Schirmer, Armin

Adaptive control schemes for parameterized heuristic scheduling

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 488

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 488

Adaptive Control Schemes for Parameterized Heuristic Scheduling

Schirmer

No. 488

Adaptive Control Schemes for Parameterized Heuristic Scheduling

Schirmer

September 1998

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 +49-431-880-1531
schirmer@bwl.uni-kiel.de
www.bwl.uni-kiel.de/bwlinstitute/prod/mab/schirmer

Figures

Tables

Abstract: For most computationally intractable problems there exists no heuristic which performs best on all instances. Usually, a heuristic characterized as best will perform good on the majority of instances but leave a minority on which other heuristics do better. In priority rule-based scheduling, attempts to remedy this have been made by combining simple priority rules in a fixed and predetermined way. We investigate another way, viz. the design of adaptive control schemes which dynamically combine algorithms as appropriate, taking into account instance-specific knowledge. We scrutinize recently proposed algorithmic approaches from the open literature. Although these have been used in various settings (e.g. lotsizing and scheduling, course scheduling), a thorough experimental investigation, comparing their performance on standard benchmark instances to that of other contemporary methods, has been lacking. Our research aims to close this gap by validating these approaches on one of the best-researched scheduling problems, viz. the resource-constrained project scheduling problem (RCPSP). We provide the results of a comprehensive computational study. We then show how to improve algorithmic effectiveness by adding a sampling stage; the arising algorithm is almost as effective as the most effective construction methods currently known, and it enjoys several additional advantages over these which facilitate the researcher's task of designing good algorithms.

Keywords: PROJECT SCHEDULING; HEURISTICS; ADAPTIVE CONTROL SCHEMES

1. Introduction

Despite the intellectual appeal of trying to find 'the best heuristic' for a problem, for most computationally intractable problems there simply exists no heuristic which performs best on all instances. Usually, a heuristic characterized as best will perform good on the majority of instances but leave 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 (Schirmer, Riesenberg 1998). A similar view is adopted 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.

We therefore begin by testing two such approaches which have been recently proposed in the literature, a randomized one (Kimms 1996) and a local search-based one (Haase 1996; Haase et al. 1998). These or similar algorithms have been used in various settings (e.g. lotsizing and scheduling, course scheduling); still a thorough experimental investigation, which would compare them to other contemporary methods by means of standard benchmark instances, has been lacking. Our research aims at closing this gap by validating the algorithms on one of the best-studied scheduling problems, the *resource-constrained project scheduling problem* (RCPSP). Our analysis reveals both approaches to be markedly less effective than well-known simpler heuristics. We demonstrate, however, that their effectiveness can be substantially improved by adding a random sampling stage. This defines a new algorithmic approach proceeding in two stages: The first, deterministic stage uses an adaptive control scheme to identify, *separately for each attempted instance*, a well-performing combination of priority rules. The second, randomized stage uses the best currently known combination of scheduling and random sampling scheme for the problem to run the composite rule found in the first stage as a sampling algorithm. This approach allows to exploit the benefits of sampling while enabling the algorithm to compose a priority rule tailored to the particular instance at hand.

Our computational results 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 state-of-the-art algorithms, which facilitate the researcher's task of designing good priority rule-based heuristics: First, by construction 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, Riesenberg 1998; Schirmer 1998a). 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 briefly recount the RCPSP as far as it is needed here. The requisite algorithmic components, viz. scheduling scheme, simple and composited priority rules, random sampling schemes, and bounding rules are covered in Section 3. Section 4 is devoted to 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. Resource-Constrained Project Scheduling

2.1. Problem Setting

The classical single-mode variant of the resource-constrained project scheduling problem can be characterized as follows: The single project consists of a number of activities of known duration; all activities have to be executed in order to complete the project. During their nonpreemptable execution the activities request renewable resources whose available amount is limited in each period by a constant capacity. Precedence relations between activities stipulate that some activities must be finished before others may be started. Table 1 summarizes the problem parameters of the RCPSP, where w.l.o.g. J, R, d_j, K_r (k_{jr}) are assumed to be positive (nonnegative) integers.

