A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalised Precedence Constraints

Ulrich Dorndorf, Erwin Pesch and Toàn Phan-Huy

Faculty of Economics, University of Bonn, Adenauerallee 24-42, D-53113 Bonn, Germany

udorndorf@acm.org, e.pesch@uni-bonn.de

Abstract

Resource-constrained project scheduling with generalised precedence constraints is a very general scheduling model with applications in areas such as make-to-order production planning. We describe a time-oriented branch-and-bound algorithm that uses constraint-propagation techniques which actively exploit the temporal and resource constraints of the problem in order to reduce the search space. Extensive computational experiments with systematically generated test problems show that the algorithm solves more problems to optimality than other exact solution procedures which have recently been proposed, and that the truncated version of the algorithm is also a very good heuristic.

1 Introduction

Resource-constrained project scheduling with generalised precedence constraints is concerned with scheduling a set of activities subject to constraints on the availability of several shared resources and temporal constraints, which allow to specify minimal and maximal time lags between the start of two activities. The objective considered in this paper is to minimise the makespan, i.e. the maximum of the completion times of all activities.

Many classical scheduling models such as the resource-constrained project scheduling problem only use minimal time lags between activities; the lags usually reflect strict activity precedence relations and are thus assumed to be equal to activity processing times. Arbitrary minimal and maximal time lags – also called generalised precedence relations or temporal constraints – are an important generalisation, as they allow to model many characteristics commonly found in practical scheduling problems. The temporal constraints can for instance be used to model activity time windows, fixed activity start times, synchronisation of start or finish times, maximal or minimal activity overlaps, non-delay execution of activities, setup times, or time varying

resource supply and demand (Bartusch et al. 1988, Elmaghraby and Kamburowski 1992, Neumann and Schwindt 1997).

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 $PS|temp|C_{\max}$, for (a) project scheduling with (b) general temporal 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 $m, 1|gpr|C_{\max}$.

While the classical resource-constrained project scheduling problem, i.e. the problem $PS|prec|C_{\max}$, has been intensively studied, algorithms for solving the problem $PS|temp|C_{\max}$ have only recently received growing attention in the OR literature as can be seen in the surveys by Herroelen et al. (1998) and Brucker et al. (1999). This may to some extent have been caused by the fact that the problem $PS|prec|C_{\max}$ itself is intractable, and as an extension, the problem $PS|temp|C_{\max}$ is, of course, also $\mathcal{NP}$-hard, and even the question whether a problem instance has a feasible solution is $\mathcal{NP}$-hard (Bartusch et al. 1988).

Different heuristics for resource-constrained project scheduling with generalised precedence constraints have been proposed, and we refer the reader to Brucker et al. (1999) for a discussion.

Exact branch-and-bound algorithms for the problem $PS|temp|C_{\max}$ have been developed by Bartusch et al. (1988), De Reyck and Herroelen (1998) (see also De Reyck et al. 1999), Schwindt (1998a,b), and Fest et al. (1999). The common idea behind these algorithms is to relax the resource constraints and compute an optimal time-feasible schedule. The resulting schedule will usually violate resource constraints and is therefore scanned for resource conflicts, i.e. times when more resources are consumed than available. The procedures then branch over the possible alternatives for resolving these conflicts. A resource conflict is resolved by adding new constraints that delay some of the activities causing the conflict (conflict set). Subject to the constraints added so far, an optimal time-feasible schedule is then re-computed and again tested for further resource conflicts. In the algorithms of Bartusch et al. (1988) and De Reyck and Herroelen (1998) activities from a conflict set are delayed by introducing additional precedence

constraints. The procedure of Schwindt (1998b) delays activities by adding special precedence constraints between pairs of disjoint sets of conflicting activities; all activities in the second set are delayed until the completion time of a first activity in the first set. The algorithm of Fest et al. (1999) resolves conflicts by dynamically increasing release dates for certain activities.

The time-oriented branch-and-bound algorithm that we describe in this paper is different in the sense that it simultaneously considers temporal and resource constraints. Instead of enumerating alternatives for resolving resource conflicts that occur in a relaxed problem, the procedure enumerates possible activity start times based on the following simple idea: 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.

Time-oriented branching schemes which branch over activity start times have previously been applied for solving several special cases of the problem considered in this paper. To the best of our knowledge 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 is to branch over all possible start time assignments of the next activity to be scheduled, and thus the number of child nodes generated at a given node of the search tree depends on the selected activity. Martin and Shmoys (1996) have used a time-oriented branching scheme for solving the job shop scheduling problem. Caseau and Laburthe (1996) have independently designed a branch-and-bound algorithm for a multi-mode project scheduling problem with classical precedence constraints which can be classified as $MPS|prec|C_{\max}$ in the scheme of Brucker et al. (1999). For the single mode case their 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 by Caseau and Laburthe has also been used in the study of Baptiste et al. (1999). Heipcke and Colombani (1997) describe 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. Since 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 $PS|temp|C_{\max}$ can be described as follows: a set of activities $\mathcal{V} = \{1, \ldots, n\}$ has to be processed with the objective of minimising the makespan. Each activity $i \in \mathcal{V}$ has a given processing time p_i and a start time S_i, which is a decision variable. By choosing sufficiently small time units we can always assume that the processing and start times are non-negative integer values. We study the non-preemptive version of the problem, which means that activities must not be interrupted during their processing.

An activity i requires $r_{ik} \in \mathbb{N}_0$ units of one or several resources $k \in \mathcal{R}$, where $\mathcal{R}$ denotes the set of all resources. For the sake of simplicity we assume that resource k is available in constant amount R_k, although the results derived in the subsequent sections also apply if we consider variable resource supply instead: for constant R_k, time varying resource supply can easily be modelled by introducing fictitious activities with fixed start times. An activity i uses exactly r_{ik} units of resource k in any interval of width one starting at time $t = S_i, \ldots, S_i + p_i - 1$, at which these units are not available for other activities, and releases them at time $t = S_i + p_i$. The set of activities which require resource k is denoted with $\mathcal{V}_k := \{i \in \mathcal{V} \mid r_{ik} > 0\}$.

In general, activities cannot be processed independently from each other due to additional technological requirements or scarcity of resources. Technological requirements will be mod-

elled by *temporal constraints* or, as a synonym, *generalised precedence constraints*. A generalised precedence constraint (i, j) specifies a minimal or maximal time lag between two activities i and j and has the general form $S_i + d_{ij} \leq S_j$. As for the activity start and processing times we will assume without loss of generality that all time lags d_{ij} are integer values. If $d_{ij} > 0$ then the constraint (i, j) can be interpreted as: activity j must start at least d_{ij} time units after the start of i (minimal time lag). If $d_{ij} \leq 0$, then the following interpretation applies: j must start at most d_{ij} time units before the start of i (maximal time lag). The set of all generalised precedence constraints is denoted with $\mathcal{E}$. The constraints can be visualised in an activity-on-node network with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ with edge weights d_{ij}, where minimal lags are usually represented as forward edges and maximal lags as backward edges.

In addition to temporal constraints, resource constraints ensure that in any processing period the resource demand never exceeds the resource supply. Given a set of activities $\mathcal{V}_k$, the difference between the supply R_k and the demand for resource k in an interval $[t_1, t_2[$ is measured by the function *slack*$(\mathcal{V}_k, t_1, t_2)$, which will be defined in detail later. The problem *PS*|*temp*|$C_{\max}$ can then conceptually be stated as follows:

$\min\{\max_{i \in \mathcal{V}}\{S_i + p_i\}\}$	s.t.	(i)
$S_i + d_{ij} \leq S_j,$	$\forall (i,j) \in \mathcal{E},$	(ii)
slack$(\mathcal{V}_k, t, t+1) \geq 0,$	$\forall t \in \mathbb{N}_0, \forall k \in \mathcal{R},$	(iii)
$S_i \in \mathbb{N}_0,$	$\forall i \in \mathcal{V}.$	(iv)

A schedule $S = (S_1, \ldots, S_{|\mathcal{V}|})$ is an assignment of all activity start times. S is *feasible* if it satisfies all precedence constraints (ii) and resource constraints (iii).

When comparing two schedules S and S' we say that $S \leq S'$ 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 *active*, if it is feasible and there exists no other feasible schedule S' such that $S' < S$. If a schedule S is not active and some activity i can therefore be started earlier than at time S_i, then we say for short that i can be left-shifted in S. It is well known that any solution method which minimises the makespan function can refrain from generating

non-active schedules, since there always exists a corresponding active schedule with a lower or identical makespan.

Finally, let us introduce the concept of activity start time domains. Each activity $i \in \mathcal{V}$ has a *current domain* $\Delta_i \subseteq \mathbb{N}_0$ of possible integral start times. We will later assume that some real or hypothetical upper bound *UB* on the optimal makespan is known or given, so that even $\Delta_i \subseteq [0, UB - p_i]$ holds. If no initial upper bound is given we use the trivial upper bound $\sum_{i \in \mathcal{V}} \max\{p_i, \max_{(i,j) \in \mathcal{E}} d_{ij}\}$.

