berlin

705/2000 *Ekkehard Köhler:* Recognizing Graphs without Asteroidal Triples

704/2000 *Ekkehard Köhler:* AT-free, coAT-free Graphs and AT-free Posets

702/2000 *Frederik Stork:* Branch-and-Bound Algorithms for Stochastic Resource-Constrained Project Scheduling

700/2000 *Rolf H. Möhring:* Scheduling under uncertainty: Bounding the makespan distribution

698/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* More-dimensional packing with order constraints

697/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Extending partial suborders and implication classes

696/2000 *Sándor P. Fekete, Ekkehard Köhler, and Jürgen Teich:* Optimal FPGA module placement with temporal precedence constraints

695/2000 *Sándor P. Fekete, Henk Meijer, André Rohe, and Walter Tietze:* Solving a "hard" problem to approximate an "easy" one: heuristics for maximum matchings and maximum Traveling Salesman Problems

694/2000 *Esther M. Arkin, Sándor P. Fekete, Ferran Hurtado, Joseph S. B. Mitchell, Marc Noy, Vera Sacristánm and Saurabh Sethia:* On the reflexivity of point sets

693/2000 *Frederik Stork and Marc Uetz:* On the representation of resource constraints in project scheduling

691/2000 *Martin Skutella and Marc Uetz:* Scheduling precedence constrained jobs with stochastic processing times on parallel machines

689/2000 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Scheduling with AND/OR precedence constraints

685/2000 *Martin Skutella:* Approximating the single source unsplittable min-cost flow problem

684/2000 *Han Hoogeveen, Martin Skutella, and Gerhard J. Woeginger:* Preemptive scheduling with rejection

683/2000 *Martin Skutella:* Convex quadratic and semidefinite programming relaxations in Scheduling

682/2000 *Rolf H. Möhring and Marc Uetz:* Scheduling scarce resources in chemical engineering

681/2000 *Rolf H. Möhring:* Scheduling under uncertainty: optimizing against a randomizing adversary

680/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Solving project scheduling problems by minimum cut computations (Journal version for the previous Reports 620 and 661)

674/2000 *Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia:* Optimal covering tours with turn costs

669/2000 *Michael Naatz:* A note on a question of C. D. Savage

667/2000 *Sándor P. Fekete and Henk Meijer:* On geometric maximum weight cliques

666/2000 *Sándor P. Fekete, Joseph S. B. Mitchell, and Karin Weinbrecht:* On the continuous Weber and k-median problems

664/2000 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* On project scheduling with irregular starting time costs

661/2000 *Frederik Stork and Marc Uetz:* Resource-constrained project scheduling: from a Lagrangian relaxation to competitive solutions

658/1999 *Olaf Jahn, Rolf H. Möhring, and Andreas S. Schulz:* Optimal routing of traffic flows with length restrictions in networks with congestion

655/1999 *Michel X. Goemans and Martin Skutella:* Cooperative facility location games

654/1999 *Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang:* Single machine scheduling with release dates

653/1999 *Andreas S. Schulz and Martin Skutella:* Scheduling unrelated machines by randomized rounding

646/1999 *Rolf H. Möhring, Martin Skutella, and Frederik Stork:* Forcing relations for AND/OR precedence constraints

640/1999 *Foto Afrati, Evripidis Bampis, Chandra Chekuri, David Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Cliff Stein, and Maxim Sviridenko:* Approximation schemes for minimizing average weighted Completion time with release dates

639/1999 *Andreas S. Schulz and Martin Skutella:* The power of α-points in preemptive single machine scheduling

634/1999 *Karsten Weihe, Ulrik Brandes, Annegret Liebers, Matthias Müller–Hannemann, Dorothea Wagner and Thomas Willhalm:* Empirical design of geometric algorithms

633/1999 *Matthias Müller–Hannemann and Karsten Weihe:* On the discrete core of quadrilateral mesh refinement

632/1999 *Matthias Müller–Hannemann:* Shelling hexahedral complexes for mesh generation in CAD

631/1999 *Matthias Müller–Hannemann and Alexander Schwartz:* Implementing weighted b-matching algorithms: insights from a computational study

629/1999 *Martin Skutella:* Convex quadratic programming relaxations for network scheduling problems

628/1999 *Martin Skutella and Gerhard J. Woeginger:* A PTAS for minimizing the total weighted completion time on identical parallel machines

624/1999 *Rolf H. Möhring:* Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen

627/1998 *Jens Gustedt:* Specifying characteristics of digital filters with FilterPro

620/1998 *Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz:* Resource constrained project scheduling: computing lower bounds by solving minimum cut problems