Math Meth Oper Res (2000) 52:413–439

Mathematical Methods
of Operations Research

A branch-and-bound algorithm for the resource-constrained project scheduling problem

U. Dorndorf, E. Pesch, T. Phan-Huy

University of Bonn, Faculty of Economics, BWL III, Adenauerallee 24-42, D-53113 Bonn, Germany (e-mail: udorndorf@acm.org, e.pesch@uni-bonn.de, phanhuy@toan.de)

Abstract. We describe a time-oriented branch-and-bound algorithm for the resource-constrained project scheduling problem which explores the set of active schedules by enumerating possible activity start times. The algorithm uses constraint-propagation techniques that exploit the temporal and resource constraints of the problem in order to reduce the search space. Computational experiments with large, systematically generated benchmark test sets, ranging in size from thirty to one hundred and twenty activities per problem instance, show that the algorithm scales well and is competitive with other exact solution approaches. The computational results show that the most difficult problems occur when scarce resource supply and the structure of the resource demand cause a problem to be highly disjunctive.

Key words: resource-constrained project scheduling, constraint propagation

1 Introduction

Resource-constrained project scheduling addresses the task of allocating scarce resources over time in order to perform a set of activities, subject to constraints on the order in which the activities may be executed. The objective considered in this paper is to minimise the project duration, i.e. the maximum of the completion times of all activities. Due to its general nature, the resource-constrained project scheduling problem (RCPSP) has important applications in diverse areas such as make-to-order production planning or construction engineering. Recent surveys of resource-constrained project scheduling have been given by Brucker et al. (1999), Herroelen et al. (1998), Kolisch and Padman (1997) and Elmaghraby (1995).

Using the classification scheme for project scheduling described by Brucker et al. (1999), which extends the well known three-field classification scheme

for machine scheduling, we will denote the problem considered in this paper with $PS|prec|C_{\max}$, for (a) project scheduling with (b) precedence constraints and (c) the objective of minimising the maximum completion time. In the classification scheme developed by Herroelen et al. (1999) the problem can be characterised as $m, 1|cpm|C_{\max}$.

It is well known that, as an extension of the job shop scheduling problem, the problem $PS|prec|C_{\max}$ is $\mathcal{NP}$-hard (Błażewicz et al., 1983). Most exact solution methods are therefore based on branch-and-bound search. Beginning with the work of Johnson (1967), a great number of branch-and-bound algorithms have been developed, and we refer the reader to the survey papers mentioned above for a description and classification of the various approaches. Currently, the most effective exact algorithms seem to be the ones of Demeulemeester and Herroelen (1997); Sprecher (1997); Mingozzi et al. (1998); Brucker et al. (1998) and the procedures of Klein and Scholl (1998a,b, see also Klein (1999)) which can solve a generalised version of the the problem $PS|prec|C_{\max}$.

In this paper, we study a time-oriented, constraint propagation based approach to project scheduling. We describe a branch-and-bound algorithm that enumerates possible activity start times based on the idea that, at a given node of the search tree, an activity must either start as early as possible or be delayed. A central feature of the algorithm is the application of constraint propagation techniques that actively exploit the temporal and resource constraints during the search in order to narrow down the set of possible activity start times and thus reduce the search space. Further reduction of the search effort is achieved by enforcing some necessary conditions that must be met by active schedules.

One of the main advantages of the time-oriented branching scheme is its conceptual simplicity which allows to modify and extend the approach for related practical scheduling problems that are often complicated by additional constraints. Furthermore, the constraint propagation techniques that we use are not custom-tailored for the problem $PS|prec|C_{\max}$ but are of an elementary nature and have a wide applicability.

The first time-oriented branching schemes for the problem $PS|prec|C_{\max}$ have been described by Elmaghraby (1977) and Talbot and Patterson (1978). The common idea behind these algorithms is to branch over all possible start time assignments of the next activity to be scheduled, and the number of child nodes generated at a given node of the search tree thus depends on the selected activity. Carlier and Latapie (1991) have proposed a binary search scheme in which branching consists of selecting an activity and splitting its interval of possible start times into two intervals of equal size. Martin and Shmoys (1996) have developed a time oriented algorithm for the job shop scheduling problem. Caseau and Laburthe (1996) have independently designed a branch-and-bound algorithm for a multi-mode project scheduling problem which can be classified as $MPS|prec|C_{\max}$ in the scheme of Brucker et al. (1999). For the single mode case the algorithm uses the same branching strategy as the procedure of Martin and Shmoys, which schedules an activity at its earliest start time and delays it upon backtracking until the earliest completion time of some other activity, resulting in a binary search tree. The branching scheme described in this paper also makes use of this elementary approach. The branching strategy described by Caseau and Laburthe has also been used in modified form in the study of Baptiste et al. (1999). Heipcke and Colombani

(1997) have developed an algorithm for a version of the problem $PS|prec|C_{\max}$ in which resource supply and demand may vary over time; the branching scheme of their algorithm is also binary; an activity is scheduled at its earliest start time and delayed upon backtracking by a single unit of time. An unusual feature of their algorithm is that activities are in general not scheduled in order of increasing start times.

In the remainder of this paper we proceed as follows. Section 2 formally states the optimisation model and introduces the notation. Because constraint propagation plays a central role in the algorithm, Section 3 presents a basic propagation algorithm and the tests used to rule out uninteresting activity start times. Section 4 then describes the branch-and-bound algorithm. Finally, the computational results obtained for a large number of benchmark test problems are discussed in Section 5.

2 The problem

The problem $PS|prec|C_{\max}$ is a generalisation of important scheduling problems like the job shop scheduling problem or the open shop scheduling problem and can be described as follows: a finite set of activities $\mathscr{V} = \{1, \ldots, n\}$ has to be scheduled with the objective of minimising the project duration, i.e. the maximum of the completion times of all operations which is also known as the makespan. Each activity $i \in \mathscr{V}$ has a specific processing time p_i, which is known in advance, and a start time S_i, which is a decision variable. By choosing sufficiently small time units we can always assume that the processing times (start times) are positive (non-negative) integer values. We further require that activities may not be interrupted during their processing (non-preemption).

In general, activities cannot be processed independently from each other due to limited resources and additional technological requirements. The problem $PS|prec|C_{\max}$ considers two kinds of constraints: *precedence constraints* and *resource constraints*.

1. Precedence constraints specify a fixed processing order between pairs of activities. Precedence constraints cover technological requirements of the kind that some activity i must finish before another activity j can start, for instance, if the output of i is the input of j.
2. Resource constraints model the resource demand of activities in a scheduling environment with scarce resource supply. More precisely, an activity i requires $r_{ik} \in \mathbb{N}_0$ units of one or several resources $k \in \mathscr{R}$, where $\mathscr{R}$ denotes the set of all resources. We further assume for the sake of simplicity that resource k is available in constant amount R_k, although many of the results derived in this work also apply if we consider a time-varying resource supply instead. Clearly, a feasible solution only exists if $r_{ik} \leq R_k$ for all activities and resources. While an activity is in process, the required resource units are exclusively assigned to it and are not available for other activities. The set of activities which require a resource k is denoted with $\mathscr{V}_k := \{i \in \mathscr{V} \mid r_{ik} > 0\}$.

A precedence constraint (i, j) can be obviously modelled by imposing a minimal time lag between the start times of the two activities i and j. This temporal

constraint has the general form $S_i + p_i \leq S_j$. We denote with $\mathscr{E}$ the set of all precedence constraints. A resource constraint for a given resource k ensures that in any processing period the resource demand never exceeds the resource supply R_k. It is possible to define these resource constraints in a quite elegant way by using the concept of a slack function which will be introduced later. For the time being it is sufficient to define the auxiliary set $\mathscr{V}(t)$ of activities in process at time t. The problem $PS|prec|C_{\max}$ can then be stated as follows:

$$\min\left\{\max_{i\in\mathscr{V}}\{S_i + p_i\}\right\} \quad \text{s.t.} \qquad \text{(i)}$$

$$S_i + p_i \leq S_j, \quad \forall (i,j) \in \mathscr{E}, \qquad \text{(ii)}$$

$$\sum_{i\in\mathscr{V}(t)} r_{ik} \leq R_k, \quad \forall t \in \mathbb{N}_0, \quad \forall k \in \mathscr{R}, \qquad \text{(iii)}$$

$$S_i \in \mathbb{N}_0, \quad \forall i \in \mathscr{V}. \qquad \text{(iv)}$$

A schedule $S = (S_1, \ldots, S_{|\mathscr{V}|})$ is an assignment of all start times. We say that S is *feasible* if it satisfies both all precedence constraints (ii) and all resource constraints (iii). Thus, solving the problem $PS|prec|C_{\max}$ is equivalent to finding a feasible schedule with a minimal makespan.

Given the set of all feasible schedules we can define a partial relation $<$, where $S \leq S'$ if no activity in S starts later than in S'. Further, $S < S'$ if $S \leq S'$ and additionally at least one activity in S starts earlier. A schedule S is then said to be *active*, if it is feasible and there exists no other feasible schedule S' such that $S' < S$. In other words, S is active, if no activity can be started earlier without violating either one of the precedence or resource constraints. Inversely, a schedule S is not active, if an activity i can be started earlier than at time S_i without having to increase the start time of some other activity. Intuitively, we will then say for short that i can be left-shifted in S. The definition of active schedules immediately leads to the following simple and well known observation: any solution method which minimises the makespan function can focus on generating active schedules, since for every non-active schedule there always exists an active schedule with an identical or even lower makespan.