The set of current domains of all activities is denoted with $\Delta := \{\Delta_i \mid i \in \mathcal{V}\}$. For an activity $i \in \mathcal{V}$, $ES_i(\Delta) := \min \Delta_i$ is the earliest start time, $LS_i(\Delta) := \max \Delta_i$ the latest start time, $EC_i(\Delta) := ES_i(\Delta) + p_i$ the earliest 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., for short.

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 $\mathcal{V}$ 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 $\mathcal{V}^s(\Delta) := \{i \in \mathcal{V} \mid |\Delta_i| = 1\}$ is the set of scheduled activities, and $\mathcal{V}^f(\Delta) := \{i \in \mathcal{V} \mid |\Delta_i| > 1\}$ is the set of free activities. For all scheduled activities $i \in \mathcal{V}^s(\Delta)$, the start time is defined through $S_i(\Delta) := ES_i(\Delta) = LS_i(\Delta)$.

3 Constraint Propagation

The branch-and-bound algorithm that will be described in the next section relies to a great extent on efficient constraint propagation techniques. Constraint propagation is an elementary method for reducing the search space of a problem instance through the repeated analysis and evaluation of variables, their domains, and the interdependence between the variables that is induced by the set of constraints. The goal is to detect and remove *inconsistent* start time assignments, which cannot participate in any feasible schedule. A whole theory is devoted to the definition of different concepts of consistency, which may serve as a theoretical background

for propagation techniques (see e.g. Tsang 1993).

Removing *all* inconsistent assignments is in general not possible due to exponentially increasing computational complexity. As a consequence we have to content ourselves with approximations. An important task will be to describe simple rules which allow efficient search space reductions but at the same time can be implemented efficiently. These rules are called *consistency tests*. Consistency tests are generally described through a condition and a domain reduction rule. Whenever the condition is satisfied, the reduction rule can be applied.

A consistency test can be identified with a function γ which has as input a set Δ of current domains and as output a hopefully, but not necessarily reduced set of current domains. A consistency test γ must only remove values that do not belong to any schedule which is domain feasible with respect to Δ.

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 reducing several domains, additional domain adjustments can possibly be derived using some of the tests which previously failed in deducing any reductions. Therefore, the reduction process is carried out until no more updates are possible, i.e. until a fixed point $CP(\Delta)$ is computed. Note that this point does not have to be unique and in general depends upon the order of the application of the consistency tests. However, we will only study consistency tests which result in a unique fixed point. These tests satisfy a monotony condition described in Dorndorf et al. (1998), which is sufficient for the uniqueness of the fixed point.

The constraint propagation algorithm that is actually used to compute $CP(\Delta)$ in our implementation is a variant of the AC-5 arc-consistency algorithm described by Van Hentenryck et al. (1992). Like all improved 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. 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.

Each constraint/value pair will enter the queue only once, if at all, and the maximum number of elements enqueued and dequeued by the algorithm thus depends on (1) the number of constraints and (2) the number of variables per constraint and their domain sizes. If $d := \max_{i \in \mathcal{V}} |\Delta_i|$ is the size of the largest domain, then we obtain for example at most $O(|\mathcal{E}|\, d)$ enqueueing and dequeueing operations for the temporal constraints. Given the number of queue operations, the overall complexity of the propagation algorithm can then be deduced from the complexity of the consistency tests. As we will see below, the worst case effort is dominated by the resource constraints and is $O(|\mathcal{R}|\, |\mathcal{V}|^3\, d)$. It is worth mentioning that the worst case in terms of computational effort is also a best case in the sense that it corresponds to reducing the domains until all activities are scheduled; the average propagation effort is usually much lower.

In the next subsections we will derive some elementary consistency tests based on precedence and resource constraints.

3.1 Precedence Consistency

A temporal or precedence constraint (i, j) of the form $S_i + d_{ij} \leq S_j$ determines the minimal or maximal time lag that must pass between the start of two activities i and j. Clearly, the left side of the constraint is minimal for ES_i, and a lower bound on the earliest possible start of activity j is thus given by $ES_i + d_{ij}$. Likewise, the right side of the constraint is maximal for LS_j, and $LS_j - d_{ij}$ is an upper bound on the latest possible start of i. This leads to the following well known test:

Consistency Test 1 (Precedence Consistency). *For any precedence constraint (i, j) the following domain reduction rules apply:*

$$\Delta_i := \Delta_i \backslash]LS_j - d_{ij}, \infty[\,, \qquad \Delta_j := \Delta_j \setminus [0, ES_i + d_{ij}[\,.$$

When used within the constraint propagation algorithm, these reduction rules, of course, lead to the same result as a "forward-backward" time window calculation in a project network. A logical contradiction in the precedence constraints (corresponding to a cycle of positive length in the project network) will lead to an empty domain for some activity.

A temporal constraint (i, j) is always satisfied (*resolved*) if the maximal value of the left side is smaller than or equal to the minimal value of the right side, i.e. if $LS_i + d_{ij} \leq ES_j$, otherwise it is unresolved.

For a given precedence constraint, consistency test 1 can be applied with constant effort. The worst case propagation effort caused by the precedence constraints is therefore determined by the $O(|\mathcal{E}|\, d)$ possible enqueueing and de-queueing operations.

3.2 Interval Capacity Consistency

Interval consistency tests are based on the comparison of resource supply and resource demand within a given time interval. Each activity requires a total amount of work $w_{ik} := r_{ik}p_i$ from resource k. We say that a time interval is capacity consistent if the amount of work requested by all activities within this time interval can be matched by the amount of work supplied.

Let us consider the work of an activity i that must fall into a time interval $[t_1, t_2[$, where $t_1 < t_2$, given the current domains Δ. The *interval processing time* $p_i(t_1, t_2)$ of an activity i is the smallest amount of time during which i has to be processed within $[t_1, t_2[$. There are five possible situations: (1) i can be completely contained within the interval, (2) completely overlap the entire interval when started either as early as possible or as late as possible, (3) have a minimum processing time in the interval that is realized when started as early as possible or (4) have a minimum processing time that is realized when started as late as possible. The fifth situation applies whenever i does not have to be processed, neither completely nor partially, within the given time interval. Consequently,

$$p_i(t_1, t_2) := \max\{0, \min\{p_i, t_2 - t_1, EC_i - t_1, t_2 - LS_i\}\}. \tag{1}$$

The corresponding *interval work* is $w_{ik}(t_1, t_2) = r_{ik}p_i(t_1, t_2)$. The interval work of a subset of activities $\mathcal{A} \subseteq \mathcal{V}_k$ is defined through $W(\mathcal{A}, t_1, t_2) := \sum_{i \in A} w_{ik}(t_1, t_2)$. Using this definition

of interval work 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 interval:

$$\mathit{slack}(\mathcal{A}, t_1, t_2) := R_k \cdot (t_2 - t_1) - W(\mathcal{A}, t_1, t_2). \tag{2}$$

Observe that the slack function depends on the actual set Δ of current domains, so we will write $\mathit{slack}_\Delta(\mathcal{A}, t_1, t_2)$ whenever necessary. An interval $[t_1, t_2[$ is capacity consistent if $\mathit{slack}(\mathcal{V}_k, t_1, t_2) \geq 0$ for every resource k. Given a domain set Δ, we can only develop a solution if this necessary condition holds for all resources and all time intervals.

The concept of interval capacity consistency as defined here has been introduced by Lopez et al. (1992) under the name *energetic reasoning*; special cases of this concept have been known for a long time (see e.g. Zaloom 1971). For an in-depth survey of consistency tests based on interval capacity considerations we refer to Dorndorf et al. (1999).

An important special case of condition (2) is obtained when we consider time intervals of unit width. If, for some time t, $slack(\mathcal{V}_k \setminus \{i\}, t, t+1)$ is less than the required resource amount r_{ik}, then activity i cannot be processed at time t. This leads to the following test, which is also known under the name timetable-based constraint propagation (Le Pape 1994).

Consistency Test 2 (Unit Interval Consistency). *Let $i \in \mathcal{V}_k$. If, for some time t, $\mathit{slack}_\Delta(\mathcal{V}_k \setminus \{i\}, t, t+1) < r_{ik}$, then the domain of i can be reduced by setting $\Delta_i := \Delta_i \setminus\,]t - p_i, t]$.*

The test can be efficiently implemented through capacity profiles reflecting available and used capacity over time. A capacity profile can be initialised and updated by using the fact that an activity i must always be in process in its *core time* between its latest start and earliest completion time, i.e. $p_i(t, t+1) = 1$ for all $t \in [LS_i, EC_i[$ and $p_i(t, t+1) = 0$ otherwise. The capacity profile can therefore only change at points in time corresponding to the latest start and earliest completion time of an activity and is thus represented using $O(|\mathcal{V}|)$ support points.

