Schirmer, Andreas

Fast iterative improvement methods for project scheduling under partially renewable resources

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 501

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 501

Fast Iterative Improvement Methods for Project Scheduling under Partially Renewable Resources

Schirmer

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 501

Fast Iterative Improvement Methods for Project Scheduling under Partially Renewable Resources

Schirmer

April 1999

Andreas Schirmer [a,b]

[a] Institut für Betriebswirtschaftslehre, Lehrstuhl für Produktion und Logistik,
Christian-Albrechts-Universität zu Kiel, Wilhelm-Seelig-Platz 1, D-24098 Kiel, Germany
[b] Institut für Informatik und Praktische Mathematik, Lehrstuhl für Systeme zur Informationsverarbeitung,
Christian-Albrechts-Universität zu Kiel, Hermann-Rodewald-Straße 3, D-24118 Kiel, Germany

Phone, Fax +49-431-880-15 31
schirmer@bwl.uni-kiel.de
www.wiso.uni-kiel.de/bwlinstitute/prod/mab/schirmer

Contents

Figures

Tables

Abstract: It is well-known that for many project scheduling problems the space **AS** of active schedules contains at least one optimal solution for each feasible instance, so restricting heuristic construction methods to **AS** will improve algorithmic efficiency without foresaking the chance to eventually find an optimal schedule. Yet, for some problems such results are not directly applicable. We address one such problem, namely the resource-constrained project scheduling problem under partially renewable resources (RCPSP/Π). Here delaying certain activities may yield better solutions than starting each activity as soon as possible. Since **AS** is the smallest space guaranteed to contain optimal schedules, it would be desirable to devise methods which sample **AS**, or at least the next-larger space **SAS** of semi-active schedules. However, no such methods are in sight because the "delay problem" of which activities should be delayed and by how many periods cannot be properly addressed during the construction process, when not all remaining resource capacities are known. Indeed, all priority rule-based construction methods for the RCPSP/Π use scheduling schemes which sample the next-larger space **FS** of feasible schedules. But while the delay problem withstands solution during the construction of schedules, it can be solved afterwards, suggesting the idea of fast iterative improvement algorithms which use appropriate shift operations to improve heuristically constructed feasible schedules. We develop several such approaches, using local and global shifts, and evaluate the effect of different design choices on effectiveness and efficiency of the algorithms. Computational results validate the efficacy of our approach.

Keywords: PROJECT SCHEDULING; PARTIALLY RENEWABLE RESOURCES; ACTIVE SCHEDULES; SHIFT; ITERATIVE IMPROVEMENT

1. Introduction

Most project scheduling problems are strongly **NP**-equivalent, in other words notoriously intractable, so the majority of algorithms developed for these are heuristic in nature. Although no theoretically provable results exist which would allow a quantitative analysis of performance measures, some theoretical results exist which at least provide some qualitative insight into the behaviour of heuristics. For sampling-based construction methods these results show that for each instance the space **AS** of active schedules - though it may fail to comprise *all* optimal schedules - will certainly contain *at least one* optimal schedule, provided the instance was feasible in the first place. Hence, restricting the sampling process to **AS** will improve algorithmic efficiency, without foresaking the chance to eventually find an optimal schedule.

For problems such as the resource-constrained project scheduling problem (RCPSP) construction methods have been devised (Kelley 1963) which can be shown to sample **AS** (Kolisch 1996). Yet, for a recent extension of the RCPSP, namely the RCPSP under partially renewable resources (RCPSP/Π), such results are not directly applicable. Partially renewable resources constitute a nontrivial generalization of all known resource concepts (Böttcher et al. 1996a, 1996b) and allow to model a variety of logical and other relations between scheduling decisions (Schirmer, Drexl 1997). Such relations are of importance for the formulation of various constraints found in real-world applications (cf. e.g. the aviation industry case studies of Haase et al. 1997, 1998; Schirmer 1999, Chapters 18, 19).

For the RCPSP/Π, we have shown that delaying certain activities may yield solutions better than those achieved from starting each and every activity as soon as possible (Schirmer 1999,

Theorem 9.5); we will discuss the properties leading to this effect later. Since **AS** is the smallest space guaranteed to contain optimal schedules, it would be desirable to devise methods which sample **AS**, or at least the next-larger space **SAS** of semi-active schedules. However, no such methods are in sight because the underlying "delay problem", viz. to decide which activities should be delayed and by how many periods, cannot be properly addressed during the construction process, where not all remaining resource capacities are known. Indeed, all authors devising priority rule-based construction methods for the RCPSP/Π (Böttcher et al. 1996b; Schirmer 1999, Chapters 11-14) use scheduling schemes which sample the next-larger space **FS** of feasible schedules (Schirmer 1999, Theorem 11.1). Nevertheless, while the delay problem withstands solution *during* the construction of schedules, it can be solved *afterwards*, when all remaining capacities are readily available.

This insight motivated our idea of devising fast iterative improvement algorithms which use appropriate shift operations to improve heuristically constructed feasible schedules in what might be called a post-processing stage. Not much research has been published that touches upon iterative improvement used that way, the only sources we are aware of are Hartmann (1997) and Sprecher et al. (1997) where so-called multi-mode left-shifts are employed within genetic algorithms and branch-and-bound methods for the multi-mode variant of the RCPSP, respectively. Albeit we concentrate on shift algorithms for the RCPSP/Π, we thus believe these or similar approaches would also be worthwhile for other scheduling problems where effective construction heuristics sample spaces larger than **AS**.

We develop several such algorithms, using local and global shifts, and evaluate the effect of different alternatives for several design choices on effectiveness and efficiency. Computational results validate the efficacy of our algorithms. We also show that straightforward approaches to shifting activities have pseudo-polynomial time complexities, making them unwise choices when tackling large instances. We demonstrate how these complexities can be cut down to a low-degree polynomial level by analyzing the schedules at hand, and we develop a number of so-called shift rules for this task. We also complement the theoretical worst-case complexity results by an experimental evaluation of the average-case performance on a test bed of 3,840 benchmark instances used in other studies on the RCPSP/Π.

The remainder of this work is organized as follows. In Section 2 we introduce the RCPSP/Π, focussing on the idiosyncrasies of partially renwable resources which make it differ substantially from other scheduling problems. In Section 3, we briefly review the concepts fundamental to schedule space classifications and shift operations, as far as they are needed here. We then use these concepts to develop three shift-based iterative improvement algorithms. Section 4 details computational results; a summary in Section 5 concludes the paper.

2. Partially Renewable Resources

2.1. Problem Setting

Partially renewable resources may be characterized by the following assumptions:

- All activities have to be processed within a number of T periods.
- Processing the activities requires the presence of one or several partially renewable resources p; the demand for resource p entailed by activity j is given by k_{jp}.
- Associated with each resource p is a so-called period subset Π_p which is a subset of the set of all periods.
- For each resource p, the amount which is available over all periods of that subset is limited by K_p.

Apart from these assumptions which specifically pertain to partially renewable resources, we employ the usual parameters of the RCPSP. All problem parameters are summarized (in alphabetical order) in Table 1. W.l.o.g. the parameters P and all d_j are assumed to be nonnegative integers, J and T to be positive integers, k_{jp} and K_p to be integers. These restrictions entail no loss of generality since they are equivalent to allowing rational numbers, i.e. fractions, and multiplying them with the smallest common multiple of their denominators. Let denote $J = \{1,...,J\}$ the set of all activities and $T = \{1,...,T\}$ the set of all periods. Finally, all Π_p are assumed to be subsets of the set T.

The goal is to find an assignment of periods to all activities (a *schedule*) that ensures for each partially renewable resource p and each period subset Π_p that the total consumption of p by all activities performed in that period subset does not exceed the total capacity of p within Π_p, respects the partial order $\angle$, and minimizes the total project length.

Model Parameter	Definition
d_j	Non-preemptable duration of activity j
J	Number of activities, indexed by j
k_{jp}	Per-period consumption of partially renewable resource p required to perform activity j
K_p	Total availability of partially renewable resource p over all periods of Π_p
P	Number of partially renewable resources, indexed by p
Π_p	Period subset associated with resource p
T	Number of periods, indexed by t
$\angle$	Partial order on the activities, representing precedence relations

Table 1: Problem Parameters of the RCPSP/Π

To simplify matters, it is assumed w.l.o.g. that activity 1 is the unique first and activity J the unique last activity w.r.t. $\angle$, i.e. $1 \angle j$ and $j \angle J$ $(2 \leq j \leq J\text{-}1)$. The fictitious activities 1 and J are called *dummy activities*, meaning that $d_1 = d_J = 0$ and $k_{1p} = k_{Jp} = 0$ $(1 \leq p \leq P)$. We also follow common practice by deriving some additional parameters from the above problem parameters; though not necessary prerequisites, they usually allow to reduce the number of variables within some of the constraints. First, let denote P_j $(1 \leq j \leq J)$ the set of all *immediate* predecessors of activity j w.r.t. $\angle$. Second, for each activity j $(1 \leq j \leq J)$ earliest finish times EFT_j and latest finish times LFT_j may be calculated by traditional forward and backward recursion, the latter starting at time T.