The branch-and-bound algorithm is based on the idea of repeatedly reducing the set of possible start times of an activity until all activities are scheduled. Each activity $i \in \mathscr{V}$ has a *current domain* $\Delta_i \subseteq \mathbb{N}_0$ of possible start times. Since most of the constraint propagation methods applied can only deduce a domain reduction if the current domains are finite, we will assume that some upper bound UB on the optimal makespan is known so that $\Delta_i \subseteq [0, UB - p_i]$ holds. If no initial upper bound is given, then we use the trivial upper bound $\sum_{i\in\mathscr{V}} p_i$. The set of current domains of all activities is denoted with $\Delta := \{\Delta_i | i \in \mathscr{V}\}$. For an activity $i \in \mathscr{V}$, $ES_i(\Delta) := \min \Delta_i$ denotes its earliest start time, and $LS_i(\Delta) := \max \Delta_i$ denotes its latest start time. Similarly, $EC_i(\Delta) := ES_i(\Delta) + p_i$ is the earliest completion time and $LC_i(\Delta)$

$:= LS_i(\Delta) + p_i$ the latest completion time of i. If no confusion is possible, then we will write ES_i, LS_i, etc.

A schedule S is called *domain feasible* with respect to a set Δ of current domains if the current domain of each activity still contains the start time of this activity in S. Given a set Δ of current domains, the set of all activities $\mathscr{V}$ can be naturally partitioned into a set of scheduled and non-scheduled (free) activities. Clearly, if the current domain of an activity i contains exactly one entry, then i must start at that time and can be considered as scheduled. Hence $\mathscr{V}^s(\Delta) := \{i \in \mathscr{V} \mid |\Delta_i| = 1\}$ is the set of scheduled activities, and $\mathscr{V}^f(\Delta) := \{i \in \mathscr{V} \mid |\Delta_i| > 1\}$ is the set of free activities. For all scheduled activities $i \in \mathscr{V}^s(\Delta)$, the start time is defined through $S_i(\Delta) := ES_i(\Delta) = LS_i(\Delta)$.

3 Constraint propagation

An exact solution method for solving the problem $PS|prec|C_{\max}$ generally consists of two components: (1) a search strategy which organises the enumeration of all potential solutions and (2) a search space reduction strategy which diminishes the number of potential solutions. In this section, we will discuss search space reduction strategies based on constraint propagation which has become more and more popular in the last decades due to its elementary nature. Constraint propagation has its origins in the field of constraint programming which models combinatorial search problems as special instances of the *constraint satisfaction problem* (CSP). The origins of constraint propagation go back to Waltz (1975) who developed a now well-known filtering algorithm for labelling three-dimensional line diagrams.

The basic idea of constraint propagation is to evaluate implicit constraints through the repeated analysis of the variables, domains and constraints that describe a specific problem instance. This analysis makes it possible to detect and remove *inconsistent* start time assignments that cannot participate in a feasible schedule by a merely partial problem analysis. Different concepts of consistency which may serve as a theoretical background for propagation algorithms have been defined; general overviews are given by Kumar (1992) and Tsang (1993), as well as Dorndorf et al. (2000), who focus on scheduling-related propagation techniques.

In general, establishing full consistency by removing *all* inconsistent start time assignments is prohibitively expensive due to a computational complexity which grows exponentially with the number of activities. Therefore, the application of constraint propagation is only sensible if we content ourselves with approximations. An important task is to derive simple rules which lead to efficient search space reductions, but at the same time can be implemented efficiently. These rules are called *consistency tests* and are generally described through a condition–instruction pair $\mathscr{A}$ and $\mathscr{B}$. Intuitively, the semantics of a consistency test are as follows: whenever condition $\mathscr{A}$ is satisfied, $\mathscr{B}$ has to be executed. $\mathscr{A}$ may be, for instance, an equation or inequation, while $\mathscr{B}$ may be a domain reduction rule. We will use the shorthand notation $\mathscr{A} \Rightarrow \mathscr{B}$ for consistency tests.

Given a set of consistency tests, these tests have to be applied in an iterative fashion rather than only once in order to obtain the maximal domain reduction possible. The reason for this is that, after the reduction of some domains, additional domain adjustments can possibly be derived using some

of the tests which have previously failed in deducing any reductions. Thus, the deduction process is carried out until no more adjustments are possible or, in other words, until the set Δ of current domains becomes a fixed point. It is important to mention that the fixed point computed does not have to be unique and usually depends upon the order of the application of the consistency tests. Thus, in general, an application order may induce a stronger search space reduction than another application order. However, we will only study *monotonous* consistency tests for which the order of application does not affect the outcome of the domain reduction process (Dorndorf et al., 2000).

The iterative application of the consistency tests is controlled by the constraint propagation algorithm; the propagation algorithm used in our implementation is a variant of the AC-5 arc-consistency algorithm described by Van Hentenryck et al. (1992). Like all advanced consistency algorithms, it works with a queue containing elements to reconsider. A queue element consists of a constraint and a value (or a set of values) that has been removed from the domain of some variable in the constraint and justifies the need to reconsider the constraint, i.e. to re-apply a consistency test. In each iteration of the propagation algorithm, a constraint/value pair is removed from the queue and all consistency tests are evaluated that are associated with this constraint. If any of these tests removes a value x from a domain, say from Δ_i, then all constraints which contain the variable S_i are stored in the queue, together with the information that x has been removed from Δ_i. This process is repeated until the queue is empty and the fixed point is reached. The reason for storing a value together with a constraint is that this may allow to use a more efficient algorithm in a consistency test.

Let us now turn to the consistency tests that are used within our algorithm. A precedence constraint determines the sequence in which two specific activities i and j have to be processed. If, for instance, activity i has to finish before activity j can start then the earliest start time of j has to be greater than or equal to the earliest completion time of i. Likewise, an upper bound for the latest completion time of i is the latest start time of j. This implies the following precedence consistency test:

$$(i,j) \in \mathscr{E}(\Delta) \Rightarrow \begin{cases} \Delta_i := \Delta_i \backslash]LS_j - p_i, \infty[, \\ \Delta_j := \Delta_j \backslash [0, ES_i + p_i[. \end{cases}$$

When used within the constraint propagation algorithm, this test, of course, leads to the same result as a "forward-backward" time window calculation in a project network. As some of the consistency tests discussed below may discover new precedence constraints, which must hold in addition to those given in the original problem, the set of all precedence constraints depends on Δ and is denoted with $\mathscr{E}(\Delta)$.

Interval consistency tests are based on the comparison of resource supply and demand within a given time interval. Each activity requires a total amount of work $w_{ik} := r_{ik}p_i$ from resource k. A time interval is said to be *capacity consistent* if the amount of work requested by all activities within this time interval can be matched by the amount of work supplied. The work of an activity i that must fall into a time interval $[t_1, t_2[$ may be smaller than w_{ik}, and it depends upon the current domain of i. We therefore introduce the concept of interval processing times which has been first proposed by Lopez et al.

(1992) under the name of *energetic reasoning*. The interval processing time $p_i(t_1, t_2)$ is the smallest amount of time during which i has to be processed within $[t_1, t_2[$. Five possibilities have to be considered: (1) i can be completely contained within the interval, (2) overlap the entire interval, (3) have a minimum processing time in the interval when started as early as possible or (4) have a minimum processing time when started as late as possible. The fifth situation applies whenever, given the current domains, i does not necessarily have to be processed within the time interval. Consequently,

$$p_i(t_1, t_2) := \max\{0, \min\{p_i, t_2 - t_1, EC_i - t_1, t_2 - LS_i\}\}.$$

The corresponding *interval work* is given by $w_{ik}(t_1, t_2) = r_{ik}p_i(t_1, t_2)$. The interval work of a subset of activities $\mathscr{A} \subseteq \mathscr{V}_k$ is defined through $W(\mathscr{A}, t_1, t_2) := \sum_{i \in A} w_{ik}(t_1, t_2)$. We can now define the slack of a time interval with respect to a set of activities as the difference between work supply and demand within the time interval:

$$slack(\mathscr{A}, t_1, t_2) := R_k \cdot (t_2 - t_1) - W(\mathscr{A}, t_1, t_2).$$

Since the slack function obviously depends on the given set Δ of current domains, we will write $slack_\Delta(\mathscr{A}, t_1, t_2)$ whenever necessary. An interval $[t_1, t_2[$ is capacity consistent if $slack(\mathscr{V}_k, t_1, t_2) \geq 0$ for every resource k. An in-depth survey of consistency tests based on the notion of interval capacity is given by Dorndorf et al. (1999).

An important special case of the interval capacity consistency condition is obtained when we consider time intervals of unit width. If, for an activity $i \in \mathscr{V}_k$ and some time t, $slack(\mathscr{V}_k \backslash \{i\}, t, t+1)$ is less than the required resource amount r_{ik}, then activity i cannot be processed at time t and must hence not start in the interval $]t - p_i, t]$. This leads to the following test, which is also known under the name timetable-based constraint propagation (Le Pape, 1994).

$$slack_\Delta(\mathscr{V}_k \backslash \{i\}, t, t+1) < r_{ik} \Rightarrow \Delta_i := \Delta_i \backslash]t - p_i, t].$$

The computational analysis will show that the most difficult problem instances tend to have a high share of *disjunctive* activities. Two activities $i, j \in \mathscr{V}$ are in disjunction if, due to limited resource availability, i and j cannot be processed simultaneously, i.e. if $r_{ik} + r_{jk} > R_k$ for some resource k. The following consistency tests, which are originally based on interval capacity, therefore address this aspect.