Whenever a resource constraint and the information, that values have been deleted from the domain of a variable S_i, are removed from the constraint propagation queue, an update of the capacity profile may be required as the domain reduction may have lead to a new or modified core time of i. Since the core time modification may overlap the entire profile, the worst case

updating effort is $O(|\mathcal{V}|)$. If the profile is updated in the interval $[t_1, t_2[$, then we may apply the unit interval consistency test for any unscheduled activity and any time $t \in [t_1, t_2[$. The effort required for applying the test in the basic form stated above thus depends on the width of the interval $[t_1, t_2[$. However, the effort can be reduced by only considering the relevant support points for $[t_1, t_2[$. Let t_k and t_{k+1} be two such consecutive support points. Clearly, if an activity cannot be in process at time t_k, then in cannot be processed anywhere in $[t_k, t_{k+1}[$. We therefore only need to test the condition of consistency test 2 at the relevant support points t_k and may strengthen the reduction rule by removing all times in the interval $]t_k - p_i, t_{k+1}[$. Updating the capacity profile and testing all activities at all modified support points requires a worst case effort $O(|\mathcal{V}|^2)$. Although this worst case effort is actually the same as would be required for checking all activities against the complete profile, the average effort is typically lower.

The unit interval consistency test affects the computational complexity of the constraint propagation algorithm in the following way. There are $|\mathcal{R}|$ resource constraints with $|\mathcal{V}|$ variables with a maximum domain size d, causing at most $O(|\mathcal{R}|\ |\mathcal{V}|\ d)$ enqueue and dequeue operations. Since the application of the test for a given queue element requires effort $O(|\mathcal{V}|^2)$, the overall propagation effort caused by the resource constraints and consistency test 2 is $O(|\mathcal{R}|\ |\mathcal{V}|^3\ d)$. This worst case effort is based on two conservative assumptions. The first assumption is that a single update of the capacity profile caused by a domain modification may affect the entire profile. If the duration of an activity is small compared to the project duration, then the core time of the activity will only cover very few support points and the update can be performed in almost constant time. The second assumption is that every domain modification leads to a core time change and to an update of the capacity profile. However, if the initial domains are large compared to the processing times, then the number of updates leading to core time modifications will be smaller. For these reasons, the average time complexity of test for typical problems can be expected to be significantly lower than in the worst case.

3.3 Disjunctive Consistency

Let us now turn to another consistency test which is based on considering pairs of disjunctive activities. Two activities $i, j \in \mathcal{V}$ are in disjunction if, due to limited resource availability, either i has to finish before j can start, or j has to finish before i can start. If i and j must be processed sequentially, then we can apply the following well known consistency test, which has first been described by Carlier and Pinson (1989):

Consistency Test 3 (Interval Based Disjunctive Consistency). *Let $i, j \in \mathcal{V}$ be in disjunction. If $LS_i - ES_j < p_j$, then i must precede j.*

If i must precede j then we may add the precedence constraint $S_i + p_i \leq S_j$. We denote the set of all such precedence constraints added through the application of a disjunctive consistency test with $\mathcal{E}^d$. The domain reduction then follows from the precedence consistency test, which is, of course, applied for all constraints $\mathcal{E} \cup \mathcal{E}^d$.

The interval based disjunctive consistency test draws conclusions based on the (absolute) start time domains of pairs of disjunctive activities. Additional domain reductions may be deduced by considering the (relative) minimal time lags between two disjunctive activities. Let $D := (d'_{ij})$ be the matrix of transitive minimal temporal distances between activities that is induced by the constraint set $\mathcal{E} \cup \mathcal{E}^d$. D can for instance be calculated with effort $O(|\mathcal{V}|^3)$ with the Floyd-Warshall Algorithm (Lawler 1976). Using these transitive time lags, we can state the following observation (De Reyck and Herroelen 1998):

Consistency Test 4 (Lag Based Disjunctive Consistency). *Let $i, j \in \mathcal{V}$ be in disjunction. If $d'_{ij} > -p_j$, then i must precede j.*

Note that the condition $d'_{ij} > -p_j$ means that j cannot finish before the start of i. Clearly, the test is only useful if $d'_{ij} < p_i$. Again, we add any precedence constraint resulting from the application of this test to the set $\mathcal{E}^d$, and the corresponding domain reduction then follows from the precedence consistency test. The test also detects infeasibilities that occur if the temporal constraints require that two activities i and j, which are in disjunction, must be processed in

parallel for some time. In this case two contradicting precedence constraints are added, and the precedence consistency test consequently leads to an empty domain.

The matrix D depends upon the temporal constraints $\mathcal{E} \cup \mathcal{E}^d$. Whenever a disjunctive consistency test adds a new precedence constraint to the set $\mathcal{E}^d$, the matrix is updated with effort $O(|\mathcal{V}|^2)$ (cf. Bartusch et al. 1988).

The consistency tests 3 and 4 are only applicable for pairs of disjunctive activities. Let us therefore consider in more detail the question when two activities i and j are in disjunction. This is obviously the case if their combined resource requirements exceed the available capacity, i.e. if $r_{ik} + r_{jk} > R_k$ for some resource k. However, this condition can be relaxed by only considering the slack for a small time interval that depends on the current domains of i and j.

Lemma 1 (Disjunctive Activities). *Consider $i, j \in \mathcal{V}^f(\Delta)$ and the interval $[t_1, t_2[$ defined by*

$$t_1 := \max\{\min\{EC_i, EC_j\} - 1, \max\{ES_i, ES_j\}\},$$

$$t_2 := \min\{\max\{LS_i, LS_j\} + 1, \min\{LC_i, LC_j\}\}.$$

Activities i and j are in disjunction, if there is a resource $k \in \mathcal{R}$ required by both i and j and if

$$\mathit{slack}_\Delta(\mathcal{V}_k \setminus \{i, j\}, t, t+1) < r_{ik} + r_{jk}, \qquad \forall t \in [t_1, t_2[\,. \tag{3}$$

Proof. If condition (3) is satisfied for the interval $[t_1, t_2[$ then either (a) i or j must finish before t_1 or start after t_2, or (b) the two activities must be in disjunction. It is now easy to show that whenever (a) holds then (b) must also be satisfied. □

Finding all pairs of disjunctive activities using Lemma 1 requires effort $O(|\mathcal{V}|^2)$. If two activities i and j are found to be disjunctive then we add the disjunctive constraint $S_i + p_i \leq S_j \ \vee \ S_j + p_j \leq S_i$, which takes part in the propagation mechanism. If a disjunctive constraint is removed from the queue then consistency test 3 is applied with constant effort. As we may add $O(|\mathcal{V}|^2)$ disjunctive constraints, the overall propagation effort caused by the application of consistency test 3 is therefore $O(|\mathcal{V}|^2\, d)$. Again, the average effort is typically much lower.

An update of the matrix D and the application of consistency test 4 as well as of Lemma 1 require effort $O(|\mathcal{V}|^2)$. Although this worst case effort seems small compared to other components of the propagation algorithm, the average case effort required for these steps is equal to

the worst case effort. We therefore found it most useful to only apply Lemma 1, consistency test 4, and updates of D during an initial fixed point algorithm at the root of the search tree.

3.4 Some Properties of the Earliest Start Times

The precedence and unit interval consistency tests that are applied within the fixed point 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 $\mathcal{V}^s(\Delta)$ and j are considered:

$$pc_j(\Delta) := \max_{i \in \mathcal{V}^s(\Delta)} \{S_i + d_{ij} \mid (i,j) \in \mathcal{E}\}. \tag{4}$$

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:

$$rc_j(\Delta) := \min_{t \geq pc_j(\Delta)} \{t \mid \forall k \in \mathcal{R}, \forall t' \in [t, \ldots, t + p_j[: slack_\Delta(\mathcal{V}_k \setminus \{j\}, t', t'+1) \geq r_{jk}\}.$$

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

$$S_j = rc_j(S) \tag{5}$$

holds for all $j \in \mathcal{V}$.

Since we may w.l.o.g. assume that an activity has at most $|\mathcal{V}| - 1$ predecessors, the calculation of pc_j requires effort $O(|\mathcal{V}|)$. The calculation of rc_j is based upon pc_j and a traversal of the support points of the remaining capacity profile and requires effort $O(|\mathcal{R}|\,|\mathcal{V}|)$.

4 The Branch-and-Bound Algorithm

The main component of the branch-and-bound algorithm is a time-oriented, binary branching scheme. We will show that this branching scheme generates at least all active schedules, so that

traversing the search tree will result in an optimal solution. Inversely, the branching scheme tries to avoid constructing non-active schedules, which cuts down the search space considerably. A detailed description of the branching scheme is given in Section 4.1.

As a peculiarity our algorithm does *not* explicitly compute lower bounds. Indeed, 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.

Additionally, the search space is 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. This is discussed in Section 4.2.