Using the above parameters, a schedule can be represented by integer variables y_j $(1 \leq j \leq J)$ denoting the period in which activity j is completed. In addition, we denote by A_t $(1 \leq t \leq T)$ the set of all activities which are processed in period t (Talbot, Patterson 1978) and thus may consume resources. Then the RCPSP/Π can be described in the following way:

Minimize

$$Z(y) = y_J \tag{1}$$

subject to

$$y_i \leq y_j - d_j \qquad (2 \leq j \leq J;\ i \angle j) \tag{2}$$

$$\sum_{t \in \Pi_p} \sum_{j \in A_t} k_{jp} \leq K_p \qquad (1 \leq p \leq P) \tag{3}$$

$$y_j \in \{EFT_j,\ldots,LFT_j\} \qquad (1 \leq j \leq J) \tag{4}$$

Minimization of the objective function (1) enforces the earliest possible completion of the last activity J and thus leads to the minimal schedule length. The precedence constraints (2) guarantee that the precedence order is respected while the capacity constraints (3) limit the total resource consumption of each partially renewable resource in each period subset to the available amount. A number of alternative model formulations as well as their respective advantages and disadvantages are discussed in Schirmer, Drexl (1997). The problem is strongly **NP**-equivalent, its feasibility variant strongly **NP**-complete (Schirmer, Drexl 1997).

2.2. *Special Properties*

Nonrenewable resources n are capacitated over the complete planning horizon, therefore the exact time when an activity j starts has no bearing on the total activity demand k_{jn} incurred. The capacity of renewable resources r, in contrast, is limited for each period separately, while the total activity demand of j for r is $k_{jr} \cdot d_j$. Hence the feasibility of a schedule depends on

the periods to which the activities are assigned: If two activities overlap in a period, they may overload a resource in that period, requiring to delay one activity until the other terminates, freeing capacities again. Large capacities allow for more parallel execution of activities while small capacities entail more sequential execution. In this sense, the total demand for and so the remaining capacities of nonrenewable resources are independent from the corresponding schedule. Yet, for renewable resources only the individual activity demand k_{jr} and the total activity demand $k_{jr} \cdot d_j$ are schedule-independent whereas the total demand for a specific resource r in a period t and thus the corresponding remaining capacities are schedule-dependent.

Partially renewable resources, in contrast, are capacitated over their respective period subsets only; in all other periods of the planning horizon there are no limitations. Also in the presence of partially renewable resources the feasibility of a schedule is determined by the position of the activities on the time axis: Two activities overlapping within a period subset may require more than the capacity of the corresponding resource; thus scarce resources may delay certain activities into uncapacitated periods, outside the period subsets. Consequentially, although in principle the total activity demand is $k_{jp} \cdot d_j$, only that part falling into capacitated periods, i.e. within a period subset, is relevant for algorithmic purposes. Hence, attention can be restricted from the total activity demand to the *relevant demand.*

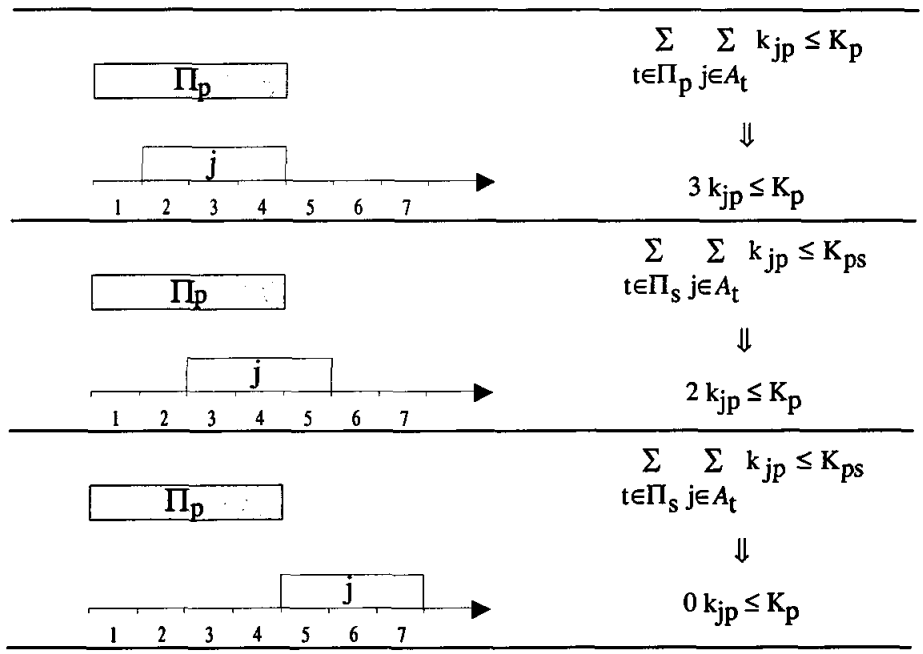

Figure 1: Total Activity Demand for Partially Renewable Resources

To determine the relevant demand RD_{jpt} of activity j for partially renewable resource p when started in period t, let Q_{jt} denote those periods of T in which (non-dummy) activity j would be active were it started in period t, i.e.

$$Q_{jt} = \{t,...,t+d_j-1\} \qquad (2 \le j \le J-1;\ 1 \le t \le T) \quad (5)$$

Then, the relevant demand can be calculated from

$$RD_{jpt} = k_{jp} \cdot |Q_{jt} \cap \Pi_p| \qquad (2 \le j \le J-1;\ 1 \le p \le P;\ EST_j+1 \le t \le LST_j+1) \quad (6)$$

as soon as earliest and latest start times EST_j and LST_j are determined in the usual fashion for each activity j. In this regard, relevant demand RD_{jpt} and thus remaining capacities are schedule-dependent although the individual activity demand k_{jp} is schedule-independent. An example is shown in Figure 1 where scheduling activity j to different periods results in different relevant demand for a partially renewable resource, depending on whether all, some, or none of the active periods fall into the corresponding period subset. A more comprehensive discussion of these concepts is given in Schirmer, Drexl (1997).

3. Iterative Improvement Algorithms

Iterative improvement algorithms belong to the class of neighborhood search algorithms: they move iteratively from solution to solution, starting from some - usually feasible - initial solution and proceeding to other solutions which are neighboring in some sense, to be defined as appropriate. The *neighborhood* of a solution comprises all *admissible moves*, which implies that certain moves may be excluded from consideration; each move is associated with the solution to which it leads. The quality of each solution - and thus of each move - is measured by an *evaluation function*. Iterative improvement methods accept only improving moves, as to avoid cycling on plateaux in the solution space; if several such moves exist in a neighborhood, the best of these is chosen (*steepest descent*, assuming a minimization objective). Such methods may also be characterized as greedy ones, as they iteratively select a candidate which is the (locally) best choice, given what has been selected before. They usually terminate if the incumbent neighborhood contains no improving move; in addition, an exogenous termination criterion, such as an iteration or a CPU time limit, may be used. We develop several such algorithms, using local and global shifts to improve heuristically constructed feasible schedules.

3.1. Classification and Connectedness of Solution Spaces

In the following, we briefly recapitulate the concepts fundamental to classifying shifts and schedules as they are needed here. Let for each scheduled activity j denote FT_j the finish time of j. For details, proofs, and examples, we refer the reader to Sprecher et al. (1995) and Nübel, Schwindt (1997). We also give a brief characterization of the algorithms to be presented.

Definition 1 A *left-shift* of an activity $j \in J$ is an operation which transforms a schedule $S = (FT_1,...,FT_J)$ into a schedule $S' = (FT'_1,...,FT'_J)$ where $FT'_j < FT_j$ and $FT'_{j'} = FT_{j'}$ for all $j' \neq j$.

Definition 2 Given a feasible schedule S, a *left-shift* of an activity $j \in J$ is called *global* iff it transforms S into a feasible schedule S'.

Definition 3 Given a feasible schedule S, a *left-shift* of an activity $j \in J$ is called a *feasible one-period left-shift* iff it transforms $S = (FT_1,...,FT_J)$ into a feasible schedule $S' = (FT'_1,...,FT'_J)$ such that $FT'_j < FT_j$ and $FT'_{j'} = FT_{j'}$ for all $j' \neq j$, *local* iff it can be produced by a sequence of feasible one-period left-shifts.

Corollary 1 Any local left-shift is global.

Definition 4 A *semi-active* (*active*) *schedule* is a feasible schedule where no activity can be locally (globally) left-shifted. Let denote **SAS** (**AS**) the space of all semi-active (active) schedules for a given instance.

We can now state the following theorem, which is due to Sprecher et al. (1995).

Theorem 1 (*Hierarchy theorem*) Let for a scheduling problem instance denote **S** the space of all schedules and **FS** the space of all feasible schedules for a given instance. Then $\mathbf{AS} \subseteq \mathbf{SAS} \subseteq \mathbf{FS} \subseteq \mathbf{S}$.

All improvement methods discussed here are neighborhood search algorithms. Following the lines of Glover et al. (1993) we employ a neighborhood decomposition strategy as we use J separate neighborhoods in each iteration, one per activity. In each iteration, the neighborhood for an activity j comprises all schedules which can be built from the current one by applying a local or global left-shift to j. From each of these one schedule is selected, which is evaluated either in terms of its objective function value or some measure of the resource demand and the resource scarcity entailed by the schedule. The algorithms terminate when in an iteration all J neighborhoods were found to be empty. Such methods may also be characterized as greedy ones, as they iteratively select candidates, each of which is the (locally) best choice, given what has been selected before.

We finish this subsection by noting an important implication of the above neighborhood definition. Of great importance for the development of neighborhood search-based methods for a problem are the topological structures, in particular the connectedness of its solution spaces. For combinatorial problems, solution space connectedness is usually defined via the graph-theoretical concept of connectedness, by defining a graph whose nodes represent solutions. Edges between these nodes reflect that the solutions are "adjacent" in some sense to be defined as appropriate, so traversing the graph is equivalent to moving among the solutions. Whenever the resulting graph is connected, the solution space is called connected (cf. e.g. Ehrgott, Klamroth 1997). For our purposes, we will consider two feasible schedules adjacent iff they have J-1 finish times in common. This definition allows to traverse the space **FS** by applying local and global shifts. Now, algorithmically a connectedness result for **FS**

would mean that all feasible schedules could be determined by repeated shifts, starting from an arbitrary feasible schedule. However, for the RCPSP/Π the space **FS**, even its subspaces **AS** and **SAS**, as well as the space **OS** of optimal solutions are disconnected.

