Böttcher, Jan; Drexl, Andreas; Kolisch, Rainer; Salewski, Frank



# Project scheduling under partially renewable resource constraints

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 398

**Provided in Cooperation with:**
Christian-Albrechts-University of Kiel, Institute of Business Administration

No. 398

# Project Scheduling Under Partially Renewable Resource Constraints*

Böttcher/Drexl/Kolisch/Salewski

July 1996



Jan Böttcher, Andreas Drexl, Rainer Kolisch, Frank Salewski

Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, D-24098 Kiel

tel & fax +49 (0) 431 / 880-1531

e-mail Drexl@bwl.uni-kiel.de, Kolisch@bwl.uni-kiel.de
Salewski@bwl.uni-kiel.de.

URL http://www.wiso.uni-kiel.de/bwlinstitute/Prod
ftp://www.wiso.uni-kiel.de/pub/operations-research

---

* Supported by the Deutsche Forschungsgemeinschaft

*Abstract:* We consider a generalization of the classical resource constrained project scheduling problem. We introduce so-called partially renewable resources by assuming for each resource a capacity on subsets of periods. The concept of partially renewable resources is a fundamental tool in order to make e.g. timetabling and shift scheduling aspects amenable to project scheduling. In addition, partially renewable resources serve to model complicated labor regulations. Furthermore, they cover traditional renewable and nonrenewable resource constraints as special cases.

We consider makespan minimization as objective. For the exact solution of the problem we employ a basic enumeration scheme. In order to speed up convergence, we formulate bounds which take into account future resource consumption of partially renewable resources. Moreover, we generalize the serial scheduling scheme in order to get fast approximation methods.

A rigorous assessment of the procedures is provided by solving ProGen instances generated under a full factorial test design. Besides the well-known problem parameters we employ additionally three parameters which control the generation of partially renewable resources.

*Keywords:* Project scheduling, resource constraints, partially renewable resources, branch-and-bound algorithm, serial scheduling scheme

## 1 Background and Motivation

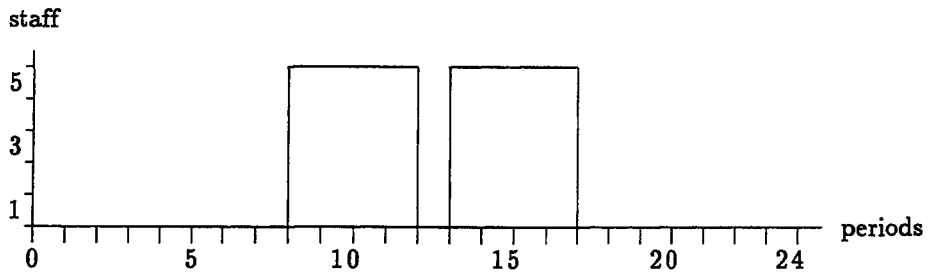
A recurring problem in project management involves the allocation of scarce resources to activities. In scheduling theory renewable and nonrenewable resources are usually distinguished. The usage of renewable resources is limited for every period while nonrenewable resources are restricted to an overall consumption within the whole planning horizon. These resource types only allow to formulate capacity constraints for exclusively one or all of the periods. Partially renewable resources are obtained by assuming for each resource a capacity on subsets of periods. The concept of partially renewable resources is a fundamental tool in order to make e.g. timetabling and shift scheduling aspects amenable to project scheduling. In addition, it serves to model complicated labor regulations. Furthermore, it covers traditional renewable and nonrenewable resource constraints as special cases. While renewable resources are defined on subsets consisting of exactly one period, nonrenewable resources are subject to the set of all periods of the planning horizon.

Resource constrained project scheduling has attracted considerable attention recently, cf. e.g. Błażewicz et al. 1986, Christofides et al. 1987, Bell and Han 1991, Demeulemeester and Herroelen 1992, 1995, Mingozzi et al. 1994, Leon and Balakrishnan 1995, and Brucker et al. 1996. More specific, the classical resource constrained project scheduling problem (RCPSP) has been the main subject of concern. We consider a generalization of the RCPSP, which makes use of partially renewable resources. For short this generalization is denoted as RCPSP/$\pi$. We consider makespan minimization as objective. For the exact solution of the RCPSP/$\pi$ we employ the basic enumeration scheme of Talbot and Patterson 1978. In order to speed up convergence, we formulate

bounds which take into account future resource consumption of partially renewable resources. In addition, we generalize the serial scheduling scheme of Kelley 1963. The methods are evaluated for a set of instances generated with the project generator ProGen, cf. Kolisch et al. 1995.

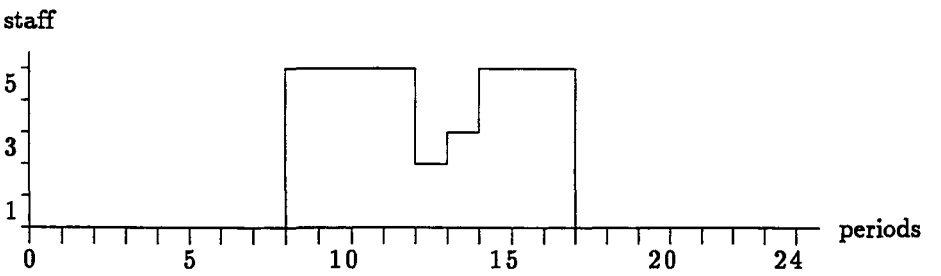
*Example 1:* In order to further motivate the RCPSP/$\pi$ we consider the case, where the resource type "staff" has to be scheduled. Figure 1 depicts that no worker is available before period 9 and after period 17, respectively. Moreover, 5 workers are available during periods 9 to 12 and during periods 14 to 17, respectively. In period 13 all the workers have their lunch break. Then e.g. no non-preemptable activity with a duration of at least 3 periods is allowed to start in period 11.

*Figure 1: Availability of Staff (Variant 1)*




More flexibility is offered if each of the workers could have his break either in period 13 or in period 14. In Figure 2 e.g. 3 (2) workers have their lunch break in period 13 (14). This would now allow to start an activity with the duration of 3 periods in period 11 if it requests less than 3 workers per period. Unfortunately, there exist several possibilities to specify in advance which workers should have their lunch break either in period 13 or in period 14. Clearly, this is impractical and does not provide the degree of flexibility which is necessary when we want to schedule a one hour lunch break for each worker either in period 13 or in period 14 without fixing the period in advance.

*Figure 2: Availability of Staff (Variant 2)*




The "skyline" (cp. Elmaghraby 1977) of resource profiles in both figures illustrates that the modelling capabilities of renewable resources are not sufficient to tackle more general and complicated situations. In this paper we show, that the RCPSP/$\pi$ is able to do this.

The outline of the paper is as follows: In Section 2 we present formal models of the RCPSP/$\pi$. The modelling capabilities of the RCPSP/$\pi$ are illustrated by some examples

in Section 3. Section 4 provides exact and heuristic solution procedures. In Section 5 we present details of the experimental evaluation. Section 6 gives a summary and an outlook to future research.

## 2 Model Formulation

The RCPSP/$\pi$ can be stated as follows:

- We consider a single project which consists of the set $J$ of activities. Let $\mathcal{V}_j$ define the set of immediate predecessors of activity $j \in J$. For ease of notation the activities are topologically ordered, i.e. each predecessor of activity $j$ has a smaller number than $j$. Furthermore, activity $j=1$ ($j=|J|$) is defined to be the unique dummy source (sink).
- Activity $j$ has a non-preemptable duration of $d_j$ periods.
- Let $T$ be the set of periods during which the activities must be processed. Furthermore, $t \in T$ denotes a specific period.
- $R'$ denotes the set of partially renewable resources. The activities are interrelated by resource constraints as follows: In order to be processed, activity $j$ requires $k_{jr}$ units of resource $r \in R'$ during every period of its duration $d_j$.
- Let $P(r,\pi)$ denote the $\pi$-th subset of periods in which resource $r \in R'$ is available with resource capacity $\kappa_{r\pi}$. For resource $r \in R'$ in total we have the set $\Pi(r)$ of subsets of periods $P(r,\pi)$, i.e. $\pi = 1,...,|\Pi(r)|$.
- The objective of the RCPSP/$\pi$ is the minimization of the makespan such that precedence and resource constraints are met.

We derive earliest and latest finish times $EF_j$ and $LF_j$, respectively, by traditional critical path analysis. Let denote $E_j := \{EF_j,...,LF_j\}$ and $Q_{jt} := \{t,...,t+d_j-1\}$. Now, based on the decision variables $x_{jt} = 1$, if activity $j$ is finished in period $t$ (0, otherwise), the RCPSP/$\pi$ can be modelled as follows:

$$\min \quad \sum_{t \in E_{|J|}} t \cdot x_{|J|t} \tag{1}$$

$$\text{s.t.} \quad \sum_{t \in E_j} x_{jt} = 1 \qquad (j \in J) \tag{2}$$

$$\sum_{t \in E_h} t \cdot x_{ht} \leq \sum_{t \in E_j} (t-d_j)\, x_{jt} \qquad (j \in J,\ h \in \mathcal{V}_j) \tag{3}$$

$$\sum_{j \in J} k_{jr} \sum_{t \in P(r,\pi)} \sum_{q \in Q_{jt} \cap E_j} x_{jq} \leq \kappa_{r\pi} \qquad (r \in R',\ \pi \in \{1,...,|\Pi(r)|\}) \tag{4'}$$

$$x_{jt} \in \{0,1\} \qquad (j \in J,\ t \in E_j) \tag{5}$$

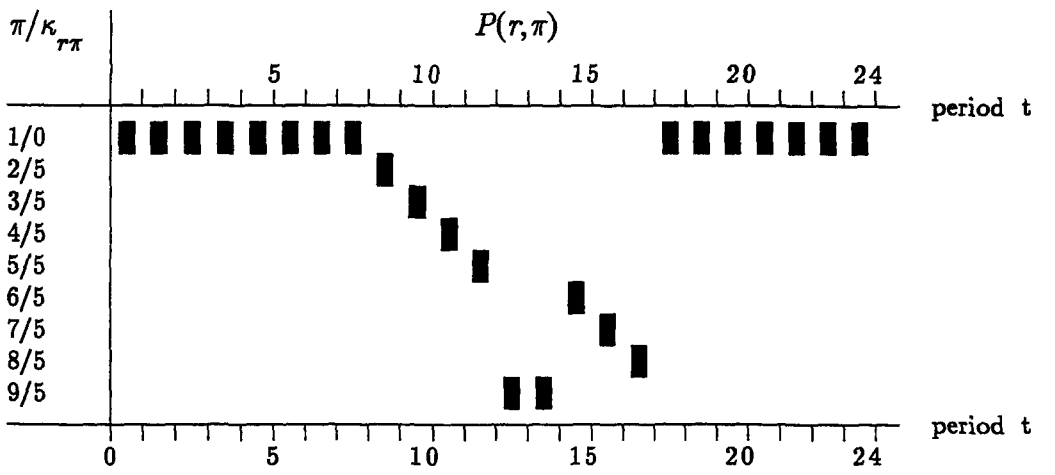
The objective function (1) minimizes the completion time of the unique sink and thus the makespan of the project. Equations (2) are activity completion constraints. Constraints (3) take into consideration the precedence relations between each pair of activities $(h, j)$, where $h$ immediately preceeds $j$. Finally, constraints (4') limit the total usage of the (partially renewable) resources to the available amount. Note that Icmeli and Rom 1994 use the term "partially renewable resource" as well. Within their framework, resources are renewable at time milestones and within time intervals they are nonrenewable. Clearly, their definition is close to so-called doubly-constrained resources (cp. e.g. Błażewicz et al. 1986), while ours is not.