4.1 The Branching Scheme

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 that will be extensively used is the application of constraint propagation. Since in general, however, constraint propagation alone does not schedule all activities, some activities additionally will 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 time $ES_j(\alpha)$ by setting $S_j(l(\alpha)) := ES_j(\alpha)$.

$r(\alpha)$: Increase the earliest start time 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.

Selection of Activities 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 $j \in \mathcal{V}^{f'}(\alpha)$, such that $ES_j = t(\alpha)$ where $t(\alpha)$ is the schedule time:

$$t(\alpha) := \min_{i \in \mathcal{V}^{f'}(\alpha)} ES_i(\alpha). \tag{6}$$

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 with *act*(α) the activity chosen in α.

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 $\mathcal{V}^{f'}(\alpha) := \mathcal{V}^{f}(\alpha)$. This, however, is not sensible, since choosing an arbitrary free activity will often 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.

It will prove useful to partition the set of free activities into a set of activities which still have to satisfy a maximal time lag and a set of activities which do not have to. Let $\mathcal{E} = \mathcal{E}^{min} \cup \mathcal{E}^{max}$, where $\mathcal{E}^{min} := \{(i,j) \in \mathcal{E} \mid d_{ij} > 0\}$ and $\mathcal{E}^{max} := \{(i,j) \in \mathcal{E} \mid d_{ij} \leq 0\}$ are the relations specifying the minimal and maximal time lags between pairs of activities. We then define

$$\mathcal{V}^{tc}(\alpha) := \{j \in \mathcal{V}^{f}(\alpha) \mid \exists i \in \mathcal{V}^{f}(\alpha) : (i,j) \in \mathcal{E}^{max}\}$$

as the set of *timemax-constrained* activities and the set $\mathcal{V}^{tu}(\alpha) := \mathcal{V}^{f}(\alpha) \setminus \mathcal{V}^{tc}(\alpha)$ of *timemax-unconstrained* activities.

We can now describe the set of free and non-delayed activities:

$$\mathcal{V}^{f'}(\alpha) := \mathcal{V}^{tc}(\alpha) \cup \{j \in \mathcal{V}^{f}(\alpha) \mid ES_j(\alpha) = rc_j(\alpha)\}$$

A free activity is a candidate for branching if it either has an "incoming" backward arc, or if its earliest start time equals its current earliest resource feasible start time $rc_i(\alpha)$. Note that the latter condition may in particular not be given if the constraint propagation algorithm has adjusted $ES_j(\alpha)$ to some value greater than $rc_j(\alpha)$, or if an activity has been delayed (by an amount of time to be defined below). The definition of the set of free and selectable activities $\mathcal{V}^{f'}$ can therefore be interpreted as follows: a delayed activity i without an incoming backward arc remains un-selectable until we know that the resource capacity "provided" by delaying i has been used by some other activity. The following lemma justifies our choice of the set $\mathcal{V}^{f'}$.

Lemma 2 (Existence of Earliest Start Time Schedules). *Let α be a node of the search tree. If there is an unscheduled activity then $\mathcal{V}^{f'}(\alpha)$ is not empty, or α cannot lead to an active schedule.*

Proof. Let S be an active schedule which is domain feasible in α, and let us assume that $\mathcal{V}^{tc}(\alpha) = \emptyset$. We then have to prove that there exists an activity $j \in \mathcal{V}^{f}(\alpha)$ 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 $j \in \mathcal{V}^{f}(\alpha)$ the identity $S_j = rc_j(\alpha)$ holds.

Suppose that $S_j > rc_j(\alpha)$ for all $j \in \mathcal{V}^{f}(\alpha)$. Observe that the set of timemax-unconstrained activities $\mathcal{V}^{tu}(\alpha)$ is not empty, since $\mathcal{V}^{f}(\alpha)$ is not empty. It is therefore possible to choose an activity $j \in \mathcal{V}^{tu}(\alpha)$ with minimal start time in S:

$$S_j = \min_{i \in \mathcal{V}^{tu}(\alpha)} S_i. \tag{7}$$

Using the obvious identity $\mathcal{V}^{s}(S) = \mathcal{V}$, equation (4) tells us that

$$pc_j(S) = \max_{i \in \mathcal{V}}\{S_i + d_{ij} \mid (i,j) \in \mathcal{E}\},$$

If there exists a precedence constraint $(i,j) \in \mathcal{E}^{min}$, then $i \in \mathcal{V}^{s}(\alpha)$, since otherwise $S_i + d_{ij} \leq S_j$ and $d_{ij} > 0$ immediately imply $S_i < S_j$, which is a contradiction to equation (7). If

$(i,j) \in \mathcal{E}^{max}$, then $i \in \mathcal{V}^s(\alpha)$ follows directly from $j \in \mathcal{V}^{tu}(\alpha)$. So for all $(i,j) \in \mathcal{E}$, we have $i \in \mathcal{V}^s(\alpha)$ and the last equation can be simplified as follows:

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

Domain feasibility allows to deduce the identity $S_i = S_i(\alpha)$ for all $i \in \mathcal{V}^s(\alpha)$, which leads to

$$pc_j(S) = \max_{i \in \mathcal{V}^s(\alpha)} \{S_i(\alpha) + d_{ij} \mid (i,j) \in \mathcal{E}\} = pc_j(\alpha). \tag{8}$$

As S is active we know from equation (5) that $S_j = rc_j(S)$, so that we can conclude $rc_j(S) > rc_j(\alpha)$. More formally

$$\begin{aligned} & \min_{t \geq pc_j(S)} \{t \mid \forall k \in \mathcal{R}, \forall t' \in [t, \ldots, t + p_j[: slack_S(\mathcal{V}_k, t', t'+1) \geq r_{jk}\} \\ > & \min_{t \geq pc_j(\alpha)} \{t \mid \forall k \in \mathcal{R}, \forall t' \in [t, \ldots, t + p_j[: slack_\alpha(\mathcal{V}_k, t', t'+1) \geq r_{jk}\}. \end{aligned}$$

Because $pc_j(S) = pc_j(\alpha)$, this means that there must be some resource $k \in \mathcal{R}$, such that for $t = S_j - 1$ the following condition holds:

$$slack_\alpha(\mathcal{V}_k - \{j\}, t, t+1) \geq r_{jk} \qquad \wedge \qquad slack_S(\mathcal{V}_k - \{j\}, t, t+1) < r_{jk}.$$

If the slack of period t in S is smaller than the slack of this period at node α, then the interval processing time $p_v(t, t+1)$ of at least one activity $v \in \mathcal{V}^f(\alpha) = \mathcal{V}^{tu}(\alpha)$ must assume the value 0 in α and 1 in S. According to the definition of interval processing times in (1), $p_v(t, t+1) = 1$ implies that $t + 1 - S_v > 0$. We thus obtain $S_v \leq t < S_j$, which is a contradiction to condition (7). So, in fact, there must exist $j \in \mathcal{V}^f(\alpha)$ with $S_j = rc_j(\alpha)$. □

Delaying Duration Let us now turn to the 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, the first case can be ruled out if all precedence constraints $(i,j) \in \mathcal{E}$ are already resolved in node α. The second case requires that the slack of all activities except j is insufficient to the left of $S_j(\alpha)$.

Intuitively, this can only be the case if $S_j(\alpha)$ matches the completion time of some activity that shares resources with j. This leads to the following lemma, in which $\mathcal{R}_i := \{k \in \mathcal{R} \mid r_{ik} > 0\}$ denotes the set of resources required by activity i.

Lemma 3 (Delaying Duration). *Let α be the current node of the search tree and all $(i,j) \in \mathcal{E}$ be resolved for $j = act(\alpha)$. The set of all activities that share resources with j and finish after $t(\alpha)$ is denoted with $\mathcal{V}' := \{i \in \mathcal{V} \setminus \{j\} \mid \mathcal{R}_i \cap \mathcal{R}_j \neq \emptyset \ \wedge \ EC_i(\alpha) > t(\alpha)\}$. Let further*

$$t^+(\alpha) := \begin{cases} \min_{i \in \mathcal{V}'} EC_i & \text{if } \mathcal{V}' \neq \emptyset, \\ t(\alpha) + 1 & \text{otherwise.} \end{cases}$$

Then $S_j \geq t^+(\alpha)$ in any active schedule S developed from $r(\alpha)$.