Theorem 2 For the RCPSP/Π, the spaces **AS**, **SAS**, **FS**, and **OS** are disconnected.

Proof: Assume an instance of the RCPSP/Π with J = 2 (discounting dummies), $d_1 = d_2 = 2$, P = 1, T = 4, $\Pi_p = \{1, 2\}$, $K_p = k_{1r} \cdot d_1 = k_{2r} \cdot d_2$, and $\angle$ empty. Confer to the examplary schedules in Figure 2. Both schedules are optimal, thus also active, semi-active, and feasible. Yet, due to the tight horizon T, there are no shifts possible at all, so both schedules are non-adjacent and the assertion holds. ■

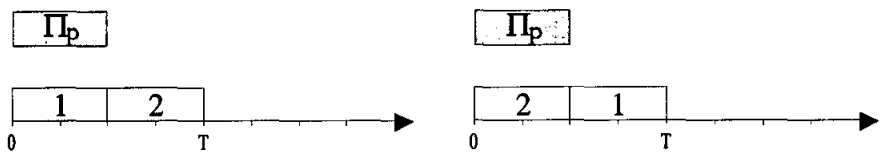

Figure 2: On the Disconnectedness of RCPSP/Π-Solution Spaces

As a consequence, any neighborhood search-based algorithm for the RCPSP/Π must either define the neighborhood in a way as to include moves leading to infeasible schedules or risk to exclude feasible and optimal schedules. Clearly, our neighborhood definition belongs to the second class; yet we accept this impediment in order to keep our algorithms as fast as possible, deliberately trading efficiency for effectiveness. More involved neighborhood search based algorithms, which trade effectiveness for efficiency, are covered in Schirmer (1999, Section 16).

3.2. Local Left-Shifting

Feasible schedules that are not semi-active can be improved by local left-shifts. The easiest way to maintain feasibility throughout this process consists of repeatedly checking for each activity j whether a one-period left-shift would still produce a feasible schedule. Notice that resource feasibility is the only real concern here since precedence feasibility can be maintained by checking only the start times between the current start time ST_j and the earliest precedence-feasible start time $EPST_j$. Doing so, however, would entail a time complexity of O(J·P·T), as T is pseudo-polynomial in the input length (Schirmer, Drexl 1997) this way appears as a somewhat undesirable choice.

In the sequel, we therefore present a more evolved approach whose time complexity depends not on T but on the number of intervals; we will see that this is indeed a polynomial quantity.

We trim the number of considered start times by exploiting the fact that it suffices to consider the start and finish times of the period subset intervals, plus some additional times around these. The well-known rule of Johnson (1967) uses a similar idea to reduce the complexity of the parallel scheduling scheme for the RCPSP. (Let us add at this point that the above time-complexity is a worst-case one, without implications on the average case. We will thus experimentally explore the pseudo-polynomial alternative later, in the context of an algorithm using global left-shifts.)

Some additional notation will facilitate the forthcoming presentation. Given a feasible schedule $S = (FT_1,...,FT_J)$, let for each activity j denote ST_j the corresponding start time, i.e.

$$ST_j \leftarrow FT_j - d_j \qquad (1 \leq j \leq J) \quad (7)$$

Let also denote, for each activity j, $EPST_j$ the (dynamically computed) earliest precedence-feasible start time of activity j, i.e.

$$EPST_j \leftarrow \max \{FT_i \mid i \angle j\} \qquad (1 \leq j \leq J) \quad (8)$$

and PR_j the set of resources requested by j, i.e.

$$PR_j \leftarrow \{1 \leq p \leq P \mid k_{jp} > 0\} \qquad (1 \leq j \leq J) \quad (9)$$

Finally, note that each period subset Π_p can be represented, rather than by listing all its periods explicitly, in terms of I_p intervals of the form $STI_{pi},...,FTI_{pi}$ $(1 \leq p \leq P;\ 1 \leq i \leq I_p)$ where STI_{pi} (FTI_{pi}) $(0 \leq STI_{pi} \leq T\text{-}1;\ 1 \leq FTI_{pi} \leq T)$ denotes the start (finish) time of interval i of period subset Π_p. Figure 3 depicts an example where $I_p = 2$, $STI_{p1} = 2$, $FTI_{p1} = 3$, $STI_{p2} = 4$, and $FTI_{p2} = 7$.

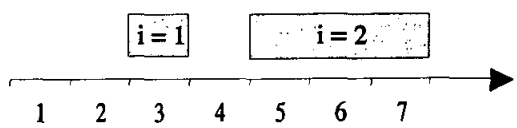

Figure 3: Exemplary Interval Structure of a Period Subset

The operating principle of the local left-shift (LLS) algorithm is to consider all activities in the order of their index (recall that they are assumed to be topologically sorted) and to apply local left-shifts to each activity as long as possible. By following the topological order each activity can be treated separately (neighborhood decomposition) without running the risk of loosing the precedence feasibility of the solutions constructed. For the sake of transparency, which a closed-form recursive equation would lack, we decompose the shifting process into a number of elementary steps. In each step, the current situation is considered - for each of the

requested resources $p \in PR_j$ separately - according to two criteria: first, whether j is incident with one or with more intervals of the corresponding period subset, and second, where j is positioned w.r.t. the incident intervals. This analysis yields one local left-shift (not necessarily the maximum possible) of j per requested resource, calculated in terms of a candidate start time (CST_{jp}). In order to assure feasibility w.r.t. all resources, the actual left-shift is then performed to CST_j, the maximum of the CST_{jp}. Afterwards, the remaining capacities of the resources requested by j are updated, if j has been moved at all. This stepwise process continues until no further local left-shift can be applied to j. The algorithm then proceeds to the next activity, attempting to left-shift it, until eventually all activities have been considered. Thus, within each neighborhood of activity j, we select that schedule which minimizes ST_j. Due to the neighborhood decomposition strategy, this criterion excludes the possibility of ties.

W.r.t. the worst-case time complexity of the algorithm, note that the fact that the shifts are feasible frees us from checking resource-feasibility of each point in time over the whole planning horizon. Their number might well reach O(T) which is pseudo-polynomial in the instance length (for details cf. Schirmer, Drexl 1997). If we let denote I the largest number of intervals over all period subsets, i.e.

$$I \leftarrow \max \{I_p \mid 1 \le p \le P\} \tag{10}$$

then, by construction, the above steps can be applied at most $2 \cdot P \cdot I$ times to each activity, so the algorithm has a complexity of $O(J \cdot P \cdot I)$ which is polynomial in the instance length.

What remains to be shown is the way in which the remaining capacities are managed. Let denote RK_p ($1 \le p \le P$) the remaining capacity of resource p. With S the schedule at hand, the remaining capacities can be initialized by

$$RK_p \leftarrow K_p - \sum_{j=2}^{J-1} RD_{j,p,ST_j} \qquad (1 \le p \le P) \tag{11}$$

and, once an activity j has been shifted from ST_j to CST_j, updated by

$$RK_p \leftarrow RK_p + RD_{j,p,ST_j} - RD_{j,p,CST_j} \qquad (p \in PR_j) \tag{12}$$

Having laid the groundwork, we can now present the local left-shift rules themselves. Recall that all rules examine one particular activity j and one particular resource p at a time. Then, the first two rules formulate conditions under which the start of j takes place *before, between, or after* the intervals of the corresponding period subset Π_p. The third rule applies to situations where the processing of j occurs completely *within* one interval of a period subset. The last rule covers conditions under which an activity starts *in one* interval and terminates *either in another or between* two intervals of period subset Π_p.

Local left-shift rule 1 (LLS1) Let j be an activity and $p \in PR_j$ a resource requested by j where j starts before any interval of the corresponding period subset Π_p, i.e.

$$EPST_j < ST_j \leq STI_{p1} \tag{13}$$

Then, w.r.t. p, j could be left-shifted to

$$CST_{jp} \leftarrow EPST_j \tag{14}$$

Proof: Under the conditions of LLS1 for an activity j, no left-shift would change either the relevant demand for or the remaining capacity of resource p. Starting j at ST_j is feasible, so starting j at $EPST_j$ is feasible as well. ■

Local left-shift rule 2 (LLS2) Let j be an activity and $p \in PR_j$ a resource requested by j where j starts either between two intervals or after the last interval of period subset Π_p, i.e.

$$((\exists\, 1 \leq i \leq I_p\text{-}1)\; FTI_{pi} < ST_j \leq STI_{p,i+1}) \vee ((\exists\, i = I_p)\; FTP_{pi} < ST_j) \tag{15}$$

Also, let Ω_{jp} denote the overlap of j with the interval i' of resource p in which j terminates, i.e. the number of periods by which i' and j overlap. Ω_{jp} can be determined from

$$\Omega_{jp} \leftarrow \begin{cases} FT_j - STI_{pi'} & \text{if}(\exists\, 1 < i' \leq I_p)\; STI_{pi'} < FT_j \leq FTI_{pi'} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Then, w.r.t. p, j could be left-shifted to

$$CST_{jp} \leftarrow \begin{cases} EPST_j & \text{if } d_j - \Omega_{jp} \leq \lfloor RK_p / k_{jp} \rfloor \\ \max\{EPST_j, FTI_{pi} - \lfloor RK_p / k_{jp} \rfloor - \Omega_{jp}\} & \text{otherwise} \end{cases} \tag{17}$$

Proof: (17) determines the maximum number of periods which may fall into intervals of Π_p without overloading the current remaining capacity of p. Since by assumption the schedule at hand is feasible, an overlap of Ω_{jp} periods is feasible in any case. $\lfloor RK_p/k_{jp} \rfloor$ states the number of additional periods by which the current overlap could be feasibly increased. Recall that $d_j \cdot k_{jp}$ is the maximum possible resource demand entailed by activity j. Under the conditions of LLS2, $d_j - \Omega_{jp}$ is the number of periods which j is processed outside of Π_p. Now, the first branch of (17) exploits that if this number is not greater than $\lfloor RK_p/k_{jp} \rfloor$ then there is no way of overloading resource p with activity j; therefore w.r.t. p activity j could be started anytime, in particular at its earliest precedence-feasible start time. Otherwise, the second branch specifies the number of periods by which j could currently be left-shifted without overloading p. Since, however, j may not be left-shifted further than $EPST_j$ without violating the precedence order on the activities, j may only shifted to the larger of both times. ■

While Ω_{jp} is empty if the processing of j does not overlap with some interval i' > i, it allows us to cover two cases, viz. that j finishes between intervals as well that it does in some interval, in one rule.

