Schirmer, Andreas



# Adaptive control schemes applied to project scheduling with partially renewable resources

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 520

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 520

# Adaptive Control Schemes Applied to Project Scheduling With Partially Renewable Resources

Schirmer

**Contents**

**Abstract:** For most computationally intractable problems there exists no heuristic that is equally effective on all instances. Rather, any given heuristic may do well on some instances but will do worse on others. Indeed, even the 'best' heuristics will be dominated by others on at least some subclasses of instances. It thus seems worthwhile to identify - for each instance anew - a heuristic that is appropriate for the instance at hand, instead of applying always the same algorithm, regardless of its suitability. Adaptive control schemes attempt to do this with iterative algorithms by dynamically exploiting the experience gained in earlier iterations to guide the underlying algorithm in later iterations. In a recent study, several adaptive control schemes have been evaluated on one of the most fundamental scheduling problems, the resource-constrained project scheduling problem (RCPSP). With this contribution, we intend to complement that research by results on a substantially more general problem, viz. the RCPSP under partially renewable resources. This new resource concept allows to formulate a variety of temporal and logical relations between scheduling objects, and is thus of interest in many practical settings pertaining to time-tabling and manpower scheduling.

**Keywords:** PROJECT SCHEDULING; PARTIALLY RENEWABLE RESOURCES; HEURISTICS; LOCAL SEARCH; CONTROL SCHEMES

## 1. Introduction

There is no doubt that trying to find 'the best heuristic' for a computationally intractable problem is intellectually appealing. However, for most such problems researchers have come to believe that there simply exists no heuristic which is best on all instances. If a heuristic is characterized as best, this usually means that it performs good on the majority of instances but leaves a minority on which other heuristics do better. For the field of priority rule-based scheduling, for example, recent experimental results suggest that some rules perform best on certain classes of instances while other rules do so on other instance classes, consequentially no single rule will perform best on all possible instances of a problem. This view is supported by Wolpert, Macready (1995) whose no-free-lunch-theorem states the same for neighborhood search algorithms in general.

For the most widely used form of fast scheduling methods, viz. priority rule-based construction ones, this implies that restricting oneself to merely one rule means to waive the capability to address the specifics of particular instances. Attempts to avoid this limitation have been made with the design of composite (or combinatorial) rules (cf. Barman 1997 and the literature cited therein), which combine simple priority rules in a fixed and predetermined way. In this contribution, we investigate another way of tackling that issue, viz. the design of *adaptive control schemes* which dynamically combine algorithms as appropriate. In essence, we aim for algorithms with a 'learning' capability, i.e. here the ability to extricate good rule combinations from an uninformed compilation of good and bad rules by gathering - for each instance attempted anew - instance-specific knowledge of which combinations work well and which do not. This concept therefore provides a unifying framework allowing to clarify commonalities and differences between certain algorithmic approaches.

Although the very name 'adaptive control scheme' has, to the best of our knowledge, not been used before, algorithmic approaches constituting such control schemes have been around for some time, being used in various settings such as lotsizing (Kimms 1996; Haase 1996) and course scheduling (Haase et al. 1997, 1998). Two such schemes have been investigated experimentally in Schirmer (1998), analyzing their performance on the resource-constrained project scheduling problem (RCPSP). Of these, a local search-based control scheme (LSCS) showed the best performance. In this contribution, we apply the LSCS to a more general problem, viz. the RCPSP under partially renewable resources (RCPSP/Π). Based upon this, we propose a new algorithmic approach to priority-rule based scheduling which proceeds in two stages. The first, deterministic stage uses the deterministic LSCS to identify, *separately for each attempted instance*, a well-performing combination of priority rules. The second, randomized stage uses the best known combination of scheduling and random sampling scheme for the problem to run the composite rule found in the first stage. This approach allows to exploit the well-known benefits of sampling while enabling the algorithm to combine priority rules which are especially suited for the instance at hand.

Our computational results on both problems show that this yields an algorithm competitive with all but the best state-of-the-art construction heuristics. In addition, adaptive control schemes enjoy two important advantages over these algorithms, which facilitate the researcher's task of designing good priority rule-based heuristics: First, by design they are robust to changes in the instances attempted, thus obviating the need of expertise on the characteristics of instances. Second, they are - at least moderately - robust to changes in the rules employed, thus lessening the importance of expertise on the characteristics of such rules for the problem at hand. Expertise on both areas would otherwise be necessary to identify good algorithms, and usually extensive experimentation is required to acquire it (Schirmer 1998; Schirmer 2000). Third, they are simple and easy to implement. Therefore, adaptive control schemes provide a commendable approach in situations where little knowledge is available on what constitutes a good rule or what makes an instance easy or difficult to solve.

The remainder of this work is structured as follows. In Section 2 we introduce the concept of partially renewable resources and the RCPSP/Π as far as they are needed here. The requisite algorithmic components, viz. scheduling scheme, simple and composited priority rules, random sampling scheme, and bounding rules are covered in Section 3. Section 4 recaps a taxonomy of control schemes, based upon which we describe the adaptive approaches to be explored. Section 5 details the experimentation carried out and analyzes the respective results. A short summary, along with some conclusions, in Section 6 wraps up this work.

## 2. Project Scheduling under Partially Renewable Resources

In this section, we will briefly state the resource-constrained project scheduling problem under partially renewable resources (RCPSP/Π), giving a formal presentation of the problem and discussing several important specifics of this new resource concept.

### *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 $\Pi_p$ 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 classical resource-constrained project scheduling problem (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 $\Pi_p$ are assumed to be subsets of the set $T$.

| 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 $\Pi_p$ |
| P | Number of partially renewable resources, indexed by p |
| $\Pi_p$ | 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*

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 $\Pi_p$ that the total consumption of p by

all activities performed in that period subset does not exceed the total capacity of p within $\Pi_p$, respects the partial order $\angle$, and minimizes the total project length.

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 $j \angle J$ $(2 \le j \le J\text{-}1)$. The fictitious activities 1 and J are called *dummy activities*, meaning that $d_1 = d_J = 0$ and $k_{1p} = k_{Jp} = 0$ $(1 \le p \le P)$. 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 $P_j$ $(1 \le j \le J)$ the set of all *immediate* predecessors of activity j w.r.t. $\angle$. Second, for each activity j $(1 \le j \le J)$ 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 $y_j$ $(1 \le j \le J)$ denoting the period in which activity j is completed. In addition, we denote by $A_t$ $(1 \le t \le T)$ the set of all activities which are processed in period t (Talbot, Patterson 1978) and thus may consume resources. Then the RCPSP/$\Pi$ can be described in the following way:

Minimize

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

subject to

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

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

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

Minimization of the objective function (1) enforces the earliest possible completion of the last activity J and thus 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. The optimization problem is shown to be strongly **NP**-equivalent and the corresponding feasibility problem to be strongly **NP**-complete in Schirmer, Drexl (1997) where also a number of alternative model formulations as well as their respective advantages and disadvantages are discussed.

### 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.*

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 (Kelley 1963). 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).

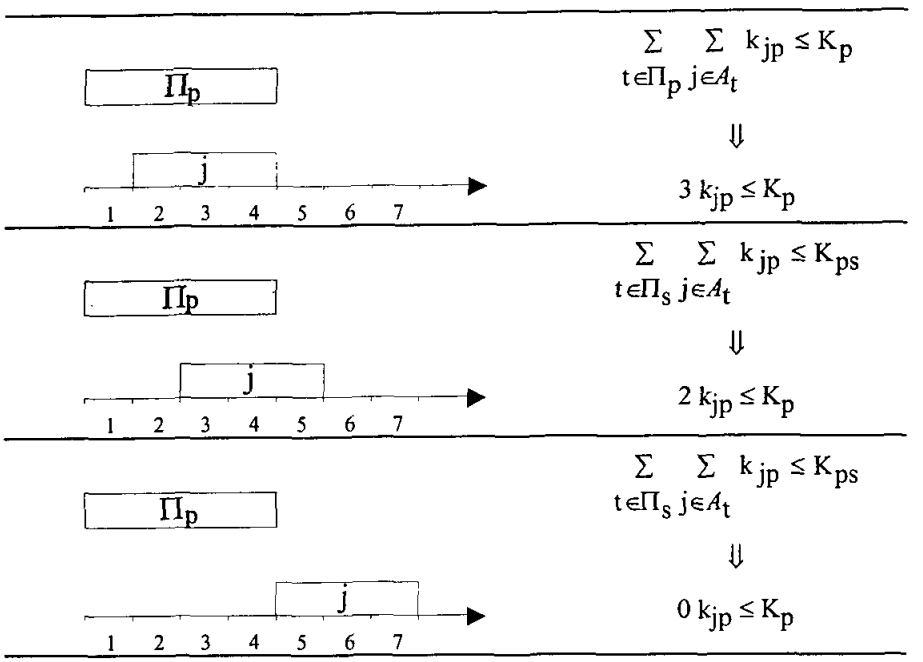


*Figure 1: Total Activity Demand for Partially Renewable Resources*

One fundamental consequence of this property is that delaying some activities may turn out beneficial in terms of solution quality. As an example consider an instance of the RCPSP/Π where $J = 2$, $d_1 = d_2 = 3$, $P = 1$, $T = 6$, $\Pi_p = \{1, 2\}$, $k_{1p} = 2$, $k_{2p} = 1$, $K_p = 5$, $\angle$ empty and confer to the schedules in Figure 2: while schedule 1 starts as soon as possible, it is actually better to delay activity 1 by one period since doing so frees enough of resource p that both activities can be processed in parallel, reducing the makespan of schedule 2 to the optimal four periods.