Proof. We need only consider the case where $\mathcal{V}' \neq \emptyset$. If j is delayed in $r(\alpha)$ and S is active then, according to equation (5), $rc_j(S) = S_j > t(\alpha)$. If $rc_j(S) > t(\alpha)$ then, obviously,

$$rc_j(S) \geq pc_j(S) > t(\alpha) \quad \vee \quad rc_j(S) > t(\alpha) \geq pc_j(S).$$

If, for the given j, all $(i,j) \in \mathcal{E}$ are resolved, then $LS_i + d_{ij} \leq ES_j$ for all $(i,j) \in \mathcal{E}$. Thus $pc_j(S) \leq t(\alpha)$ and the first condition cannot hold. Now consider the second condition. We will show that any time $t = rc_j(S)$ satisfying this condition must correspond to some completion time. If $rc_j(S) > t(\alpha) \geq pc_j(S)$ then there must be some time t and some resource $k \in \mathcal{R}$ for which:

$$\mathit{slack}_S(\mathcal{V}_k \setminus \{j\}, t-1, t) < r_{jk} \ \wedge \ \mathit{slack}_S(\mathcal{V}_k \setminus \{j\}, t, t+1) \geq r_{jk}.$$

This immediately implies that there must be some activity in $\mathcal{V}_k \setminus \{j\}$ that is processed in the interval $[t-1, t[$ but not in $[t, t+1[$, i.e. an activity which finishes at time t.

We have thus derived that if j is delayed from $ES_j(\alpha)$ and the resulting schedule is active, then $S_j = rc_j(S)$ must equal some completion time $t > t(\alpha)$. Therefore we can conclude that $ES_j(r(\alpha))$ must be greater than or equal to an earliest completion time greater than $t(\alpha)$. Of course, we need only consider activities that share a common resource with j. □

It is worth mentioning that the precedence constraints $(i,j) \in \mathcal{E}$ are always resolved if j has only incoming arcs with positive weight, i.e. if $j \in \mathcal{V}^{tu}(\alpha)$.

At the root ρ Let $\Delta_i{}' := [0, UB - p_i]$ for all $i \in \mathcal{V}$. Then: $\Delta(\rho) := CP(\Delta')$.

In node α Let $\Delta(\alpha) = \{\Delta_i(\alpha) \mid i \in \mathcal{V}\}$ be the set of current domains in α and $j := act(\alpha)$ the activity chosen in α.

Branching to the left $l(\alpha)$ Let $\Delta'(\alpha) := \{\Delta_1(\alpha), \ldots, \Delta_j{}'(\alpha), \ldots, \Delta_n(\alpha)\}$, where

$$\Delta_j{}'(\alpha) := \{t(\alpha)\}.$$

Then: $\Delta(l(\alpha)) := CP(\Delta'(\alpha))$.

Branching to the right $r(\alpha)$ Let $\Delta''(\alpha) := \{\Delta_1(\alpha), \ldots, \Delta_j{}''(\alpha), \ldots, \Delta_n(\alpha)\}$, where

$$\Delta_j{}''(\alpha) := \begin{cases} \Delta_j(\alpha) \cap [t(\alpha)+1, \infty[& \text{if there is an unresolved } (i,j) \in \mathcal{E}^{max} \\ \Delta_j(\alpha) \cap [t^+(\alpha), \infty[& \text{otherwise.} \end{cases}$$

Then: $\Delta(r(\alpha)) := CP(\Delta''(\alpha))$.

Figure 1: *The branching scheme*

Summary of the Branching Scheme We are now able to define the branching scheme recursively. This is done in Figure 1; recall that we only have to specify $\Delta(\alpha)$, since this determines all other sets and values.

The search tree is traversed in depth-first order until a leaf node is generated. This happens whenever $\mathcal{V}^{f'}(\alpha) = \emptyset$. This leaf node represents a solution, if $\mathcal{V}^s(\alpha) = \mathcal{V}$. The makespan of an initial or improved schedule is, of course, used as 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 $i \in \mathcal{V}$.

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 is possible 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 $|\mathcal{V}| - 1$ activities. As delayed activities may remain un-selectable at least until some other activity is scheduled, i.e.

until we can conclude that the resource capacity which has been released by delaying the activities has been used, we must then schedule some activity or backtracking would be initiated. Next we can at most branch $|\mathcal{V}| - 2$ 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 $1/2|\mathcal{V}|(|\mathcal{V}| + 1)$.

The following theorem states that our time-oriented branching scheme is complete, i.e. that an optimal schedule is generated. As we have already discussed in Section 2, it is sufficient to prove that all active schedules can be generated.

Theorem 4 (Completeness of Time-Oriented Branching). *The time-oriented branching scheme generates 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 $i \in \mathcal{V}$.*

Proof. Let S be an active schedule. We will first prove the following assertion: if S is domain feasible in α, then S is domain feasible in either $l(\alpha)$ or $r(\alpha)$.

Lemma 2 ensures that $\mathcal{V}^{f'}(\alpha)$ is not empty, so that there exists an activity $j \in \mathcal{V}^{f'}$ which is selected in α. Now, if $S_j = ES_j(\alpha)$, then S is domain feasible with respect to $\Delta'(\alpha)$ as defined in Figure 1. Constraint propagation only removes values from current domains Δ_i that do not belong to any schedule which is domain feasible with respect to Δ. This implies that S must be domain feasible with respect to $\Delta(l(\alpha)) = \mathit{CP}(\Delta'(\alpha))$. If $S_j > ES_j(\alpha)$ then a similar argumentation in combination with Lemma 3 shows that S must be domain feasible with respect to $\Delta(r(\alpha))$.

We can conclude, that there exists a path $\rho, \alpha_1, \alpha_2, \ldots$, along which S is domain feasible. Let $|\Delta| := \sum_{i \in \mathcal{V}} |\Delta_i|$. Given the finiteness of the current domains, $\infty > |\Delta(\rho)| > |\Delta(\alpha_1)| > |\Delta(\alpha_2)| > \ldots \geq n$ must hold. This implies, that S is domain feasible in some node α_m satisfying $|\Delta(\alpha_m)| = n$, i.e. $\mathcal{V}^s(\alpha_m) = \mathcal{V}$. This completes the proof. □

4.2 Some Properties of Active Schedules

This section describes some additional conditions and a simple left-shift test that aim at further reducing the search space by ruling out non-active schedules. We make use of an effect caused by the activity selection rule: choosing an activity $j \in \mathcal{V}^{f'}(\alpha)$ with minimal earliest start time,

which, according to equation (6), determines the schedule time $t(\alpha)$, ensures that any time point smaller than $t(\alpha)$ does not have to be considered any more.

Clearly, the selection rule implies that in any schedule S developed from α the condition $S_j \geq t(\alpha)$ must hold for all $j \in \mathcal{V}^{f'}(\alpha)$. But there might be free and delayed activities $j \in \mathcal{V}^{f}(\alpha) \setminus \mathcal{V}^{f'}(\alpha)$ for which $ES_j(\alpha) < t(\alpha)$ 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, the following lemma states that this cannot happen if the resulting schedule is active.

Lemma 5 (Start of Delayed Activities). *Let α be a node of the search tree and let S be an active schedule that is domain feasible in α. Then:*

$$S_j > t(\alpha), \quad \textit{for all } j \in \mathcal{V}^{f}(\alpha) \setminus \mathcal{V}^{f'}(\alpha).$$

Proof. Similar to proof of Lemma 2. □

We can directly use Lemma 5 in order to reduce the search space in the following way. At node α we additionally set

$$ES_i(\alpha) := t(\alpha) + 1, \quad \text{for all } i \in \mathcal{V}^{f}(\alpha) \setminus \mathcal{V}^{f'}(\alpha) : ES_i \leq t(\alpha),$$

before applying constraint propagation. The start time adjustment can be further improved by applying a similar argument as in Lemma 3. 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. Because the adjustment of a single start time requires constant effort, the total adjustment effort is $O(|\mathcal{V}|)$.

Lemma 5 and the fact that $S_i \geq t(\alpha)$ for all $i \in \mathcal{V}^{f}(\alpha)$ also imply the following result.

Corollary 6 (Constant Slack to the Left of $t(\alpha)$). *Let α be a node of the search tree. Then the slack in any period $t < t(\alpha)$ does not change in descendant nodes of α that lead to an active schedule.*

This allows us to apply a simple left-shift dominance test. If, for any free, timemax-unconstrained activity $j \in \mathcal{V}^{tu}(\alpha)$ 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. While it is possible to formulate more powerful left-shift conditions that consider sets of activities rather than just a single activity (Schwindt 1998b), the advantage of the test described here is that it can be easily evaluated. The effort for the left-shift dominance test for all free, timemax-unconstrained activities is $O(|\mathcal{V}|^2)$ since rc_j must be calculated for every activity in $\mathcal{V}^{tu}(\alpha)$.

The fact that the slack to the left of $t(\alpha)$ remains constant can be exploited further. Let j be an activity scheduled at node α at time $t(\alpha)$. If $rc_j(\alpha) < t(\alpha)$ then sufficient slack and the temporal constraints involving the currently scheduled activities admit a left-shift of j. Hence a resulting schedule S can only be active if a temporal constraint involving a currently *unscheduled* activity prevents this left-shift. This means that the following condition must hold in order for S to be active:

$$\exists (i,j) \in \mathcal{E}^{max} : \ (i \in \mathcal{V}^f(\alpha) \ \wedge \ S_i + d_{ij} = S_j).$$

We add a corresponding constraint that takes part in the propagation mechanism. The consistency test for this constraint works in the following way. If no temporal constraint can satisfy this condition, then the node is fathomed. Otherwise, if only one single temporal constraint (i,j) can satisfy the condition, then the domain of activity i can be adjusted.