Local left-shift rule 3 (LLS3) Let j be an activity and $p \in PR_j$ a resource requested by j where j is active completely within one interval of period subset Π_p, i.e.

$$(\exists\, 1 \le i \le I_p)\ STI_{pi} < ST_j \wedge FT_j \le FTI_{pi} \tag{18}$$

Then, w.r.t. p, j could be left-shifted to

$$CST_{jp} \leftarrow EPST_j \tag{19}$$

Proof: Shifting activity j within interval i would not change the remaining capacity RK_p of p, left-shifting j out of i would possibly even increase RK_p. Since starting j at ST_j is feasible, so is starting j at $EPST_j$. ■

Local left-shift rule 4 (LLS4) Let j be an activity and $p \in PR_j$ a resource requested by j where j starts within an interval i of period subset Π_p without finishing in i, i.e.

$$(\exists\, 1 \le i \le I_p\text{-}1)\ STI_{pi} < ST_j \le FTI_{pi} \wedge FTI_{pi} < FT_j \tag{20}$$

Then, w.r.t. p, j could be left-shifted to

$$CST_{jp} \leftarrow \max\ \{EPST_j,\ ST_j - (\lfloor RK_p/k_{jp} \rfloor + \Omega_{jp})\} \tag{21}$$

Proof: Again, $\lfloor RK_p/k_{jp} \rfloor + \Omega_{jp}$ specifies the number of periods which activity j may be left-shifted from its current start time w.r.t. resource p. Yet, to maintain precedence feasibility, the shift may not exceed $EPST_j$. ■

Note that the conditions of the rules (cf. Figure 4) are mutually exclusive, so for each combination of activity j and requested resource p we can use LLS_{jp} to denote the currently applicable local left-shift rule and CST_{jp} to denote the candidate start time derived from LLS_{jp}. Note also that even one shift may free enough capacity to allow another activity to be shifted; therefore we have to reconsider all activities whenever at least one of them has actually been moved. Now, the LLS algorithm can be formulated as in Table 2 (the bounds PB and RB are discussed below).

Input

a feasible schedule S

Initialization

for each $(1 \le p \le P)$
  initialize RK_p **according to** (11)
determine PR_j **according to** (9)
some_j_may_be_shiftable ← TRUE

Execution

while (some_j_may_be_shiftable)
  some_j_may_be_shiftable ← FALSE
  for j ← 2 **to** J
    calculate $EPST_j$ **according to** (8)
    if $(ST_j = EPST_j)$
      j_may_be_shiftable ← FALSE
    else
      j_may_be_shiftable ← TRUE
    while (j_may_be_shiftable)
      for each $p \in PR_j$
        determine LLS_{jp}
        calculate CST_{jp} **according to** LLS_{jp}
      endfor
      $CST_j \leftarrow \max \{CST_{jp} \mid p \in PR_j\}$
      if (PB or RB)
        $ST_j \leftarrow CST_j$
        for each $p \in PR_j$
          update RK_p **according to** (12)
        j_may_be_shiftable ← FALSE
        some_j_may_be_shiftable ← TRUE
      else if $(CST_j = ST_j)$
        j_may_be_shiftable ← FALSE
      else if
        $ST_j \leftarrow CST_j$
        for each $p \in PR_j$
          update RK_p **according to** (12)
        some_j_may_be_shiftable ← TRUE
      endif
    endwhile
    $FT_j \leftarrow ST_j + d_j$
  endfor
endwhile

Result

For each activity $j \in S$, FT_j denotes the period in which j is finished. ■

Table 2: Local Left-Shift Algorithm

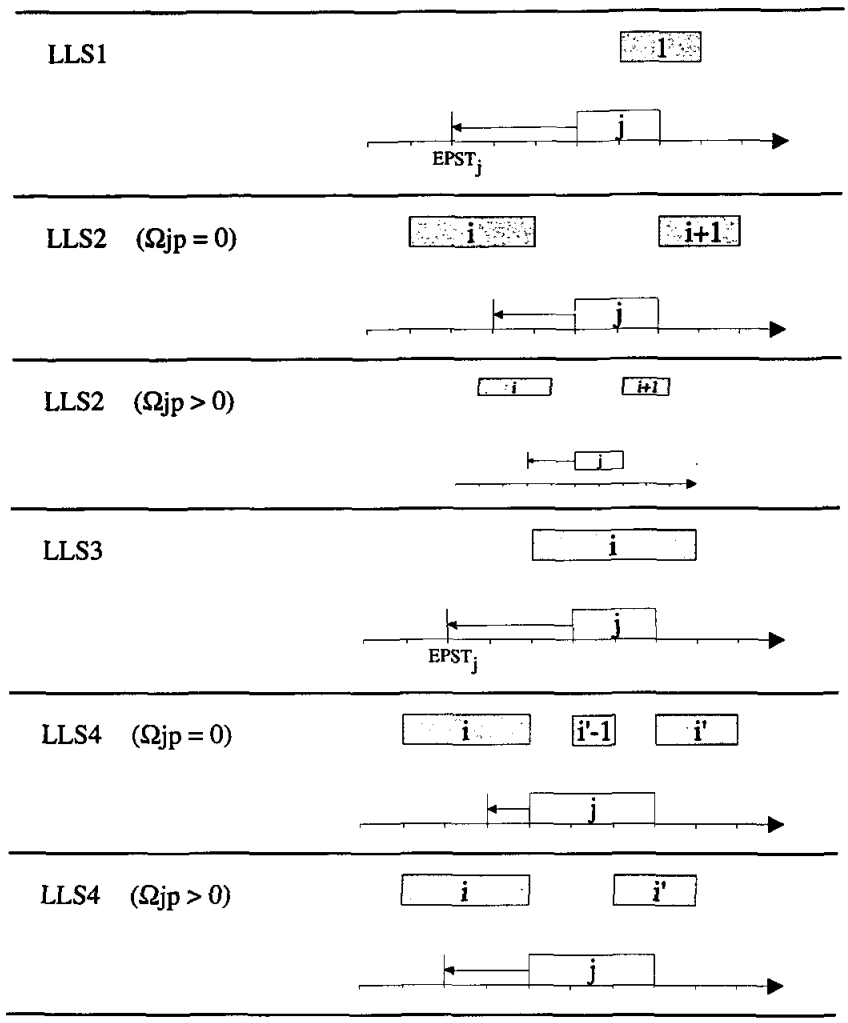

Figure 4: On the Local Left-Shift Rules

Even if the time complexity of the above algorithm is polynomial in the length of the respective instance tackled, the effort of considering possible shifts can be further reduced by employing lower bounding techniques similar to those used in other constructive scheduling heuristics (Kolisch, Drexl 1996; Schirmer, Riesenberg 1998).

Precedence-based bound (PB) Whenever $ST_j = EPST_j$ for an activity j, shifting j can be terminated because in this case j cannot possibly be scheduled any earlier. ■

Resource-based bound (RB) Whenever, for an activity j and one of its requested resources $p \in PR_j$, CST_j lies within an interval of Π_p whereas $CST_j + d_j$ lies outside all intervals of Π_p, i.e.

$$(\exists\, 1 \le i \le I_p\text{-}1)\ STI_{pi} < CST_j \le FTI_{pi} \wedge (\exists\, i < i' \le I_p)\ FTI_{p,i'-1} < CST_j + d_j \le STI_{p,i'} \quad (22)$$

shifting j can be terminated as j cannot be locally left-shifted further without overloading p. ■

3.3. Global Left-Shifting

Non-active feasible schedules can be improved by administering global left-shifts until eventually they become active. The straightforward way to do this is to check for each activity j all start times between the current one and $EPST_j$ by iteratively applying one-period left-shifts; the activity can then be shifted to the earliest feasible start time encountered during this process. Yet this way is rather inefficient: as it proceeds from the right to the left on the time-axis, it requires to check ST_j - $EPST_j$ start times *in any case*. A more efficient way is to proceed from the left to the right, by checking whether j can be started at $EPST_j$, and checking the subsequent start times for resource-feasibility if that start time is infeasible. This way requires ST_j - $EPST_j$ start times to be evaluated *in the worst case only*. Although, as we have noted above, this entails a pseudo-polynomial time complexity in the worst case, preliminary tests showed computation times not to be prohibitive for the benchmark instances used.