Example 1 now serves to illustrate the RCPSP/$\pi$. The following parameter instantiations appropriately define the problem under consideration in terms of the RCPSP/$\pi$ and allow to schedule a one our break for each of the five workers without fixing in advance whether the break is in period 13 or in period 14.

$P(1,1) = \{1,...,8,18,...,24\}$, $\kappa_{1,1} = 0$;
$P(1,2) = \{9\}$, $\kappa_{1,2} = 5$; $P(1,3) = \{10\}$, $\kappa_{1,3} = 5$; $P(1,4) = \{11\}$, $\kappa_{1,4} = 5$;
$P(1,5) = \{12\}$, $\kappa_{1,5} = 5$; $P(1,6) = \{15\}$, $\kappa_{1,6} = 5$; $P(1,7) = \{16\}$, $\kappa_{1,7} = 5$;
$P(1,8) = \{17\}$, $\kappa_{1,8} = 5$; $P(1,9) = \{13,14\}$, $\kappa_{1,9} = 5$.

Figure 3 provides a graphical representation of the availability of the partially renewable resource staff; the entry $\pi/\kappa_{r\pi}$ in the first column denotes the capacity of the resource $r$ = staff in the period subset $\pi$, $\pi = 1,...,9$, while "▮" denotes that the period belongs to $P(r,\pi)$.

*Figure 3: Graphical Representation of the Instance*




Clearly, a partially renewable resource is a nonrewable one if the period subset covers the whole planning horizon. Moreover, the RCPSP/$\pi$ is a generalization of the RCPSP, where we have $|\Pi(r)| = |T|$, $P(r,1) = \{1\}$, $P(r,2) = \{2\}$ ,..., $P(r,|\Pi(r)|) = \{|T|\}$ for all $r \in R'$ as well as $\kappa_{r\pi} = \kappa_r$ for all $r \in R'$ and $\pi \in \{1,...,|\Pi(r)|\}$, respectively. Clearly, constraints (4') then reduce to the resource constraints (4") of the RCPSP.

$$\sum_{j \in J} k_{jr} \sum_{q \in Q_{jt} \cap E_j} x_{jq} \leq \kappa_r \qquad (r \in R', t \in T) \quad (4'')$$

The purpose of the following is to classify the complexity of the feasibility problem of the RCPSP/$\pi$. In fact, even the feasibility problem of the RCPSP with given deadline $|T|$ is NP-complete. The *feasibility problem* of the RCPSP can be stated as follows:

*Given an instance of the RCPSP, i.e. a network, a set of capacity constraints and a deadline $|T|$ does this problem have a feasible solution?*

**Theorem 1** *The feasibility problem of the RCPSP is NP-complete in the strong sense.*

**Proof** By restriction to 3-PARTITION. □

Note that the NP-completeness of the feasibility problem in turn implies the NP-hardness of the corresponding optimization problems (cf. Garey and Johnson 1979).

We finish this section by presenting the RCPSP/$\pi$ in a "normalized" form which allows to drop one of the parameters introduced so far. More precisely, normalized means that $|\Pi(r)| = 1$ holds for every resource. Starting from the "ordinary" RCPSP/$\pi$ (1)-(3), (4'), (5) we get the normalized RCPSP/$\pi$ as follows: Generate one additional resource $r$ with the availability level $\kappa_{r\pi}$ for each period subset $\pi = 2,\ldots,|\Pi(r)|$ and define the resource requirement $k_{jr}$ for each activity $j \in J$ appropriately. These transformations allow to state the RCPSP/$\pi$ as follows:

min (1)

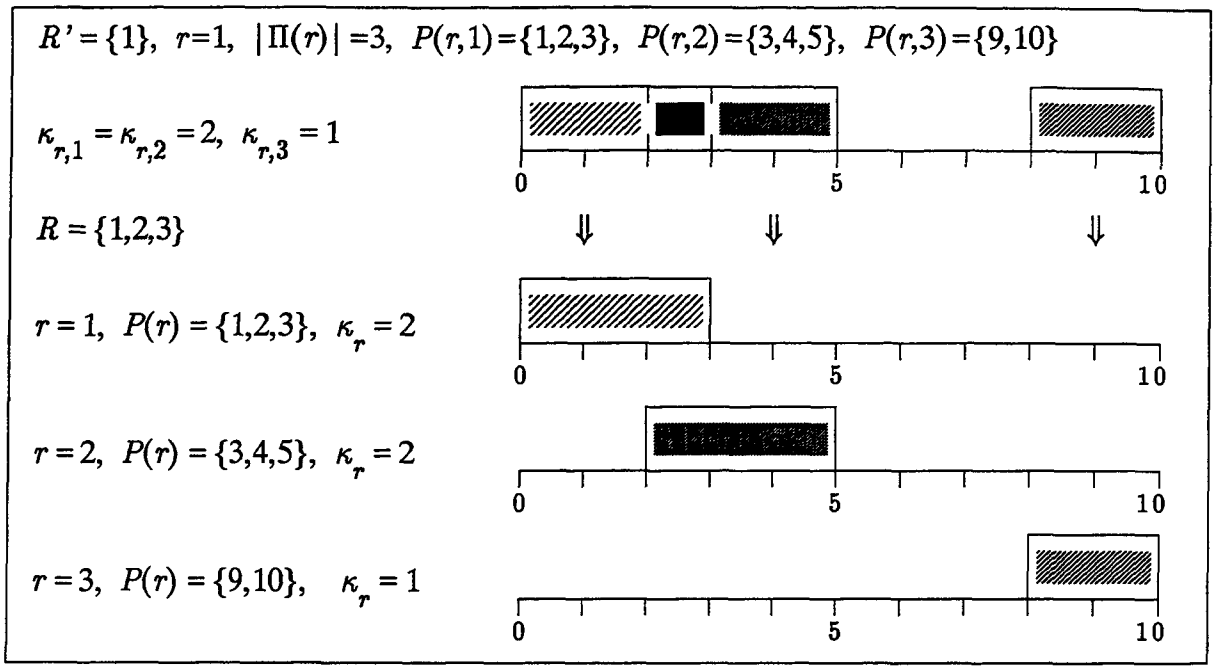
s.t. (2), (3), (5), and

$$\sum_{j \in J} k_{jr} \sum_{t \in P(r)} \sum_{q \in Q_{jt} \cap E_j} x_{jq} \leq \kappa_r \qquad (r \in R) \quad (4)$$

Note we have $|\Pi(r)| = 1$ in the model formulation (1)-(5), therefore $|\Pi(r)|$ is omitted in comparison to (4'). Furthermore, $P(r,\pi)$ reduces to $P(r)$ and $\kappa_{r\pi}$ to $\kappa_r$, respectively. Clearly, the reduction of $|\Pi(r)|$ to 1 is done at the "price" of enlarging the set of resources from $R'$ to $R$. It is easy to prove that the ordinary RCPSP/$\pi$ and the normalized RCPSP/$\pi$ have the same objective function and the same set of constraints. Therefore, both model formulations are equivalent.

The process of normalizing the RCPSP/$\pi$ is illustrated by the use of an example in Figure 4. Starting with $R' = \{1\}$ and $\Pi(r) = 3$ in the ordinary RCPSP/$\pi$ we get $R = \{1,2,3\}$ in the normalized RCPSP/$\pi$. Note that "▨", "▮" and "▨" denote the period subsets in both cases while "▮" denotes period 3 which is covered by $P(r,1)$ and $P(r,2)$, respectively, in the ordinary RCPSP/$\pi$.

*Figure 4: Normalizing the RCPSP/π*

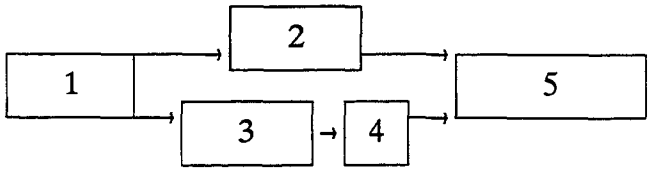



As a consequence in the following we will solely relate to the formulation (1)-(5), because then we must not take care of $\Pi(r)$ and $P(r,\pi)$. Clearly, then it is difficult to interprete the transformed "resources" in terms of an application.

## 3 Modelling Capabilities

In order to further demonstrate the modelling capabilities of the RCPSP/$\pi$, we consider additional examples.

*Figure 5: Network of Example 2*




*Example 2:* Let $T := \{1,2,...,11\}$, $J := \{1,2,...,5\}$. Moreover, consider the network structure given in Figure 5. In addition, let $d_1 := d_2 := d_3 := 2$, $d_4 := 1$, $d_5 := 3$,
$R := \{1,2,3\}, P(1) := \{1,2,3,4\}$, $\kappa_1 := 7$,
$P(2) := \{5,6,7,8\}$, $\kappa_2 := 3$,
$P(3) := \{9,10,11\}$, $\kappa_3 := 2$, and
$k_{1,r} := k_{2,r} := k_{4,r} := k_{5,r} := 1$, $k_{3,r} := 2$ $(r \in R)$; then we compute $E_1 := \{2,...,5\}$, $E_2 := \{4,...,8\}$, $E_3 := \{4,...,7\}$, $E_4 := \{5,...,8\}$, $E_5 := \{8,...,11\}$. Figure 6 depictes the single feasible schedule. The rectangules represent the activities $j$, their length indicates the

duration $d_j$, and the numbers in the rectangles give the per period resource requirements $k_{jr}$ ($j \in J$, $r \in R$), respectively.