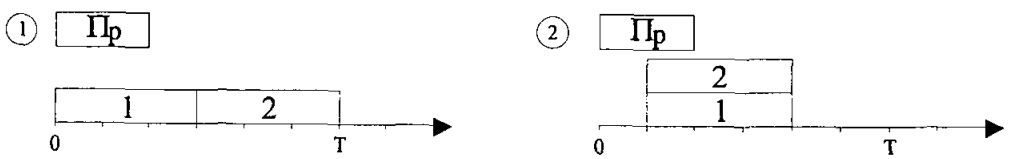


*Figure 2: On the Benefit of Delaying RCPSP/Π-Activities*

## 3. Components of Priority Rule-Based Algorithms

*Priority rule-based methods* consist of at least two components, viz. one (or more) scheduling schemes and one (or more) priority rules. A *scheduling scheme* determines how a schedule is constructed, building feasible *full schedules* (which cover all activities) by augmenting *partial schedules* (which cover only a proper subset of the activities) in a stage-wise manner. On each stage, the scheme determines the set of all activities which are currently eligible for schedul-

ing. In this, *priority rules* serve to resolve conflicts where more than one activity could be feasibly scheduled. Also, such methods may be complemented by *bounding rules* which serve to improve their performance.

### *3.1. Scheduling Scheme*

In the following, we describe an adaptation of the well-known *serial scheduling scheme* (SSS) for the RCPSP to the specifics of the RCPSP/Π (SSS/Π). The motivation to use a serial scheme was twofold. First, recall that the capability to delay some activities from their earliest feasible start time is crucial to finding high-quality schedules. This capability is relatively easy to add to the SSS whereas introducing this idea to its well-known parallel counterpart, which essentially relies on scheduling activities - once selected - as soon as possible, requires some substantial bending. Second, preliminary experiments showed the SSS/Π to outperform several other scheduling schemes developed for the RCPSP/Π.

The SSS partitions the set of activities into *scheduled*, *eligible,* and *ineligible* activities. 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* otherwise. 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 *decision set* $D_n$. 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 its earliest feasible start time. Then j is moved from $D_n$ to $S_n$ which may render some ineligible activities eligible if now all their predecessors are scheduled. The scheme terminates either on stage J when all activities are scheduled (feasible solution found) or if no further selection is possible (no feasible solution found).

In its common form, the decision set of the SSS contains all precedence-feasible, unscheduled activities, i.e. those whose predecessors have already been scheduled. A selected activity is then scheduled to its earliest resource-feasible start period. Yet, for the RCPSP/Π, the above example has demonstrated that delaying certain activities behind these periods may be beneficial. In order to allow activities to be started in later periods, here $D_n$ is defined as to comprise all pairs of an activity j and a period t where j is precedence- and resource-feasibly schedulable in t; note that for the dummy activities only the first such period is considered.

To facilitate the subsequent formal description, some additional notation will prove helpful. For each scheduled activity j let denote $FT_j$ the finish time of j as determined by the schedule. Let also 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

$$RK_{p1} \leftarrow K_p \qquad (1 \le p \le P) \quad (7)$$

and updated dynamically according to

$$RK_{pn} \leftarrow RK_{p,n-1} - RD_{j,p,FT_j-d_j+1} \qquad (1 \le p \le P;\ 2 \le n \le N) \quad (8)$$

Let denote $D'_n$ the set of activities currently eligible for scheduling, i.e.

$$D'_n \leftarrow \{1 \le j \le J \mid j \notin S_n \wedge P_j \subseteq S_n\} \qquad (1 \le n \le N) \quad (9)$$

Then, using $EPST_j$ to denote the earliest precedence-feasible start time of activity j, i.e.

$$EPST_j \leftarrow \max\{FT_i \mid i \angle j\} \qquad (j \notin S_n) \quad (10)$$

---

*Initialization*

$FT_1 \leftarrow 0$
$S_n \leftarrow \{1\}$
$n \leftarrow 1$
FEASIBLE $\leftarrow$ TRUE
**for each** $(1 \le p \le P)$
  $RK_{pn} \leftarrow K_p$

*Execution*

**while** $|S_n| < J$ **and** FEASIBLE
  **calculate** $D_n$ **according to** (12)
  **if** $D_n = \varnothing$
    FEASIBLE $\leftarrow$ FALSE[1]
  **else if** $D_n = \{(j, t)\}$
    $(j^*, t^*) \leftarrow (j, t)$
  **else**
    **select** $(j^*, t^*)$ **according to** (13)
  $FT_{j^*} \leftarrow t^* + d_{j^*}$
  $S_n \leftarrow S_n \cup \{j^*\}$
  **for each** $(1 \le p \le P)$
    **update** $RK_{pn}$ **according to** (8)
  $n \leftarrow n+1$
**endwhile**

*Output/Result*

If FEASIBLE = TRUE, a feasible solution has been found and for each activity $j \in S_n$, $FT_j$ denotes the period in which j is finished; otherwise no feasible schedule has been found. ■

---

*Table 2: Serial Scheduling Scheme SSS/Π*

[1] A more efficient implementation would be to check during the computation of $D_n$ whether there exists an activity $j \in D'_n$ with $W_{jn} = \varnothing$. If so, set FEASIBLE $\leftarrow$ FALSE.

the set $W_{jn}$ of all feasible periods for activity j on stage n can be determined from

$$W_{jn} \leftarrow \{EPST_j + 1 \le t \le LST_j + 1 \mid RD_{jpt} \le RK_{pn}\ (1 \le p \le P)\} \quad (j \notin S_n;\ 1 \le n \le N) \quad (11)$$

Now, finally, $D_n$ can be defined as

$$D_n \leftarrow \{(j, t) \mid j \in D'_n \wedge t \in W_{jn}\} \quad (1 \le n \le N) \quad (12)$$

Having laid this groundwork, the adapted serial scheduling scheme SSS/Π can be formulated as done in Table 2 (the details of selecting a job according to (13) are discussed below).

### *3.2. Simple Priority Rules*

Priority rules allow to resolve conflicts between candidates competing for the allocation of scarce resources. In situations where the decision set contains more candidates than can be feasibly scheduled, priority values are calculated from numerical measures which are related to properties of the activities, the complete project, or the incumbent partial schedule. Formally, any priority rule can be specified by two components: One is a mapping v: $D_n \rightarrow \mathbb{R}_{\ge 0}$ that accords a numerical measure v(d) to each candidate in the decision set. The other is a dichotomical parameter extr $\in$ {max, min} which specifies which extremum of the priority values determines the candidate to be selected (Kolisch 1995, pp. 83-84). Ties can be broken arbitrarily, e.g. by smallest activity index (as done in the sequel) or randomly. Then, any such (deterministic) selection of a candidate d (as used above in Table 2 where d = (j*, t*)) can be expressed as

$$d^* \leftarrow \min \{d \in D_n \mid v(d) = \text{extr}\ \{v(d') \mid d' \in D_n\}\} \quad (13)$$

Elementary insight into the problem at hand points at two fundamental considerations for selecting a set of simple priority rules from which the composite rules will be constructed. First, bowing to the objective function, we wish to construct short schedules which minimize total project duration. So, some priority rules should reflect the desirability of scheduling activities early to get them "done and out of the way" (Alvarez-Valdés, Tamarit 1989), rather than delaying them and their successors into later periods. Second, scarce resources may delay certain activities into noncapacitated periods. Thus, certain rules should measure the resource-related importance of activities, e.g. in terms of its resource usage or the scarcity of the resources used. In the sequel, we therefore describe several simple precedence- and resource-based rules.

Before we can introduce these rules, some formal preparations are in place. For each activity j, let denote $P_j$ the set of resources requested by j, i.e.

$$P_j \leftarrow \{1 \le p \le P \mid k_{jp} > 0\} \quad (1 \le j \le J) \quad (14)$$

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.

$$MD_{jpt} \leftarrow \min \{RD_{jpt'} \mid t \leq t' \leq LST_j\} \quad (2 \leq j \leq J\text{-}1;\ p \in P_j;\ EST_j+1 \leq t \leq LST_j+1) \quad (15)$$

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

$$EST_{j'jt} \leftarrow \max \{EST_{j'},\ t + LP_{jj'} - 1\} \qquad (j \angle j';\ EST_j+1 \leq t \leq LST_j+1) \quad (16)$$

Now, $MDE_{jpt}$ can be calculated from

$$MDE_{jpt} \leftarrow \sum_{j \angle j'} MD_{j',p,EST_{j'jt}} \qquad (2 \leq j \leq J\text{-}1;\ 1 \leq p \leq P;\ EST_j+1 \leq t \leq LST_j+1) \quad (17)$$

Finally, if we let $TD_{pn}$ denote the total demand for resource p on stage n of the algorithm, calculated over all activities still unscheduled, i.e.