Recall that for each activity j, the local left-shift algorithm begins with an existing, feasible schedule, where j starts at ST_j. As we intend to decrease this start time, the algorithm iteratively tests earlier start times which might possibly be feasible. This process continues until no earlier feasible start time may exist, i.e. after $EPST_j$ has been tested. Note that only resource-feasibility has to be verified since all times considered are precedence-feasible by construction. Now, the global left-shift (GLS) algorithm starts from a virtual schedule where j is assumed to start at its earliest precedence-feasible start time $EPST_j$. This candidate start time may or may not be feasible. In the latter case, the algorithm iteratively tests later start times which might possibly be feasible. This process terminates as soon as the first feasible one has been found or if all possible start times up to ST_j-1 have been examined. Again, it suffices to check resource-feasibility. If an activity has been actually shifted, the remaining capacities of its requested resources are updated. Each activity is considered at least once, in index order. Again, any shift may free resource capacities and thus allow other shifts as well, so all activities must be reexamined once one of them has been shifted. Again, we select that unique schedule from each neighborhood which minimizes ST_j.

The time complexity of this procedure is $O(J \cdot P \cdot T)$. The GLS algorithm can be formulated as in Table 3.

Input

a feasible schedule S

Initialization

for each $(1 \le p \le P)$
  initialize RK_p **according to** (11)
determine $PR_j \leftarrow \{1 \le p \le P \mid k_{jp} > 0\}$
some_j_may_be_shiftable $\leftarrow$ TRUE

Execution

while (some_j_may_be_shiftable)
  some_j_may_be_shiftable $\leftarrow$ FALSE
  for $j \leftarrow 2$ **to** J
    calculate $EPST_j$ **according to** (8)
    if $(ST_j > EPST_j)$
      $CST_j \leftarrow EPST_j$
      while $((\exists\, p \in PR_j)\ (RD_{j,p,CST_j} > RK_p + RD_{j,p,ST_j}))$
        $CST_j \leftarrow CST_j + 1$
      if $(CST_j < ST_j)$
        $ST_j \leftarrow CST_j$
        for each $p \in PR_j$
          update RK_p **according to** (12)
        some_j_may_be_shiftable $\leftarrow$ TRUE
      endif
    endif
    $FT_j \leftarrow ST_j + d_j$
  endfor
endwhile

Result

For each activity $j \in S$, FT_j denotes the period in which j is finished. ■

Table 3: Global Left-Shift Algorithm

3.4. Hybrid Left-Shifting

Another approach to turning feasible schedules into active ones proceeds in two stages. We call this approach a hybrid left-shift (HLS) algorithm as it employs a combination of local and global left-shifts. While the second stage is identical to the GLS algorithm, the first stage, using local left-shifts, can be seen as preparing the initial schedule for the GLS algorithm. Rather than shifting the activities to the left-most, i.e. earliest, of the start times reachable by feasible one-period left-shifts, as done by the LLS algorithm, they are shifted to one of the start times minimizing their total relevant demand, thus maximizing the remaining capacities. Freeing more capacities in the beginning, we hoped, would pave the way for more effective global shifts in the end.

In the following, we will detail the first stage of the HLS. Notice that, since we use local left-shifts, we do not necessarily consider all start times t between $EPST_j$ and ST_j but only those which can be reached by feasible one-period left-shifts, starting from ST_j. The left-most of these, which we denote by $\tilde{t}_j$, can be determined from

$$\tilde{t}_j \leftarrow \max \{EPST_j \leq t \leq ST_j \mid RD_{jpt} \leq RK_p \, (p \in PR_j)\} \qquad (2 \leq j \leq J) \quad (23)$$

such that we consider all start times t between $\tilde{t}_j$ and ST_j.

In order to avoid bottlenecks formed by resources with scarce remaining capacities, not all resources are treated equal. Rather, an implicit resource levelling is performed by preferring shifts which free capacities of scarce resources over those which free capacities of non-scarce ones. This is achieved by means of weights ω_p expressing the scarcity of resource p and thus the importance of freeing some of its capacity. These weights, derived dynamically from

$$\omega_p \leftarrow \frac{1}{RK_p + 1} \qquad (1 \leq p \leq P) \quad (24)$$

are used to prioritize the relevant demand of activities for resources: the scarcer a resource, the higher the priority to free some of its capacity. We therefore aim at finding, for each activity j, a candidate start time CST^*_j ($\tilde{t}_j \leq CST^*_j \leq ST_j$) such that the total weighted relevant demand $TWRD_{jt}$ of activity j starting at time t is minimal, i.e.

$$TWRD_{jt} \leftarrow \sum_{p \in PR_j} \omega_p \cdot RD_{jpt} \qquad (25)$$

$$CST^*_j \leftarrow \text{argmin} \{TWRD_j \mid \tilde{t}_j \leq t < ST_j\} \qquad (26)$$

Activity j is then shifted to start at CST^*_j. In other words, we chose from each neighborhood that schedule whose start time for activity j minimizes $TWRD_{jt}$, ties to be broken by smallest activity index. Again this process is repeated for each activity in index order, as long as some activity has been shifted. The time complexity of this procedure is O(J·P·T). Now, the HLS algorithm can be couched as in Table 4 where the GLS algorithm is called as the second stage.

In order to illustrate the potential advantage of the HLS over the GLS algorithm, assume an instance where J = 2 (we omit the dummy activities), $d_1 = d_2 = 4$, P = 2, $k_{11} = k_{12} = k_{21} = 1$, $k_{22} = 2$, T = 12, and $\angle$ empty. The period subsets are depicted in Figure 5, where the intervals i of the period subsets p (denoted by p, i) are put in shading. Let us also assume that $EPST_1 = 0$ and that $RK_1 = 1$ and $RK_2 = 1$ for the initial schedule 1.

Input

a feasible schedule S

Initialization

for each $(1 \le p \le P)$
  initialize RK_p **according to** (11)
determine $PR_j \leftarrow \{1 \le p \le P \mid k_{jp} > 0\}$
some_j_may_be_shiftable $\leftarrow$ TRUE

Execution

while (some_j_may_be_shiftable)
  some_j_may_be_shiftable $\leftarrow$ FALSE
  for $j \leftarrow 2$ **to** J
    calculate $EPST_j$ **according to** (8)
    calculate ω_p **according to** (24)
    if $(ST_j > EPST_j)$
      $CST_j \leftarrow ST_j$
      calculate $TWRD_j$ **according to** (25)
      $CST_j \leftarrow ST_j - 1$
      $CST^*_j \leftarrow ST_j$
      $TRWD^*_j \leftarrow TWRD_j$
      while $((\forall\, p \in PR_j)\ (RD_{j,p,CST_j} \le RK_p + RD_{j,p,ST_j}))$ and $(CST_j \ge EPST_j)$
        calculate $TWRD_j \leftarrow$ **according to** (25)
        if $(TWRD_j < TRWD^*_j)$
          $TRWD^*_j \leftarrow TWRD_j$
          $CST^*_j \leftarrow CST_j$
        endif
        $CST_j \leftarrow CST_j - 1$
      endwhile
      if $(CST^*_j < ST_j)$
        $ST_j \leftarrow CST^*_j$
        for each $p \in PR_j$
          update RK_p **according to** (12)
        some_j_may_be_shiftable $\leftarrow$ TRUE
      endif
    endif
    $FT_j \leftarrow ST_j + d_j$
  endfor
endwhile