*Figure 6: Gantt-Chart for Example 2*

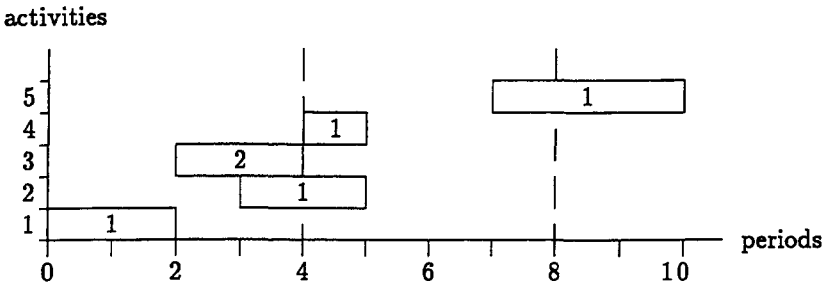


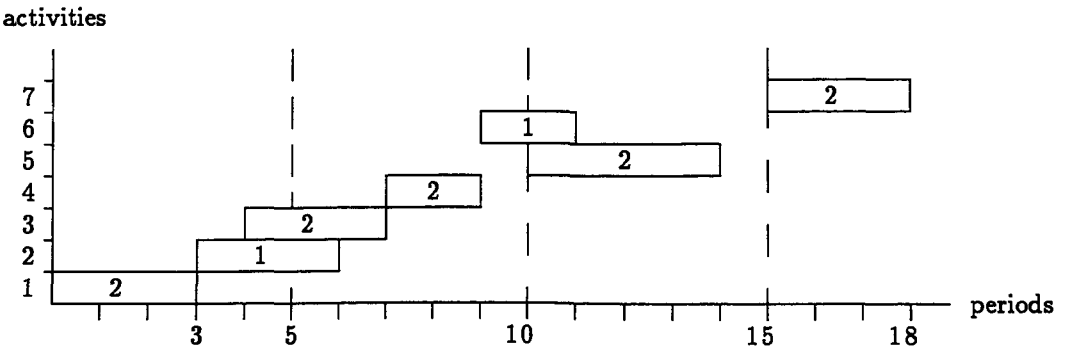

Figure 6 leads to the following observations: Each activity $j$ finishes within its feasible interval $E_j$ as required by constraints (2). If activity $j$ has a positive resource usage $k_{jr}$ ($r \in R$) and if it is processed in period subset $P(r)$, then resource $r$ is in fact consumed by activity $j$. Activities 1 and 3 are totally, activity 2 is partially processed in period subset $P(1)$; alltogether they consume $\kappa_1 = 7$. Activities 2 and 5 are partially, activity 2 is totally processed in period subset $P(2)$; in sum they polish off $\kappa_2 = 3$. Finally, activity 5 is partially processed in period subset $P(3)$ and consumes $\kappa_3 = 2$. Clearly, activity 5 is delayed not because of the precedence constraints, but due to resource constraints, i.e. the limited availability of resource $r = 2$ in period subset $P(2)$.

*Example 3:* Assume we have to plan the daily payment of workers on basis of weekly budgets for the next four weeks. With each week one resource is associated. Four weeks with five days each in total give $4 \times 5 = 20$ periods. Without reproducing all the data of the instance, then the parameter instantiations

$$P(1) = \{1,...,5\}, \quad \kappa_1 = 10,$$
$$P(2) = \{6,...,10\}, \quad \kappa_2 = 10,$$
$$P(3) = \{11,...,15\}, \kappa_3 = 10,$$
$$P(4) = \{16,...,20\}, \kappa_4 = 10,$$

allow to compute the Gantt-chart depicted in Figure 7.

*Figure 7: Daily Payment of Workers, Weekly Budgets*

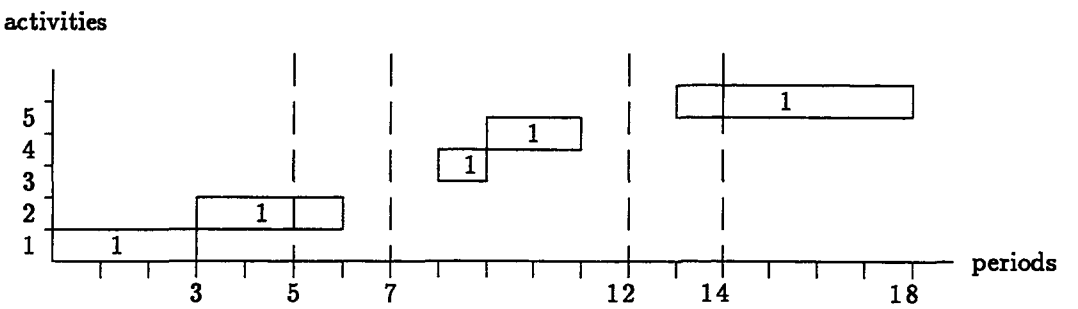


*Example 4:* Assume we have to plan the work weeks for an assembly worker for the next weeks. Moreover assume, that the worker is allowed to work only on eight non-weekend days (two weekend days) within two consecutive weeks. Without reproducing all the data of the instance, then the parameter instantiations

$$
\begin{array}{ll}
P(1) = \{1\} & \kappa_1 = 1 \\
P(2) = \{2\} & \kappa_2 = 1 \\
\vdots & \vdots \\
P(|T|) = \{|T|\} & \kappa_{|T|} = 1 \\
P(|T|+1) = \{6,7,13,14\} & \kappa_{|T|+1} = 2 \\
P(|T|+2) = \{1,\ldots,5,8,\ldots,12\} & \kappa_{|T|+2} = 8 \\
\vdots & \vdots
\end{array}
$$

allow to compute the Gantt-chart depicted in Figure 8. Note that activities 3 and 4 could be scheduled in other periods while activity 5 cannot start before period 13 because of the restriction of at most 2 weekend days.

*Figure 8: Work Weekends for Assembly Worker*



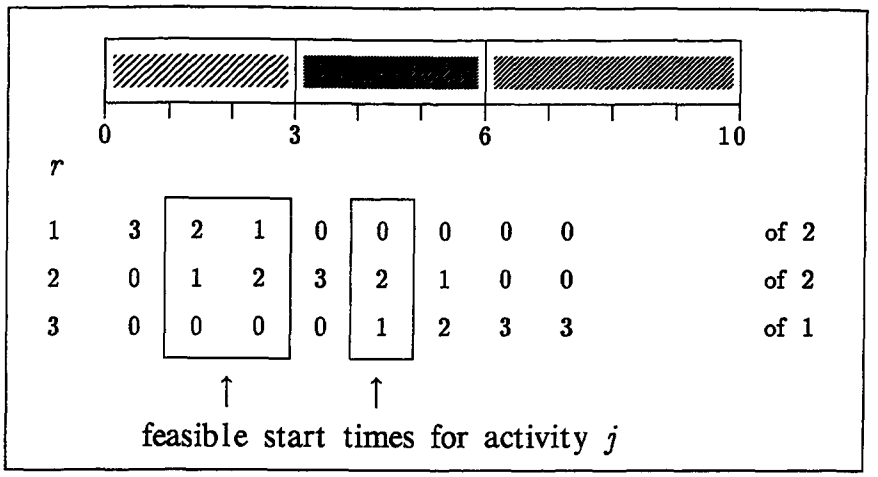

We finish this section by presenting part of an instance which *(i)* illustrates special cases covered by different constellations of period subsets $P(r)$ and which *(ii)* shows the impact of these subsets on the feasible start times of activity $j = 1$. The instance is characterized as follows: $T = \{1,\ldots,10\}$, activity $j$ with $d_j = 3$, $E_j = \{3,\ldots,10\}$ and $k_{jr} = 1$.

*Case 1:* Period subsets $P(r)$ are disjoint and cover the set of periods completely; more specific we have period subsets and resource availabilities as follows:

$P(1) = \{1,2,3\}$, $\kappa_1 = 2$
$P(2) = \{4,5,6\}$, $\kappa_2 = 2$
$P(3) = \{7,8,9,10\}$, $\kappa_3 = 1$

Case 1 is illustrated in Figure 9. The last three rows provide the consumptions of each of the three resources if activity $j$ starts in the corresponding period. Feasible start times are marked with boxes. E.g., if activity $j$ starts in period 2 then it uses 2 units of resource 1 (periods 2 and 3) and 1 unit of resource 2 (period 4).

*Figure 9: Illustration of Case 1*




*Case 2:* Period subsets $P(r)$ are not disjoint and we have periods without resource constraints; more specific we have period subsets and resource availabilities as follows:

$P(1) = \{1,2,3\}, \quad \kappa_1 = 2$
$P(2) = \{3,4,5\}, \quad \kappa_2 = 2$
$P(3) = \{9,10\}, \quad \kappa_1 = 1$

Case 2 is illustrated in Figure 10. Note that none of the resources is consumed if processing of activity $j$ starts in period 6.

*Figure 10: Illustration of Case 2*

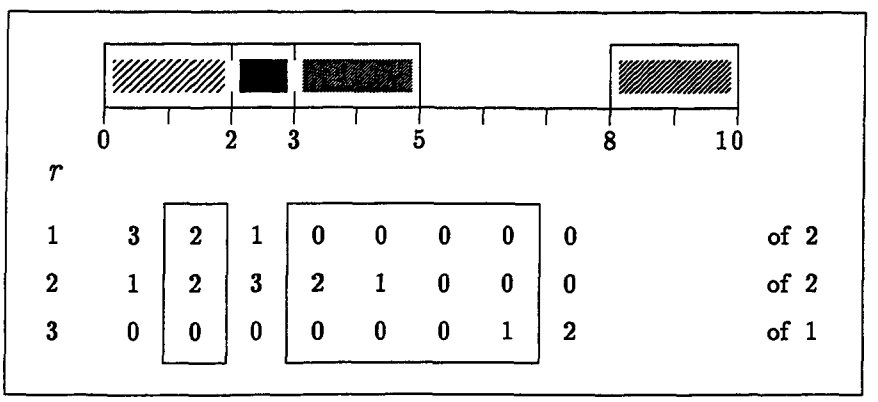
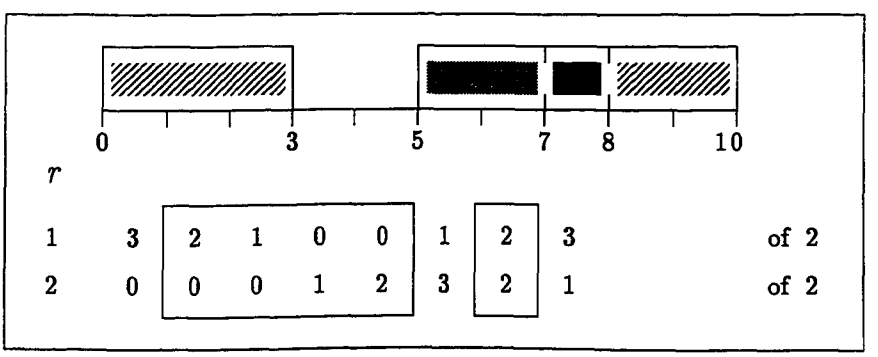



*Figure 11: Illustration of Case 3*



*Case 3:* Some period subsets $P(r)$ contain more than one interval; more specific we have period subsets and resource availabilities as follows (cf. the cases illustrated in Figure 11):

$P(1) = \{1,2,3,8,9,10\}, \quad \kappa_1 = 2$

$P(2) = \{6,7,8\}, \quad \kappa_2 = 2$

## 4 Solution Procedures

In the literature there is no tailored method available for the solution of the RCPSP/$\pi$. For the development of special algorithms we start off with methods known for the RCPSP.