The effort required to test whether the constraint may be added is dominated by the calculation of rc_j. The constraint is a disjunction over the temporal constraints with $O(|\mathcal{V}|)$ possible predecessors and can be defined in time $O(\mathcal{V}|)$. Since a constraint of this type can be added whenever an activity is scheduled, there may be $O(|\mathcal{V}|)$ of these constraints. The constraints may thus cause $O(|\mathcal{V}|^2\, d)$ enqueueing and dequeueing operations in the constraint propagation algorithm. The corresponding consistency test can be performed with effort $O(|\mathcal{V}|)$. The overall worst case propagation effort caused by this constraint and test is therefore $O(|\mathcal{V}|^3\, d)$. Again, if the number of predecessors of an activity is small as in typical project scheduling problems, then the average effort is lower.

5 Computational Results

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 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 temporal constraints and the corresponding consistency test 1 are realised through the built-in linear constraints provided by SOLVER. The resource constraints and the unit interval consistency test 2 are provided by SCHEDULER. For the administration of the temporal and resource constraints we have used the activity and resource classes of SCHEDULER. Consistency test 3 is implemented as a generic disjunctive SOLVER constraint. The logic of the branch-and-bound algorithm, the other consistency tests and the additional node fathoming rules described in Section 4 have 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.

We have tested the algorithm on two large sets of benchmark problems that were systematically generated by Schwindt (1998b) using the problem generator ProGen/Max (Schwindt

1996). The first test set contains 1080 problems with 100 activities; only 1059 problems have a feasible solution. The second set consists of 120 problems with 500 activities; 119 of these problems have a feasible solution. A detailed description of the characteristics of the test sets is given by Schwindt (1998b).

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

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 (see e.g. Klein 1999). Intuitively, if the difficult part, or 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 chose 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.

Table 1 shows the impact of the different modules of our algorithm for the set of 1080 problems with 100 activities. For a given algorithm version, which is characterised by the presence or absence of the modules, and a given run time limit $t_{\max}$ the table shows the percentage of problems for which (1) a feasible solution could be found, (2) for which an optimal solution was found and verified, (3) for which infeasibility was proven, and (4) the average deviation MRE_{LB} from the lower bounds calculated by Schwindt (1998b). Except for the MRE_{LB} values, all percentages are given with respect to the total number of 1080 problems. For comparison purposes the percentages for the average deviation from the lower bound are given with respect to the number of problems solved to feasibility, including those solved to optimality[1].

[1]The deviation of a problem instance with (possibly optimal) upper bound UB_i and lower bound LB_i is $(UB_i - LB_i)/LB_i$. This means that problems that were solved to optimality but where the lower bound is not tight have a positive deviation and that the lowest possible MRE_{LB} value is therefore greater than zero. The average deviations are approximately 0.1 percentage points smaller if the deviation of an instance solved to optimality is always set

Version of the algorithm					$t_{\max}$	*feasible*	*optimal*	*infeasibility proven*	MRE_{LB}
No.	B[a]	D[b]	A[c]	BP[d]	*(sec)*	(%)	(%)	(%)	(%)
1	–	–	–	–	3	91.1	55.9	0.0	5.7
2	+	–	–	–	3	95.6	61.7	0.0	5.6
3	+	+	–	–	3	96.3	63.3	1.9	5.5
4	+	+	+	–	3	96.7	64.2	1.9	5.5
5	+	+	+	+	3	97.8	66.2	1.9	5.2
6	–	–	–	+	3	96.0	61.1	0.0	5.8
7	+	–	–	+	3	97.4	63.5	0.0	5.3
8	+	+	–	+	3	97.7	65.7	1.9	5.2
9	–	–	–	–	30	91.1	56.0	0.0	5.7
10	+	–	–	–	30	96.8	63.1	0.0	5.6
11	+	+	–	–	30	97.2	65.6	1.9	5.5
12	+	+	+	–	30	97.5	67.0	1.9	5.2
13	+	+	+	+	30	98.1[e]	70.4	1.9	4.8
14	–	–	–	+	30	96.0	61.2	0.0	5.8
15	+	–	–	+	30	98.0	65.1	0.0	5.1
16	+	+	–	+	30	98.1[e]	67.9	1.9	5.0

[a]Branching: + indicates that $\mathcal{V}^{f'}$ and $t^+(\alpha)$ are defined as in Section 4.1; otherwise $\mathcal{V}^{f'} := \mathcal{V}^f$ and $t^+(\alpha) := t(\alpha) + 1$.

[b]Disjunctive consistency tests: + indicates use of consistency tests 3 and 4.

[c]Active schedules: + indicates use of the tests and conditions described in Section 4.2.

[d]Bidirectional planning.

[e]Corresponding to 100% of the problems that have a feasible solution.

Table 1: *Impact of different modules of the algorithm for 1080 problems with 100 activities*

The first five columns of the table characterise different versions of the algorithm; in addition to a reference number they show whether a particular module has been used (+) or omitted (–) in a version. In order to keep the size of the table within reasonable limits we have grouped related features into modules and present data for several interesting module combinations.

Rows 1 and 9 of the table show the results obtained for the minimal version of our algorithm in which only the precedence and the unit interval consistency tests are applied within the constraint propagation algorithm. Observe that these tests are always required as they are the only means by which the algorithm will obey the temporal and resource constraints. In the minimal version, we use a very basic activity selection rule where any free activity is selectable, i.e. we set $\mathcal{V}^{f'} := \mathcal{V}^f$, and the simple delaying strategy of always postponing an activity by a single time unit, i.e. we set $t^+(\alpha) := t(\alpha) + 1$. The advanced activity selection and delaying rules described in Section 4.1 are referred to as the branching module which is shown as column B. The minimal version does not use the disjunctive consistency tests 3 and 4 (column D), it does not apply the tests and conditions for active schedules described in Section 4.2 (column A), and it does not use bidirectional planning (column BP). Row 1 of the table shows that within a time limit of 3 seconds the minimal algorithm solves 91.1% of the problems to feasibility and 55.9% to optimality; it cannot prove the infeasibility of any of the 21 infeasible problems, and the average deviation from the lower bound is 5.7%. As Row 9 shows, these results are hardly improved within the tenfold run time.

The minimal version is then improved by activating the advanced branching module; the results are shown in Row 2 (10). Rows 3 (11) and 4 (12) show the effect of adding the disjunctive consistency tests described in Section 3.3 and the active schedule dominance rules described in Section 4.2. When the disjunctive tests are used infeasibility can be proven at the root node for 20 of the 21 infeasible instances. Row 5 (13) shows the impact of applying the full algorithm bidirectionally, i.e. to the original problem and to the mirror problem. The table shows that the more advanced versions of the algorithm solve more problems to feasibility and optimality than their simpler counterparts while at the same time achieving a smaller average deviation from

to zero.

the lower bound.

Row 6 (14) shows the results for the minimal version of the algorithm with bidirectional planning. For the smaller time limit the improvement with respect to Version 1 is comparable to the effect obtained by the advanced branching module shown in Row 2. However, Row 14 shows that in contrast to the other modules bidirectional planning alone does hardly lead to further improvements within the higher run time. By comparing Rows 7 and 8 (15 and 16) to Rows 2 and 3 (10 and 11) we can see that the combination of bidirectional planning and the other modules has a positive effect. It is interesting to note that in contrast to the minimal version with or without bidirectional planning the higher run time always leads to improved results and that all modules contribute to the improvements.

Table 2 compares the results obtained with our algorithm for the test set of 1080 problems with 100 activities to those of the three most recent other exact solution approaches by — in historical order — De Reyck and Herroelen (1998), Schwindt (1998a, and personal communication), and Fest et al. (1999), who have all used the same test set. De Reyck et al. (1999) describe a newer version of the procedure of De Reyck and Herroelen; the improvements mainly concern a different conflict detection and resolution mechanism (the conflicts are resolved in a different, more effective, sequence) as well as more efficient coding, which has led to slightly improved results (personal communication De Reyck 1999); however, as test data for this new version for Schwindt's benchmark problem set is not available, Table 2 shows the results published in De Reyck and Herroelen (1998). For run time limits t_{max} of 3, 30, and 100 seconds, including a scaling factor to account for different hardware, the table shows the percentage of problems for which (1) a feasible solution could be found, (2) for which an optimal solution was found and verified, (3) for which infeasibility was proven, and (4) the average deviation MRE_{LB} from the lower bounds calculated in the study of Schwindt (1998b). Dashes indicate that the corresponding information is not available.