The goal is to find an assignment of periods to activities (a *schedule*) that covers all activities, ensures for each renewable resource r that in each period the total usage of r by all activities performed in that period does not exceed the per-period availability of r, respects the partial order $\angle$, and minimizes the total project length. This problem is notoriously intractable. Its optimization variant is known to be strongly **NP**-hard, even strongly **NP**-equivalent (Schirmer 1998b, pp. 10-14); indeed, assuming a deadline to be given for the project completion even its feasibility variant is strongly **NP**-complete (Garey, Johnson 1975).

Problem Parameter	Definition
d_j	Duration of activity j
J	Number of activities, indexed by j
k_{jr}	Per-period usage of renewable resource r required to perform activity j
K_r	Per-period availability of renewable resource r
R	Number of renewable resources, indexed by r
$\angle$	Partial order on the activities, representing precedence relations

Table 1: Problem Parameters of the RCPSP

2.2. Model Formulation

To simplify the formulation of the model, w.l.o.g. it is assumed that the activities 1 and J are *dummy activities*, having durations and resource requirements of zero, and that activity 1 (J) is the unique first (last) activity w.r.t. $\angle$. Also, several parameters are derived from the above problem parameters. First, in order to restrict the number of periods to be considered, let de-

note $\overline{T}$ an upper bound for the makespan of the project. Second, let denote P_j $(2 \le j \le J)$ the set of all *immediate* predecessors of activity j w.r.t. $\angle$. Third, for each activity j $(1 \le j \le J)$ earliest finish times EFT_j and latest finish times LFT_j are calculated (Kelley 1963). Finally, let A_t $(1 \le t \le \overline{T})$ denote the set of all (non-dummy) activities being active in period t and T the set of all periods in which at least one activity begins. Then, using integer variables y_j $(1 \le j \le J)$ to denote the period in which activity j is finished, the RCPSP can be couched as:

Minimize

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

subject to

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

$$\sum_{j \in A_t} k_{jr} \le K_r \qquad (1 \le r \le R;\ t \in T) \tag{3}$$

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

Minimization of the objective function (1) enforces the earliest possible completion of the last activity J and thus 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 renewable resource in each period to the available amount.

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 scheduling. 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

Two variants of scheduling schemes are usually distinguished, viz. serial and parallel ones. We restrict our attention to the serial scheduling scheme (SSS) for two reasons. First, when priority rules are applied in a deterministic manner, the SSS allows for better discrimination between good and bad rules than its parallel counterpart (Schirmer, Riesenberg 1997;

Schirmer 1997). Second, we intend to apply those composite rules identified as good in a random sampling scheme afterwards (in a second stage). Due to the neglectable computation times required per iteration (at the order of milliseconds for the instances considered here), such algorithms are commonly run for at least 100 iterations, and so they will be in the second stage. As the serial scheme is clearly dominant (Kolisch 1996b; Schirmer 1997) for higher iteration numbers, we intend to find those rules which perform good under the SSS.

The SSS divides the set of activities into three disjoint subsets or states: *scheduled, eligible,* and *ineligible.* 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. D_n is determined dynamically from

$$D_n \leftarrow \{j \mid j \notin S_n \wedge P_j \subseteq S_n\} \qquad (1 \leq n \leq N) \quad (5)$$

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 on stage N when all activities are scheduled. The SSS has been dealt with, among others, in numerous studies (cf. the literature cited in Schirmer (1997) where also a formal description is given). Note that for each feasible RCPSP-instance the SSS probes the set of active schedules (Kolisch 1996b) which always contains at least one optimal schedule.

3.2. Simple Priority Rules

We briefly introduce the priority rules to be used. Let for each activity j ($1 \leq j \leq J$) denote S_j the set of all immediate successors w.r.t. $\angle$ and $EFST_j$ ($1 \leq j \leq J$) the - dynamically updated - earliest feasible start time w.r.t. all constraints. Using this notation, the rules used can be defined as done in Table 2 where also a classification in terms of several straightforward criteria is given. The first well-known four of these rules were selected because in several studies (Davis, Patterson 1975; Alvarez-Valdés, Tamarit 1989; Ulusoy, Özdamar 1989; Boctor 1990; Kolisch 1996a; Schirmer 1997) they have been found to hold particular promise for the RCPSP. The other four were found to perform rather poorly (Schirmer 1997); they were deliberately included for reasons about to unfold later.