*Optimal procedures* are dynamic programming (cf. e.g. Davis and Heidorn 1971), zero-one programming (cf. e.g. Pritsker et al. 1969, Patterson and Roth 1976), as well as a variety of branch-and-bound-based implicit enumeration methods (cf. Stinson et al. 1978, Talbot and Patterson 1978, Radermacher 1985/86, Christofides et al. 1987, Bell and Park 1990, Carlier and Latapie 1991, Demeulemeester and Herroelen 1992, 1995, Mingozzi et al. 1994, Brucker et al. 1996). Note that currently the branch-and-bound-approach of Demeulemeester and Herroelen 1995 is the most powerful optimal procedure available.

*Heuristic procedures* for the RCPSP basically involve five different solution methodologies: Single- and multi-pass priority rule based scheduling, truncated branch-and-bound-procedures (cf. Alvarez-Valdes and Tamarit 1989), integer programming based heuristics (cf. Oguz and Bala 1994), disjunctive arc concepts (cf. Alvarez-Valdes and Tamarit 1989, Bell and Han 1991), and local search techniques (cf. Sampson and Weiss 1993, Leon and Balakrishnan 1995).

Although belonging to the oldest solution methodology to solve the RCPSP, priority rule based scheduling is still the most important (heuristic) solution technique. This is due to several reasons: The method is intuitive and easy to use which makes it highly suitable to be employed within commercial packages; the method is fast in terms of the computational effort. Finally, multi-pass implementations of the method show the best results obtainable by heuristics for the RCPSP today (cf. Kolisch and Drexl 1996).

### 4.1 Exact Branch-and-Bound-Algorithm

Unfortunately, the powerful branch-and-bound-algorithm of Demeulemeester and Herroelen 1995 is not capable to handle problems with partially renewable resources. The same is true for almost all the exact procedures mentioned above. The only exception is the algorithm of Talbot and Patterson 1978. Therefore we implemented the basic branch-and-bound-procedure of Talbot and Patterson 1978 in order to solve the RCPSP/$\pi$. The network cuts which are not applicable in the presence of partially renewable resources

have been dropped. To speed up convergence of the algorithm we added newly developed bounding rules.

The basic working principle of the algorithm may be described as follows: It is a depth-first-search branch-and-bound-algorithm. The depth of the search tree equals the number of activities. In each stage, one precedence- and resource-feasible activity $j$ (beginning with the unique start activity) is added to the partial schedule at the earliest convenient start time $t_j \in \{ES_j, \ldots, LS_j\}$ where $ES_j$ and $LS_j$ define the critical path based earliest and latest start time, respectively. Backtracking occurs if *(i)* there is no augmentation of the partial schedule under consideration which does not violate resource-constraints or if *(ii)* the partial schedule exceeds the upper bound $\overline{T}$ on the project's makespan. In case of backtracking the current start time of the activity scheduled in the ancestral node of the search tree is augmented by one period. Clearly, we start with $\overline{T} := T$ and update $\overline{T}$ when a complete schedule has been constructed with a makespan less than $\overline{T}$. The algorithm terminates when backtracking to stage 0 occurs.

Clearly, the computational effort which has to be undertaken by this basic scheme is about that of complete enumeration of the set of feasible schedules. Therefore, we want to speed-up convergence by properly designing feasibility bounds, which take care of the partially renewable resources. As the name suggests, feasibility bound does not mean to compute lower bounds for the objective function value, but to calculate lower bounds for the resource consumption of the activities not yet scheduled.

For the description of the bounds we need some definitions. Let denote $W_j := \{ES_j, \ldots, LS_j\}$ the (maximal) time window which takes care of the precedence constraints and the upper bound for the makespan. Let $n \in J$ and $\mathcal{S} := \{(i, j_i, t_i) \mid i = 1,\ldots,n\}$, where $i$ denotes the level of the branch-and-bound tree, $j_i$ is the activity scheduled on level $i$ and $t_i \in W_i$ denotes the start time assignment of activity $j_i$ for $1 \leq i \leq n$. Then $\mathcal{S}$ is called (partial) schedule if $n = |J|$ $(n < |J|)$. Now,

$$SC_{jrt} := k_{jr} \, |Q_{jt} \cap P(r)|$$

denotes (single) activity $j$ consumption of resource $r$ if $j$ starts in period $t$. Then we define

$$TC_r(\mathcal{S}) := \sum_{i=1}^{n} SC_{j_i r t_i}$$

as total consumption of resource $r$ w.r.t. the (partial) schedule $\mathcal{S}$. Accordingly,

$$\kappa_r^0(\mathcal{S}) := \kappa_r - TC_r(\mathcal{S})$$

defines the left-over capacity of resource $r$. Then a feasible (partial) schedule $\mathcal{S}$ is defined as follows:

$$t_h + d_h \leq t_j \qquad (j \in J,\ j \leq n,\ h \in V_j)$$

$\kappa_r^o(S) \geq 0 \qquad (r \in R)$

Clearly, a feasible schedule $S$ is a solution iff $n = |J|$. $S' = \{(i, j_i, t_i) \mid i = 1,...,n\}$ is called an extension of $S = \{(i, j_i, t_i) \mid i = 1,...,h\}$ if $1 \leq h < n$.

Now, for $t \in W_j$

$MC_{jrt} := \min\{ SC_{jr\tau} \mid t \leq \tau \leq LS_j \}$

denotes minimum activity $j$ consumption of resource $r$ if $j$ starts not earlier than period $t$. For $MC_{jrt}$ an important property holds.

**Property 1** *For each activity $j$ and each resource $r$ the sequence*

$$(MC_{jr(ES_j)}, MC_{jr(ES_j+1)}, ..., MC_{jr(LS_j)})$$

*is monotonically increasing.*

**Proof** For $|W_j| = 1$ the proof is trivial. Therefore we consider the case $|W_j| > 1$ and $t \in \{ES_j, ..., LS_j - 1\}$. Then

$MC_{jr(t+1)} = \min\{ SC_{jr\tau} \mid t+1 \leq \tau \leq LS_j \} \geq$

$\min\{ SC_{jrt}, \min\{ SC_{jr\tau} \mid t+1 \leq \tau \leq LS_j \} \} = \min\{ SC_{jr\tau} \mid t \leq \tau \leq LS_j \} = MC_{jrt}$.

As a consequence we have $MC_{jr(t+1)} \geq MC_{jrt}$ for $t \in \{ES_j, ..., LS_j - 1\}$ which completes the proof. □

Property 1 shows that the minimum resource consumption of activity $j$ does not decrease if the time window of activity $j$ is diminished.

Let $U_j$ denote the set of all (direct and indirect) successors of activity $j$ and $LP_{jh}$ the length of the longest path from the start of activity $j$ to the start of activity $h$. Then

$$MCI_{jrt} := \sum_{h \in U_j} MC_{hr(\max\{ES_h,\, t+LP_{jh}\})}$$

defines the minimum consumption of resource $r$ by the successors of activity $j$ induced by its start in period $t$. For $MCI_{jrt}$ an important property holds, too.

**Property 2** *For each activity $j$ and each resource $r$ the sequence*

$$(MCI_{jr(ES_j)}, MCI_{jr(ES_j+1)}, ..., MCI_{jr(LS_j)})$$

*is monotonically increasing.*

**Proof** For $h \in U_j$ and $t \in \{ES_h, ..., LS_h - 1\}$

$\max\{ES_h, t + LP_{jh}\} \leq \max\{ES_h, t + 1 + LP_{jh}\}$ holds.

Because of property 1

$$MC_{hr(\max\{ES_h,\, t+LP_{jh}\})} \leq MC_{hr(\max\{ES_h,\, t+1+LP_{jh}\})}$$

also holds. Therefore, we have

$$MCI_{jrt} = \sum_{h \in U_j} MC_{hr(\max\{ES_h,\, t+LP_{jh}\})} \leq$$

$$\sum_{h \in U_j} MC_{hr(\max\{ES_h,\, t+1+LP_{jh}\})} = MCI_{jr(t+1)}$$

and the proof is complete. □

The computation of $SC_{jrt}$, $MC_{jrt}$ and $MCI_{jrt}$ is illustrated in the Appendix by the use of an example.

Let denote

$$TCI_r(S) := TC_r(S) + MCI_{j_n r t_n}$$

the minimal total induced consumption of resource $r$ w.r.t. the (partial) schedule $S := \{(i, j_i, t_i) \mid i = 1,...,n\}$. Accordingly,

$$\bar{\kappa}_r(S) := \kappa_r - TCI_r(S)$$

defines the maximal left-over capacity of resource $r$. Now we can state

**Theorem 2** *(Feasibility Bound 1)*
*Let $S = \{(i, j_i, t_i) \mid i = 1,...,n\}$ and $r \in R$. If $\bar{\kappa}_r(S) < 0$, then every completion of the partial schedule $S$ violates the resource constraints and hence backtracking occurs.*

**Proof** Obvious. □

*Feasibility Bound 2* dynamically generates conditions where each condition is related to a particular activity. A condition has to be fulfilled before the related activity can be feasible scheduled. If $S$ is a partial schedule that does not schedule activity $j$ and which does not fulfill the condition related to activity $j$ then it is not possible to extend $S$ into a feasible solution.

Although this informal description does not clarify any detail, we refrain, however, from the tedious task of citing technical details, due to several reasons: *(i)* An in-depth description is very lengthy and would neccessitate to introduce a couple of additional definitions, symbols, corollaries and theorems. *(ii)* In comparison with the basic version the feasibility bound 1 already provides a considerable speed boast (cp. Section 5.3). *(iii)* The technical details can be found in Böttcher 1995 on our ftp-site. *(iv)* Last but not least, also in the presence of the feasibility bound 2 even medium-sized instances remain untractable, and therefore our primary concern now is on the description and evaluation of fast and reliable heuristics.

## 4.2 Greedy Randomized Adaptive Search Procedures

The only line of attack for tackling practical problem sizes comprising hundreds of activities is provided by approximation methods. Deterministic greedy priority rule-based methods have been widely adopted in scheduling (cp. the surveys in Haupt 1989, Panwalkar and Iskander 1977). Partial schedules are extended, starting with the empty set of scheduled activities, i.e. the intialization $x_{jt} := 0$ for all the decision variables. These methods are commonly used when scheduling large problem instances and they yield only one solution for an instance, even if applied several times. Having in mind that this solution may be arbitrarily bad or even infeasible, determinism seems to be a major deficiency of such methods. Semi-greedy (cp. Hart and Shogan 1987), greedy randomized (cp. Laguna et al. 1994), or regret-based biased random sampling methods (cp. Drexl 1991, Kolisch 1995, Kolisch and Drexl 1996) try to overcome the shortcoming of determinism by performing the selection process randomly, but according to probabilities which are proportional to priority values. In this way, in each step every schedulable activity may be chosen, though those with higher probabilities will have a greater chance of being selected. Due to their nondeterminism, repeated application of randomized methods will produce a set of solutions rather than one sole solution. Usually some of these solutions will be better than the one found with the deterministic version of the same method. Moreover, no tiebreaker needs to be specified for randomized methods, since ties cannot occur.