For a given pair of activities i, j in disjunction, the well known *pair test*, which has been proposed by Carlier and Pinson (1989), can be applied:

$$LS_i - ES_j < p_j \Rightarrow \mathscr{E}(\Delta) := \mathscr{E}(\Delta) \cup \{(i, j)\}.$$

Clearly, if the condition on the left hand side is satisfied, i must precede j and we can update the set of precedence constraints.

The consistency test for disjunctive activity pairs can be extended by considering larger sets $\mathscr{A}$ of pairwise disjunctive activities, which must be processed sequentially. Given such a set $\mathscr{A}$, we can check for the existence of a feasible schedule under the hypothesis that some activity $i \in \mathscr{A}$ is *not* scheduled as last activity in $\mathscr{A}$. If no such schedule exists then we have falsified the

hypothesis and can conclude that i must be processed after all other activities in $\mathscr{A}$; we can thus add the corresponding precedence constraints and adjust the earliest start time of i to the earliest possible completion time of all activities in $\mathscr{A}\backslash\{i\}$. This value is equal to the minimal completion time of a corresponding one-machine problem with release dates and due dates, which is itself $\mathscr{NP}$-hard; it is therefore usually approximated with the solution of the corresponding preemptive one-machine problem, denoted with $EC^{pr}(\mathscr{A}\backslash\{i\})$, which can be computed efficiently. The resulting consistency test, which is also due to Carlier and Pinson (1989), can be formalised as follows:

$$\max_{j\in\mathscr{A}\backslash\{i\}} LS_j - \min_{j\in\mathscr{A}} ES_j < \sum_{j\in\mathscr{A}} p_j \Rightarrow \begin{cases} \mathscr{E}(\Delta) := \mathscr{E}(\Delta) \cup \{(j,i) \mid j \in \mathscr{A}\backslash\{i\}\}; \\ \Delta_i := \Delta_i \backslash [0, EC^{pr}(\mathscr{A}\backslash\{i\})[. \end{cases}$$

A symmetrical test can be applied to show that some activity must be processed *before* a set of other activities. The tests are called *input/output tests* and are sometimes also referred to as immediate selection.

Let us now consider the question how the set $\mathscr{A}$ should be chosen. Several algorithms have been proposed which, given a set $\mathscr{V}'$ of pairwise disjunctive activities, can apply the consistency test for all subsets $\mathscr{A} \subseteq \mathscr{V}'$ with low polynomial effort (cf. Dorndorf et al., 2000); the implementation used in our procedure is based on the algorithm of Nuijten (1994) and requires effort $O(|\mathscr{V}|^2)$. The question to be answered thus is how to choose $\mathscr{V}' \subseteq \mathscr{V}$. Clearly, as the algorithm will apply the test for all subsets of $\mathscr{V}'$, the set $\mathscr{V}'$ itself should be maximal in the sense that none of the activities in $\mathscr{V}\backslash\mathscr{V}'$ is in disjunction with all those in $\mathscr{V}'$ (maximal clique).

The possible choices of $\mathscr{V}'$ can be determined by considering an undirected graph G with nodes corresponding to activities and edges between any pair of disjunctive activities, i.e. pairs $i,j \in \mathscr{V}$ for which $r_{ik} + r_{jk} \geq R_k$ for some resource $k \in \mathscr{R}$. A decomposition of G into all maximal cliques then gives all possible choices of $\mathscr{V}'$. Although the number of maximal cliques may in general be exponential in the size of the graph, the decomposition can for practical purposes be quickly calculated with the algorithm of Bron and Kerbosch (1973). Nevertheless, the number of maximal cliques may still be large and many of these cliques may be overlapping. As the gain of information deduced by the consistency tests may be outweighed by the computational effort for applying the tests, if this is done too frequently, we restrict our attention to a small number of cliques chosen at the beginning of the search according to the following heuristic suggested by Phan Huy (1999).

Given the decomposition of G into all maximal cliques, the first phase of the heuristic consists of repeatedly selecting a maximal clique which contains the largest number of edges that are not already covered by some previously chosen clique, until all edges are covered. In the second phase, the algorithm repeatedly chooses an additional clique in order of decreasing size, if the new clique does not overlap with any previously chosen clique for more than two thirds.

Other heuristics for chosing some sets $\mathscr{V}'$ in order to apply the input/output tests for the problem $PS|prec|C_{\max}$ have been desribed by Brucker et al. (1998) and Baptiste et al. (1999); in contrast to the approach described here, these heuristics are not based on an initial decomposition into all maximal cliques.

4 Branch-and-Bound algorithm

The branch-and-bound algorithm uses a time-oriented branching scheme that can generate all active schedules, so that traversing the search tree will lead to an optimal solution. Inversely, the branching scheme avoids constructing non-active schedules, which cuts down the search space considerably. As a peculiarity our algorithm does *not* explicitly compute any lower bounds. Instead, the bound-oriented fathoming of nodes is a useful by-product of constraint propagation techniques, that have to be applied anyway in the "branching" part of the algorithm. The search space is further reduced by adding constraints that must be satisfied by all active schedules that can be developed from a given node, and through the application of a simple left-shift dominance test.

4.1 Branching scheme

The branching structure is based on a simple time-oriented schedule generation scheme, which results in a binary search tree. Each node α of the search tree is associated a set $\Delta(\alpha) = \{\Delta_i(\alpha) \mid i \in \mathcal{V}\}$ of current domains, which uniquely determine the sets $\mathcal{V}^s(\Delta(\alpha))$ and $\mathcal{V}^f(\Delta(\alpha))$ of scheduled and non-scheduled activities, respectively. In order to simplify the notation we will write $\mathcal{V}^s(\alpha)$ instead of $\mathcal{V}^s(\Delta(\alpha))$, etc., whenever possible. Generating a specific schedule is equivalent to reducing the current domains until all activities are appropriately scheduled. One method of domain reduction is the application of constraint propagation at every node of the search tree. Since in general, however, constraint propagation alone does not schedule all activities, some activities additionally have to be scheduled by an explicit assignment of their start time variables.

At every node α of the search tree an unscheduled activity $j \in \mathcal{V}^f(\alpha)$ is chosen and two child nodes are generated. Denoting the left child node with $l(\alpha)$ and the right child node with $r(\alpha)$, the branching scheme relies on the following simple node generation rule.

$l(\alpha)$: Start j at its earliest start $ES_j(\alpha)$ by setting $S_j(l(\alpha)) := ES_j(\alpha)$.
$r(\alpha)$: Increase the earliest start of j by choosing $ES_j(r(\alpha)) > ES_j(\alpha)$.

A complete specification of the branching scheme now requires the answer to two questions. The first question deals with the problem of which activity $j \in \mathcal{V}^f(\alpha)$ to choose in node α. The second question is how the earliest start time of j should be increased in $r(\alpha)$. We will first describe the choice of an activity j and then derive an earliest start time adjustment for the right child node.

4.2 Activity selection

At node α, an activity can be selected for branching if it is free and *non-delayed*. For the time being, it is not necessary to describe this attribute more closely. We only assume that the set of non-delayed activities $\mathcal{V}^{f'}(\alpha)$ is a non-

empty subset of the set of free activities. An activity j is then selected according to the following rule:

> Choose $j \in \mathscr{V}^{f'}(\alpha)$, such that $ES_j = t(\alpha)$ where $t(\alpha)$ is the schedule time, which is defined as $t(\alpha) := \min_{i \in \mathscr{V}^{f'}(\alpha)} ES_i(\alpha)$.

Ties are first broken by selecting an activity which satisfies some secondary criterion, then randomly. In general, we use the minimal *time slack*, i.e. $|\Delta_i|$, as secondary criterion; this means that we use the well known first fail principle which consists of first instantiating the variable with the fewest remaining possible values. We will denote the activity chosen in α as $act(\alpha)$.

After the description of the selection rule, we are left with the problem of how to identify the set of non-delayed activities. Of course, we can always set $\mathscr{V}^{f'}(\alpha) := \mathscr{V}^{f}(\alpha)$. This, however, is not sensible, since choosing an arbitrary free activity may lead to a non-active schedule. We will therefore show how to specify the set of delayed activities, so as to capture the notion of active schedules more closely.

The precedence and unit interval consistency tests that are applied within the constraint propagation algorithm affect the earliest activity start times as follows. Let $pc_j(\Delta)$ be the minimal start time of j if only the precedence constraints (i, j) between activities i in $\mathscr{V}^{s}(\Delta)$ and j are considered:

$$pc_j(\Delta) := \max_{i \in \mathscr{V}^{s}(\Delta)} \{S_i + p_i \,|\, (i, j) \in \mathscr{E}\}.$$

Here, we have used the convention that the maximum of the empty set is 0. Let further $rc_j(\Delta)$ be the minimal start time of j if additionally resource constraints are considered:

$$rc_j(\Delta) := \min_{t \geq pc_j(\Delta)} \{t | \forall k \in \mathscr{R}, \forall t' \in [t, \ldots, t + p_j[:$$

$$slack_{\Delta}(\mathscr{V}_k \backslash \{j\}, t', t' + 1) \geq r_{jk}\}.$$

Then, obviously,

$$ES_j(\Delta) \geq rc_j(\Delta) \geq pc_j(\Delta).$$

A schedule S can be naturally identified with a set of current domains, where each domain Δ_i contains the corresponding start time, i.e. $\Delta_i := \{S_i\}$. This justifies the notation $rc_j(S)$ and $pc_j(S)$. Clearly, S can only be active if for all activities either a precedence constraint or insufficient resource capacity prevents a left-shift. Thus, in any active schedule S, the identity

$$S_j = rc_j(S) \tag{1}$$

holds for all $j \in \mathscr{V}$. Inversely, all schedules satisfying this condition are active.