For comparison purposes the MRE_{LB} values for our algorithm and for the algorithms of Fest et al. (1999) and Schwindt (1998b) were all calculated in the way described above using the

Procedure	$t_{\max}$	*feasible*	*optimal*	*infeasibility proven*	MRE_{LB}
	(sec)	(%)	(%)	(%)	(%)
Time-oriented B&B	3	97.8	66.2	1.9	5.2
	30	98.1[a]	70.4	1.9	4.8
	100	98.1[a]	71.7	1.9	4.6
Fest, Möhring, Stork & Uetz	3	92.2	58.1	1.9	10.9
	30	98.1[a]	69.4	1.9	7.7
	100	98.1[a]	71.1	1.9	7.0
Schwindt	3	98.1[a]	58.0	1.9	7.5
	30	98.1[a]	62.5	1.9	7.0
	100	98.1[a]	63.4	1.9	6.9
De Reyck & Herroelen	3[b]	97.3	54.8	1.4	—[c]
	30[b]	97.5	56.4	1.4	—[c]
	100[b]	—	—	—	—

[a]Corresponding to 100% of the problems that have a feasible solution.
[b]Corresponding to 60/200 of the real computation time.
[c]Published values are based on different lower bounds than values for the other procedures.

Table 2: *Results of exact algorithms for 1080 problems with 100 activities*

lower bounds of Schwindt[2]. Since the deviations reported by De Reyck and Herroelen (1998) are based on different, possibly weaker bounds, the corresponding fields are left empty.

The results of De Reyck and Herroelen have been obtained on a Pentium/60 PC; the run time limits used in their study were 1, 10, and 100 seconds. Schwindt has used a Pentium/200 PC and Fest et al. have used a Sun Ultra with 200 MHz clock pulse. As mentioned above, our results have been obtained on a Pentium Pro/200 PC. For comparison purposes the run time

[2]In contrast to the values shown for our algorithm and the procedure of Schwindt, the values shown for the algorithm of Fest et al. have been calculated by setting the deviation of a problem instance solved to optimality to zero, leading to a slightly smaller average value. However, in our experience the resulting difference is negligible.

limits for all procedures but the one of De Reyck and Herroelen were set to 3, 30, and 100 seconds, thus reflecting the clock pulse ratio.

The table shows that the time-oriented branch-and-bound algorithm solves more problems to optimality than the other procedures. With respect to this criterion, the results obtained within 3 seconds are already better than the results obtained with the procedures of Schwindt or De Reyck and Herroelen within the maximum allowed time. Within a limit of 30 seconds, a feasible solution for all 1059 problems that can be feasibly solved is found; only Schwindt's algorithm, which applies a cycle structure based decomposition heuristic at the root node for finding initial upper bounds, finds a feasible solution for all problems within 3 seconds and does better on this criterion.

The interpretation of the mean remaining error with respect to the lower bound (MRE_{LB}) can be problematic since this value depends on the individual problems that are solved to feasibility as well as on the lower bounds used for calculating MRE_{LB}. Strictly speaking, two MRE_{LB} values can therefore only be compared if they are both based on the same bounds and on the same subset of problems that were solved feasibly. The MRE_{LB} values shown for the first three algorithms are all based on Schwindt's lower bounds, and the values shown for time limits of 30 seconds or more are based upon all instances that have a feasible solution. We would therefore like to point out that the mean remaining error of the solutions found by our algorithm is significantly lower than that of the procedures of Fest et al. and Schwindt.

Because our algorithm does not use explicit lower bounds, we were interested in the possible improvement that could be achieved by adding such bounds. To partially answer this question we have used the lower bounds of Schwindt and have examined those test problems for which our algorithm could find a solution matching a lower bound without being able to prove optimality within the time limit. We found that for one of the 1080 test problems our algorithm finds a solution matching a lower bound but cannot prove optimality within 3 seconds. Within 30 seconds, this solution is proven to be optimal, and for another problem a solution matching a lower bound is found without proof of optimality; this problem remains open after 100 seconds. This means that the results of our algorithm could only be marginally improved by using these

Procedure	t_{max} (sec)	t_{avg} (sec)	*feasible* (%)	*optimal* (%)	$C_{\max} = LB$ (%)	MRE_{LB} (%)
Time-oriented B&B	3	1.3	99.7	67.5	64.6	5.2
	30	9.4	100.0	71.8	66.0	4.8
	100	29.7	100.0	73.1	66.1	4.6
Franck & Neumann						
Direct	—	0.5	99.4	—	56.8	7.7
Contraction	—	1.3	100.0	—	42.5	9.4
Franck & Selle						
GA_{prec}	—	16.0[a]	100.0	—	59.9	5.3
GA_{vary}	—	16.0[a]	81.1	—	61.0	2.0
Tabu Search	—	16.6[a]	100.0	—	56.0	5.8
Simulated Annealing	—	10.4[a]	100.0	—	59.5	5.7

[a]Corresponding to 266/200 of the real computation time.

Table 3: *Comparison of heuristics for 1059 of the 1080 problems with 100 activities*

lower bounds. Data concerning the tightness of the lower bounds can be found in Table 3.

Table 3 compares our algorithm to the best heuristic results that have been reported for the same problem set, this time using only the 1059 solvable instances. In addition to the columns shown in the previous tables, column "t_{avg}" shows the average required run time, and column "$C_{\max} = LB$" contains the percentage of problems for which a solution with a value matching a lower bound was found. The results for our algorithm are identical to those shown in the corresponding rows in Tables 1 and 2, except that all percentages in the columns "feasible", "optimal", and "$C_{\max} = LB$" are now given with respect to the 1059 solvable problems. Again, all values regarding lower bounds shown in the table are based on the bounds of Schwindt. The time-oriented algorithm has been tested on a Pentium Pro/200 PC; the algorithms of Franck and Neumann (1998) have been run on a Pentium/200 PC, and Franck and Selle (1998) have used a Pentium/266 PC. As before, we have scaled the run times according to the clock pulse.

The results of Franck and Neumann (1998) have been obtained by applying a combination of serial and parallel list scheduling algorithms using several different priority rules; the algorithms include limited backtracking capabilities. The basic idea behind the direct and the contraction method is to give preferential treatment to activities which are on cycle structures induced by the temporal constraints. The two approaches differ in the specific way in which they handle cycle structures; the contraction heuristic initially solves subproblems defined by the activities and corresponding precedence constraints on the same cycle structure and then integrates these solutions in a complete schedule. The results of Franck and Neumann greatly improve upon the results reported by Schwindt (1998b) for the older priority rule based heuristics of Zhan (1994) and Brinkmann and Neumann (1996), which can solve approximately 98% of the problems with an average deviation from the lower bound of roughly 80%. This indicates the progress that has been made in this area in the past years.

Franck and Selle (1998) have improved these results by embedding a variant of the direct method in four meta-heuristics, specifically in two genetic algorithms (GA) based on two different solution encodings and in a tabu search and simulated annealing framework. The meta-heuristics all manipulate the order in which activities are scheduled by the list scheduling algorithm, which thus serves for evaluating (neighbouring) solutions. The table shows that, at the cost of an increased average run time, the meta-heuristics solve more problems to optimality than the priority rule based methods and achieve a significantly smaller average deviation from the lower bound. The low average deviation from the lower bound shown for the second genetic algorithm is probably caused by the fact that this procedure reaches the smallest number of feasible solutions; this conjecture is supported by the observation that the 81.1% of the problems with lowest individual deviation that are found by our algorithm within a maximum time of 3 seconds have an average deviation of 0.9%.

Other heuristics have been developed by Schwindt (1998b) based upon truncated versions of his branch-and-bound algorithm. However, since the newer version of his algorithm described in Schwindt (1998a) and cited in Table 2 improves upon the results of these heuristics, we do not present them in Table 3.

Procedure	$t_{\max}$ *(sec)*	t_{avg} *(sec)*	*feasible* (%)	*optimal* (%)	$C_{\max} = LB$ (%)	MRE_{LB} (%)
Time-oriented B&B	200	98	97.5	71.4	61.3	0.5
	1000	306	99.2	77.3	61.3	0.5
Fest, Möhring, Stork & Uetz	200	—	100.0	58.8	—	5.2
	1000	—	100.0	58.8	—	3.8
Franck & Neumann						
Direct	—	56	84.9	—	40.3	1.2
Contraction	—	18	100.0	—	5.0	5.1
Neumann & Zimmermann						
Filtered Beam Search	—	14	80	—	62	0.1
Decomposition	200	51	100	—	6	5.0

Table 4: *Results for 119 of 120 large problems with 500 activities*

When comparing the time-oriented algorithm to the priority rule based heuristics of Franck and Neumann we can observe that for average run times in the order of magnitude of one second the algorithm finds more solutions matching a lower bound while achieving a very small average deviation. However, the contraction method is faster at finding feasible solutions for all problems. It can also be seen that for average times in the order of magnitude of 10 seconds the time-oriented algorithm performs better with respect to all criteria shown in the table than any of the meta-heuristics that can solve all problems.