$$TD_{pn} \leftarrow \Sigma_{j \notin Sn}\, k_{jp} \cdot d_j \qquad (1 \leq p \leq P;\ 1 \leq n \leq N) \quad (18)$$

then the set $SP_n$ of those resources which can be identified on stage n as potentially scarce may be defined as

$$SP_n \leftarrow \{1 \leq p \leq P \mid TD_{pn} > RK_{pn}\} \qquad (1 \leq n \leq N) \quad (19)$$

| Extremum | Measure | Definition | Static vs. Dynamic | Local vs. Global |
|---|---|---|---|---|
| MAX | $DRC/ES_{it}$ | $\Sigma_{p \in SP_n} (RK_{pn} - RD_{jpt} - MDE_{jpt})$ | D | G |
| MAX | $DRC/E_{jt}$ | $\Sigma_p (RK_{pn} - RD_{jpt} - MDE_{jpt})$ | D | G |
| MIN | $DRS/E_{jt}$ | $\Sigma_p (RD_{jpt} + MDE_{jpt}) / RK_{pn}$ | D | G |
| MIN | $DRS/ES_{it}$ | $\Sigma_{p \in SP_n} (RD_{jpt} + MDE_{jpt}) / RK_{pn}$ | D | G |
| MAX | $MTS_j$ | $\lvert \{j' \mid j \angle j'\} \rvert$ | S | L |
| MIN | $EFT_j$ | $EFT_j$ | S | L |
| MIN | $TRS/E_{jt}$ | $\Sigma_p (RD_{jpt} + MDE_{jpt}) / K_p$ | S | G |
| MIN | $TRS/ES_{jt}$ | $\Sigma_{p \in SP_n} (RD_{jpt} + MDE_{jpt}) / K_p$ | S | G |
| MIN | $RRD/E_{jt}$ | $\Sigma_p (RD_{jpt} / (k_{jp} \cdot d_j) + \Sigma_{j \angle j'} MD_{j',p,EST_{j'}} / (k_{j'p} \cdot d_{j'}))$ | S | G |
| MIN | $TRS_{jt}$ | $\Sigma_p RD_{jpt} / K_p$ | S | L |

*Table 3: Simple Priority Rules - Definition and Classification*

Using this notation, the rules can be defined as in Table 3 where also a classification in terms of several straightforward criteria is given. The decision to select these rules draws upon the study in Schirmer (1999, pp. 181-191) where a total of 32 rules for the RCPSP/Π were evaluated. We have selected the best and the worst four rules from this sample, to allow to evaluate the performance of the LSCS in the presence of suited as well as unsuited rules.

### *3.3. Parameterized Composite Priority Rules*

As we aim at minimizing the project makespan, it is straightforward that certain (precedence-based) rules perform better on less capacitated instances while other (resource-based) ones do so on instances with scarce resources. It would therefore be unreasonable to expect one particular rule to outperform all its companions on all problem instances; we will come back to this issue below. In addition, some rules may accord similar or even identical priority values to several candidate activities, especially so when scheduling large projects; therefore the discriminative potential of simple rules may be rather limited in certain situations[2]. This insight motivates the concept of composite rules, which are made up from several simple priority rules. Such rules combine several numerical measures, each reflecting information about the desirability to select a specific candidate, into one priority value. It is also noteworthy that experimental results seem to suggest that good composite rules need not neccessarily be made up from good simple rules: for the RCPSP Ulusoy, Özdamar (1989) report that, although the rules MAX MIS (most immediate successors) and MAX TRS perform poorly as individuals, the weighted combination WRUP performs rather well. In other words, an appropriately balanced combination of different priority rules may do better than each of its parts.

Let I denote the number of priority rules to be combined. Let also for each activity $j \in D_n$ denote $v_i(j)$ the priority value accorded by rule i, and let finally for each rule i be $\alpha_i \in [-1, 1]$ an (exponential) control parameter or weight assigned to rule i. Now, we can define a composite rule in general as

$$v(j) \leftarrow \sum_{i=1}^{I} v_i(j)^{\alpha_i} \qquad \text{extr} = \max \quad (j \in D_n) \quad (20)$$

Obviously, for I = 1 this definition includes the special case of a simple rule. For each priority rule i, the corresponding control parameter $\alpha_i$ allows to vary the influence of the rule as well as whether candidates with high or with low priority values are to be preferred. On one hand, $\alpha_i \in (0,1]$ implies extr = max and increasing values tend to pronounce the differences between the candidate activities; on the other hand, $\alpha_i \in [-1, 0)$ implies extr = min and increas-

---

[2] "In order to exercise purported logic relative to a specific criterion, a heuristic scheduling rule must be able to discriminate among activities." (Patterson 1976, p. 97).

ing values tend to reduce the differences between the activities. For $\alpha_i = 0$, all candidates receive the same priority value. In our implementation, ties are broken by activity index j first, followed by period index t second.

It stands to reason that combining rules with different domains may overemphasize the influence of some rules. Suppose for instance that one rule draws priorities from {0,...,1} while another draws from {1,...,100}; a composite rule using these with identical weights will essentially behave like the latter. This imbalance of scales can either be adjusted explicitly by normalizing or scaling the priorities or implicitly by adjusting the weights (Whitehouse, Brown 1979). Following the former approach, we counter this effect by scaling the priority values to the interval [0,1] by

$$v'_i(j) \leftarrow \begin{cases} 1 - \dfrac{v_i(j)}{\max\left\{v_i(j') \middle| j' \in D_n\right\}} & \text{if } extr_i = \min \\ \dfrac{v_i(j)}{\max\left\{v_i(j') \middle| j' \in D_n\right\}} & \text{if } extr_i = \max \end{cases} \qquad (1 \le i \le I;\ j \in D_n) \quad (21)$$

If the denominator is zero, the priorities remain unchanged. Note that equation (21) transforms the priority values of all rules i with $extr_i = \min$ to values with $extr_i = \max$, so it suffices to let $\alpha_i \in [0,1]$. We thus compose the individual priority values only after having scaled them, i.e.

$$v(j) \leftarrow \sum_{i=1}^{I} v'_i(j)^{\alpha_i} \qquad extr = \max \quad (j \in D_n) \quad (22)$$

If a value of $v'_i(j)$ is zero, then $v'_i(j)^{\alpha_i}$ is set to zero. We should point out that we have also tried the following, multiplicative variant of composing the individual priorities. Yet, as it yielded noticeably worse results, we use the exponential one in the sequel.

$$v(j) \leftarrow \sum_{i=1}^{I} \alpha_i \cdot v'_i(j) \qquad extr = \max \quad (j \in D_n) \quad (23)$$

Note that, given a scheduling scheme and a set of I priority rules, each control parameter vector $\vec{\alpha} = (\alpha_1,...,\alpha_I)$ fully specifies one scheduling algorithm. Since each algorithm can be conceived as a mapping from the problem to the solution space, it is straightforward to regard any $\vec{\alpha}$-vector as encoding a specific solution of the instance tackled. Hence, for any algorithm that employs composite rules, the parameter space $[0,1]^I$ can be said to represent that subspace of the solution space that is sampled by the algorithm.

### *3.4. Random Sampling Scheme*

The concept of regrets, well-known from the field of decision theory, was introduced to randomized algorithms in terms of the *regret-based biased random sampling* scheme (RBRS; cf. Drexl 1991; Drexl, Grünewald 1993). In an extensive study, Kolisch (1996) found the scheme to dominate all other randomization schemes proposed so far. We use a recent modification of this scheme (MRBRS) that outperforms even the RBRS (Schirmer 1999, pp. 52-53; Schirmer 2000).

The regret value of a candidate d measures the worst-case consequence that could possibly result from selecting another candidate. Let denote $V(D_n)$ the set of priority values of all candidates in a decision set $D_n$. Then, the regrets are computed as

$$v'(d) \leftarrow \begin{cases} \max V(D_n) - v(d) & \text{iff extr} = \min \\ v(d) - \min V(D_n) & \text{iff extr} = \max \end{cases} \qquad (d \in D_n) \quad (24)$$

Now, let denote $V'(D_n)^+$ the set of all positive transformed priorities of the candidates in a decision set $D_n$, i.e.

$$V'(D_n)^+ \leftarrow \{v'(d) \mid d \in D_n \wedge v'(d) > 0\} \qquad (25)$$

The priorities are then modified by

$$v''(d) \leftarrow (v'(d) + \varepsilon_n)^\alpha \qquad (d \in D_n) \quad (26)$$

where $\alpha \in \mathbb{R}_{\geq 0}$ while $\varepsilon_n$ is determined from

$$\varepsilon_n \leftarrow \begin{cases} \min V'(D_n)^+ / 10 & \text{iff } (\exists d \in D_n)\, v'(d) > 0 \\ 1 & \text{otherwise} \end{cases} \qquad (27)$$

$\varepsilon_n$ guarantees v''(d) to be nonzero; otherwise at least one candidate in each decision set could never be selected, an undesirable consequence in the presence of scarce resources or tight time windows. The first alternative of $\varepsilon_n$ in (27) tends to keep its influence small whenever some transformed priorities are zero and some are not. The second alternative applies only if all candidates in the decision set have zero priorities such that $\varepsilon_n$ would be undefined; since then all priorities are identical, adding an arbitrary constant will give all candidates the same probability of being selected. $\alpha$ allows to diminish or enforce the differences between the modified priorities for $\alpha<1$ or $\alpha>1$, respectively. Note that for $\alpha = 0$ the selection process becomes pure random sampling, since all candidates will share the same probability of being selected. On the other extreme, for $\alpha = \infty$ the process behaves deterministic: since with increasing $\alpha$ the difference between the highest and the second-highest modified priority increases, the probability of the highest-prioritized candidate being selected converges to one.

Having determined these values, the selection probabilities are derived from

$$p(d) \leftarrow \frac{v''(d)}{\sum_{d' \in D_n} v''(d')} \qquad (d \in D_n) \quad (28)$$

The first iteration is always run in a deterministic manner to tighten the time windows early on, in order to alleviate the adverse effect of the large initial decision set cardinalities on algorithmic efficiency.

### *3.5. Bounding Rules*

Although bounding rules are mostly used in conjunction with exact methods, serving to speed these up when applied to (strongly) **NP**-hard or **NP**-equivalent problems, they can also be put to good use with heuristics. In their plain vanilla version, iterative construction algorithms, such as sampling methods, cannot utilize experience from earlier iterations, so each iteration starts from scratch. In the worst possible case, the optimum is found in the first iteration - and in all subsequent iterations as well. To avoid this waste of effort, bounds can be employed to restrict the solution space searched or to stop searching it when no better solution may be found.