This motivates the following definition of the set of free and *non-delayed* activities based on the precedence and resource feasible earliest start times:

$$\mathscr{V}^{f'}(\alpha) := \{j \in \mathscr{V}^{f}(\alpha) \,|\, ES_j(\alpha) = rc_j(\alpha)\}$$

This means that a free activity is a candidate for branching if a precedence constraint or low slack prohibits a left shift of the activity. It is worth mentioning that this condition may not hold for a free activity which has been delayed. The definition of the set of free and selectable activities $\mathscr{V}^{f'}$ can therefore also be interpreted as follows by considering the complementary set $\mathscr{V}\backslash\mathscr{V}^{f'}$ of free and un-selectable activities: a delayed activity i remains un-selectable until we know that the resource capacity "provided" by delaying i has been used by some other activity. The following lemma, which is formally proven in Appendix A, justifies our choice of the set $\mathscr{V}^{f'}$.

Lemma 1. *If, at node α of the search tree, there is an unscheduled activity then $\mathscr{V}^{f'}(\alpha)$ is not empty, or α cannot lead to an active schedule.*

4.3 Delaying duration

The activity selection rule implies that in any schedule S developed from α the condition $S_i \geq t(\alpha)$ must hold for all $i \in \mathscr{V}^{f'}(\alpha)$. But there might be free and delayed activities $i \in \mathscr{V}^{f}(\alpha)\backslash\mathscr{V}^{f'}(\alpha)$ for which $ES_i(\alpha) < t(\alpha)$ and which could therefore possibly be scheduled at a time earlier than $t(\alpha)$, either by the propagation algorithm or through an explicit start time assignment, once they have become selectable again. However, by using a similar argumentation as in the proof of Lemma 1, it can be shown that this cannot happen if the resulting schedule is active.

Lemma 2. *In any active schedule developed from a node α of the search tree, all free and delayed activities in the set $\mathscr{V}^{f}(\alpha)\backslash\mathscr{V}^{f'}(\alpha)$ start after $t(\alpha)$.*

It follows from the activity selection rule and from Lemma 2 that, in any active schedule developed from α, no activity which was free at α can start earlier than at time $t(\alpha)$. This implies that the slack at any time $t < t(\alpha)$ does not change in descendant nodes of α that lead to an active schedule. Furthermore, we can conclude that the predecessors of all activities that can start at $t(\alpha)$ must already be scheduled at α. These observations can be used in several ways.

Let us first return to the initial question of how to increase the earliest start time of a selected activity $j = act(\alpha)$ if we branch to the right. A first simple alternative is to delay the activity by a single time unit. However, we can do better by observing that the resulting schedule S can only be active if either (1) a precedence constraint or (2) low slack prohibits a left-shift of the selected activity. Since the activity will be delayed by at least one time unit and because all predecessors are scheduled, the first case can be ruled out. The second case requires that the slack of all activities except j is insufficient to the left of $S_j(\alpha)$. This can only be the case if $S_j(\alpha)$ matches the completion time of some activity that shares resources with j.

Let $\mathscr{R}_i := \{k \in \mathscr{R} \mid r_{ik} > 0\}$ denote the set of resources required by an activity i. The set of activities which share resources with activity j and finish after the current schedule time $t(\alpha)$ can then be defined as

$$\mathscr{V}'(j) := \{i \in \mathscr{V}\backslash\{j\} \mid \mathscr{R}_i \cap \mathscr{R}_j \neq \varnothing \wedge EC_i(\alpha) > t(\alpha)\}.$$

If $\mathscr{V}'(j)$ is not empty then the earliest start time of j can be adjusted by setting

$$ES_j(r(\alpha)) := \min_{i \in \mathscr{V}'(j)} EC_i(\alpha).$$

We can use the same argument together with Lemma 2 to re-adjust the earliest start times of all free and delayed activities. At node α, before applying constraint propagation and branching, we set

$$ES_i(\alpha) := \min_{l \in \mathscr{V}'(i)} EC_l(\alpha), \quad \text{for all } i \in \mathscr{V}^f(\alpha) \backslash \mathscr{V}^{f'}(\alpha).$$

Here, we have used the convention that the minimum of the empty set is zero. In both cases, if the set $\mathscr{V}'(j)$ is empty then the node cannot lead to an active schedule. Observe that the adjustment of the earliest start time will lead to an empty domain for all delayed activities i for which $LS_i \leq t(\alpha)$, i.e. for those activities which have been "needlessly" delayed.

The fact that the slack in periods $t < t(\alpha)$ remains constant in descendant nodes also allows us to apply a simple left-shift dominance test. If, for any free activity $j \in \mathscr{V}^f(\alpha)$ with $p_j > 0$, the condition $rc_j(\alpha) + p_j \leq t(\alpha)$ holds, i.e. if j can resource and precedence-feasibly be scheduled so that it finishes not later than at time $t(\alpha)$, then node α cannot lead to an active schedule.

4.4 Search strategy and properties of the branching scheme

The search tree is traversed in depth-first order until a leaf node is generated. This happens whenever the set of free and selectable activities becomes empty, i.e. if $\mathscr{V}^{f'}(\alpha) = \varnothing$. This leaf node represents a solution if all activities are scheduled. The makespan of an initial or improved schedule is, of course, used as new upper bound. Backtracking occurs when a leaf node is reached or when an inconsistency has been detected, i.e. when $\Delta_i(\alpha)$ has become empty for some activity $i \in \mathscr{V}$.

The minimum possible depth of the tree is zero and is obtained if all activities are scheduled through constraint propagation at the root node. The maximum depth of the search tree that can occur in the worst case is reached when branching to the very right side of the tree in the following way. Starting at the root node, we can initially at most delay $|\mathscr{V}| - 1$ activities. As delayed activities remain un-selectable at least until some other activity is scheduled, i.e. until we can conclude that the resource capacity, released by delaying activities, has been used, we must then schedule some activity or backtracking would be initiated. Next we can at most branch $|\mathscr{V}| - 2$ times to the right before branching a single time to the left. By continuing in this way, we may reach a theoretical worst case depth of $1/2|\mathscr{V}|(|\mathscr{V}| + 1)$.

The time-oriented branching scheme can generate all active schedules, i.e. if S is an active schedule, then the search tree contains a leaf node α in which all activities are scheduled and $S_i = S_i(\alpha)$ for all $i \in \mathscr{V}$. This completeness property of the branching scheme follows from the fact that if S is domain feasible in α, then it is also domain feasible in either $l(\alpha)$ or $r(\alpha)$; we can thus construct a path in the tree along which (1) S remains domain feasible and (2) the domains are continuously reduced until S is obtained. Inversely, all complete schedules that are constructed are active.

5 Computational analysis

5.1 Implementation of the algorithm

The branch-and-bound algorithm has been implemented in C++ using the constraint programming libraries ILOG SOLVER and ILOG SCHEDULER which support the implementation of tree search algorithms that apply constraint propagation at the nodes of the tree (Le Pape, 1994). The basic propagation algorithm used in SOLVER is a variant of the AC-5 arc consistency algorithm of Van Hentenryck et al. (1992).

The most important features of the SOLVER library are (1) fundamental data types such as integer domain variables, (2) generic constraints upon these variables together with corresponding domain reduction rules, e.g. linear constraints on integer domain variables, (3) the propagation algorithm, (4) classes for defining a search (branching) scheme, and (5) support for reversible actions that are automatically undone upon backtracking, for instance the definition and propagation of constraints. Based upon the generic data types and algorithms found in SOLVER, the SCHEDULER library provides an object model and algorithms that facilitate the development of scheduling applications. For instance, SCHEDULER includes classes for representing activities and resources as well as associated constraints such as precedence or resource constraints.

Besides the support for implementing backtracking algorithms and the generic propagation mechanism, we have used the following features of the libraries. The decision variables S_i are represented as integer domain variables. The precedence constraints and the corresponding consistency test are realised through the built-in linear constraints provided by SOLVER. The resource constraints, the unit interval consistency test and the input/output tests are provided by SCHEDULER. For the administration of the temporal and resource constraints we have used the activity and resource classes of SCHEDULER. The disjunctive activity pair test is implemented as a generic disjunctive SOLVER constraint on integer domain variables.

The logic of the branch-and-bound algorithm described in Section 4 has been hand coded. By using the SOLVER search tree classes, the amount of code required for the branching and backtracking part has been kept low.

All results reported for our algorithm in the following tables have been obtained on a Pentium Pro/200 PC with Windows NT 4.0 as operating system.