In order to demonstrate the scalability of our algorithm Table 4 presents results for the second test set of 120 problem instances with 500 activities. For comparison, the table also shows the results reported by Fest et al. (1999), the only other exact procedure for which results have been published for this test set. The table also contains the results obtained by Franck and Neumann (1998) for their priority rule based heuristics, and by Neumann and Zimmermann (1999) for the two branch-and-bound based heuristics that they found most effective for this test

set in terms of the criteria reported in Table 4. The latter heuristics are based on the algorithm of Schwindt (1998b). Similar to the priority rule based contraction method, the decomposition heuristic initially solves subproblems corresponding to the cycle structures. All percentages except for those in the MRE_{LB} column are based only on the 119 problem instances that have a feasible solution. Again, the lower bounds used for calculating the mean remaining errors have been found in the study of Schwindt (1998b). The results of Neumann and Zimmermann as well as those of Franck and Neumann have been obtained on Pentium/200 PCs.

The results in Table 4 show that our algorithm scales quite well. Within 200 seconds, the algorithm solves 71.4% of the problems to optimality and leaves only 3 of the 119 feasible problems unsolved; the infeasibility of the remaining problem is proven at the root node. For a time limit of 1000 seconds, 118 of the 119 problems that have a feasible solution are solved to feasibility and 92 instances or 77.3% to optimality; the time-oriented algorithm also achieves a very small average deviation from the lower bound. The table shows that those procedures which can also solve the remaining problem(s) left open by the time-oriented algorithm can only do so at the price of a significantly lower solution quality, as indicated by the MRE_{LB} values. The number of problems solved to optimality within the maximum allowed time is 18.5 percentage points, corresponding to 22 problems, higher than for the algorithm of Fest et al.

We have also experimented with more sophisticated consistency tests than described in this paper (cf. Dorndorf et al. 1999), but have been unable to improve the results presented above. In all cases the additional search space reduction has been outweighed by the increase in computation time; the additional tests that we have tried require effort $O(|\mathcal{V}^2|)$ or $O(|\mathcal{V}^3|$. Baptiste et al. (1999) report a similar observation for the problem $PS|prec|C_{\max}$.

6 Conclusions

We have presented a branch-and-bound algorithm for a very general scheduling model, the resource-constrained project scheduling problem with generalised precedence relations, with the objective of minimising the project makespan. The algorithm uses a binary, time-oriented branching scheme that relies on efficient constraint propagation techniques for reducing the

search space. The power of constraint propagation lies in the systematic and computationally efficient application of basic consistency tests. The search effort is reduced further by adding some necessary conditions that must be satisfied by active schedules. The algorithm can also easily be applied for optimising other regular measures of performance.

Given the conventional wisdom that the efficiency of branch-and-bound procedures depends largely on good lower bounds, it is quite interesting to note that our algorithm does not use any *explicit* lower bounds. Instead, lower bounding is implicitly achieved through the constraint propagation process.

Computational experiments on large test sets of systematically generated benchmark problems taken from the literature demonstrate the effectiveness of the approach. On a data set of over thousand problem instances with one hundred activities each, the algorithm finds feasible solutions for all problems and it solves more problems to optimality than other methods, while at the same time achieving a significantly smaller deviation from a lower bound for those instances for which optimality cannot be proven. The results obtained for another test set consisting of problems with five hundred activities show that the algorithm also scales very well. In addition, the truncated version of the algorithm compares favourably to the best heuristic procedures for the problem.[3]

References

Baptiste, P., C. Le Pape and W. P. Nuijten. 1999. Satisfiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems. *Annals of Oper. Res.* **92**, 305–333.

Bartusch, M., R. H. Möhring and F. J. Radermacher. 1988. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Oper. Res.* **16**, 201–240.

Brinkmann, K. and K. Neumann. 1996. Heuristic Procedures for Resource-Constrained Project Scheduling with Minimal and Maximal Time Lags: The Resource Levelling and Minimum Project-Duration Problems. *J. Decision Systems* **5**, 129–156.

[3]We would like to thank Rolf Möhring, Frederik Stork, and Marc Uetz for their valuable comments on an earlier draft of this paper. We are indebted to Christoph Schwindt for making his test data available to us. We would also like to acknowledge the helpful comments of two anonymous referees.

Brucker, P., A. Drexl, R. Möhring, K. Neumann and E. Pesch. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *EJOR* **112**, 3–41.

Carlier, J. and E. Pinson. 1989. An Algorithm for Solving the Job-Shop Problem. *Management Sci.* **35**, 164–176.

Caseau, Y. and F. Laburthe. 1996. Cumulative Scheduling with Task Intervals. In *Proc. of the Joint International Conf. on Logic Programming*. MIT-Press.

De Reyck, B., E. Demeulemeester and W. Herroelen. 1999. Algorithms for Scheduling Projects with Generalised Precedence Relations. In *Project Scheduling — Recent Models, Algorithms and Applications*, J. Węglarz, ed. Kluwer Academic Publ., Boston, 77–105.

De Reyck, B. and W. Herroelen. 1998. A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Constraints. *EJOR* **111**, 152–174.

Dorndorf, U., E. Pesch and T. Phan-Huy. 1998. Constraint Propagation Techniques for the Disjunctive Scheduling Problem. Tech. rep., Univ. Bonn.

Dorndorf, U., T. Phan-Huy and E. Pesch. 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 Publ., Boston, 213–238.

Elmaghraby, S. E. 1977. *Activity Networks: Project Planning and Control by Network Models*. Wiley, New York.

Elmaghraby, S. E. and J. Kamburowski. 1992. The Analysis of Activity Networks under Generalized Precedence Relations. *Management Sci.* **38**, 1245–1263.

Fest, A., R. H. Möhring, F. Stork and M. Uetz. 1999. Resource Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates. Tech. Rep. 596, TU Berlin.

Franck, B. and K. Neumann. 1998. Resource-Constrained Project Scheduling with Time Windows: Structural Questions and Priority Rule Methods. Tech. Rep. WIOR-492, Univ. Karlsruhe.

Franck, B. and T. Selle. 1998. Metaheuristics for the Resource-Constrained Project Scheduling

Problem with Schedule-Dependent Time Windows. Tech. Rep. WIOR-546, Univ. Karlsruhe.

Heipcke, S. and Y. Colombani. 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., E. Demeulemeester and B. De Reyck. 1998. Resource-Constrained Project Scheduling: A Survey of Recent Developments. *Computers & Oper. Res.* **25**, 279–302.

Herroelen, W., E. Demeulemeester and B. De Reyck. 1999. A Classification Scheme for Project Scheduling Problems. In *Project Scheduling — Recent Models, Algorithms and Applications*, J. Węglarz, ed. Kluwer Academic Publ., Boston, 1–26.

Klein, R. 1999. *Scheduling of Resource-Constrained Projects*. Ph.D. thesis, TU Darmstadt.

Lawler, E. L. 1976. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, New York.

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.

Lopez, P., J. Erschler and P. Esquirol. 1992. Ordonnancement de tâches sous contraintes: une approche énergétique. *RAIRO Automatique, Productique, Informatique Industr.* **26**, 453–481.

Martin, P. and D. B. Shmoys. 1996. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. In *Proc. of the* 5^{th} *Internat. IPCO Conf.*.

Neumann, K. and C. Schwindt. 1997. Activity-on-Node Networks with Minimal and Maximal Time Lags and their Application to Make-to-Order Production. *OR Spektrum* **19**, 205–217.

Neumann, K. and J. Zimmermann. 1999. Methods for Resource-Constrained Project Scheduling with Regular and Nonregular Objective Functions and Schedule-Dependent Time Windows. In *Project Scheduling — Recent Models, Algorithms and Applications*, J. Węglarz, ed. Kluwer Academic Publ., Boston, 213–287.

Schwindt, C. 1996. ProGen/max: Generation of Resource-Constrained Scheduling Problems with Minimal and Maximal Time Lags. Tech. Rep. WIOR-489, Univ. Karlsruhe.

Schwindt, C. 1998a. A Branch-and-Bound Algorithm for the Resource-Constrained Project

Duration Problem Subject to Temporal Constraints. Tech. Rep. WIOR-544, Univ. Karlsruhe.

Schwindt, C. 1998b. *Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern*. Shaker Verlag, Aachen.

Talbot, F. B. and J. H. Patterson. 1978. An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems. *Management Sci.* **24**, 1163–1174.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press, London.

Van Hentenryck, P., Y. Deville and C. Teng. 1992. A Generic Arc-Consistency Algorithm and its Specializations. *Artificial Intelligence* **57**, 291–321.

Zaloom, V. 1971. On the Resource Constrained Project Scheduling Problem. *AIIE Trans.* **3**, 302–305.

Zhan, J. 1994. Heuristics for Scheduling Resource-Constrained Projects in MPM Networks. *EJOR* **76**, 192–205.