We employed three bounds which we briefly discuss in the sequel. For details we refer the reader to Schirmer (1999, pp. 54-56) where the bounds are discussed more fully.

**General precedence-based lower bound (GPLB)** This bound is generally applicable whenever a feasible full schedule has been found. It is based upon relaxing the resource constraints, so a lower bound on the project makespan is determined as the length of a critical path in the project network. Whenever $FT_J = EFT_J$ holds, the whole run can be terminated. In this case, the heuristically derived upper bound $FT_J$ on the makespan equals the precedence-based lower bound $EFT_J$ such that the schedule at hand is optimal. ■

**Time window bound (TWB)** In the form presented here, this bound can be employed only under the SSS/Π. Let denote $\underline{FT}_J$ the objective function value of the incumbent best solution. Whenever a feasible solution with a better value of $FT_J < \underline{FT}_J$ is found, the $LST_j$-values of all activities are decremented by

$$LST_j \leftarrow LST_j - \underline{FT}_J + FT_J - 1 \qquad (1 \le j \le J) \quad (29)$$

Then terminate the whole run whenever there exists an activity $j \in D'_n$ which would have to be started outside its updated time window, i.e.

$$EST_j > LST_j \qquad (30)$$

Note that by construction of the bound any newly constructed schedule will be better than the previous one as its makespan must be smaller than the previous makespan[3]. Now, if above condition holds, no further schedule can be built, so the previous schedule is actually optimal. This bound may be seen as transforming, by means of tightening time windows, the issue of improving upon a known solution into one of achieving feasibility of the solutions constructed. Notice that this bound can be conveniently employed by testing whether $EST_J > LST_J$. ■

The tightening of the time windows gains particular importance in early iterations where the initial time windows may be relatively large; also it benefits the computational effort to determine the MDE-values if GRFB (to be discussed below) is used.

The feasibility bound used (Böttcher et al. 1999) is motivated by the fact that the relevant resource demand of an activity may vary depending on its start time. The approach consists of calculating lower bounds on the demand that all unscheduled activities place on the resources and checking these against their remaining capacities: if there exists a resource for which this lower bound exceeds the remaining capacity, the current partial schedule cannot be extended to a feasible solution. Before we introduce the corresponding bounding rule, we need some formal preparations. Let, for each activity j, denote $P_j$ the set of resources requested by j, i.e.

$$P_j \leftarrow \{1 \le p \le P \mid k_{jp} > 0\} \qquad (1 \le j \le J) \quad (31)$$

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.

$$MD_{jpt} \leftarrow \min \{RD_{jpt'} \mid t \le t' \le LST_j\} \quad (2 \le j \le J\text{-}1;\ p \in P_j;\ EST_j\text{+}1 \le t \le LST_j\text{+}1) \quad (32)$$

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

$$EST_{j'jt} \leftarrow \max \{EST_{j'},\ t + LP_{jj'} - 1\} \qquad (j \angle j';\ EST_j\text{+}1 \le t \le LST_j\text{+}1) \quad (33)$$

Now, $MDE_{jpt}$ can be calculated from

$$MDE_{jpt} \leftarrow \sum_{j \angle j'} MD_{j',p,EST_{j'jt}} \qquad (2 \le j \le J\text{-}1;\ 1 \le p \le P;\ EST_j\text{+}1 \le t \le LST_j\text{+}1) \quad (34)$$

---

[3] Such approaches, which iteratively impose tighter bounds and then try to refute them, have recently been named a destructive improvement technique (Klein, Scholl 1999).

**General resource-based feasibility bound (GRFB)** Let denote $j_n$ the highest-numbered activity, i.e. the last one w.r.t. to the precedence order, in the partial schedule $S_n$ and $t_n$ the start period assigned to $j_n$. Then terminate the current iteration if there exists a resource p where the minimum demand entailed by all unscheduled activities exceeds the remaining capacity, i.e.

$$RK_{pn} < MDE_{j_n,p,t_n} \tag{35}$$

because any extension of $S_n$ to a full schedule will be resource-infeasible. ■

This bound can easily be integrated into the SSS/Π by appropriately restricting the set $W_{jn}$ of feasible periods defined in (11) to

$$W_{jn} \leftarrow \{EPST_j + 1 \le t \le LST_j + 1 \mid RD_{jpt} + MDE_{jpt} \le RK_{pn}\ (1 \le p \le P)\}$$

$$(j \notin S_n;\ 1 \le n \le N) \tag{36}$$

This implementation has the additional benefit of restricting the cardinality of the decision set which also tends to increase algorithmic efficiency.

## 4. Control Schemes

Parameterized algorithms possess (one or more) *control parameters* which allow to direct the way in which they proceed. While these may be understood to encompass numerical parameters only, we adopt a broader view and include also the choice of certain algorithmic components, such as priority rules, scheduling and sampling schemes, in the concept. To algorithmic schemes which govern the instantiation of the control parameters we refer as *control schemes*. Although this very name has, to the best of our knowledge, not been used before, we maintain that some kind of control scheme is always present whenever parameterized methods are used. In the sequel, we propose a taxonomy of control schemes, distinguishing between classical and adaptive forms.

### *4.1. Classical Control Schemes*

#### *4.1.1. Fixed Control*

The most simple class of control schemes might be called *fixed control schemes* (FCS), in that the values or the value sequences used to instantiate the control parameters are prescribed in advance and for all instances alike. As an example, consider a numerical control parameter $\alpha \in \mathbb{N}$ of an algorithm to be run for three iterations. Under a FCS, $\alpha$ could e.g. be instantiated thrice by the same value or by a different value in each iteration. These precepts will, of

course, reflect the results of previous experimentation, having been found to produce the best results on average over all test instances used. Exemplary applications of FCS can be found in Kolisch, Drexl (1996), Schirmer, Riesenberg (1997), Böttcher et al. (1999).

#### *4.1.2. Class-Based Control*

Another approach is motivated by the observation that for most computationally intractable problems there is no algorithm which performs best for all instances. Thus, as the prescriptions of FCS apply to all instances alike, they are unable to take into account the specifics of particular instances. This observation has already been made, among others, by Davis, Patterson (1975). Therefore, a more refined class of schemes, to which we refer as *class-based control schemes* (CCS), proceeds by instantiating the control parameters depending on some characteristics of the instances attempted. Such schemes utilize a partition of the problems instances into equivalence classes and prescribe the application of specific parameter instantiations for each of the classes. Kolisch, Drexl (1996) propose a control scheme which selects the scheduling scheme to be applied according to two quantities: the number of iterations to be performed and the resource strength of the instance. Other examples of CCS can be found in Kimms (1998) and Schirmer; Riesenberg (1998). Also these schemes are based upon extensive experimentation to develop sufficient insight into the influence of the parameters on the algorithms' performance. Yet, the individual performance of an algorithm on a given instance may still not be reflected by its average performance on the corresponding equivalence class.

### *4.2. Adaptive Control Schemes*

#### *4.2.1. Motivation and Concept*

While the instantiation of control parameters reflects the results of previous experimentation, these results usually represent best averages, either over all test instances or over some subset of these; rarely will the resulting values be the most appropriate ones for each and every possible instance of the problem at hand. For the related RCPSP this has been shown in Schirmer (1998) where a number of simple priority rules were run deterministically under the SSS. On one hand, even the best rule ranked best on only 68% of the instances. On the other hand, even the worst rules ranked best on 7% of the instances, on some of these they were even the only ones to do so. It stands to reason that this kind of result extends to other problems as well.

We therefore advocate selecting such values by some more flexible device, capable of exploiting the outcome of previous instantiations in order to steer the algorithm towards better performance. Since such a scheme chooses an instantiation of the control parameters based on past selections and the respective outcome, we refer to them as *adaptive control schemes* (ACS) because, in the words of Glover, Laguna (1997), the particular instantiation "chosen at

any iteration [...] is a function of those previously chosen. That is, the method is adaptive in the sense of updating relevant information from iteration to iteration". In contrast, we might say that class-based as well as fixed control schemes are adapted rather than adaptive in nature. By saying this we mean that the former are (hopefully) tailored to the specifics of the problem in general while the latter incorporate some capability of adapting the values of control parameters to the specifics of a given instance (cf. Table 4). Throughout the literature that we are aware of, control schemes which are actually adaptive in this sense are scarce. Bölte, Thonemann (1996) apply a genetic algorithm-based control scheme to identify good cooling schemes for a simulated annealing method. Kraay, Harker (1996) discuss a number of related approaches for repetitive combinatorial optimization problems. An experimental study of several adaptive control schemes for the RCPSP is conducted by Schirmer (1998).

| Control Scheme | Fixed in Advance | Fixed for all Instances |
|---|---|---|
| FCS | yes | yes |
| CCS | yes | no |
| ACS | no | no |

*Table 4: Classification of Control Schemes*

### *4.2.2. Deterministic Local Search-Based Control*

This approach is a *local search control scheme* (LSCS), which systematically applies and appraises different control parameter values in order to identify promising ones, hopefully guiding the computational effort to promising regions of the parameter and thus the solution space. The scheme is based upon the algorithm proposed in Haase (1996) and has been utilized in the field of course scheduling by Haase et al. (1997, 1998). An evaluation of the scheme on the RCPSP can be found in Schirmer (1998). Following the classification of local search procedures used in Leon, Ramamoorthy (1997), the scheme can be characterized as a steepest-descent search method. Recall from above that each parameter vector $\vec{\alpha}$ encodes one particular solution. In each iteration, the neighborhood comprises a fixed number of such vectors. From each of these one solution is constructed, which is evaluated in terms of its objective function value. If the best so-found solution is better than the incumbent best one, the search process is intensified in the surrounding region by focussing on the neighborhood of the corresponding vector; otherwise, the procedure terminates.