5.2 Test data

We have tested the algorithm on four large sets of benchmark problems that were systematically generated with the problem generator ProGen (Kolisch et al., 1995), which allows to specify several control parameters that characterise a resulting problem instance:

- The *network complexity* $C \geq 1$ indicates the average number of immediate successors of an activity and is a measure of the complexity of the precedence constraints.
- The *resource factor* $RF \in [0, 1]$ indicates the average percentage of resources

Table 1. Characteristics of the test sets

Test set	Size	Fixed parameters				Variable parameters		
		$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{R}\rvert$	p_i	r_{ik}	C	RF	RS
j30	480	30	4	$\{1\ldots10\}$	$\{1\ldots10\}$	1.5	0.25	0.2
j60	480	60	4	$\{1\ldots10\}$	$\{1\ldots10\}$	1.8	0.50	0.5
j90	480	90	4	$\{1\ldots10\}$	$\{1\ldots10\}$	2.1	0.75	0.7
							1.00	1.0
j120	600	120	4	$\{1\ldots10\}$	$\{1\ldots10\}$	1.5	0.25	0.1
						1.8	0.50	0.2
						2.1	0.75	0.3
							1.00	0.4
								0.5

required to process an activity. It takes a value of 1 if every activity requires every resource.

– The *resource strength* $RS \in [0, 1]$ describes the average tightness of the resource constraints. A resource strength of 0 indicates maximal tightness, which results from the minimal feasible resource availability, i.e. a supply equal to the maximum requirement of any single activity. For a resource strength of 1, the earliest start schedule does not contain any resource conflicts.

Table 1 shows the detailed characteristics of the test sets, which are collected in the project scheduling problem library PSPLIB (Kolisch and Sprecher, 1996; Kolisch et al., 1999). The number of activities, $|\mathscr{V}|$, does not include the fictitious start and end activities. All processing times and resource requirements were randomly drawn from the set $\{1, \ldots, 10\}$. The first three test sets with 30, 60, and 90 activities per problem contain ten instances for each combination of the three control parameter values shown in the three rightmost columns and four top-most rows of the table, leading to a total number of 480 instances. The last test set, which contains problems with 120 activities, has been generated with different, more difficult resource strength values; again, the set contains 10 problem instances for each combination of the variable parameters shown in the last 5 rows of the table, resulting in a total number of 600 problems.

5.3 Bidirectional planning

When trying to solve a given problem instance, we apply our algorithm in forward and backward direction (*bidirectional planning*). A problem can be solved in backward fashion by simply reversing the project network and applying the algorithm to the resulting mirror-network. Forward and backward scheduling methods have for instance been discussed by Li and Willis (1992) and Klein (1998). While no scheduling direction is uniformly superior for all test problems some instances are easier to solve in one direction than in the other. Intuitively, a branch-and-bound algorithm works best if the difficult part of the problem (bottleneck) is handled at the beginning of the search, since otherwise a solution for the difficult subproblem has to be rediscovered many times in different branches of the search tree. This means that if the

bottleneck is towards the beginning of the project then forward planning is advantageous; otherwise, if the bottleneck is at the end then backward planning works best. Since it is hard to predict the location of the bottleneck in order to choose a favourable planning direction, we simply proceed as follows. We allocate half of the run-time to solve the problem in forward direction; if the problem remains open after this time then we apply the algorithm to the mirror problem, now using the makespan of the best schedule found so far, if any, as initial upper bound.

5.4 Results

Table 2 shows the results obtained with the time-oriented branch-and-bound algorithm for the smallest test set with 30 activities per problem. For a given run time limit t_{max} the table shows the average run time t_{avg}, the percentage of problems solved to optimality within the time limit, and the remaining average and maximum deviation from the optimal solution (all optimal solutions for this test set are known). For example, the table shows that within a time limit of 300 seconds 95.4% or 458 problem instances can be solved to optimality within an average run time of 19.4 seconds and a remaining average deviation from the optimal solution of 0.05%. Within the maximum allowed run time of 1800 seconds, 97.3% of the problems are solved. We found that the difficulty of the problem instances for the time-oriented algorithm strongly depends on the resource strength. While all instances with a resource strength greater than 0.2 can be solved within less than 10 seconds, the problems with a resource strength of 0.2 are considerably more difficult.

We must mention that the currently most effective algorithms for this problem set, which have been developed by Klein and Scholl (1999a), Demeulemeester and Herroelen (1997), Sprecher (2000) and Mingozzi et al. (1998), perform better on this problem set and can solve more instances within shorter time. For example, Klein (1999) reports that the scatter search algorithm of Klein and Scholl can solve all problems within a maximum time of 361 seconds on a Pentium/166 computer.

Table 3 shows the results of our algorithm (T-O B&B) for the larger test set with 60 activities per problem instance and compares them to the results of the procedures of Brucker et al. (1998), Sprecher (2000), and Klein and Scholl (1999a), which have been tested on the same problem set. The table shows the algorithms in inverse historical order. For a given time limit, the table presents the average run time, the percentage of problems solved to optimality, and the

Table 2. Results of exact algorithms for test set j30

Procedure	t_{max} (sec)	t_{avg} (sec)	Opt. (%)	$dev._{opt}$	
				avg. (%)	max (%)
Time-oriented B&B	1	0.3	80.2	0.57	10.9
	10	1.6	88.3	0.19	8.9
	60	6.0	92.7	0.10	6.0
	300	19.4	95.4	0.05	6.0
	1800	66.4	97.3	0.03	4.5

Table 3. Results of exact algorithms for test set j60

Procedure	t_{max} (sec)	t_{avg} (sec)	Opt. (%)	dev.$_{LB}$[a]		dev.$_{LB}$[b]		dev.$_{LB_0}$ avg. (%)	dev.$_{UB}$[c] avg. (%)
				avg. (%)	max (%)	avg. (%)	max (%)		
T-O B&B[d]	1	0.4	73.5	4.3	34.0	5.8	39.8	13.7	1.9
	10	2.7	75.4	3.6	25.8	5.1	34.9	12.8	1.3
	60	14.7	76.2	3.4	24.4	4.8	34.2	12.5	1.1
	300	69.2	78.5	3.2	23.3	4.6	32.9	12.3	0.9
	1800	386.0	80.0	3.0	22.6	4.5	30.3	12.0	0.8
Klein and Scholl[e]	10	3.7	69.6	–	–	5.3	32.5	–	–
	60	17.8	73.3	–	–	4.8	31.3	–	–
	300	77.7	76.0	–	–	4.6	29.7	–	–
	1800	396.7	80.2	–	–	4.3	29.7	–	–
	3600	736.1	81.9	–	–	4.2	29.0	–	–
Sprecher[f]	300	88.1	72.7	–	–	5.7	45.8	13.6	–
	1800	472.7	75.8	–	–	5.3	40.7	13.0	–
Brucker et al.[g]	3600	–	67.9	–	–	4.8	30.8	–	–

[a] Based on the best known lower bounds collected in the PSPLIB.
[b] Based on the lower bounds of Brucker et al. (1998).
[c] Based on the best known solutions collected in the PSPLIB.
[d] Impl. in C++, results obtained on Pentium Pro/200 with Windows NT.
[e] Impl. in C++, results obtained on Pentium/166 with Windows 95.
[f] Impl. in C++, results obtained on Pentium/166 with Linux.
[g] Impl. in C, results obtained on SUN/Sparc 20/801 (80 MHz) with Solaris 2.5.

average and maximum deviations from several lower bounds as well as the average deviation from the best known solutions collected in the corresponding benchmark file of the project scheduling problem library PSPLIB. Dashes indicate that the corresponding information was not available. When comparing the results of different algorithms, the different computer platforms, which are described in the table footnotes, must be taken into account; observe that we have *not* scaled the run time values.

The development of tight lower bounds for the problem $PS|prec|C_{max}$ is an area of active research (see e.g. Klein and Scholl, 1999b; Brucker and Knust, 1999; Möhring et al., 1998; Heilmann and Schwindt, 1997). In Table 3 and in the following tables we show the deviations of our algorithm with respect to the best lower bounds that are currently available in the corresponding PSPLIB benchmark files. A comparison of the performance of different algorithms with respect to deviations from lower bounds is, of course, only meaningful if the deviations are based on the same bounds. Table 3 and Table 4 below therefore also include deviations from the lower bounds of Brucker et al. (1998), which have been used in the other studies. For easy reproducibility we also give the deviations with respect to the precedence based lower bound LB_0 which corresponds to the optimal solution of the resource relaxation of the problem.

Table 3 shows that the time-oriented algorithm is competitive with the other procedures and that, for small run times, it achieves the highest percentage of optimally solved problems. For large run times, the algorithm of Klein and Scholl seems to perform slightly better than our algorithm.

Table 4 compares the results of the time-oriented algorithm for the test set

Table 4. Results of exact algorithms for test set j90

Procedure	t_{max} (sec)	t_{avg} (sec)	Opt. (%)	dev.$_{LB}$[a]		dev.$_{LB}$[b]		dev.$_{LB_0}$ avg. (%)	dev.$_{UB}$[c] avg. (%)
				avg. (%)	max (%)	avg. (%)	max (%)		
T-O B&B[d]	1[e]	0.6	71.2	4.7	35.5	6.2	43.4	13.4	2.2
	10	3.0	74.2	4.0	28.8	5.4	37.0	12.4	1.5
	60	15.9	75.0	3.8	26.5	5.2	37.0	12.2	1.4
	300	76.1	76.0	3.7	26.4	5.1	36.1	12.1	1.3
Sprecher[f]	300	120.3	61.5	–	–	8.3	58.7	15.7	–

[a] Based on the best known lower bounds collected in the PSPLIB.
[b] Based on the lower bounds of Brucker et al. (1998).
[c] Based on the best known solutions collected in the PSPLIB.
[d] Impl. in C++, results obtained on Pentium Pro/200 with Windows NT.
[e] Based only on forward planning.
[f] Impl. in C++, results obtained on Pentium/166 with Linux.