call GLS

Result

For each activity $j \in S$, FT_j denotes the period in which j is finished. ■

Table 4: Hybrid Left-Shift Algorithm

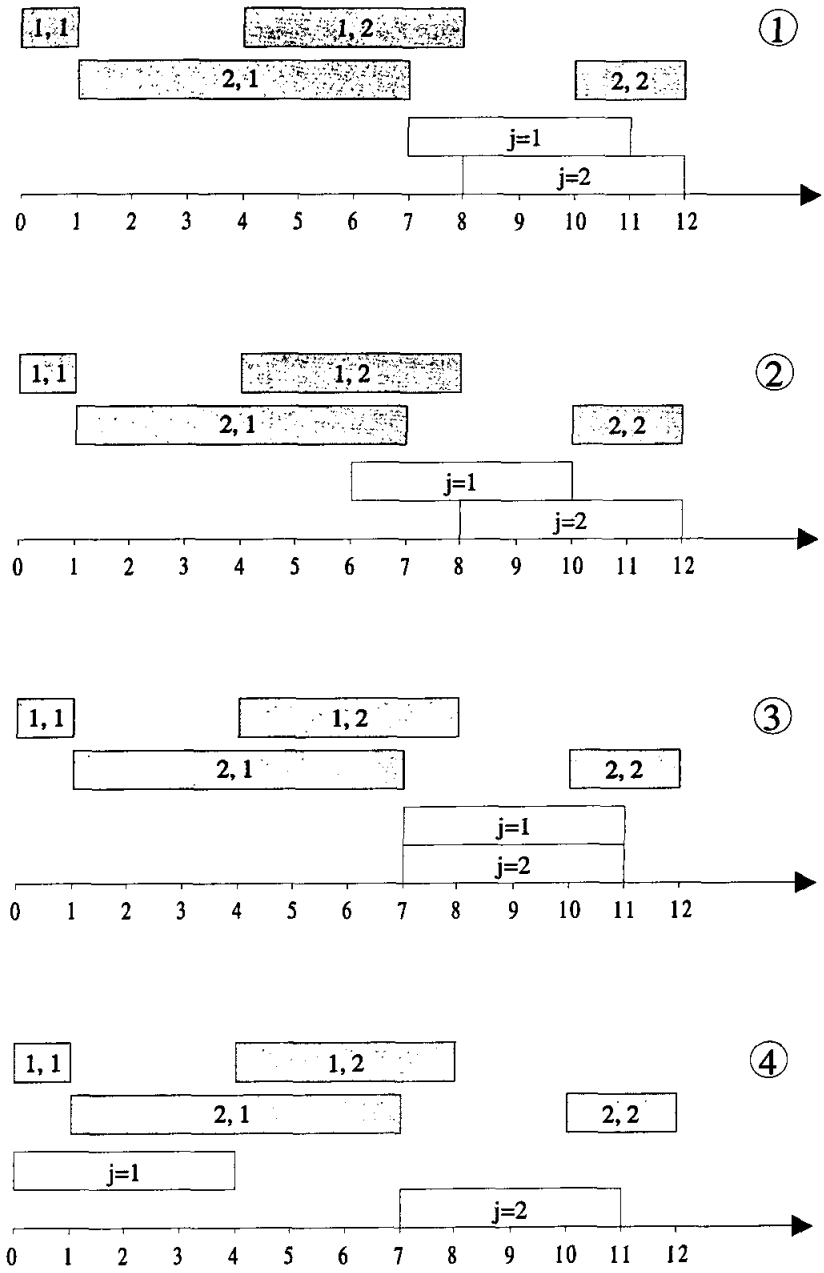

Figure 5: On Global vs. Hybrid Left-Shift Algorithm

Now, the GLS algorithm shifts activity 1 by one period, after which no further shifts are possible (schedule 2). The first stage of the HLS algorithm, however, begins by computing the weights $\omega_1 = 0.5$ and $\omega_2 = 1$. It then evaluates the $TWRD_{jt}$ for activity 1, checking the start times 7 ($TWRD_{1,7} = 1.5$) and 6 ($TWRD_{1,6} = 2$) (further shifts of activity 1 are infeasible due to $RK_1 = 0$), and leaves activity 1 unmoved since $TWRD_{1t}$ is minimal for the current start time $ST_1 = 7$. For activity 2, start times 8 ($TWRD_{2,8} = 4$) and 7 ($TWRD_{2,7} = 2.5$) are checked (further shifts of activity 2 are infeasible due to $RK_1 = 0$), and activity 2 is shifted by one period as $TWRD_{2t}$ is minimal for $t = 7$. No further local shifts are feasible for both activities, so the first stage terminates (schedule 3). Now, RK_2 has been increased from

0 to 2 units, which allows the second stage to globally shift activity 1 to start at $EPST_1 = 0$. The result is a decrease of the project makespan by one period (schedule 4).

3.5. Roads Not Travelled

As any local shift is also global by Corollary 1, one might expect the GLS algorithm to produce schedules at least as good as those produced by the LLS algorithm. Although this relationship holds in general, there are cases where the GLS algorithm returns a schedule worse than the LLS algorithm, due to the fact that both algorithms prescribe the exact order in which the activities are to be examined.

Lemma 1 There exist schedules for RCPSP/Π-instances which are improved more by the LLS algorithm than by the GLS algorithm.

Proof: As an example, assume an instance where $J = 2$ (we omit the dummy activities), $d_1 = 1$, $d_2 = 3$, $P = 2$, $k_{jp} = 1$ ($1 \leq j \leq J$; $1 \leq p \leq P$), $K_1 = K_2 = 1$, and $1 \angle 2$. Consider the schedule depicted in Figure 6 where the intervals i of the period subsets p (denoted by p, i) are put in shading. Both algorithms examine activity 1 first. Applying the LLS algorithm results in a one-period left-shift of activity 1 which frees enough capacity of resource 2 that also activity 2 can be left-shifted by one period, improving the makespan by one period. Yet, the GLS, albeit moving activity 1 by two periods, does not allow activity 2 to be moved at all. ■

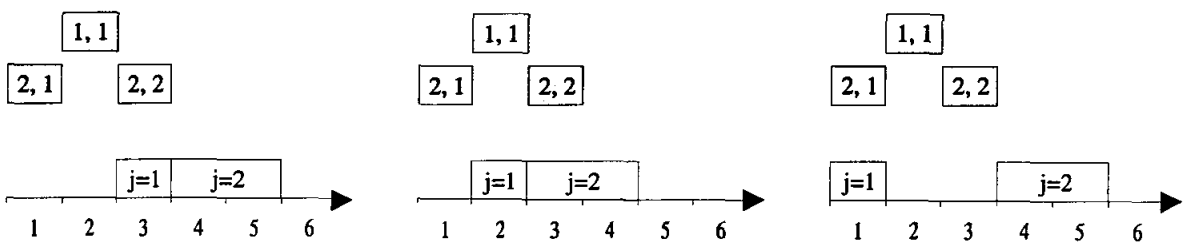

Figure 6: On Local vs. Global Left-Shift Algorithm

Consequently, an interesting question is whether changing the sequence in which activities are considered might improve the effectiveness of the algorithms. Yet, changing the sequence of the activities tends to produce negative effects on efficiency. As most sequences other than that prescribed by the index order deviate from the activities' topological order, some shifts may have to be shorter in order to maintain feasibility, and some activities may not be shiftable at all. Hence, if we require an iteration to consider each activity in some order, shifting algorithms may need more iterations.

Lemma 2 Considering activities in non-topological order may entail more iterations than considering them in topological order.

Proof: Consider an instance where J = 3 (we omit the dummy activities), $d_j = 1$ ($1 \leq j \leq J$), P = 1, $\Pi = \{3, 4\}$, $k_{j1} = 2$ ($1 \leq j \leq J$), $K_1 = 2$, and $1 \angle 2 \angle 3$, and the schedule depicted in Figure 7. If activity 1 is considered first, then activity 1 (2, 3) can be shifted by one (two, two) period(s), in one iteration. However, when activity 2 is shifted first it can be shifted by only one period, due to the precedence relation between 1 and 2. Shifting activity 3 first is infeasible because of $2 \angle 3$; notice also the resource is already used to capacity. Therefore, at least two iterations are required to achieve the same result.

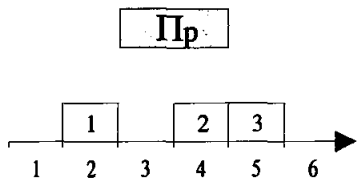

Figure 7: On the Effect of Shift Sequences

As the loss in efficiency entailed by attempting other sequences would conflict with the idea of using the algorithms presented as fast improvement procedures, we have chosen not to examine such algorithmic variants.

4. Experimental Analysis

4.1. Experimental Design

The initial schedules were constructed with a randomized construction method (Schirmer 1999, Chapter 12) using an appropriate adaptation of the well-known serial scheduling scheme. The scheme divides the set of activities into three disjoint subsets or states: scheduled, eligible, and ineligible (cp. Kurtulus, Narula 1985): an activity that is already in the partial schedule is scheduled, otherwise, an activity is called eligible if all its predecessors are scheduled, and ineligible else. The scheme proceeds in N = J stages, indexed by n. For notational purposes, we refer on stage n to the set of scheduled activities as S_n and to the set of eligible activities as D_n. On each stage n, one activity j from D_n is selected - using a priority rule if more than one activity is eligible - and scheduled to begin at some feasible start time. Then j is moved from D_n to S_n which may render some ineligible activities eligible. The scheme terminates either on stage J if all activities are scheduled (feasible solution found) or if no further selection is possible (no feasible solution found).

Eight priority rules were used, as listed in Table 5; formal definitions missing here are provided in the Appendix. These rules were selected since w.r.t. effectiveness these cover the whole spectrum of priority rules for the RCPSP/Π: the frist three are the most effective rules currently known whereas RRD is the worst-performing one; the other rules settle between these extrema. For further details on these algorithms, we refer the reader to Schirmer (1999, Chapter 12) from which also the control parameter settings were taken.

Extre-mum	Measure	Definition	Static vs. Dynamic	Local vs. Global	Simple vs. Composite
MAX	DRC/E_{jt}	$\leftarrow \Sigma_p (RK_{pn} - RD_{jpt} - MDE_{jpt})$	D	G	S
MAX	DRC/ES_{jt}	$\leftarrow \Sigma_{p \in SPn} (RK_{pn} - RD_{jpt} - MDE_{jpt})$	D	L	S
MIN	DRS/ES_{jt}	$\leftarrow \Sigma_{p \in SPn} (RD_{jpt} + MDE_{jpt}) / RK_{pn}$	D	G	S
MIN	EST_j	$\leftarrow EFT_j - d_j$	S	G	S
MAX	MTS_j	$\leftarrow \lvert \{j' \mid j \angle j'\} \rvert$	S	L	S
MIN	RRD_{jt}	$\leftarrow \Sigma_p RD_{jpt} / (k_{jp} \cdot d_j)$	S	L	S
MIN	SPT_j	$\leftarrow d_j$	S	L	S
MIN	TRS_{jt}	$\leftarrow \Sigma_p RD_{jpt} / K_p$	S	L	S

Table 5: Priority Rules

The factors examined in our experimentation comprise shifting algorithm, priority rule employed in the construction method, and instance size, measured in terms of the number of activities J. Specifying a set of values for each factor describes over which levels it is varied during an experiment, while one value for each factor determines a run of an experiment. The outcome of each run is reported in terms of several result variables, related to effectiveness and efficiency. Effectiveness is measured in terms of the percentage deviation of the schedule returned from the initial schedule, the number of schedules improved, and the number of schedules improved to optimality. Efficiency is measured in terms of CPU-time required in seconds. In addition, some related quantities are reported, viz. the number of activities actually shifted, the average number of periods by which each such activity has been shifted, and the number of iterations. All algorithms were implemented and compiled using Borland C, data points were taken on a 486DX/66 personal computer with 12 MB RAM run under Windows 95.