The method proceeds by spanning an I-dimensional grid over the control parameter space by defining a set of equidistant points. In each iteration the grid, which initially embraces the entire parameter space $[0,1]^I$, is refined to ever smaller subspaces. The number of gridpoints, which remains constant throughout, is determined by an integral control parameter, the *grid granularity* $\sigma$, in the following way. Let for each iteration denote $L\alpha_i$ and $U\alpha_i$ the lower and

upper bound of the parameter subspace of rule i. Then for each iteration the so-called *grid width* $\Delta_i$ of rule i is defined as

$$\Delta_i \leftarrow (U\alpha_i - L\alpha_i) / \sigma \qquad (1 \leq i \leq I) \quad (37)$$

The initial iteration of the procedure starts off with $\alpha_i = 0$, $L\alpha_i = 0$, and $U\alpha_i = 1$ for each rule i and determines a solution. It then increments $\alpha_1$ by its grid width $\Delta_1$ and constructs another solution, and so forth, until eventually $\alpha_1 = U\alpha_1$. Then $\alpha_1$ is reset to $L\alpha_1$ whereas $\alpha_2$ is incremented by $\Delta_2$. When also $\alpha_2$ has reached its upper bound, it is reset to its lower bound while $\alpha_3$ is incremented by $\Delta_3$ and so forth, until eventually all $\alpha_i$ equal their upper bound. Thus, the number of gridpoints considered per iteration is $(\sigma+1)^I$. For each gridpoint, which corresponds to one parameter vector, one solution is constructed. If the iteration fails to improve the incumbent best solution, the algorithm halts (in order to restrict computation times, the algorithm could also be stopped after some time limit; however, this approach is not pursued here since computation times have been found to be modest for the test instances attempted). Otherwise, let denote $\alpha^*_i$ that weight of rule i which produced the best solution in that iteration; ties are broken by taking the first such weight. Now, new bounds are calculated from

$$L\alpha_i \leftarrow \max\{0, \alpha^*_i - \Delta_i \cdot (\sigma-1) / \sigma\} \qquad (1 \leq i \leq I) \quad (38)$$

$$U\alpha_i \leftarrow \min\{1, \alpha^*_i + \Delta_i \cdot (\sigma-1) / \sigma\} \qquad (1 \leq i \leq I) \quad (39)$$

the grid widths $\Delta_i$ of all rules are updated according to (37), and the next iteration starts. Notice that, due to the feasibility variant of the RCPSP/Π being strongly **NP**-complete, schedules constructed during the proceeding of the algorithm may be infeasible, i.e. may have a makespan greater than the planning horizon allotted. We chose, however, to tolerate these, hoping that they will be improved in due course to feasible schedules; thus, only the final schedule for each instance is actually checked for feasibility.

An algorithmic description of the scheme is given in Table 5.

*Initialization*

**for each** $(1 \le i \le I)$
$L\alpha_i \leftarrow 0$
$U\alpha_i \leftarrow 1$
**endfor**
$FT^* \leftarrow \infty$

*Execution*

**repeat**
no_better_solution_found ← TRUE
**for each** $(1 \le i \le I)$
$\alpha_i \leftarrow L\alpha_i$
**calculate** $\Delta_i$ **according to** (37)
**endfor**
**for** $z \leftarrow 1$ **to** $(\sigma + 1)^I$
$FT_J \leftarrow$ SchedulingAlgorithm(I, $\vec{\alpha}$)
if $(FT_J < FT^*)$
$FT^* \leftarrow FT_J$
$\vec{\alpha}^* \leftarrow \vec{\alpha}$
no_better_solution_found ← FALSE
**endif**
**call** UpdateAlpha
**endfor**
**for each** $(1 \le i \le I)$
**calculate** $L\alpha_i$ **according to** (38)
**calculate** $U\alpha_i$ **according to** (39)
**endfor**
**until** no_better_solution_found

*Subroutine UpdateAlpha*

$i \leftarrow I$
**while** $\alpha_i = U\alpha_i \wedge i \ge 1$
$\alpha_i \leftarrow L\alpha_i$
$i \leftarrow i - 1$
**endwhile**
$\alpha_i \leftarrow \alpha_i + \Delta_i$

*Result*

If $FT^* < \infty$, then for each priority rule i $(1 \le i \le I)$ $\alpha^*_i$ denotes that control parameter value which produced the best solution and $FT^*$ denotes the corresponding objective function value. Otherwise, no feasible solution was found. ■

*Table 5: Local Search-Based Control Scheme*

We illustrate the procedure for I = 2 priority rules and a grid granularity of $\sigma = 3$. Figure 3 depicts the first three iterations of the scheme. Each evaluated parameter value vector is represented by an intersection point, so the shaded area shows the parameter subspace spanned by the grid searched in one iteration. Again, the symbol "•" marks the best control parameter vector of each iteration, in whose vicinity the search for good vectors (and solutions) is intensified.

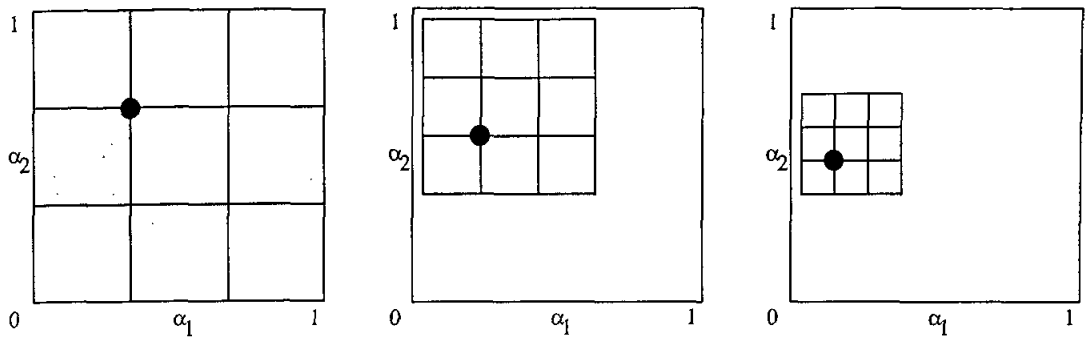


*Figure 3: Exemplary Application of Local Search-Based Control Scheme*

#### *4.2.3. Randomized Local Search-Based Control*

It is well-known that the effectiveness of construction methods can be increased by adding a good randomization scheme to turn them into sampling methods. This strategy has been very successful for simple priority rule-based methods for the RCPSP, improving e.g. the effectiveness of the rule LST from deterministic 6.56% to sampling-based 1.89% (Schirmer 1999, pp. 42-43, 60-64).

In oirder to follow up on the question whether similar improvements can also be reaped for the RCPSP/Π, we augmented the LSCS by the the MRBRS as a subsequent randomization stage. To the resulting two-stage scheme we refer as a *randomized local search control scheme* (RLSCS).

## 5. Experimental Analysis

### *5.1. Experimental Setting*

#### *5.1.1. Test Instances and Reference Solutions*

As a test bed, we used the benchmark instance set J10 generated with ProGen/Π (Schirmer 1999, pp. 155-168). Each instance comprises 10 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 capturing characteristics of these instances 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 controls the number

of resources that are requested by each activity, RS governs the resource scarcity measured between minimum and maximum demand. CF controls the cardinality of the period subsets, HF the tightness of the planning horizon T. 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 in Böttcher et al. (1999). J10 comprises ten instances per cluster, or 960 instances. All instances were attempted with the most effective of the branch-and-bound algorithms described in Böttcher et al. (1999). 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 (TBB) of this algorithm. The CPU time limit was set to 5 minutes, 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. For details we refer the reader to Schirmer (1999, pp. 155-168). Of course, the nine instances proven to be infeasible were excluded from our experimentation.

| Instance Set | Non-Optimally Solved | Optimally Solved | Feasibly Solved | Undecided | Proven Infeasible | Total |
|---|---|---|---|---|---|---|
| J10 | 39 | 901 | 940 | 11 | 9 | 960 |

*Table 7: Characteristics of Instance Set J10*

### 5.1.2. *Priority Rule Sets and Scheduling Algorithm*

Particular care has to be taken in defining the rule sets to be used. In order to assess the control schemes' ability to cope with sets of mostly good or bad rules as well as large and small sets, we defined several such rule sets. As indicated earlier, our classification of rules as good or bad stems from the results in Schirmer (1999, pp. 183-187), where the effectiveness of numerous priority rules applied to the RCPSP/Π under the SSS/Π was measured in terms of the number of unsolved instances. In Table 8, we report each such number as a percentage from the average number of unsolved instances. Note that negative percentages indicate that a rule left less instances unsolved than the average such number over all rules; in other words, the effectiveness of a rule is the better, the lower its percentage is. From these rules, we blended several rule sets as shown in Table 9.

| *Good Rules* | | | | | | *Bad Rules* | | | |
|---|---|---|---|---|---|---|---|---|---|
| DRC/ES | DRC/E | DRS/E | DRS/ES | MTS | EFT | TRS/E | TRS/ES | RRD/E | TRS |
| -64% | -59% | -43% | -37% | -30% | -21% | 28% | 29% | 31% | 35% |