Table 5. Results for test set j120

Procedure	t_{max} (sec)	t_{avg} (sec)	Opt. (%)	dev.$_{LB}$[a]		dev.$_{LB_0}$ avg. (%)	dev.$_{UB}$[b] avg. (%)
				avg. (%)	max (%)		
Time-oriented B&B	10	7.4	31.0	9.9	40.6	38.0	3.6
	60	41.9	32.2	9.5	40.6	37.5	3.3
	300	205.3	33.3	9.2	40.6	37.1	3.0

[a] Based on the best known solutions collected in the PSPLIB.
[b] Based on the best known lower bounds collected in the PSPLIB.

j90 to those of the procedure of Sprecher (2000), which is the only algorithm for which results on this test set have been published. The format of the table is the same as in Table 3.

Table 5 shows the results of our algorithm for the largest test set with 120 activities per problem instance. Recall that this problem set has been generated with more difficult resource strength values than the three smaller sets. As we will see in Table 6 below, this appears to be the main reason for the strong decrease in the percentage of problems solved to optimality when compared to the smaller test sets. We can also observe that the average deviations from the lower bounds are roughly three times as high as for the smaller and easier test sets with 60 and 90 activities per instance. As before, the percentage of problems solved to optimality grows only slowly when the run time is increased.

Data on the performance of other exact procedures for this problem set has not been published. We have compared our results with respect to the average deviation from the precedence based lower bound LB_0 to that of several state of the art heuristics reported by Kolisch and Hartmann (1999), who have analysed the performance of eight heuristics within a maximum number of 1000 and 5000 iterations; an iteration corresponds to the application of a serial or parallel schedule generation scheme. The minimal deviation obtained by the best heuristic within 1000 iterations is 39.4%. Within the

Table 6. Influence of problem characteristics

Param.	Value	Optimal[a]				dev.$_{LB}$[b]			
		j30 (%)	j60 (%)	j90 (%)	j120 (%)	j30 (%)	j60 (%)	j90 (%)	j120 (%)
RS	0.1	–	–	–	2.5	–	–	–	19.9
	0.2	81.7	30.8	20.0	9.2	0.2	11.6	14.0	13.4
	0.3	–	–	–	25.0	–	–	–	8.3
	0.4	–	–	–	49.2	–	–	–	3.9
	0.5	100.0	83.3	84.2	80.8	0.0	1.2	0.8	0.8
	0.7	100.0	100.0	100.0	–	0.0	0.0	0.0	–
	1.0	100.0	100.0	100.0	–	0.0	0.0	0.0	–
RF	0.25	100.0	100.0	95.0	84.2	0.0	0.1	0.5	3.3
	0.50	100.0	80.8	73.3	38.3	0.0	3.1	4.6	14.5
	0.75	93.3	71.7	67.5	25.0	0.0	4.7	5.2	15.4
	1.00	88.3	61.7	68.3	19.2	0.2	4.9	4.5	13.1
C	1.5	95.0	80.0	78.8	46.3	0.1	4.0	4.2	13.2
	1.8	94.4	78.8	74.4	44.4	0.1	4.4	5.0	14.7
	2.1	96.7	76.9	75.0	34.4	0.0	4.4	5.6	18.3

[a] Within a time limit of 300 seconds.
[b] Based on the best known lower bounds collected in the PSPLIB.

maximum number of iterations, only the best of the eight heuristics, the genetic algorithm of Hartmann (1998), achieves a lower deviation (36.7%) than our algorithm within the maximum allowed time.

Table 6 analyses the influence of the resource strength *RS*, the resource factor *RF*, and the network complexity *C* on the difficulty of the problem instances. For the four test sets, the table gives the percentage of problems with a particular characteristic that could be solved to optimality and the average deviation from the best known lower bounds collected in the corresponding PSPLIB benchmark files. For example, line five of the table shows that 80.8% of the problem instances with 120 activities that were generated with a resource strength of 0.5 could be solved to optimality, and the remaining average deviation from the lower bound for these problems was 0.8%. The data shown in Table 6 confirms the results of earlier studies, see e.g. Kolisch (1995), regarding the influence of the problem characteristics.

The table shows that the hardest problems are those with a low resource strength. For a resource strength of 0.2, the percentage of problems that could be solved to optimality sharply decreases with growing problem size; for the lowest resource strength value of 0.1, only three of the problems with 120 activities could be solved to optimality. Problems with $RS \geq 0.7$ appear to be easy independent of problem size, and the benchmark lower bounds for these instances are always tight. For $RS = 0.5$, we can observe that the percentage of problems that can be solved remains roughly constant when the problem size grows from 60 to 120 activities, although the time limit is not increased.

The influence of the resource factor is also clearly visible: problems become harder as the average number of resource types required by an activity increases. For example, for the minimal resource factor of 0.25, which means that on average each activity requires only a single resource type, the algorithm can solve 84.2% of the problems with 120 activities. As the resource factor grows, the value drops to 19.2%.

The influence of the network complexity is not as significant as that of the other two control parameters. While the results for test set j120 indicate that the problems become more difficult with increasing network complexity, the data for the smaller test sets is inconclusive.

As to be expected after examining Table 6, the hardest problems occur when a low resource strength is combined with a high resource factor. For example, roughly speaking, the 30.8% of the problems with 60 activities and a resource strength of 0.2 that can be solved to optimality include all those instances for which the resource factor takes a value of 0.25 and a few instances with a resource factor of 0.5; the tables in Appendix B show detailed results for each combination of the three control parameters. Intuitively, a low resource strength causes many activity pairs to be disjunctive and thus leads to cliques of pairwise disjunctive activities of considerable size. Additionally, if the average number of resource types required by an activity is high, then, simply speaking, there are many "links" between the cliques induced by each resource type. This combined effect leads to large and difficult disjunctive sub-problems.

We also analysed in how many cases our algorithm could find values matching a best known lower bound without being able to prove optimality within the maximum allowed run time. We found that this occurs for none of the instances in the test sets j60 and j90 and for only a single instance of the test set j120. This means that even the best known lower bounds, if calculated at the root of the search tree, would only marginally improve the results of our algorithm. Also, it seems questionable if a re-calculation of bounds during the search would pay off in terms of overall computation time. For example, Klein (1999) has found that for his branch-and-bound algorithm the pruning power of the bounds described by Klein and Scholl (1999b) does often not outweigh the associated computational effort and does in general not lead to a reduction of computation times.

We also experimented with a dominance rule based on storing partial schedules, which is similar to the well known cutset rule described by Demeulemeester and Herroelen (1992). While the use of this rule led to some improvements, the overall effect for the larger test sets was rather small; for example, when using this rule, only a single additional instance of the test set j60 could be solved within the maximum time limit of 1800 seconds. Since the performance of the rule within our algorithm was disappointing and because the rule cannot easily be adapted for generalisations of the problem $PS|prec|C_{\max}$ with arbitrary minimal and maximal time lags between activities, we did not further consider it in our study.

6 Conclusions

We have described a branch-and-bound algorithm for the problem $PS|prec|C_{\max}$. The algorithm is based on an elementary time-oriented branching scheme and uses constraint propagation techniques for reducing the size of the search space. The search space is further restricted by enforcing necessary conditions for active schedules and through a simple left-shift test.

Computational experiments with four large, systematically generated sets of benchmark problems, ranging in size from 30 to 120 activities per problem instance, indicate that the algorithm scales well and, especially for larger in-

stances, is competitive to other exact procedures for this problem. The results for the largest test set show that the time truncated version of the algorithm may be a useful heuristic for solving large real world project scheduling problems. Surprisingly, many exact algorithms for the problem $PS|prec|C_{\max}$ have mainly been evaluated on the smallest of the four test sets. The good performance of the time-oriented algorithm on the larger test sets is also interesting because the algorithm does not include features such as partial schedule based dominance pruning or explicit lower bound (re-)computation; while these features often make exact algorithms perform well on the small test set, they have the disadvantage that they are usually not easy to extend or to adapt for generalised or modified versions of the problem $PS|prec|C_{\max}$. We have also found that, for the larger test sets, even the use of the best known lower bound values available in the benchmark files of the project scheduling library PSPLIB would only marginally improve the results of the algorithm with respect to the number of optimally solved problems.

The computational analysis has shown that the difficulty of the problem instances for the algorithm depends primarily on the problem characteristics, in particular on the combination of resource supply and demand as measured by the resource strength and resource factor, and that the problem size is not the most important factor. As the hardest problems are characterised by a high share of disjunctive activities, we expect that the greatest further improvements may be achieved by concentrating on the disjunctive aspects of the problem.

Acknowledgements. This research has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Pe 514/7-2 and by the Friedrich Ebert Stiftung.

Appendix A

Proof (Proof of Lemma 1). Let S be an active schedule which is domain feasible in α. We have to prove that there exists an activity $j \in \mathscr{V}^f(\alpha)$ satisfying $ES_j(\alpha) = rc_j(\alpha)$. Since $S_j \geq ES_j(\alpha) \geq rc_j(\alpha)$, we only have to show that for some $j \in \mathscr{V}^f(\alpha)$ the identity $S_j = rc_j(\alpha)$ holds.