Generally, common priority rule-based methods for (project) scheduling are distinguished to be serial or parallel (cp. the early work of Kelley 1963 and the recent improvements obtained by Kolisch 1996a). While the former schedule one of the precedence-feasible activities as early as possible w.r.t. resource constraints, the latter proceed chronologically over all periods of the planning horizon trying to schedule in each period as many activities as possible. Though the basic principle of both methods is simple and intuitive, some specific details have to be designed appropriately in order to get reliable and fast methods for the problem class under consideration.

We modified the *serial scheduling scheme* for solving the RCPSP/$\pi$. It consists of at most $|J|$ stages. In each stage one activity is selected and assigned a start time, i.e. added to the current partial schedule $\mathcal{S} := \{(i, j_i, t_i) \mid i = 1,...,n\}$. Then

$$ES_j(\mathcal{S}) := \max\{t_h + d_h \mid h \in \mathcal{V}_j\}$$

denotes the earliest start time of activity $j > n$, adapted w.r.t. $\mathcal{S}$. Accordingly,

$$W_j(\mathcal{S}) := \{ES_j(\mathcal{S}), ES_j(\mathcal{S}) + 1, ..., LS_j\}$$

denotes the (adapted) time window of activity $j$.

Two disjoint sets $C$ and $D$ are computed in each stage. In the complete set $C$ are the activities which have already been scheduled and thus belong to the partial schedule $S$. Recall $SC_{jrt}$, $\kappa_r^0(S)$ and $MCI_{jrt}$ as defined in Subsection 4.1.
Then

$$D := \{(j,t) \mid (j \notin C) \wedge (\forall h \in V_j : h \in C) \wedge (t \in W_j(S)) \wedge (\forall r \in R : \kappa_r^0(S) \geq SC_{jrt} + MCI_{jrt})\}$$

defines the decision set, i.e. the activity-period combinations $(j,t)$ which are feasible w.r.t. the current partial schedule $S$.

In each stage one activity-period combination $(j,t)$ from the decision set is selected with a priority rule $\omega_{jt}$. Afterwards, the selected activity $j$ is removed from the decision set (together with all activity-period combinations $(j,t)$ of activity $j$ and those activity-period combinations $(i,\tau)$ which now are infeasible) and put into the complete set. This, in turn, may place a number of activity-period combinations into the decision set, since all their predecessors are now completed. The algorithm terminates either if $D = \phi$ or if no feasible activity-period combination $(j,t)$ exists for activity $j$ with $j \notin C$ and $V_j \subseteq C$. If $C = J$ then $|J|$ stages have been performed and a feasible solution has been computed. A formal description of the serial scheduling scheme is given in Table 1. Note that as tie breaker in the $(j^*, t^*)$ selection step the activity and/or period number is chosen.

*Table 1: Serial Scheduling Scheme*

*Initialization*

$C := \phi,\ S := \phi,\ \kappa_r^0(S) := \kappa_r\ (r \in R)$

*Execution*

compute $D$
**while** $D \neq \phi$ **do**
{
$(j^*, t^*) \in \{(j,t) \mid \omega_{jt} = \inf\{\omega_{ir} \mid (i,\tau) \in D\}\}$
update $C$, $S$, $\kappa_r^0(S)$, and $D$
}
If $C = J$ **then** feasible solution found

The way the serial method has been described so far is termed as single-pass approach, i.e. one single pass and one priority rule are employed to derive one (feasible) solution. Contrary, multi-pass procedures perform Z single passes in order to generate a sample of at most Z unique (feasible) solutions, where the best one is chosen. Basically, two different kinds of multi-pass methods can be distinguished: The multi-priority rule approach (cf. Boctor 1990, Li and Willis 1992) employs one scheduling scheme and

different priority rules while sampling (cf. Wiest 1964, Cooper 1976, and Alvarez-Valdes and Tamarit 1989) makes use of one scheduling scheme and one priority rule. Different schedules are obtained by biasing the selection of the priority rule through a random device. The use of a random device can be interpreted as a mapping

$$\psi : (j,t) \in D \rightarrow [0,1] \tag{6}$$

which assigns to each activity-period combination in the decision set $D$ a probability $\psi_{jt}$ of being selected. Three different methods can be distinguished: *(i) Random sampling* assigns each activity in the decision set the same probability. *(ii) Biased random sampling* biases the probabilities dependent on the priority values of the activities to favour those activities which seem to be a more sensible choice. *(iii)* A special case of biased random sampling is the utilization of regret measures for determining the selection probabilities. It has been introduced by Drexl 1991 and Drexl and Grünewald 1993 and is referred to as *regret based biased random sampling*. The three different randomization schemes have been compared in Kolisch and Drexl 1996 when solving the RCPSP and it has been shown that the last one clearly is superior in terms of the solution quality.

Let a priority rule be defined by the mapping

$$\omega : (j,t) \in D \rightarrow \mathbb{R}_{\geq 0} \tag{7}$$

which assigns to each activity-period combination $(j,t)$ in the decision set $D$ a priority value $\omega_{jt}$ and an objective $O$ stating whether the activity-period combination of the decision set with the minimal ($O = \min$) or maximal ($O = \max$) priority value is selected. Then, the regret $\rho_{jt}$ compares the priority value of activity-period combination $(j,t)$ with the worst consequence in the decision set as follows:

$$\rho_{jt} := \begin{cases} \max\{\omega_{i\tau} \mid (i,\tau) \in D\} - \omega_{jt}, & \text{if } O = \min \\ \omega_{jt} - \min\{\omega_{i\tau} \mid (i,\tau) \in D\}, & \text{if } O = \max \end{cases} \qquad (j,t) \in D \tag{8}$$

Therewith, the parameterized probability mapping arises to:

$$\psi_{jt} := (\rho_{jt}+1)^{\alpha} \Big/ \sum_{(i,\tau)\in D} (\rho_{i\tau}+1)^{\alpha} \qquad (j,t) \in D \tag{9}$$

Adding the constant "1" to the regret value $\rho_{jt}$ assures that the selection probability $\psi_{jt}$ for each activity-period combination $(j,t)$ in the decision set $D$ will be greater zero and thus every schedule $S := \{(i, j_i, t_i) \mid i = 1,\ldots,|J|\}$ may be generated. By choice of the parameter $\alpha$, the amount of bias can be controlled. Associated with an arbitrarily large $\alpha$ will be no bias and thus deterministic selection on the basis of the employed priority rule (with random selection as a tie breaker), while an $\alpha$ of 0 will give way for random activity selection.

We have implemented the serial scheduling scheme as a multi-pass version. Thus, in Table 1 the activity-period combination $(j^*, t^*)$ is chosen from $D$ with selection probabilities $\rho_{jt}$ computed according to a priority rule $\omega$. Following the lines of Kolisch and Drexl 1996 the control parameter $\alpha$ has been set for pass number $z = 1,...,Z$ as follows:

| | | | | |
|---|---|---|---|---|
| Pass | $z = 1$: | | | $\alpha = \infty$ |
| Passes | $z = 2$ | to | $z = 5$: | $\alpha = 3$ |
| Passes | $z = 6$ | to | $z = 10$: | $\alpha = 2$ |
| Passes | $z = 11$ | to | $z = Z$: | $\alpha = 1$ |

Clearly, when $Z$ passes are performed the upper bound $\overline{T}$ is updated whenever an improved makespan has been found.

We employed several priority rules (a survey of advanced priority rules for solving the RCPSP is given in Kolisch 1996b). Some of them are well-known from literature others are newly developed ones.

### MINEFT, MINLFT, MINSLK, MTSUCC

First, we implemented the well-known priority rules MINEFT (minimum earliest finishing time), MINLFT (minimum latest finishing time), MINSLK (minimum slack), and MTSUCC (most total successors), respectivly.

### MAXSRU, MINSRU

Second, we implemented two static priority rules. Let denote

$$SRU_j := \sum_{r \in R} k_{jr} \qquad (j \in J)$$

the maximum static resource usage of activity $j$. In the case of $O = \max$ $\omega_{jt} := SRU_j$ defines the maximum static resource usage priority rule for $(j,t) \in D$, denoted as MAXSRU. Accordingly, the minimum static resource usage priority rule, denoted as MINSRU, is defined in the case of $O = \min$.

Unfortunately, the rules MAXSRU and MINSRU have the disadvantage, that $k_{jr}$ does not take care of the resource usage which depends on the starting times of jobs $j$ of the partial schedule (cf. cases 1 to 3 in Section 3 also).

### MAXTRU, MINTRU, MAXRRU, MINRRU

Now four new but still static priority rules will be defined. Recall $SC_{jrt}$ and $MCI_{jrt}$ as defined in Subsection 4.1. Let denote

$$TRU_{jt} := \sum_{r \in R} (SC_{jrt} + MCI_{jrt}) \qquad (j,t) \in D$$

the start time dependent lower bound of the total resource usage. In the case of $O = \max$ $\omega_{jt} := TRU_{jt}$ defines the maximum start time dependent total resource usage priority

rule, denoted as MAXTRU. Accordingly, the minimum start time dependent total resource usage priority rule, denoted as MINTRU, is defined in the case of $O$ = min.

The rules MAXTRU and MINTRU have the drawback that resource usages are not related to the available resource capacity.

Let denote

$$RRU_{jt} := \sum_{\substack{r \in R \\ \kappa_r > 0}} (SC_{jrt} + MCI_{jrt}) / \kappa_r \qquad (j,t) \in D$$

the relative resource usage. In the case of $O$ = max $\omega_{jt} := RRU_{jt}$ defines the maximum relative resource usage priority rule, denoted as MAXRRU. Accordingly, the minimum relative resource usage priority rule, denoted as MINRRU, is defined in the case of $O$ = min.

### MAXDRRU, MINDRRU, MAXTRC, MINTRC

Note for a given partial schedule $S$ the overall available resource capacity $\kappa_r$ does not properly reflect the capacity which is still available. Recall $\kappa_r^0(S)$ to be the left-over capacity of resource $r$ which regards the current partial schedule $S$. Accordingly,

$$DRRU_{jt} := \sum_{\substack{r \in R \\ \kappa_r^0(S) > 0}} (SC_{jrt} + MCI_{jrt}) / \kappa_r^0(S) \qquad (j,t) \in D$$

defines the dynamic relative resource usage. In the case of $O$ = max $\omega_{jt} := DRRU_{jt}$ defines the maximum relative resource usage priority rule, denoted as MAXDRRU. Accordingly, the minimum relative resource usage priority rule, denoted as MINDRRU, is defined in the case of $O$ = min.