*Table 8: Effectiveness and Classification of Priority Rules*

| Rule Sets | I | Characterization | Priority Rules |
|---|---|---|---|
| 1 | 8 | 4 good, 4 bad rules | DRC/ES, DRC/E, DRS/E, DRS/ES, TRS/E, TRS/ES, RRD/E, TRS |
| 2 | 6 | 3 good, 3 bad rules | DRC/ES, DRC/E, DRS/E, TRS/ES, RRD/E, TRS |
| 3 | 4 | 2 good, 2 bad rules | DRC/ES, DRC/E, RRD/E, TRS |
| 4 | 2 | 1 good, 1 bad rules | DRC/ES, TRS |
| 5 | 4 | 3 good, 1 bad rules | DRC/ES, DRC/E, DRS/E, TRS |
| 6 | 4 | 1 good, 3 bad rules | DRC/ES, TRS/ES, RRD/E, TRS |
| 7 | 4 | 4 good rules | DRC/ES, DRC/E, DRS/E, DRS/ES |
| 8 | 3 | 3 good rules | DRC/ES, DRC/E, DRS/E |
| 9 | 2 | 2 good rules | DRC/ES, DRC/E |
| 10 | 4 | 4 bad rules | TRS/E, TRS/ES, RRD/E, TRS |
| 11 | 3 | 3 bad rules | TRS/ES, RRD/E, TRS |
| 12 | 2 | 2 bad rules | RRD/E, TRS |
| 13 | 4 | 4 good rules | DRC/ES, MTS, EFT, DRS/E |
| 14 | 3 | 3 good, 1 bad rules | DRC/ES, MTS, EFT, TRS |
| 15 | 2 | 2 good, 2 bad rules | DRC/ES, MTS, RRD/E, TRS |

*Table 9: Priority Rule Sets*

Among other factors, in the remainder of this work we will analyze the effect of two factors, viz. the number of priority rules I and the proportion of good to bad rules, expressed in terms of the percentage Θ of good rules. We will refer to these factors as *rule set cardinality* and *composition*. Note that we can vary the first factor, while holding the second one constant, by

regarding rule sets 7, 8, 9 for the case of $\Theta = 100\%$, 1, 2, 3, and 4 for $\Theta = 50\%$, and 10, 11, 12 for $\Theta = 0\%$. Vice versa, to vary the second factor, while keeping the first one fixed, we merely need to consider rule sets 7, 5, 3, 6, 10 for the case of I = 4 and rule sets 9, 4, 12 for the case of I = 2.

As instantiation of SchedulingAlgorithm($I$, $\vec{\alpha}$) we use the SSS/Π.

### *5.1.3. Performance Measures*

Specifying a set of values for each factor within an experiment describes over which levels it is varied during an experiment, so one value for each factor determines a run of an experiment. We regard, for each algorithm, the outcome of a run in terms of both effectiveness and efficiency. The latter captures the CPU-time required for the run, measured in terms of seconds; where appropriate, we also report the number of schedules generated. Measurements were taken using an implementation in C, running on a Pentium 266 personal computer with 32 MB RAM under Windows 95. As the feasibility problem is strongly **NP**-complete, the principal effectiveness measure is the number of unsolved instances. Apart from that, we will also regard the average percentage deviation, over all solved instances, of the best found schedule from the respective reference solution.

## *5.2. Deterministic Local Search-Based Control*

### *5.2.1. Experimental Design*

The factors examined here are control parameter $\sigma$, rule set cardinality I, and rule set composition $\Theta$.

| σ | I | Sol/Iter | I | Sol/Iter | I | Sol/Iter | I | Sol/Iter | I | Sol/Iter |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 3 | 8 | 4 | 16 | 6 | 64 | 8 | 256 |
| 2 | 2 | 9 | 3 | 27 | 4 | 81 | 6 | 729 | 8 | 6,561 |
| 3 | 2 | 16 | 3 | 64 | 4 | 256 | 6 | 4,096 | 8 | 65,536 |
| 4 | 2 | 25 | 3 | 125 | 4 | 625 | 6 | 15,625 | 8 | 390,625 |
| 5 | 2 | 36 | 3 | 216 | 4 | 1,296 | 6 | 46,656 | 8 | 1,679,616 |
| 6 | 2 | 49 | 3 | 343 | 4 | 2,401 | 6 | 117,649 | 8 | 5,764,801 |
| 7 | 2 | 64 | 3 | 512 | 4 | 4,096 | 6 | 262,144 | 8 | 16,777,216 |
| 8 | 2 | 81 | 3 | 729 | 4 | 6,561 | 6 | 531,441 | 8 | 43,046,721 |
| 9 | 2 | 100 | 3 | 1,000 | 4 | 10,000 | 6 | 1,000,000 | 8 | 100,000,000 |
| 10 | 2 | 121 | 3 | 1,331 | 4 | 14,641 | 6 | 1,771,561 | 8 | 214,358,881 |

*Table 10: Exemplary Numbers of Gridpoints*

In order to demonstrate the influence of I and σ on the number of gridpoints, Table 10 summarizes some exemplary numbers. Due to the prohibitive effort required to cover large gridpoint numbers, the experiments were restricted to combinations of I and σ lying in the non-shaded parts of Table 10. Indeed, if not specified otherwise, σ is set - for each rule set cardinality I - to the maximum value lying in the non-shaded area.

### *5.2.2. Effect of Sigma*

From the design of the LSCS it is clear that increasing σ will tend to increase effectiveness but also CPU times entailed. The marginal benefit from increasing σ, i.e. the ratio of CPU times to deviations, of course decreases at an exponential rate. To verify this, we employed the rule sets 7 - 12 with all σ-values up to the maximum one used. The effect of σ on effectiveness is illustrated in Tables 11 and 12, the effect on efficiency in Table 13. The results are representative for those obtained from varying σ on other rule sets. Note that for the rule sets 9 and 12 not only the number of unsolved instances but also the respective deviations remain constant for $\sigma \geq 6$, so very large levels of σ need not necessarily be beneficial.

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 73 | 72 | 70 | | | | | | | |
| 8 | 75 | 73 | 71 | 71 | 71 | | | | | |
| 9 | 91 | 90 | 88 | 88 | 88 | 88 | 88 | 88 | 88 | 88 |
| 10 | 51 | 40 | 40 | | | | | | | |
| 11 | 56 | 45 | 43 | 41 | 41 | | | | | |
| 12 | 106 | 103 | 101 | 100 | 99 | 98 | 98 | 98 | 98 | 98 |

*Table 11: Effect of Sigma - Unsolved Instances (deterministic LSCS)*

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 3.04% | 2.93% | 2.96% | | | | | | | |
| 8 | 3.46% | 3.11% | 3.13% | 3.14% | 3.13% | | | | | |
| 9 | 4.13% | 3.49% | 3.52% | 3.51% | 3.51% | 3.51% | 3.51% | 3.51% | 3.51% | 3.51% |
| 10 | 2.64% | 1.95% | 1.79% | | | | | | | |
| 11 | 4.02% | 2.20% | 2.08% | 2.10% | 1.99% | | | | | |
| 12 | 5.11% | 3.70% | 3.60% | 3.68% | 3.66% | 3.71% | 3.66% | 3.66% | 3.66% | 3.66% |

*Table 12: Effect of Sigma - Deviations (deterministic LSCS)*

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 0.08 | 0.26 | 0.83 | | | | | | | |
| 8 | 0.05 | 0.10 | 0.17 | 0.30 | 0.58 | | | | | |
| 9 | 0.04 | 0.05 | 0.06 | 0.08 | 0.10 | 0.12 | 0.14 | 0.18 | 0.21 | 0.28 |
| 10 | 0.18 | 0.54 | 1.71 | | | | | | | |
| 11 | 0.12 | 0.23 | 0.42 | 0.73 | 1.41 | | | | | |
| 12 | 0.21 | 0.14 | 0.16 | 0.21 | 0.29 | 0.36 | 0.46 | 0.57 | 0.68 | 0.96 |

*Table 13: Effect of Sigma - CPU Times (deterministic LSCS)*

As can be seen, larger values of $\sigma$ will generally improve effectiveness, albeit at the expense of increasing computation times. However, note that for the rule sets 9 and 12 not only the number of unsolved instances but also the respective deviations remain constant for $\sigma \geq 6$, so very large levels of $\sigma$ need not necessarily be beneficial.

### *5.2.3. Effect of Rule Set Cardinality and Composition*

To demonstrate the effect that the number of rules in a set exerts on the effectiveness, we present the average deviations of the respective rule sets in Table 14.

| Rule Set | l | $\Theta$ | *Unsolv* | *Avg Dev* |
|---|---|---|---|---|
| 7 | 4 | 100% | 75 | 2.96% |
| 8 | 3 | 100% | 76 | 3.13% |
| 9 | 2 | 100% | 93 | 3.51% |
| 1 | 8 | 50% | 47 | 1.87% |
| 2 | 6 | 50% | 47 | 2.09% |
| 3 | 4 | 50% | 49 | 1.88% |
| 4 | 2 | 50% | 59 | 2.10% |
| 10 | 4 | 0% | 45 | 1.79% |
| 11 | 3 | 0% | 46 | 1.99% |
| 12 | 2 | 0% | 103 | 3.66% |

*Table 14: Effect of Rule Set Cardinality - Effectiveness (deterministic LSCS)*

It turns out that increasing (decreasing) the number of rules while keeping the proportion of good to bad rules fixed produces better (worse) results since the pool is larger from which an appropriate rule can be selected for each instance.