Extremum	Measure	Definition	Static vs. Dynamic	Local vs. Global
MIN	SLK_j	$\leftarrow LST_j - EFST_j$	D	L
MIN	LST_j	$\leftarrow LFT_j - d_j$	S	G
MIN	LFT_j	$\leftarrow LFT_j$	S	L
MAX	MTS_j	$\leftarrow \lvert \{j' \mid j \angle j'\} \rvert$	S	L
MIN	SPT_j	$\leftarrow d_j$	S	L
MIN	DRD_j	$\leftarrow \Sigma_r k_{jr} / \max \{k_{j'r} \mid j' \in D_n\}$	D	G
MIN	TRD_j	$\leftarrow \Sigma_r k_{jr}$	S	L
MIN	TRS_j	$\leftarrow \Sigma_r k_{jr} / K_r$	S	L

Table 2: Simple Priority Rules - Definition and Classification

3.3. Parameterized Composite Priority Rules

All above priority rules can be characterized as simple ones (cp. Kolisch 1995, p. 86). 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. 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 suggest that good composite rules need not neccessarily be made up from good simple rules: Ulusoy, Özdamar (1989) report that, although the rules MAX MIS (most immediate successors) and MAX TRS perform poorly as individuals, their 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

$$v(j) \leftarrow \sum_{i=1}^{I} v_i(j)^{\alpha_i} \qquad \text{extr} = \max \quad (j \in D_n) \quad (6)$$

Obviously, for I = 1 this definition includes the special case of a simple rule. For each priority rule i, the corresponding control parameter α_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 increasing 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.

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 explicitely by normalizing or scaling the priorities or implicitely 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

$$v'_i(j) \leftarrow \begin{cases} 1 - \dfrac{v_i(j)}{\max\{v_i(j') \mid j' \in D_n\}} & \text{if } \text{extr}_i = \min \\ \dfrac{v_i(j)}{\max\{v_i(j') \mid j' \in D_n\}} & \text{if } \text{extr}_i = \max \end{cases} \qquad (1 \le i \le I; j \in D_n) \quad (7)$$

If the denominator is zero, the priorities remain unchanged. Note that formula (7) 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.

$$v(j) \leftarrow \sum_{i=1}^{I} v'_i(j)^{\alpha_i} \qquad \text{extr} = \max \quad (j \in D_n) \quad (8)$$

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 a multiplicative variant (cf. Eq. 9) of composing the individual priorities. Yet, as it yielded noticeably worse results, we use the exponential one in the sequel.

$$v(j) \leftarrow \sum_{i=1}^{I} \alpha_i \cdot v'_i(j) \qquad \text{extr} = \max \quad (j \in D_n) \quad (9)$$

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 employing 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 Schemes

The idea of randomization in selecting between several alternative candidates is based on measures of attractiveness which are mapped monotonically into selection probabilities. Thus, a randomized method chooses among the available candidates according to probability values which are biased to favour apparently attractive selections. In the framework considered here, these measures are determined by priority rules. A *biased random sampling scheme* consists of a mapping p: $D_n \rightarrow [0,1]$ assigning a probability value p(j) to each candidate in the decision set. In essence, this mapping simply transforms the priority value of each candidate into a probability value. Of course, all probabilities sum to unity. To formalize the (randomized) selection once the probabilities are calculated, let denote $\vec{P}(D_n) = (p(\pi(1)),...,p(\pi(|D_n|)))$ the sequence of probability values of all candidates in a decision set D_n; w.l.o.g. we assume $\vec{P}(D_n)$ to be ordered by ascending activity index. Also, let denote $\zeta \in [0,1]$ a random number. Then, the activity j* to be selected is determined as

$$j^* \leftarrow \min \{ j \in D_n \mid j = \pi(k) \wedge \zeta \leq \sum_{i=1}^{k} p(\pi(i)) \} \tag{10}$$