Finally, let denote

$$TRC_{jt} := \sum_{r \in R} (\kappa_r^0(S) - SC_{jrt} - MCI_{jrt}) \qquad (j,t) \in D$$

an upper bound of the total remaining capacity. In the case of $O$ = max $\omega_{jt} := TRC_{jt}$ defines the maximum total remaining capacity priority rule, denoted as MAXTRC. Accordingly, the minimum total remaining capacity priority rule, denoted as MINTRC, is defined in the case of $O$ = min.

## 5 Experimental Evaluation

First, we will describe an advanced generator for the parametric characterization of instances. Second, we want to elaborate the problem parameters which make an instance "hard" or "easy". Third, the average speed-up obtained by the feasibility bounds shall be described. Fourth, the serial scheduling method will be analyzed.

All algorithms have been coded in C and implemented on an IBM RS6000 41T workstation under the AIX 3.2.5 operating system.

## 5.1 Generation of Instances

Generally, two possible approaches can be found adopted in literature when having to come up with test instances. First, practical cases. Their strength is their high practical relevance while the obvious drawback is the absence of any systematical structure to infer any general properties. Thus, even if an algorithm performs well on some practical instances, it is not guaranteed that it will continue to do so on other instances. Second, artificial instances. Since they are generated randomly according to predefined specifications, their plus lies in the fact that fitting them to certain requirements such as given probability distributions poses no problem. However, they may reflect situations with little or no resemblance to any problem setting of practical interest. Hence, an algorithm performing well on several such artificial instances may or may not perform satisfactorily in practice.

In this research, we decided to start with the ProGen-code which has (among others) been designed for generating instances of the RCPSP variety. More specific, the problem parameters *NC*, *RF* and *DF* are the same as in ProGen (for details cf. Kolisch et al. 1995), *RS* has been modified in contrast to ProGen while the parameters *CF* and *PF* are entirely new.

- The *network complexity* $NC \geq 0$ defines the ratio of non-redundant precedence relations to the number of activities.
- The *resource factor* $RF \in [0,1]$ reflects the density of the matrix $(k_{jr})$. For $RF = 0$ no activity has a resource usage, while for $RF = 1$ each activity (despite the dummy source and dummy sink) uses every resource.
- The *duedate factor* $DF \in [0,1]$ determines the horizon $T$ as follows: $T := \text{ROUND}[ES_{|J|}(1 - DF) + \text{DF} \cdot \Sigma_{j \in J} d_j]$. Note that $DF = 0$ forces the project to be finished as early as possible (i.e. to start activity $|J|$ in period $ES_{|J|}$), while for $DF = 1$ the horizon is long enough to schedule "one activity after the other". Unfortunately, for the RCPSP/$\pi$ this does not ensure the existence of a feasible solution.
- The *resource strength* $RS \in [0,1]$ measures the degree of resource-constrainedness. Let denote $K_r^{\min} := \Sigma_{j \in J} \min\{SC_{jrt} \mid t \in W_j\}$, $K_r^{\max} := \Sigma_{j \in J} \max\{SC_{jrt} \mid t \in W_j\}$, then $\kappa_r := \text{ROUND}[K_r^{\min}(1\text{-RS}) + K_r^{\max} \cdot \text{RS}]$ defines the available capacity of resource $r$. Clearly, only for the RCPSP $RS = 0$ defines the lowest resource feasible level, while for $RS = 1$ no resource might become scarce in any schedule. Unfortunately, for the RCPSP/$\pi$ there is no guarantee for resource feasibility.
- The subsets *cardinality factor* $CF \in [0,1]$ determines the cardinality $M$ of the period subsets as follows: $M := \text{ROUND}(1 - CF + T \cdot CF)$. Clearly, $CF = 0$ implies that all period subsets have cardinality one (i.e. only renewable resources are generated),

while for $CF = 1$ they cover the whole planning horizon (i.e. only nonrenewable resources are generated).

- The subset of period *partition factor* $PF \in [0, 1]$ determines the number of intervals $I$ (i.e. subsequent periods) of the period subsets as follows: $I := \text{ROUND}\ (1 - PF + \min\{\mu, T - \mu + 1\} \cdot PF)$. Note that $PF = 0$ implies that $I$ equals one, while for $PF = 1$ we get the maximum possible $I$ (given $T$ and $\mu$).

Table 2 gives a summary of the variable problem parameter levels. In order to generate the test instances we used a full factorial design where we fixed the number of non-dummy activities to 10, i.e. $|J| = 12$, and the number of normalized partially renewable resources to 30, i.e. $|R| = 30$. The duration of non-dummy activities and the level of a positive resource requirement were randomly drawn from the uniform distribution [1, 10]. All other problem parameters were set as documented in Kolisch et al. 1995.

We generated 10 instances for each combination of *NC*, *RF*, *RS*, *DF*, *CF*, and *PF* which gave a total of $2 \times 3 \times 3 \times 2 \times 3 \times 2 = 216$ benchmark instances. All instances used in the computational study are available from the authors upon request.

*Table 2: Levels of Variable Problem Parameters*

| Parameter | Levels |
|---|---|
| *NC* | {1.5, 2.0} |
| *RF* | {0.1, 0.5, 0.9} |
| *RS* | {0.25, 0.5, 0.75} |
| *DF* | {0.3, 0.6} |
| *CF* | {0.2, 0.5, 0.8} |
| *PF* | {0.0, 0.5} |

## 5.2 What Makes Instances "Hard" or "Easy"

While clearly all problem instances covered by the RCPSP/$\pi$ belong to the class of NP-hard ones (cf. Theorem 1), the computational tractability of a specific instance depends on the problem parameters introduced above. In this subsection we report the computational results which have been achieved with the basic version (BV) of the exact algorithm without any bounding rules, because then the impact of the problem parameters becomes more obvious.

Solving all 2,160 instances with BV we needed an average CPU-time of 4.264 seconds where 27,963 leaves were generated on the average. For the present we will not distinguish between feasible and infeasible instances. Later we will come back to this aspect in detail.

Table 3 shows the impact of each parameter on the tractability. We report both the number of generated leaves (# of leaves) and the CPU-time required to solve an instance to optimality or to show that no feasible solution exists. PAR denotes the parameter under consideration, VAL the value of the parameter, AVE the average over all instances, and STD the standard deviation, respectively. SIG denotes the ANOVA-based error of rejecting the hypothesis "PAR does not have a significant impact on the computational effort of the algorithm". SIG < 0.05 indicates that PAR influences AVE.

The following observations can be made: *PF* and *MC* have no significant effect on the tractability. Contrary to the RCPSP (cf. Kolisch et al. 1995) increasing *NC* seems to decrease the tractability. All other parameters do have a high impact on the tractability.

*Table 3: Impact of Parameters on Tractability*

| PAR | VAL | # of leaves AVE | # of leaves STD | # of leaves SIG | CPU-time in sec AVE | CPU-time in sec STD | CPU-time in sec SIG |
|---|---|---|---|---|---|---|---|
| *NC* | 1.5 | 22,265 | 153,651 | 0.161 | 3.462 | 20.332 | 0.164 |
| | 2.0 | 33,661 | 218,166 | | 5.067 | 31.974 | |
| *RF* | 0.1 | 3,858 | 26,125 | 0.0 | 0.818 | 4.268 | 0.0 |
| | 0.5 | 16,879 | 147,076 | | 2.520 | 17.328 | |
| | 0.9 | 63,152 | 287,569 | | 9.455 | 42.380 | |
| *RS* | 0.25 | 3,063 | 24,959 | 0.0 | 0.611 | 3.157 | 0.0 |
| | 0.5 | 33,818 | 167,662 | | 5.448 | 27.428 | |
| | 0.75 | 47,008 | 277,853 | | 6.735 | 37.059 | |
| *DF* | 0.3 | 2,724 | 12,918 | 0.0 | 0.629 | 2.110 | 0.0 |
| | 0.6 | 53,203 | 264,250 | | 7.900 | 37.498 | |
| *CF* | 0.2 | 2,461 | 20,612 | 0.0 | 0.621 | 3.090 | 0.0 |
| | 0.5 | 22,879 | 155,176 | | 3.894 | 25.756 | |
| | 0.8 | 58,549 | 284,317 | | 8.279 | 38.131 | |
| *PF* | 0.0 | 27,594 | 182,555 | 0.928 | 4.115 | 25.346 | 0.796 |
| | 0.5 | 28,332 | 194,791 | | 4.413 | 28.187 | |

*Table 4: Level of Significance for Interactions of Two Parameters*

| | *RF* | *RS* | *DF* | *CF* | *PF* |
|---|---|---|---|---|---|
| *NC* | 0.117 | 0.546 | 0.209 | 0.488 | 0.526 |
| *RF* | | 0.000 | 0.000 | 0.000 | 0.205 |
| *RS* | | | 0.000 | 0.000 | 0.517 |
| *DF* | | | | 0.000 | 0.844 |
| *CF* | | | | | 0.876 |

Table 4 shows the mutual interactions of two parameters as an outcome of multiple variance analysis in terms of SIG, the level of significance. More precisely, SIG denotes the multiple ANOVA-based error of rejecting the hypothesis "Both parameters do not effect each other with respect to the tractability". It can be seen that the three parameters *RF*, *RS* and *DF* show a significant interaction while the interactions of all other parameter pairs are not significant.

### 5.3 Evaluation of the Feasibility Bounds

In this subsection we report the speed-up of the basic version (BV) of the algorithm obtained by the use of the feasibility bounds described in Section 4.1. In the sequel FB 1, FB 2 and FB 1&2 denotes the version which utilizes the feasibility bound 1, the feasibility bound 2 and the feasibility bounds 1 and 2, respectively.

Tables 5 to 7 provides average comparative results. FAC denotes the speed-up factor. Table 5 presents the results for all the instances, i.e. based on the set of 2,160 problems introduced in Subsection 5.1. Table 6 reports averages over the subset of the 40 instances where the speed-up was highest. Table 7 shows averages over the subset of the 40 instances which turned out to be "very hard" for BV in terms of the number of leaves generated in the search tree. Clearly, FB 1&2 is more efficient than FB 1 with respect to the number of backtracking steps needed in order to solve an instance. Unfortunately, the effort to check the assumptions of the more elaborate method FB 1&2 is very high. Therefore, FB 1&2 does not outperform FB 1 that much in terms of the CPU-time required.

*Table 5: Averages over all Instances*

| | # of leaves | | | CPU-time in sec | | |
|---|---|---|---|---|---|---|
| | AVE | STD | FAC | AVE | STD | FAC |
| BV | 27,963 | 188,729 | 1.0 | 4.3 | 26.8 | 1.0 |
| FB 1 | 4,441 | 50,873 | 6.3 | 1.0 | 8.1 | 4.3 |
| FB 2 | 3,470 | 35,144 | 8.1 | 1.9 | 16.1 | 2.3 |
| FB 1&2 | 580 | 6,696 | 48.1 | 0.7 | 5.5 | 6.0 |

As expected, the tractability of instances by the different versions depends on the problem parameters as well; details can be found in Böttcher 1995.