In Table 15 we arrange the results pertaining to the influence of the rule set composition, given a fixed set cardinality. Let us concentrate for a moment on the first two groups of results, up to rule set 12. Including good rules in a rule set is clearly important, as the presence of bad rules only cannot be compensated by the LSCS. However, rule sets of only good rules are not guaranteed to be the most effective ones. Actually those rule sets where $\Theta$ is in the

middle ranges show the highest effectivity. Although one might expect effectiveness to be the higher (lower), the higher (lower) the proportion of good rules is, given that then the number of good rules is larger (smaller) from which an appropriate rule can be selected for each instance, this finding conforms with similar findings on the RCPSP (Schirmer 1998).

| Rule Set | I | Θ | *Unsolv* | *Avg Dev* |
|---|---|---|---|---|
| 7 | 4 | 100% | 75 | 2.96% |
| 5 | 4 | 75% | 46 | 1.97% |
| 3 | 4 | 50% | 49 | 1.88% |
| 6 | 4 | 25% | 44 | 1.78% |
| 10 | 4 | 0% | 45 | 1.79% |
| 9 | 2 | 100% | 93 | 3.51% |
| 4 | 2 | 50% | 59 | 2.10% |
| 12 | 2 | 0% | 103 | 3.66% |
| 13 | 4 | 100% | 60 | 1.98% |
| 14 | 4 | 75% | 47 | 1.60% |
| 15 | 4 | 50% | 43 | 1.58% |
| 6 | 4 | 25% | 44 | 1.78% |
| 10 | 4 | 0% | 45 | 1.79% |

*Table 15: Effect of Rule Set Composition - Effectiveness (deterministic LSCS)*

To seek out the cause of this effect, we reanalyzed the results in Schirmer (1999, pp. 169-179) on the simple priority rules which make up the rule sets used here. This analysis revealed that the four resource-based good rules are rather close to each other in terms of effectiveness: for 766 out of 951 instances they produce similar solutions, i.e. solutions with the same objective function value. It is thus straightforward to suspect that in such cases adding some bad rules adds a healthy dose of diversity to the rule set, which then helps to explore other regions of the parameter space searched. We therefore conducted an additional experiment in which we replaced some of the resource-based good rules by other ones, viz. MTS and EFT. The outcome supports our conjecture as indeed the more diverse rule sets 13 - 15 markedly outperform the corresponding, less diverse sets 7, 5, and 3.

### *5.2.4. Efficiency*

The computation times observed for the LSCS approach are reported in Table 16, for each rule set the maximum value of σ as defined in Table 10 was used.

| Rule Set | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Avg CPU | 1.88 | 0.50 | 1.39 | 0.21 | 0.60 | 1.64 |
| Rule Set | 7 | 8 | 9 | 10 | 11 | 12 |
| Avg CPU | 0.83 | 0.58 | 0.28 | 1.71 | 1.41 | 0.96 |
| Rule Set | 13 | 14 | 15 | | | |
| Avg CPU | 0.52 | 0.46 | 1,38 | | | |

*Table 16: Effect of Rule Sets - CPU Times (deterministic LSCS)*

Note that, as we use a dynamic termination criterion for the LSCS approach instead of just setting a fixed sample size, the number of iterations performed by the LSCS algorithm may vary substantially for different instances. Also, the number of gridpoints actually visited, and thus the number of schedules generated per iteration of the LSCS, may be smaller than $(\sigma+1)^I$ due to the inclusion of bounding rules in the scheduling algorithm. We therefore give the average numbers of generated schedules per rule set in Table 17.

| Rule Set | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Avg Schedules | 156 | 41 | 164 | 82 | 161 | 158 |
| Rule Set | 7 | 8 | 9 | 10 | 11 | 12 |
| Avg Schedules | 184 | 159 | 96 | 159 | 139 | 89 |

*Table 17: Effect of Rule Sets - Generated Schedules (deterministic LSCS)*

### *5.3. Randomized Local Search-Based Control*

#### *5.3.1. Experimental Design*

It is well-known that the effectiveness of construction methods can be increased when a good randomization scheme is added to turn them into sampling methods. This strategy has been very successful for simple priority rule-based methods for the RCPSP, improving e.g. the effectiveness of the rule LST from 6.56% when applied deterministically to 1.89% when applied in a randomized manner (Schirmer 2000). In this section, we follow up on the question whether the same improvements can also be reaped for the LSCS. We therefore added a second, randomized stage to the LSCS; we refer to the two-stage LSCS as RLSCS. The factors examined in the following are control parameter $\sigma$, rule set cardinality I, and rule set composition $\Theta$. In line with the settings found best in Schirmer (1998), we let $\alpha = 1$ and $\delta = 10$.

#### *5.3.2. Effect of Sigma*

In Tables 19 and Table 20 we summarize the results for rule sets 7 - 12, using all applicable $\sigma$-values. For the randomized, second stage control parameters were set as $\alpha = 1$ and $\delta = 10$. The CPU times reported are the totals of both stages, using a sample size of 100 iterations.

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 31 | 31 | 30 | | | | | | | |
| 8 | 32 | 31 | 30 | 30 | 30 | | | | | |
| 9 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 |
| 10 | 29 | 25 | 25 | | | | | | | |
| 11 | 31 | 28 | 27 | 25 | 25 | | | | | |
| 12 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 | 42 |

*Table 18: Effect of Sigma - Unsolved Instances (randomized LSCS)*

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 1.44% | 1.56% | 1.61% | | | | | | | |
| 8 | 1.54% | 1.56% | 1.57% | 1.63% | 1.66% | | | | | |
| 9 | 1.52% | 1.47% | 1.48% | 1.44% | 1.41% | 1.40% | 1.42% | 1.42% | 1.42% | 1.42% |
| 10 | 1.32% | 1.18% | 1.17% | | | | | | | |
| 11 | 1.62% | 1.27% | 1.25% | 1.32% | 1.25% | | | | | |
| 12 | 1.72% | 1.87% | 1.95% | 1.99% | 1.95% | 1.94% | 1.97% | 1.99% | 1.99% | 2.02% |

*Table 19: Effect of Sigma - Deviations (randomized LSCS)*

| Rule Set | *Sigma* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 0.18 | 0.35 | 0.94 | | | | | | | |
| 8 | 0.13 | 0.17 | 0.25 | 0.38 | 0.67 | | | | | |
| 9 | 0.11 | 0.11 | 0.12 | 0.14 | 0.16 | 0.18 | 0.21 | 0.24 | 0.27 | 0.36 |
| 10 | 0.43 | 0.77 | 1.99 | | | | | | | |
| 11 | 0.38 | 0.45 | 0.64 | 0.96 | 1.67 | | | | | |
| 12 | 0.84 | 0.43 | 0.41 | 0.47 | 0.55 | 0.63 | 0.72 | 0.84 | 0.95 | 1.28 |

*Table 20: Effect of Sigma - CPU Times (randomized LSCS)*

Unsurprisingly, the effect is similar to the one observed for the deterministic LSCS, although less pronounced as the beneficial effect of the randomization stage decreases the numbers of instances remaining unsolved as well as the average deviations.

### *5.3.3. Effect of Rule Set Cardinality and Composition*

From the results in Tables 21 and 22 it is obvious that the effectiveness of all rule sets is improved by adding randomization as a second stage. As already good rule sets can be said to start into the second stage with an advantage, their improvements are less dramatical than those of the bad rule sets. Still, the ranking of the rule sets remains virtually unchanged such that again sets comprising some but not only good rules sport the highest effectiveness.

| Rule Set | I | Θ | *Unsolv* | *Avg Dev* |
|---|---|---|---|---|
| 7 | 4 | 100% | 35 | 1.61% |
| 8 | 3 | 100% | 35 | 1.66% |
| 9 | 2 | 100% | 40 | 1.42% |
| 1 | 8 | 50% | 32 | 1.03% |
| 2 | 6 | 50% | 32 | 1.15% |
| 3 | 4 | 50% | 32 | 1.30% |
| 4 | 2 | 50% | 37 | 1.28% |
| 10 | 4 | 0% | 30 | 1.17% |
| 11 | 3 | 0% | 30 | 1.25% |
| 12 | 2 | 0% | 47 | 2.02% |

*Table 21: Effect of Rule Set Cardinality - Effectiveness (randomized LSCS)*

| Rule Set | I | Θ | *Unsolv* | *Avg Dev* |
|---|---|---|---|---|
| 7 | 4 | 100% | 35 | 1.61% |
| 5 | 4 | 75% | 30 | 1.27% |
| 3 | 4 | 50% | 32 | 1.30% |
| 6 | 4 | 25% | 30 | 1.18% |
| 10 | 4 | 0% | 30 | 1.17% |
| 9 | 2 | 100% | 40 | 1.42% |
| 4 | 2 | 50% | 37 | 1.28% |
| 12 | 2 | 0% | 47 | 2.02% |
| 13 | 4 | 100% | 31 | 1.23% |
| 14 | 4 | 75% | 33 | 1.15% |
| 15 | 4 | 50% | 33 | 1.16% |
| 6 | 4 | 25% | 30 | 1.18% |
| 10 | 4 | 0% | 30 | 1.17% |

*Table 22: Effect of Rule Set Composition - Effectiveness (randomized LSCS)*

An interesting question, finally, is how often the additional randomization stage actually improves the effectiveness. Table 23 shows, again for $\alpha = 1$ and $\delta = 10$, the number of instances for which randomization produced a better solution than the one deterministically found. The results are representative for other settings of $\alpha$ and $\delta$.

| Rule Set | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Instances | 108 | 120 | 98 | 117 | 96 | 86 |
| Rule Set | 7 | 8 | 9 | 10 | 11 | 12 |
| Instances | 167 | 172 | 210 | 86 | 90 | 145 |
| Rule Set | 13 | 14 | 15 | | | |
| Instances | 124 | 101 | 82 | | | |

*Table 23: Effect of Randomization - Improved Instances*

### 5.3.4. *Efficiency*

The computation times measured for the randomized LSCS approach are given in Table 24.