As a test bed, we used the benchmark instance sets J10, J20, J30, and J40 generated with ProGen/Π (Schirmer 1999, Chapter 10). Each instance comprises 10, 20, 30, or 40 non-dummy activities, as indicated by the naming. Each activity has a nonpreemptable duration of between one and ten periods and may require one or several of thirty partially renewable resources present. The number of successors and predecessors w.r.t. the precedence order varies

between one and three for each activity. Systematically varied design parameters are complexity index (CI), resource factor (RF), resource strength (RS), cardinality factor (CF), horizon factor (HF), and interval factor (IF). As CI we implemented an estimator of the restrictiveness of a network developed and evaluated by Thesen (1977); RF determines the number of resources that are requested by each activity, and RS expresses resource scarcity measured between minimum and maximum demand. CF controls the cardinality of the period subsets, HF the tightness of the planning horizon T up to which all activities must be completed. IF, finally, serves to establish the degree of fragmentation of the period subsets (cf. Table 6).

Design Parameter	Related To	Level = 0	Level = 1
CI Complexity Index	Network	network is parallel digraph	network is serial digraph
RF Resource Factor	Resources	each activity uses no resources	each activity uses all resources
RS Resource Scarcity	Resources	minimum resource capacities	maximum resource capacities
CF Cardinality Factor	Period Subsets	each period subset comprises one period	each period subset comprises all periods
HF Horizon Factor	Period Subsets	planning horizon equals EFT_J	planning horizon equals upper bound
IF Interval Factor	Period Subsets	each period subset consists of one interval	each period subset consists of maximum number of intervals

Table 6: Design Parameters of ProGen/Π

Due to the complexity of the RCPSP/Π, not all of its instances are feasible. Hence, 96 instance clusters - defined by level combinations of the design parameters - were identified experimentally which consist of mostly feasible instances; a similar approach has been pursued by Böttcher et al. (1996b). Each benchmark instance set comprises ten instances per cluster, or 960 instances, for a total of 3,840 instances. All instances were attempted with the most effective of the truncated branch-and-bound algorithms described in Böttcher et al. (1996a). As the problem is strongly **NP**-equivalent, it is no surprise that the computation times required to solve harder instances increase exponentially; we hence employed a truncated version of this algorithm. To compensate for the fact that larger instances require more computation time, the CPU time limit was set to 5 (5, 10, 20) minutes for the instances with 10 (20, 30, 40) non-dummy activities, using an implementation in GCC, running under AIX on an IBM RS/6000 computer with 66 MHz clockpulse and 64 MB RAM. The corresponding results are summarized in Table 7.

Instance Set	Non-Optimally Solved	Optimally Solved	Feasibly Solved	Undecided	Proven Infeasible	Total
J10	39	901	940	11	9	960
J20	203	734	937	23	0	960
J30	181	757	938	22	0	960
J40	183	743	926	34	0	960
Total	606	3,135	3,741	90	9	3,840

Table 7: Characteristics of Instance Sets

Of course, in the following experiments only those instances were included which were feasibly solved by the respective scheduling algorithm. Also, in order to prevent already optimal schedules, which simply cannot be improved, from distorting the results, we had the algorithms check each initial schedule for optimality by comparing them against the reference solution from the truncated branch-and-bound algorithm.

4.2. Effectiveness

Results pertaining to the effectiveness of the algorithms are compiled in Tables 8 - 10 where Avg Improv (Instances, Improved, To Opt) refers to the average percentage improvement achieved over the initial schedule (the number of instances attempted, the number of schedules improved, the number of schedules improved to optimality). Note that the differing number of instances (and schedules) attempted depend on the different capabilities of the scheduling algorithms to construct feasible but non-optimal schedules.

Jobs		DRC/E	DRC/ES	DRS/ES	EST	MTS	RRD	SPT	TRS
10	Avg Improv	0.89%	1.01%	1.40%	1.38%	0.85%	1.56%	1.02%	1.55%
	Instances	162	160	158	161	122	161	146	171
	Improved	40	50	48	61	29	62	42	55
	To Opt	19	29	21	22	17	27	19	24
20	Avg Improv	0.95%	0.81%	0.65%	1.38%	1.37%	3.08%	1.60%	2.58%
	Instances	259	258	247	243	222	251	229	244
	Improved	111	88	84	105	81	121	93	119
	To Opt	45	36	31	40	34	35	38	33
30	Avg Improv	1.18%	1.49%	1.29%	2.79%	2.10%	8.21%	2.85%	8.68%
	Instances	215	211	218	195	186	225	193	227
	Improved	99	88	88	82	76	165	79	166
	To Opt	20	13	23	15	15	45	15	51
40	Avg Improv	2.35%	3.14%	9.60%	2.54%	1.99%	17.82%	2.43%	17.96%
	Instances	235	271	413	200	200	441	189	441
	Improved	134	156	318	71	61	399	69	404
	To Opt	21	51	190	8	9	257	9	250

Table 8: Effectiveness of Local Left-Shift Algorithm

Jobs		DRC/E	DRC/ES	DRS/ES	EST	MTS	RRD	SPT	TRS
10	Avg Improv	1.77%	1.77%	2.50%	2.10%	1.90%	2.86%	2.17%	2.98%
	Instances	162	160	158	161	122	161	146	171
	Improved	58	60	64	66	41	70	55	78
	To Opt	25	33	31	27	21	30	29	39
20	Avg Improv	1.13%	1.10%	0.92%	2.66%	2.18%	5.90%	2.64%	5.62%
	Instances	259	258	247	243	222	251	229	244
	Improved	102	97	89	116	86	142	102	137
	To Opt	48	42	34	46	34	48	43	48
30	Avg Improv	1.97%	1.63%	1.89%	4.13%	3.05%	11.62%	3.91%	12.09%
	Instances	215	211	218	195	186	225	193	227
	Improved	101	83	91	90	79	158	85	170
	To Opt	20	16	25	17	16	57	17	61
40	Avg Improv	2.65%	3.60%	9.87%	4.26%	3.30%	20.34%	4.14%	20.38%
	Instances	235	271	413	200	200	441	189	441
	Improved	125	154	316	82	73	409	78	409
	To Opt	21	54	201	10	11	280	11	277

Table 9: Effectiveness of Global Left-Shift Algorithm

Jobs		DRC/E	DRC/ES	DRS/ES	EST	MTS	RRD	SPT	TRS
10	Avg Improv	1.75%	1.73%	2.54%	2.16%	1.88%	2.83%	2.20%	2.96%
	Instances	162	160	158	161	122	161	146	171
	Improved	57	60	65	67	41	70	56	77
	To Opt	27	33	32	29	21	30	32	39
20	Avg Improv	1.14%	1.11%	0.87%	2.68%	2.24%	5.89%	2.67%	5.70%
	Instances	259	258	247	243	222	252	229	244
	Improved	103	99	89	115	87	144	102	138
	To Opt	48	41	34	46	34	48	44	49
30	Avg Improv	1.82%	1.62%	1.87%	4.11%	3.20%	11.62%	3.85%	12.14%
	Instances	215	211	218	195	186	225	193	227
	Improved	103	83	93	90	79	158	84	170
	To Opt	20	16	26	16	16	57	17	61
40	Avg Improv	2.60%	3.55%	9.87%	4.30%	3.23%	20.34%	4.14%	20.41%
	Instances	235	271	413	200	200	441	189	441
	Improved	124	154	313	80	73	408	78	409
	To Opt	22	54	201	11	11	281	11	277

Table 10: Effectiveness of Hybrid Left-Shift Algorithm

It comes as no surprise that the effectiveness of all our algorithms is greater on the schedules generated by the worst priority rules, as is evident from comparing e.g. the results for the best rules DRC/E and DRC/ES with those of the worst rule RRD. This result was expected and it is consistent with elementary insight as worse algorithms leave more room for further improvement than better one do. The more interesting result is that even the schedules produced by the best construction methods can still be improved by between 1.1% and 3.6% by the GLS and HLS algorithms. Even the application of mere local shifts allows to reap improvements between 0.8% and 3.1%. This picture is corroborated by the numbers of instances improved and improved to optimality. Whereas for the best rules between 35% and 57% of the eligible schedules are improved, and between 15% and 20% are improved to optimality, the corresponding percentages lie between 43% and 93% (between 19% and 64%) for the worst rule. For the LLS algorithm, the situation is similar, albeit of course on a lower level.

4.3. Efficiency

For all three algorithms, the computation times are given in Table 11. As these directly depend on the number of activities shifted and the number of iterations, for the GLS and the HLS algorithm also on the number of periods shifted, we provide the corresponding results along with the computation times. We refrain from detailing the results for the different priority rules since the respective differences are minor only; for all corresponding result variables, averages and standard deviations over all schedules are listed.