*Table 6: Averages over 40 Instances with Highest Speed-up*

| | # of leaves | | | CPU-time in sec | | |
|---|---|---|---|---|---|---|
| | AVE | STD | FAC | AVE | STD | FAC |
| BV | 157,489 | 235,251 | 1.0 | 22.3 | 35.8 | 1.0 |
| FB 1 | 5,840 | 18,293 | 27.0 | 1.2 | 3.2 | 18.3 |
| FB 2 | 4,743 | 10,372 | 33.2 | 2.5 | 4.9 | 8.9 |
| FB 1&2 | 124 | 336 | 1267.0 | 0.5 | 0.3 | 47.1 |

*Table 7: Averages over 40 Very Hard Instances*

| | # of leaves | | | CPU-time in sec | | |
|---|---|---|---|---|---|---|
| | AVE | STD | FAC | AVE | STD | FAC |
| BV | 556,981 | 863,217 | 1.0 | 77.9 | 119.8 | 1.0 |
| FB 1 | 123,825 | 314,838 | 4.5 | 20.1 | 49.6 | 3.9 |
| FB 2 | 66,360 | 151,350 | 8.4 | 31.2 | 68.3 | 2.5 |
| FB 1&2 | 15,243 | 39,745 | 36.5 | 12.9 | 33.3 | 6.1 |

So far, we did not differentiate between instances with and without feasible solutions. In the following 'feasible instances' will be of primary concern, due to several reasons: *(i)* Practical examples, though not available to the authors so far, are supposed to be feasible. *(ii)* To solve 'infeasible instances' is on the average more time consuming than solving 'feasible instances'. *(iii)* Last but not least, it does not make any sense trying to solve 'infeasible instances' by the use of greedy heuristics.

The instance generator has not been designed in order to provide 'feasible instances'. Therefore we counted the 'feasible instances' within each of the 216 parameter cells collected in Table 2. Then we excluded the cells for which less than six of the instances did not have a feasible solution. This way we restricted the generator ex post to provide 'feasible instances' with a high probability. Table 8 gives a summary of the restricted variable problem parameter levels. Note that we have $NC = 2.0$ and $DF = 0.6$ in all the 25 cells identified in Table 8.

*Table 8: Restricted Levels of Variable Problem Parameters*

| Cell | *PF* | *CF* | *RS* | *RF* |
|---|---|---|---|---|
| 1 | .0 | .2 | .5 | .1 |
| 2 | .0 | .2 | .5 | .5 |
| 3 | .0 | .2 | .5 | .9 |
| 4 | .0 | .2 | .75 | .1 |
| 5 | .0 | .2 | .75 | .5 |
| 6 | .0 | .2 | .75 | .9 |
| 7 | .0 | .5 | .75 | .9 |
| 8 | .0 | .8 | .25 | .1 |
| 9 | .0 | .8 | .25 | .9 |
| 10 | .0 | .8 | .5 | .1 |
| 11 | .0 | .8 | .5 | .5 |
| 12 | .0 | .8 | .5 | .9 |
| 13 | .0 | .8 | .75 | .1 |
| 14 | .0 | .8 | .75 | .5 |
| 15 | .0 | .8 | .75 | .9 |
| 16 | .5 | .2 | .5 | .1 |
| 17 | .5 | .2 | .5 | .5 |
| 18 | .5 | .2 | .5 | .9 |
| 19 | .5 | .2 | .75 | .1 |
| 20 | .5 | .2 | .75 | .5 |
| 21 | .5 | .2 | .75 | .9 |
| 22 | .5 | .5 | .75 | .1 |
| 23 | .5 | .5 | .75 | .5 |
| 24 | .5 | .5 | .75 | .9 |
| 25 | .5 | .8 | .75 | .9 |

We generated four sets of instances with 15, 20, 30 and 60 non-dummy activities, respectively, where the instances of each set were generated according to the 25 parameter cells of Table 8. The number of normalized resources was set to 30. For each of the 4 problem sizes and each of the 25 cells 10 instances were generated. A summary of the results obtained with the algorithm FB 1&2 can be given as follows:

- Within a time limit of 5 CPU minutes all but 10 of the 250 instances having 15 activities could be solved to optimality. For the 250 instances with 20 activities this number is 21.
- For the 250 instances with 30 activities we were unable to find a feasible solution within a time limit of 5 CPU minutes for 19 instances. Additional 14 instances could not be solved to optimality within 5 minutes. Thus in total for 33 instances the algorithm terminated prematurely within this problem class.
- Concerning the 250 instances with 60 activities for 21 instances a time-out occurred after 5 CPU minutes. For 10 of these 21 instances no feasible solution could be found.

- Within the 30 and 60 non-dummy activity classes of problem sizes a time-out occurs only for instances belonging to the parameter cells 8, 9, 12, and 25, respectively. The instances for which no feasible solution could be found before reaching the time limit are primarily located in cell 25.

### 5.4 Evaluation of the Priority Rule Based Serial Scheduling Method

Recall that $\omega$ denotes a priority rule while $Z$ denotes the sample size. First we conducted some experiments in order to evaluate the priority rules presented in Subsection 4.2. In the first experiment we employed the instance set with 20 non-dummy activities used above.

*Table 9: Comparison of the Priority Rules* ($|J| = 22$)

| priority rule $\omega$ | UNSOLVED $Z$=10 | 100 | 1,000 |
|---|---|---|---|
| MINEFT | 51 | 50 | 41 |
| MINLFT | 40 | 36 | 33 |
| MINSLK | 49 | 47 | 42 |
| MTSUCC | 45 | 40 | 39 |
| MAXSRU | 56 | 54 | 54 |
| MINSRU | 62 | 55 | 47 |
| MAXTRU | 105 | 70 | 56 |
| MINTRU | 37 | 36 | 35 |
| MAXRRU | 104 | 66 | 54 |
| MINRRU | 38 | 37 | 36 |
| MAXDRRU | 104 | 67 | 52 |
| MINDRRU | 34 | 33 | 32 |
| MAXTRC | 37 | 37 | 35 |
| MINTRC | 100 | 68 | 54 |

Table 9 provides the results. In columns 2 to 4 we find the number of instances for which no feasible solution has been found (denoted as UNSOLVED) after $Z = 10$, 100, and 1,000 passes, respectively.

The results of Table 9 indicate that the rules MAXSRU, MINSRU, MAXTRU, MAXRRU, MAXDRRU and MINTRC are inferior w.r.t. the ability to derive feasible solutions. Therefore we decided to further evaluate the eight rules MINEFT, MINLFT, MINSLK, MTSUCC, MAXTRC, MINTRU, MINRRU and MINDRRU for larger instance sets comprising 30 and 60 non-dummy activities, respectively.

In order to evaluate the solution quality obtained with the priority rules we only considered for the 250 instances with 30 activities those 217 (14) instances for which the exact algorithm was able to prove the optimal solution (identify at least one feasible solution) within the imposed time limit. Table 10 provides the average percentage deviations of the objective function value computed with the serial algorithm (within $Z$ passes) from the objective function value computed by the exact algorithm.

*Table 10: Evaluation of Priority Rules for Larger Instances* ($|J| = 32$)

| priority rule $\omega$ | 217 instances $Z$=10 | 100 | 1,000 | 14 instances $Z$=10 | 100 | 1,000 | 250 instances CPU ($Z$=1,000) |
|---|---|---|---|---|---|---|---|
| MINEFT | 12.23 | 9.17 | 6.38 | 134.74 | 133.10 | 100.40 | 10.6 |
| MINLFT | 1.63 | 0.81 | 0.81 | 7.87 | 7.87 | 6.50 | 8.8 |
| MINSLK | 8.70 | 7.53 | 6.50 | 75.13 | 74.83 | 40.25 | 11.3 |
| MTSUCC | 4.71 | 3.31 | 2.77 | 55.52 | 40.06 | 39.86 | 11.0 |
| MAXTRC | 49.48 | 9.41 | 5.26 | 4.93 | 3.84 | 1.63 | 27.2 |
| MINTRU | 49.48 | 10.01 | 5.60 | 4.71 | 3.95 | 1.63 | 19.6 |
| MINRRU | 50.15 | 10.38 | 5.93 | 5.04 | 3.73 | 1.73 | 22.3 |
| MINDRRU | 50.15 | 10.38 | 6.04 | 5.37 | 3.73 | 1.52 | 32.5 |

The results can be interpreted as follows: The "classical" rules MINEFT, MINLFT, MINSLK, and MTSUCC perform better on the subset of the "easy" 217 instances while the new rules MINTRU, MINRRU, MINDRRU, and MAXTRC do a better job on the "hard" 14 ones, respectively. This is especially evident for a small number of passes $Z$. In summary the rule MINLFT seems to be the most promising candidate.

In order to evaluate the run time performance of the different priority rules, the average CPU times for solving all the 250 instances under investigation in sec for 1,000 passes are reproduced in Table 10, too.

Table 11 provides the results for the 229 (11) instances with 60 non-dummy activities for which the exact algorithm was able to prove the optimal solution (identify at least one feasible solution). We restricted the umber of passes $Z$ to at most 100 because of the increasing computational effort. Average CPU times for solving all the 250 instances under concern in sec for $Z$=100 passes are reproduced also.

The results already obtained on the 30 activity set are maintained, i.e. the rule MINLFT performs best over the "easy" 229 instances while the rules MINEFT and MINSLK are inferior for the "hard" 11 instances. In summary the rule MINLFT gives reasonable good results for both "easy" and "hard" instances.

*Table 11: Evaluation of Priority Rules for Larger Instances* ($|J| = 62$)

| priority rule $\omega$ | 229 instances $Z$=10 | 100 | 11 instances $Z$=10 | 100 | 250 instances CPU ($Z$=100) |
|---|---|---|---|---|---|
| MINEFT | 8.11 | 7.87 | 41.84 | 41.84 | 5.8 |
| MINLFT | 1.63 | 1.52 | 0.70 | 0.70 | 5.9 |
| MINSLK | 5.49 | 5.26 | 41.64 | 41.64 | 6.1 |
| MTSUCC | 2.67 | 2.46 | 1.83 | 1.83 | 6.8 |
| MAXTRC | 58.73 | 17.51 | 4.17 | 4.17 | 28.2 |
| MINTRU | 58.73 | 17.65 | 4.17 | 4.17 | 20.6 |
| MINRRU | 58.73 | 17.51 | 4.17 | 4.06 | 21.9 |
| MINDRRU | 58.73 | 17.51 | 4.17 | 4.17 | 32.8 |

## 6 Summary and Future Work

In this paper we consider a generalization of the classical resource constrained project scheduling problem. So-called partially renewable resources are introduced by assuming for each resource a capacity on subsets of periods. The concept of partially renewable resources is a fundamental tool in order to make e.g. timetabling and shift scheduling aspects amenable to project scheduling. Furthermore, they cover traditional renewable and nonrenewable resource constraints as special cases. We consider makespan minimization as objective. For the exact solution of the problem we employ a basic enumeration scheme. We formulate bounds which take into account future resource consumption of partially renewable resources in order to speed up convergence. Moreover, we generalize the serial scheduling scheme in order to get fast approximation methods. A rigorous assessment of the procedures is provided by solving ProGen instances generated under a full factorial test design. Besides the well-known problem parameters we employ additionally three parameters which control the generation of partially renewable resources.