| Rule Set | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Avg CPU | 2.16 | 0.75 | 1.62 | 0.27 | 0.69 | 1.90 |
| Rule Set | 7 | 8 | 9 | 10 | 11 | 12 |
| Avg CPU | 0.94 | 0.67 | 0.36 | 1.99 | 1.67 | 1.28 |
| Rule Set | 13 | 14 | 15 | | | |
| Avg CPU | 0.58 | 0.51 | 1.59 | | | |

*Table 24: Effect of Rule Sets - CPU Times (randomized LSCS)*

## 5.4. *Comparison with Other Algorithms*

In order to assess the merits of the algorithmic approaches discussed in this paper, we compare them to several sampling-based construction methods proposed for the problem, namely the FCS-based one (FCS-B) of Böttcher et al. (1999), the best priority rules (DRC/E, DRC/ES) and the CCS-based one (CCS/Π) of Schirmer (1999, pp. 181-198). For the sake of brevity, we restrict the presentation of results for the adaptive control algorithms to the most effective rule set 6. As the first stage of the LSCS-algorithms usually performs between 100 and 500 iterations (cf. Table 17), we provide the results from both these sizes in Table 25.

| Algorithm | Reference | Control Scheme | *Sample Size* | | | |
|---|---|---|---|---|---|---|
| | | | Unsolv | | Avg Dev | |
| | | | 100 | 500 | 100 | 500 |
| CCS/Π | Schirmer (1999) | CCS | 26 | 17 | 1.44% | 0.69% |
| DRC/E | Schirmer (1999) | FCS | 27 | 18 | 1.40% | 0.72% |
| RLSCS, rule set 6 | – | ACS | 30 | | 1.18% | |
| DRC/ES | Schirmer (1999) | FCS | 31 | 17 | 1.16% | 0.70% |
| FCS-B | Böttcher et al. (1999) | FCS | 40 | 21 | 1.49% | 0.81% |
| LSCS, rule set 6 | – | ACS | 44 | | 1.78% | |

*Table 25: Comparison of Several Construction Methods*

The performance of the deterministic LSCS is clearly inferior to that of other construction algorithms, which is no surprise as it proceeds strictly deterministic. Augmenting LSCS by a randomization stage to turn it into a sampling algorithm improves its effectiveness drastically, an effect which is well-known for other construction methods. The arising RLSCS approach is almost as effective as more traditional sampling algorithms; note, however, that these

require less iterations. To summarize, state-of-the-art sampling-based methods still outperform all ACS-based algorithms considered here.

## 6. Summary and Conclusions

For many computationally intractable problems findings suggest that there is no single heuristic which performs best on all possible instances. Therefore, adaptive control schemes try to dynamically compose good algorithms from an unsorted collection of - usually rather simple - algorithms, thus tailoring one individual algorithm to each instance.

In this work, we chose a collection of priority rule-based construction methods to start with, and some simple search-based mechanisms to combine well-suited algorithms from these. In doing so, one hopes to close the performance gap between construction and search methods, turning out methods nestled between pure construction and evolved search methods, in terms of both effectiveness and efficiency. Building upon a recent study in which several adaptive control schemes were analyzed on the RCPSP (Schirmer 1998), we scrutinized the performance of the best of these schemes on the more general RCPSP/Π. We find that the RLSCS, a combination of local search and randomized sampling, achieves an effectiveness comparable to that of other good construction methods, but it is hampered by a lower efficiency.

As even the RLSCS fails to outperform state-of-the-art construction heuristics for both problems, our results seem to indicate that adaptive control schemes cannot compete with the more classical sampling heuristics using fixed control schemes. Still, adaptive control schemes should not be disregarded prematurely, on the basis of effectiveness alone. Such a focus turns the process of improving algorithms into a competitive race which concentrates on 'beating' other algorithms rather than on insight why some algorithms perform better than others (Hooker 1995). Given the long history of more and more effective heuristics for project scheduling problems, evaluating algorithms on these clearly puts new algorithms at a disadvantage if the outcome is considered in terms of effectiveness alone. One has to bear in mind that the analytical and experimental advances achieved there are the result of several decades of inventive and persistent research. We conclude that the rather straightforward algorithms proposed so far are too simple to unlock the full potential of adaptive control schemes.

Indeed, we believe the concept of algorithms utilizing adaptive control schemes to have merit for several reasons. First, by design they are robust to changes in the characteristics of instances attempted, thus obviating the need for prior knowledge on their effects which is otherwise required to devise good algorithms. Second, they are relatively immune against the adverse effects of bad rule sets, so they can be applied in situations where little knowledge is available on what constitutes a good or a bad rule. Third, they are simple and easy to imple-

ment. We thus hope for this contribution to pave the way for better adaptive control schemes which will eventually be competitive with state-of-the-art heuristics.

Future work to achieve this goal might begin with refining the RLSCS. In each iteration of the LSCS, the search is intensified around the best-performing point in the parameter space; if several such points exist, the tie is broken by using the one found first. A canonical extension of this idea would be to store several or all tied candidates in a list, to intensify the search around these in a breadth-first fashion, and to continue with the best candidate found. Also, for certain problems, the assumption that the best rules under a deterministic regime perform best also as sampling algorithms may not be justified. In that case, a better idea might be to replace the deterministic call of SchedulingAlgorithm($I$, $\tilde{\alpha}$) by a number of randomized iterations, where the best makespan found is returned. Finally, note that all these refinements would still keep the algorithm within the realm of local search-based control schemes. Still, the basic idea of adaptivity easily allows for more evolved control schemes which employing not only sampling-based construction but metaheuristic approaches, as well. Exploring such options will certainly prove worthwhile.

## Acknowledgements

We are indebted to Andreas Drexl and Peter Kandzia for their ongoing support. Also, we gratefully acknowledge the excellent research assistance of Carsten Scholz. Parts of this work were conducted while the author held a teaching position at the Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel. Partly support was granted by the Deutsche Forschungsgemeinschaft (German National Science Foundation) under Grant Dr 170/4-1.

## References

ALVAREZ-VALDÉS, R. AND J.M. TAMARIT (1989), "Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis", in: *Advances in project scheduling*, R. Slowinski and J. Weglarz (eds.), Elsevier, Amsterdam, pp. 113-134.

BARMAN, S. (1997), "Simple priority rule combinations: An approach to improve both flow time and tardiness", International Journal of Production Research 35, pp. 2857-2870.

BÖLTE, A. AND U.W. THONEMANN (1996), "Optimizing simulated annealing schedules with genetic programming", European Journal of Operational Research 92, pp. 402-416.

BÖTTCHER, J., A. DREXL, R. KOLISCH, AND F. SALEWSKI (1999), "Project scheduling under partially renewable resource constraints", Management Science 45, pp. 543-559.

DAVIS, E.W. AND J.H. PATTERSON (1975), "A comparison of heuristic and optimum solutions in resource-constrained project scheduling", *Management Science* 21, pp. 944-955.

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

DREXL, A. AND J. GRÜNEWALD (1993), "Nonpreemptive multi-mode resource-constrained project scheduling", IIE Transactions 25, pp. 74-81.

GLOVER, F. AND M. LAGUNA (1997), *Tabu search*, Kluwer, Norwell, MA.

HAASE, K. (1996), "Capacitated lot-sizing with sequence dependent setup costs", Operations Research Spektrum 18, pp. 51-59.

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.

HOOKER, J.N. (1995), "Testing heuristics: We have it all wrong", Journal of Heuristics 1, pp. 33-42.

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.

KIMMS, A. (1996), "Competitive methods for multi-level lot sizing and scheduling: Tabu search and randomized regrets", International Journal of Production Research 34, pp. 2279-2298.

KIMMS, A. (1998), "Fallbasiertes Schließen auf Methoden zur Produktionsplanung", Wirtschaftsinformatik 40, pp. 417-423 (in German).

KLEIN, R. AND A. SCHOLL (1999), "Computing lower bounds by destructive improvement - An application to resource-constrained project scheduling", European Journal of Operational Research 112, pp. 322-346.

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

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.

KRAAY, D.R. AND P.T. HARKER (1996), "Case-based reasoning for repetitive combinatorial optimization problems, Part I: Framework", Journal of Heuristics 2, pp. 55-85.

LEON, V.J. AND B. RAMAMOORTHY (1997), "An adaptable problem-space based search method for flexible flow line scheduling", IIE Transactions 29, pp. 115-125.

SCHIRMER, A. (1998), "Adaptive control schemes for parameterized heuristic scheduling", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 488, ftp://ftp.bwl.uni-kiel.de/pub/operations-research/wp488.ps.gz.

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

SCHIRMER, A. (2000), "Case-based reasoning and improved adaptive search for project scheduling", to appear in Naval Research Logistics.

SCHIRMER, A. AND A. DREXL (1997), "Allocation of partially renewable resources - Models and applications", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 455, ftp://ftp.bwl.uni-kiel.de/pub/operations-research/wp455.ps.gz.

SCHIRMER, A. AND S. RIESENBERG (1997), "Parameterized heuristics for project scheduling - Biased random sampling methods", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 456, ftp://ftp.bwl.uni-kiel.de/pub/operations-research/wp456.ps.gz.

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, ftp://ftp.bwl.uni-kiel.de/pub/operations-research/wp471.ps.gz.

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.

ULUSOY, G. AND L. ÖZDAMAR (1989), "Heuristic performance and network/resource characteristics in resource-constrained project scheduling", Journal of the Operational Research Society 40, pp. 1145-1152.

WHITEHOUSE, G.E. AND J.R. BROWN (1979), "GENRES: An extension of Brookes algorithm for project scheduling with resource constraints", Computers and Industrial Engineering 3, pp. 261-268.

WOLPERT, D.H. AND W.G. MACREADY (1995), "No free lunch theorem for search", Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, www.santafe.edu/sfi/publications/Working-Papers/95-02-010.ps.