Suppose that $S_j > rc_j(\alpha)$ for all $j \in \mathscr{V}^f(\alpha)$. It is always possible to choose an activity $j \in \mathscr{V}^f(\alpha)$ with minimal start time in S. The choice of j implies that all its predecessors have been scheduled at α, so that obviously

$$pc_j(S) = \max_{i \in \mathscr{V}} \{S_i + p_i \mid (i,j) \in \mathscr{E}\} = \max_{i \in \mathscr{V}^s(\alpha)} \{S_i + p_i \mid (i,j) \in \mathscr{E}\} = pc_j(\alpha).$$

As S is active we know from Equation (1) that $S_j = rc_j(S)$, so that we can conclude $rc_j(S) > rc_j(\alpha)$. Since $pc_j(S) = pc_j(\alpha)$, the last inequation can only hold if there is an activity $i \in \mathscr{V}^f(\alpha)$ that finishes at time S_j and consumes resources required by j, which implies that $S_i < S_j$. This, however, contradicts the fact that the start of j is minimal among all free activities.

Appendix B: Detailed computational results

Tables 7, 8, 9, and 10 show the detailed results of the time-oriented algorithm for the problem sets with 30, 60, 90, and 120 activities, respectively. For each

Table 7. Results per problem group for test set j30

Group	Parameters			Optimal[a]	$dev._{opt}$ (%)
	C	*RF*	*RS*		
1	1.50	0.25	0.20	10	0.0
2	1.50	0.25	0.50	10	0.0
3	1.50	0.25	0.70	10	0.0
4	1.50	0.25	1.00	10	0.0
5	1.50	0.50	0.20	10	0.0
6	1.50	0.50	0.50	10	0.0
7	1.50	0.50	0.70	10	0.0
8	1.50	0.50	1.00	10	0.0
9	1.50	0.75	0.20	8	0.0
10	1.50	0.75	0.50	10	0.0
11	1.50	0.75	0.70	10	0.0
12	1.50	0.75	1.00	10	0.0
13	1.50	1.00	0.20	4	1.5
14	1.50	1.00	0.50	10	0.0
15	1.50	1.00	0.70	10	0.0
16	1.50	1.00	1.00	10	0.0
17	1.80	0.25	0.20	10	0.0
18	1.80	0.25	0.50	10	0.0
19	1.80	0.25	0.70	10	0.0
20	1.80	0.25	1.00	10	0.0
21	1.80	0.50	0.20	10	0.0
22	1.80	0.50	0.50	10	0.0
23	1.80	0.50	0.70	10	0.0
24	1.80	0.50	1.00	10	0.0
25	1.80	0.75	0.20	6	0.2
26	1.80	0.75	0.50	10	0.0
27	1.80	0.75	0.70	10	0.0
28	1.80	0.75	1.00	10	0.0
29	1.80	1.00	0.20	5	0.7
30	1.80	1.00	0.50	10	0.0
31	1.80	1.00	0.70	10	0.0
32	1.80	1.00	1.00	10	0.0
33	2.10	0.25	0.20	10	0.0
34	2.10	0.25	0.50	10	0.0
35	2.10	0.25	0.70	10	0.0
36	2.10	0.25	1.00	10	0.0
37	2.10	0.50	0.20	10	0.0
38	2.10	0.50	0.50	10	0.0
39	2.10	0.50	0.70	10	0.0
40	2.10	0.50	1.00	10	0.0
41	2.10	0.75	0.20	8	0.1
42	2.10	0.75	0.50	10	0.0
43	2.10	0.75	0.70	10	0.0
44	2.10	0.75	1.00	10	0.0
45	2.10	1.00	0.20	7	0.0
46	2.10	1.00	0.50	10	0.0
47	2.10	1.00	0.70	10	0.0
48	2.10	1.00	1.00	10	0.0

[a] Within a time limit of 300 seconds.

Table 8. Results per problem group for test set j60

Group	Parameters			Optimal[a]	dev.$_{LB}$[b] (%)
	C	*RF*	*RS*		
1	1.50	0.25	0.20	10	0.1
2	1.50	0.25	0.50	10	0.0
3	1.50	0.25	0.70	10	0.2
4	1.50	0.25	1.00	10	0.0
5	1.50	0.50	0.20	2	12.4
6	1.50	0.50	0.50	10	0.4
7	1.50	0.50	0.70	10	0.0
8	1.50	0.50	1.00	10	0.0
9	1.50	0.75	0.20	0	17.8
10	1.50	0.75	0.50	10	0.0
11	1.50	0.75	0.70	10	0.0
12	1.50	0.75	1.00	10	0.0
13	1.50	1.00	0.20	0	15.5
14	1.50	1.00	0.50	6	1.4
15	1.50	1.00	0.70	10	0.0
16	1.50	1.00	1.00	10	0.0
17	1.80	0.25	0.20	10	0.5
18	1.80	0.25	0.50	10	0.0
19	1.80	0.25	0.70	10	0.0
20	1.80	0.25	1.00	10	0.0
21	1.80	0.50	0.20	2	12.5
22	1.80	0.50	0.50	10	0.4
23	1.80	0.50	0.70	10	0.0
24	1.80	0.50	1.00	10	0.0
25	1.80	0.75	0.20	0	19.3
26	1.80	0.75	0.50	8	1.5
27	1.80	0.75	0.70	10	0.0
28	1.80	0.75	1.00	10	0.0
29	1.80	1.00	0.20	0	16.1
30	1.80	1.00	0.50	6	2.9
31	1.80	1.00	0.70	10	0.0
32	1.80	1.00	1.00	10	0.0
33	2.10	0.25	0.20	10	0.3
34	2.10	0.25	0.50	10	0.0
35	2.10	0.25	0.70	10	0.0
36	2.10	0.25	1.00	10	0.0
37	2.10	0.50	0.20	3	10.4
38	2.10	0.50	0.50	10	1.0
39	2.10	0.50	0.70	10	0.0
40	2.10	0.50	1.00	10	0.0
41	2.10	0.75	0.20	0	16.3
42	2.10	0.75	0.50	8	1.9
43	2.10	0.75	0.70	10	0.0
44	2.10	0.75	1.00	10	0.0
45	2.10	1.00	0.20	0	18.4
46	2.10	1.00	0.50	2	4.4
47	2.10	1.00	0.70	10	0.0
48	2.10	1.00	1.00	10	0.0

[a] Within a time limit of 300 seconds.
[b] Based on the best known lower bounds collected in the PSPLIB.

Table 9. Results per problem group for test set j90

Group	Parameters			Optimal[a]	dev.$_{LB}$[b] (%)
	C	*RF*	*RS*		
1	1.50	0.25	0.20	7	2.1
2	1.50	0.25	0.50	10	0.0
3	1.50	0.25	0.70	10	0.0
4	1.50	0.25	1.00	10	0.0
5	1.50	0.50	0.20	0	14.6
6	1.50	0.50	0.50	9	0.4
7	1.50	0.50	0.70	10	0.0
8	1.50	0.50	1.00	10	0.0
9	1.50	0.75	0.20	0	18.3
10	1.50	0.75	0.50	10	0.0
11	1.50	0.75	0.70	10	0.0
12	1.50	0.75	1.00	10	0.0
13	1.50	1.00	0.20	0	15.4
14	1.50	1.00	0.50	10	0.0
15	1.50	1.00	0.70	10	0.0
16	1.50	1.00	1.00	10	0.0
17	1.80	0.25	0.20	8	2.0
18	1.80	0.25	0.50	10	0.0
19	1.80	0.25	0.70	10	0.0
20	1.80	0.25	1.00	10	0.0
21	1.80	0.50	0.20	0	19.8
22	1.80	0.50	0.50	9	0.2
23	1.80	0.50	0.70	10	0.0
24	1.80	0.50	1.00	10	0.0
25	1.80	0.75	0.20	0	19.0
26	1.80	0.75	0.50	5	1.5
27	1.80	0.75	0.70	10	0.0
28	1.80	0.75	1.00	10	0.0
29	1.80	1.00	0.20	0	15.6
30	1.80	1.00	0.50	7	1.4
31	1.80	1.00	0.70	10	0.0
32	1.80	1.00	1.00	10	0.0
33	2.10	0.25	0.20	9	1.5
34	2.10	0.25	0.50	10	0.2
35	2.10	0.25	0.70	10	0.0
36	2.10	0.25	1.00	10	0.0
37	2.10	0.50	0.20	0	19.7
38	2.10	0.50	0.50	10	0.5
39	2.10	0.50	0.70	10	0.0
40	2.10	0.50	1.00	10	0.0
41	2.10	0.75	0.20	0	21.3
42	2.10	0.75	0.50	6	2.7
43	2.10	0.75	0.70	10	0.0
44	2.10	0.75	1.00	10	0.0
45	2.10	1.00	0.20	0	18.4
46	2.10	1.00	0.50	5	2.7
47	2.10	1.00	0.70	10	0.0
48	2.10	1.00	1.00	10	0.0

[a] Within a time limit of 300 seconds.
[b] Based on the best known lower bounds collected in the PSPLIB.