Future work should concentrate on the development of additional bounding rules in order to speed-up convergence of the exact algorithm. In addition, fast and reliable local search methods have to be developed for solving the RCPSP/$\pi$ to suboptimality.

**Appendix:** Computation of $SC_{jrt}$, $MC_{jrt}$ and $MCI_{jrt}$ for example 2

*Table A.1:* $SC_{jrt}$

| start time $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $SC_{1,1,t}$ | 2 | 2 | 2 | 1 | | | | | |
| $SC_{1,2,t}$ | 0 | 0 | 0 | 1 | | | | | |
| $SC_{1,3,t}$ | 0 | 0 | 0 | 0 | | | | | |
| $SC_{2,1,t}$ | | | 2 | 1 | 0 | 0 | 0 | | |
| $SC_{2,2,t}$ | | | 0 | 1 | 2 | 2 | 2 | | |
| $SC_{2,3,t}$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $SC_{3,1,t}$ | | | 4 | 2 | 0 | 0 | | | |
| $SC_{3,2,t}$ | | | 0 | 2 | 4 | 4 | | | |
| $SC_{3,3,t}$ | | | 0 | 0 | 0 | 0 | | | |
| $SC_{4,1,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $SC_{4,2,t}$ | | | | | 1 | 1 | 1 | 1 | |
| $SC_{4,3,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $SC_{5,1,t}$ | | | | | | 0 | 0 | 0 | 0 |
| $SC_{5,2,t}$ | | | | | | 3 | 2 | 1 | 0 |
| $SC_{5,3,t}$ | | | | | | 0 | 1 | 2 | 3 |

| start time $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $MC_{1,1,t}$ | 1 | 1 | 1 | 1 | | | | | |
| $MC_{1,2,t}$ | 0 | 0 | 0 | 1 | | | | | |
| $MC_{1,3,t}$ | 0 | 0 | 0 | 0 | | | | | |
| $MC_{2,1,t}$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $MC_{2,2,t}$ | | | 0 | 1 | 2 | 2 | 2 | | |
| $MC_{2,3,t}$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $MC_{3,1,t}$ | | | 0 | 0 | 0 | 0 | | | |
| $MC_{3,2,t}$ | | | 0 | 2 | 4 | 4 | | | |
| $MC_{3,3,t}$ | | | 0 | 0 | 0 | 0 | | | |
| $MC_{4,1,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $MC_{4,2,t}$ | | | | | 1 | 1 | 1 | 1 | |
| $MC_{4,3,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $MC_{5,1,t}$ | | | | | | 0 | 0 | 0 | 0 |
| $MC_{5,2,t}$ | | | | | | 0 | 0 | 0 | 0 |
| $MC_{5,3,t}$ | | | | | | 0 | 1 | 2 | 3 |

*Table A.2:* $MC_{jrt}$

| start time $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $MCI_{1,1,t}$ | 0 | 0 | 0 | 0 | | | | | |
| $MCI_{1,2,t}$ | 1 | 4 | 7 | 7 | | | | | |
| $MCI_{1,3,t}$ | 0 | 1 | 2 | 3 | | | | | |
| $MCI_{2,1,t}$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $MCI_{2,2,t}$ | | | 0 | 0 | 0 | 0 | 0 | | |
| $MCI_{2,3,t}$ | | | 0 | 0 | 1 | 2 | 3 | | |
| $MCI_{3,1,t}$ | | | 0 | 0 | 0 | 0 | | | |
| $MCI_{3,2,t}$ | | | 1 | 1 | 1 | 1 | | | |
| $MCI_{3,3,t}$ | | | 0 | 1 | 2 | 3 | | | |
| $MCI_{4,1,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $MCI_{4,2,t}$ | | | | | 0 | 0 | 0 | 0 | |
| $MCI_{4,3,t}$ | | | | | 0 | 1 | 2 | 3 | |
| $MCI_{5,1,t}$ | | | | | | 0 | 0 | 0 | 0 |
| $MCI_{5,2,t}$ | | | | | | 0 | 0 | 0 | 0 |
| $MCI_{5,3,t}$ | | | | | | 0 | 0 | 0 | 0 |

*Table A.3:* $MCI_{jrt}$

## References


Alvarez-Valdes, R. and J.M. Tamarit (1989): "Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis", in: Słowiński, R. and J. Węglarz (eds.): *Advances in project scheduling,* Elsevier, Amsterdam, pp. 113-134.

Bell, C.E. and J. Han (1991): "A new heuristic solution method in resource-constrained project scheduling", *Naval Research Logistics,* Vol. 38, pp. 315-331.

Bell, C.E. and K. Park (1990): "Solving resource-constrained project scheduling problems by A* search", *Naval Research Logistics,* Vol. 37, pp. 61-84.

Błażewicz, J.; W. Cellary; R. Słowiński and J. Węglarz (1986): *Scheduling under resource constraints – deterministic models,* Baltzer, Basel (*Annals of Operations Research,* Vol. 7).

Boctor, F.F. (1990): "Some efficient multi-heuristic procedures for resource-constrained project scheduling", *European J. of Operational Research,* Vol. 49, pp. 3-13.

Böttcher, J. (1995): "Projektplanung: ein exakter Algorithmus zur Lösung des Problems mit partiell erneuerbaren Ressourcen", Master Thesis, University Kiel (in German).

Brucker, P.; A. Schoo and O. Thiele (1996): "A branch and bound algorithm for the resource-constrained project scheduling problem", Working Paper, University Osnabrück.

Carlier, J. and B. Latapie (1991): "Une methode arborescente pour resoudre les problemes cumulatifs", *Recherche Operationnelle,* Vol. 25, pp. 311-340.

Christofides, N.; R. Alvarez-Valdes and J.M. Tamarit (1987): "Project scheduling with resource constraints: A branch and bound approach", *European J. of Operational Research,* Vol. 29, pp. 262-273.

Cooper, D.F. (1976): "Heuristics for scheduling resource-constrained projects: An experimental investigation", *Management Science,* Vol. 22, pp. 1186-1194.

Davis, E.W. and G.E. Heidorn (1971): "An algorithm for optimal project scheduling under multiple resource constraints", *Management Science,* Vol. 17, pp. 803-816.

Demeulemeester, E. and W. Herroelen (1992): "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem", *Management Science,* Vol. 38, pp. 1803-1818.

Demeulemeester, E. and W. Herroelen (1995): "New benchmark results for the resource-constrained project scheduling problem", Working Paper, Katholieke Universiteit Leuven.

Drexl, A. (1991): "Scheduling of project networks by job assignment", *Management Science,* Vol. 37, pp. 1590-1602.

Drexl, A. and J. Grünewald (1993): "Nonpreemptive multi-mode resource-constrained project scheduling", *IIE Transactions,* Vol. 25/5, pp. 74-81.

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

Garey, M.R. and D.S. Johnson (1979): *Computers and intractability – A guide to the theory of NP-completeness.* Freeman, San Francisco.

Hart, J.P. and A.W. Shogan (1987): "Semi-greedy heuristics: an empirical study", *Operations Research Letters,* Vol. 6, pp. 107-114.

Haupt, R. (1989): "A survey of priority rule-based scheduling", *OR Spektrum,* Vol. 11, pp. 3-16.

Icmeli, O. and W.O. Rom (1994): "Solving the resource constrained project scheduling problem with Optimization Subroutine Library", Working Paper, Cleveland State University.

Kelley, J.E. Jr. (1963): "The critical-path method: resources planning and scheduling", in: Muth, J.F. and G.L. Thompson (eds.): *Industrial scheduling,* Prentice-Hall, New Jersey, pp. 347-365.

Kolisch, R. (1995): *Project scheduling under resource constraints – Efficient heuristics for several problem classes,* Physica, Heidelberg.

Kolisch, R. (1996a): "Serial and parallel resource-constrained project scheduling methods revisited: theory and computation", *European J. of Operational Research,* Vol. 90, pp. 320-333.

Kolisch, R. (1996b): "Efficient priority rules for the resource-constrained project scheduling problem", *J. of Operations Management* (to appear).

Kolisch, R. and A. Drexl (1996): "Adaptive search for solving hard project scheduling problems", *Naval Research Logistics,* Vol. 43, pp. 23-40.

Kolisch, R.; A. Sprecher and A. Drexl (1995): "Characterization and generation of a general class of resource-constrained project scheduling problems", *Management Science,* Vol. 41, pp. 1693-1703.

Laguna, M.; T.A. Feo and H.C. Elrod (1994): "A greedy randomized adaptive search procedure for the two-partition problem", *Operations Research,* Vol. 42, pp. 677-687.

Leon, V.J. and R. Balakrishnan (1995): "Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling", *OR Spektrum,* Vol. 17, pp. 173-182.

Li, R.K.-Y. and J. Willis (1992): "An iterative scheduling technique for resource-constrained project scheduling", *European J. of Operational Research,* Vol. 56, pp. 370-379.

Mingozzi, A.; V. Maniezzo; S. Ricciardelli and L. Bianco (1994): "An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation", Working Paper, University of Bologna.

Panwalkar, S.S. and W. Iskander (1977): "A survey of scheduling rules", *Operations Research,* Vol. 25, pp. 45-61.

Oguz, O. and H. Bala (1994): "A comparative study of computational procedures for the resource constrained project scheduling problem", *European J. of Operational Research,* Vol. 72, pp. 406-416.

Patterson, J.H. and G.W. Roth (1976): "Scheduling a project under multiple resource constraints: a zero-one programming approach", *AIIE Transactions,* Vol. 8, pp. 449-455.

Pritsker, A.A.B.; L.J. Watters and P.M. Wolfe (1969): "Multiproject scheduling with limited resources: a zero-one programming approach", *Management Science,* Vol. 16, pp. 93-107.

Radermacher, F.J. (1985/86): "Scheduling of project networks", *Annals of Operations Research,* Vol. 4, pp. 227-252.

Sampson, S.E. and E.N. Weiss (1993): "Local search techniques for the generalized resource constrained project scheduling problem", *Naval Research Logistics,* Vol. 40, pp. 365-375.

Stinson, J.P.; E.W. Davis and B.M. Khumawala (1978): "Multiple resource-constrained scheduling using branch and bound", *AIIE Transactions,* Vol. 10, pp. 252-259.

Talbot, F.B. and J.H. Patterson (1978): "An efficient integer programming algorithm with network cuts for solving resource-constrained project scheduling problems", *Management Science,* Vol. 24, pp. 1163-1174.

Wiest, J.D. (1964): "Some properties of schedules for large projects with limited resources", *Operations Research,* Vol. 12, pp. 395-418.