Jobs		*LLS*		*GLS*		*HLS*	
		Avg	StdDev	Avg	StdDev	Avg	StdDev
10	Jobs Shifted	4.96	0.45	5.82	0.51	5.75	0.45
	Periods Shifted	2.56	0.21	4.67	0.43	4.61	0.21
	Iter	3.90	0.26	2.51	0.07	2.50	0.26
	CPU	0.00	0.00	0.03	0.00	0.01	0.00
20	Jobs Shifted	12.93	1.45	13.76	1.48	13.73	1.45
	Periods Shifted	5.18	1.09	8.27	1.76	8.25	1.09
	Iter	6.85	0.94	2.76	0.15	2.75	0.94
	CPU	0.08	0.01	0.02	0.00	0.02	0.01
30	Jobs Shifted	22.73	2.54	23.39	2.31	23.39	2.54
	Periods Shifted	9.43	3.93	14.02	4.61	14.01	3.93
	Iter	10.26	1.71	2.99	0.05	2.98	1.71
	CPU	0.18	0.03	0.03	0.00	0.04	0.03
40	Jobs Shifted	33.93	4.89	34.54	4.48	34.55	4.89
	Periods Shifted	16.35	7.70	20.76	7.37	20.75	7.70
	Iter	10.94	1.11	2.78	0.21	2.76	1.11
	CPU	0.45	0.05	0.05	0.01	0.06	0.05

Table 11: Efficiency of the Left-Shift Algorithms

While theoretical analysis revealed the worst-case time complexity of the LLS to be lower than that of the other algorithms, Table 11 demonstrates that - at least on the instances tackled - its actual running time is higher than that of the other algorithms. One can conclude that here the computational effort to evaluate the LLS rules more than compensates the effort of checking a - here, not too large - number of periods for resource-feasibility.

Note also that for the HLS algorithm the values reported for Jobs Shifted, Periods Shifted, and Iter refer to the second stage only, in order to facilitate comparisons with the GLS algorithm. Clearly, the effect of the first stage of the HLS algorithm is relatively minor and does neither allow larger shifts nor more activities to be shifted.

Now, it is also apparent why we chose to explore the polynomial variant of selecting start times in the context of the LLS algorithm, and the pseudo-polynomial variant in that of the GLS and the HLS algorithms. As by the definition of global shifts the latter algorithms usually examine more start times, they require longer computation times, regardless of the selection variant used. Thus, the finding that the LLS algorithm with a polynomial selection of candidate start times takes actually longer than its counterparts using a pseudo-polynomial selection allows us to infer that - at least for the instances tackled - the pseudo-polynomial selection variant is the better choice. So, our experimental design frees us from having to run additional experiments on a LLS algorithm with the pseudo-polynomial approach to finding candidate start times, or a GLS or HLS algorithm with a polynomial approach. Note, however, that for large instances the 'curse of dimensionality' associated with larger-than-polynomial complexities may prevail nevertheless.

5. Summary

Starting from the finding that currently all known heuristics for the RCPSP/Π sample the space of feasible rather than the desirable space of active schedules, we have proposed several iterative improvement algorithms which use local and global shifts to improve heuristically constructed schedules. We have also shown that straightforward approaches to selecting candidate start times for shifting activities have pseudo-polynomial time complexities. We have developed a number of shift rules which analyze the schedules at hand and allow to bring these complexities down to a polynomial level. These theoretical complexity results are complemented by an experimental evaluation of the average-case performance, employing a test bed of 3,840 benchmark instances that has already been used in other studies on the RCPSP/Π. Summarizing the results, we can note that both the GLS and the HLS algorithm produce good results in terms of schedule makespan reduction. Even of the eligible schedules constructed by the best priority rule-based methods, between 35 and 57% are improved by between 1.1 and 3.6%, and between 15 and 20% are improved to optimality.

Regarding efficiency, it is interesting to note that - for the benchmark instances used in this study - the LLS algorithm, albeit constructed in a way as to ensure polynomial worst-case time complexity, is less efficient than its pseudo-polynomial companions. Indeed, the GLS algorithm shows the least computation times. Clearly, the planning horizons to be considered here are not large enough to let the pseudo-polynomial complexities of the latter become dominant. From these results, using the GLS algorithm as a post-processing procedure appears to be a sensible choice. For substantially larger instances, however, additional experimentation would have to verify whether the pseudo-polynomial time complexity induced by the way in which periods are checked would have to be replaced by a more evolved, polynomial version.

An always enticing question is whether research results can also be applied or extended to other problem settings. For the closely related classical resource-constrained project scheduling problem (RCPSP), construction methods are well-known for decades which always produce active (or semi-active) schedules, so the application of local or global left-shifts to schedules of RCPSP-instances would be a futile effort. It seems, however, worthwhile to circle out other scheduling problems where construction methods sampling the space of active or semi-active schedules are unknown. Developing post-processing procedures based on appropriately defined shift operations would allow to improve also the effectiveness of such methods.

Acknowledgements

Part of this work was conducted while the author held a teaching position at the Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel; the author is most grateful for the support, in particular of Peter Kandzia, that made this work possible. Also, the helpful comments of Andreas Drexl and the able research assistance of Lars Urbszat are gratefully acknowledged. Partial funding was granted by the Deutsche Forschungsgemeinschaft (German National Science Foundation) under Grant Dr 170/4-1.

Appendix

Let denote RK_{pn} ($1 \le p \le P$; $1 \le n \le N$) the remaining capacity of resource p on stage n. It is initialized by

$$RK_{p1} \leftarrow K_p \qquad (1 \le p \le P) \quad (27)$$

and updated dynamically according to

$$RK_{pn} \leftarrow RK_{p,n-1} - RD_{j,p,FT_j - d_j + 1} \qquad (1 \le p \le P;\ 2 \le n \le N) \quad (28)$$

Let denote MD_{jpt} the minimum (relevant) demand for any requested resource p incurred by starting j no earlier than in period t, i.e.

$$MD_{jpt} \leftarrow \min \{RD_{jpt'} \mid t \le t' \le LST_j\} \qquad (2 \le j \le J-1;\ p \in PR_j;\ EST_j+1 \le t \le LST_j+1) \quad (29)$$

and MDE_{jpt} the minimum (relevant) demand entailed for resource p by all successors of activity j when started in period t. Assuming $j \angle j'$, let us refer to the longest duration path from j to j' w.r.t. the precedence order $\angle$ by $LP_{jj'}$. Also, assuming that j is started in period t, updated earliest start times $EST_{j'jt}$ of its successors j' could be derived from

$$EST_{j'jt} \leftarrow \max \{EST_{j'},\ t + LP_{jj'} - 1\} \qquad (j \angle j';\ EST_j+1 \le t \le LST_j+1) \quad (30)$$

Then, MDE_{jpt} can be calculated from

$$MDE_{jpt} \leftarrow \sum_{j \angle j'} MD_{j',p,EST_{j'jt}} \qquad (2 \le j \le J-1;\ 1 \le p \le P;\ EST_j+1 \le t \le LST_j+1) \quad (31)$$

Finally, let denote SP_n the set of those resources which can be identified on stage n as potentially scarce. SP_n can be formally defined as

$$SP_n \leftarrow \{1 \le p \le P \mid \Sigma_{j \in S_n} k_{jp} \cdot d_j > RK_{pn}\} \qquad (1 \le n \le N) \quad (32)$$

References

BÖTTCHER, J., A. DREXL, AND R. KOLISCH (1996a), "A branch-and-bound procedure for project scheduling with partially renewable resources", Proceedings of the Fifth International Workshop on Project Management and Scheduling, Scientific Publishers OWN PAN, Poznan, pp. 48-51.

BÖTTCHER, J., A. DREXL, R. KOLISCH, AND F. SALEWSKI (1996b), "Project scheduling under partially renewable resource constraints", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 398, to appear in Management Science.

GLOVER, F., E. TAILLARD, AND D. DE WERRA (1993), "A user's guide to tabu search", Annals of Operations Research 41, pp. 3-28.

HAASE, K., J. LATTEIER, AND A. SCHIRMER (1997), "Course planning at Lufthansa Technical Training - Constructing more profitable schedules", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 442, to appear in Interfaces.

HAASE, K., J. LATTEIER, AND A. SCHIRMER (1998), "The course scheduling problem at Lufthansa Technical Training", European Journal of Operational Research 110, pp. 441-456.

JOHNSON, T.J.R. (1967), *An algorithm for the resource-constrained project scheduling problem*, unpublished Ph.D. thesis, Massachusetts Institute of Technology.

KELLEY, J.E. (1961), "Critical path planning and scheduling: Mathematical basis", Operations Research 9, pp. 296-320.

KELLEY, J.E. (1963), "The critical-path method: Resources planning and scheduling", in: *Industrial scheduling*, Muth, J.F. and G.L. Thompson (eds.), Prentice-Hall, Englewood Cliffs, NJ, pp. 347-365.

KOLISCH, R. (1996), "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation", European Journal of Operational Research 90, pp. 320-333.

KOLISCH, R. AND A. DREXL (1996), "Adaptive search for solving hard project scheduling problems", Naval Research Logistics 43, pp. 23-40.

KURTULUS, I.S. AND S.C. NARULA (1985), "Multi-project scheduling: Analysis of project performance", IIE Transactions 17, pp. 58-66.

NÜBEL, H. AND C. SCHWINDT (1997), "A classification of shifts, schedules, and objective functions in project scheduling", Report WIOR-509, Universität Karlsruhe, Germany.

SCHIRMER, A. (1999), *Project scheduling with scarce resources - Models, methods, and applications*, Kovac, Hamburg.

SCHIRMER, A. AND A. DREXL (1997), "Allocation of partially renewable resources - Models and applications", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 455.

SCHIRMER, A. AND S. RIESENBERG (1998), "Class-based control schemes for parameterized project scheduling heuristics", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 471.

SPRECHER, A., R. KOLISCH, AND A. DREXL (1995), "Semi-active, active, and non-delay schedules for the resource-constrained scheduling problem", European Journal of Operational Research 80, pp. 94-102.

SPRECHER, A., S. HARTMANN, AND A. DREXL (1997), "An exact algorithm for project scheduling with multiple modes", Operations Research Spektrum 19, pp. 195-203.

TALBOT, F.B. AND J.H. PATTERSON (1978), "An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems", Management Science 24, pp. 1163-1174.

THESEN, A. (1977), "Measures of the restrictiveness of project networks", Networks 7, pp. 193-208.