Table 10. Results per problem group for test set j120

Group	Parameters			Optimal[a]	dev.$_{LB}$[b] (%)
	C	*RF*	*RS*		
1	1.50	0.25	0.10	1	7.9
2	1.50	0.25	0.20	2	4.0
3	1.50	0.25	0.30	9	0.4
4	1.50	0.25	0.40	10	0.0
5	1.50	0.25	0.50	10	0.0
6	1.50	0.50	0.10	0	21.2
7	1.50	0.50	0.20	0	17.4
8	1.50	0.50	0.30	0	10.2
9	1.50	0.50	0.40	9	1.0
10	1.50	0.50	0.50	10	0.0
11	1.50	0.75	0.10	0	22.7
12	1.50	0.75	0.20	0	14.9
13	1.50	0.75	0.30	0	10.1
14	1.50	0.75	0.40	3	5.2
15	1.50	0.75	0.50	10	0.0
16	1.50	1.00	0.10	0	18.4
17	1.50	1.00	0.20	0	9.7
18	1.50	1.00	0.30	0	9.9
19	1.50	1.00	0.40	3	5.1
20	1.50	1.00	0.50	7	0.9
21	1.80	0.25	0.10	1	8.3
22	1.80	0.25	0.20	5	3.6
23	1.80	0.25	0.30	10	0.0
24	1.80	0.25	0.40	10	0.2
25	1.80	0.25	0.50	10	0.4
26	1.80	0.50	0.10	0	27.5
27	1.80	0.50	0.20	0	16.4
28	1.80	0.50	0.30	2	8.1
29	1.80	0.50	0.40	7	2.2
30	1.80	0.50	0.50	8	1.0
31	1.80	0.75	0.10	0	24.8
32	1.80	0.75	0.20	0	16.0
33	1.80	0.75	0.30	0	12.5
34	1.80	0.75	0.40	2	5.2
35	1.80	0.75	0.50	6	0.7
36	1.80	1.00	0.10	0	20.1
37	1.80	1.00	0.20	0	14.1
38	1.80	1.00	0.30	0	8.1
39	1.80	1.00	0.40	2	6.4
40	1.80	1.00	0.50	8	0.6
41	2.10	0.25	0.10	1	7.9
42	2.10	0.25	0.20	4	5.1
43	2.10	0.25	0.30	9	1.0
44	2.10	0.25	0.40	9	0.1
45	2.10	0.25	0.50	10	0.2
46	2.10	0.50	0.10	0	28.4
47	2.10	0.50	0.20	0	21.2
48	2.10	0.50	0.30	0	12.7
49	2.10	0.50	0.40	2	6.1
50	2.10	0.50	0.50	8	0.9
51	2.10	0.75	0.10	0	28.6
52	2.10	0.75	0.20	0	21.2
53	2.10	0.75	0.30	0	13.8

Table 10 (Continued)

Group	Parameters			Optimal[a]	dev.$_{LB}$[b] (%)
	C	*RF*	*RS*		
54	2.10	0.75	0.40	2	7.6
55	2.10	0.75	0.50	7	1.3
56	2.10	1.00	0.10	0	22.9
57	2.10	1.00	0.20	0	17.1
58	2.10	1.00	0.30	0	12.5
59	2.10	1.00	0.40	0	7.3
60	2.10	1.00	0.50	3	3.9

[a] Within a time limit of 300 seconds.
[b] Based on the best known lower bounds collected in the PSPLIB.

group of ten problem instances that were generated with the same parameter settings, the tables show the number of problems solved to optimality within a given time limit and the average deviation from the best known lower bound taken from the corresponding PSPLIB benchmark file.

References

Baptiste P, Le Pape C, Nuijten WP (1999) Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research 92:305–333

Błażewicz J, Lenstra JK, Rinnooy Kan A (1983) Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5:11–24

Bron C, Kerbosch J (1973) Algorithm 457: Finding all cliques of an undireced graph. Communications of the ACM 16:575–577

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112:3–41

Brucker P, Knust S (1999) A linear programming and constraint propagation based lower bound for the RCPSP. Tech. rep., University of Osnabrück

Brucker P, Knust S, Schoo A, Thiele O (1998) A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107:272–288

Carlier J, Latapie B (1991) Une méthode arborescente pour résoudre les problèmes cumulatifs. RAIRO Recherche Opérationelle 25:311–340

Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Management Science 35:164–176

Caseau Y, Laburthe F (1996) Cumulative scheduling with task intervals. In Proceedings of the Joint International Conference on Logic Programming. MIT-Press

Demeulemeester EL, Herroelen WS (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38:1803–1818

Demeulemeester EL, Herroelen WS (1997) New benchmark results for the resource-constrained project scheduling problem. Management Science 43:1485–1492

Dorndorf U, Pesch E, Phan-Huy T (2000) Constraint propagation techniques for the disjunctive scheduling problem. Artificial Intelligence 122:189–240

Dorndorf U, Phan-Huy T, Pesch E (1999) A survey of interval capacity consistency tests for time- and resource-constrained scheduling. In Project Scheduling – Recent Models, Algorithms and Applications, J. Węglarz, ed. Kluwer Academic Publishers, Boston, pages 213–238

Elmaghraby S (1977) Activity networks: Project planning and control by network models. Wiley, New York

Elmaghraby S (1995) Activity nets: A guided tour through some recent developments. European Journal of Operational Research 82:371–432
Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45:733–750
Heilmann R, Schwindt C (1997) Lower bounds for RCPSP/max. Tech. Rep. WIOR-511, University of Karlsruhe
Heipcke S, Colombani Y (1997) A new constraint programming approach to large scale resource constrained scheduling. In Third Workshop on Models and Algorithms for Planning and Scheduling Problems, Cambridge, UK
Herroelen W, Demeulemeester E, De Reyck B (1998) Resource-constrained project scheduling: A survey of recent developments. Computers & Operations Research 25:279–302
Herroelen W, Demeulemeester E, De Reyck B (1999) A classification scheme for project scheduling problems. In Project Scheduling – Recent Models, Algorithms and Applications, J. Węglarz, ed. Kluwer Academic Publishers, Boston, pages 1–26
Johnson T (1967) An algorithm for the resource-constrained project scheduling problem. Ph.D. thesis, Massachusets Institute of Technology
Klein R (1998) Bidirectional planning: Improving priority rule based heuristics for scheduling resource-constrained projects. Tech. rep., Technical University of Darmstadt
Klein R (1999) Scheduling of resource-constrained projects. Ph.D. thesis, Technical University of Darmstadt
Klein R, Scholl A (1998) Optimally solving the generalized resource-constrained project scheduling problem. Tech. rep., Technical University of Darmstadt
Klein R, Scholl A (1999a) Scattered branch and bound – an adaptive search strategy applied to resource-constrained project scheduling. Central European Journal of Operations Research 7:177–201
Klein R, Scholl A (1999b) Computing lower bounds by destructive improvement – an application to resource-constrained project scheduling. European Journal of Operational Research 112:322–346
Kolisch R (1995) Project scheduling under resource constraints: Efficient heuristics for several problem classes. Physica-Verlag, Heidelberg
Kolisch R, Hartmann S (1999) Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In Project Scheduling – Recent Models, Algorithms and Applications, J. Węglarz, ed. Kluwer Academic Publishers, Boston, pages 147–178
Kolisch R, Padman R (1997) An integrated survey of project scheduling. Tech. Rep. 463, University of Kiel
Kolisch R, Schwindt C, Sprecher A (1999) Benchmark instances for project scheduling problems. In Project Scheduling – Recent Models, Algorithms and Applications, J. Węglarz, ed. Kluwer Academic Publishers, Boston, pages 197–212
Kolisch R, Sprecher A (1996) PSPLIB – a project scheduling problem library. European Journal of Operational Research 96:205–216
Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41:1693–1703
Kumar V (1992) Algorithms for constraint-satisfaction problems: A survey. A.I. Magazine 13:32–44
Le Pape C (1994) Implementation of resource constraints in ILOG SCHEDULE: A library for the development of constraint-based scheduling systems. Intelligent Systems Engineering 3:55–66
Li R-Y, Willis J (1992) An iterative scheduling technique for resource-constrainted project scheduling. European Journal of Operational Research 56:370–379
Lopez P, Erschler J, Esquirol P (1992) Ordonnancement de tâches sous contraintes: une approche énergétique. RAIRO Automatique, Productique, Informatique Industrielle 26:453–481
Martin P, Shmoys DB (1996) A new approach to computing optimal schedules for the job-shop scheduling problem. In Proceedings of the 5th International IPCO Conference
Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. Management Science 44:715–729
Möhring RH, Schulz A, Stork F, Uetz M (1998) Resource constrained project scheduling: Computing lower bounds by solving minimum cut problems. Tech. Rep. 620, Technical University of Berlin

Nuijten WP (1994) Time and resource constrained scheduling: A constraint satisfaction approach. Ph.D. thesis, Eindhoven University of Technology
Phan Huy T (1999) Constraint propagation in flexible manufacturing. Ph.D. thesis, University of Bonn
Sprecher A (2000) Scheduling resource-constrained projects competitively at modest memory requirements. Management Science 46:710–723
Talbot FB, Patterson JH (1978) An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Management Science 24:1163–1174
Tsang E (1993) Foundations of constraint satisfaction. Academic Press, London
Van Hentenryck P, Deville Y, Teng C (1992) A generic arc-consistency algorithm and its specializations. Artificial Intelligence 57:291–321
Waltz DL (1975) Understanding line drawings of scenes with shadows. In The Psychology of Computer Vision, P. H. Winston, ed. McGraw-Hill, pages 19–91