One of the two schemes that we use here is the *regret-based biased random sampling* (RBRS) one presented in Drexl, Grünewald (1993). The regret value of a candidate j measures the worst-case consequence that might 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

$$v'(j) \leftarrow \begin{cases} \max V(D_n) - v(j) & \text{iff extr = min} \\ v(j) - \min V(D_n) & \text{iff extr = max} \end{cases} \qquad (j \in D_n) \tag{11}$$

and modified by

$$v''(j) \leftarrow (v'(j) + \varepsilon)^{\alpha} \qquad (j \in D_n) \tag{12}$$

where $\varepsilon \in R_{>0}$ and $\alpha \in R_{\geq 0}$; note that this parameter α is not identical to any weight parameter α_i defined in the previous section. Now the selection probabilities are derived from

$$p(j) \leftarrow \frac{v''(j)}{\sum_{j' \in D_n} v''(j')} \qquad (j \in D_n) \quad (13)$$

ε guarantees v''(j) to be nonzero; otherwise those candidates with priorities of zero could never be selected, an undesirable consequence in the presence of scarce resources. α allows to diminish or enforce the differences between the modified priorities for $\alpha < 1$ or $\alpha > 1$, respectively. Note that for $\alpha \to 0$ 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 α the difference between the highest and the second-highest modified priority increases, the probability of the highest-prioritized candidate being selected converges to one.

The second scheme (MRBRS) we use is a modification of the RBRS, in that it computes regrets from the original priorities according to (11) and modifies them according to (12). However, rather than using a constant value of ε, ε is determined dynamically. Letting denote $V'(D_n)^+$ the set of all positive transformed priorities of the candidates in D_n, i.e.

$$V'(D_n)^+ \leftarrow \{v'(j) \mid j \in D_n \wedge v'(j) > 0\} \quad (14)$$

ε can be derived from

$$\varepsilon \leftarrow \begin{cases} \min V'(D_n)^+ / \delta & \text{iff } (\exists j \in D_n)\, v'(j) > 0 \\ 1 & \text{otherwise} \end{cases} \quad (15)$$

where δ is a positive integer. Selection probabilities are again derived from (13). Thus, the above actually defines a family MRBRS/δ of sampling schemes. This scheme has been found to be particularly effective in conjunction with the SSS. Details of these schemes are discussed elsewhere (Schirmer 1997; Schirmer, Riesenberg 1997).

3.5. Bounding Rules

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

and a time window bound. Resource-based bounds have not been included since their effect has been found to be rather small (Kolisch 1995; Schirmer, Riesenberg 1998).

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

$$FT_J = EFT_J \tag{16}$$

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. ■

Serial time window bound (STWB) The fundamental idea is to decrease the latest start times of all unscheduled activities whenever an iteration finds an improved upper bound on the makespan. Let denote $\underline{FT}_J$ the makespan of the incumbent best solution and LST_j the latest start time of activity j (Kelley 1963). Whenever a feasible solution with a shorter makespan of $FT_J < \underline{FT}_J$ has been found, updated LST-values can be determined for all activities by

$$LST_j \leftarrow LST_j - \underline{FT}_J + FT_J - 1 \qquad (1 \le j \le J) \tag{17}$$

Now, the current iteration can be terminated whenever a selected activity j* would have to start outside its updated time window, i.e.

$$EFST_{j^*} > LST_{j^*} \tag{18}$$

the reason being that the incumbent partial schedule could not be completed to a full schedule with a makespan better than the best known one. By construction of the STWB, any feasible schedule will be better than the previous one since its makespan will at most equal the makespan of that one, less one. ■

We will not detail computational results for the algorithms presented with or without bounding rules since the general suitability of bounding rules for scheduling heuristics has been analyzed in depth in several studies (Kolisch 1995, pp. 120-127; Schirmer, Riesenberg 1998).

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 compo-

nents, 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, α 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), Böttcher et al. (1996), and Schirmer, Riesenberg (1997).

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 (1997) 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*

When recommendations on the instantiation of control parameters reflect the results of previous experimentation, their values are usually related to the best averages observed, 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. To demonstrate this, we reanalyzed some results from a computational study on priority rule-based algorithms for the RCPSP (Schirmer 1997). We focus on the performance of the simple rules presented above, run deterministically under the SSS, such that the rule fully specifies the algorithm used. Notice that, since we intend to expose the occurrence, not the degree of mutual dominance, for each instance the best heuristically found solution is used as a reference point. The column "Best" in Table 3 reports the number of instances for which a rule did find the best heuristic solution - an honor it may share with other rules - whereas the column "Best Alone" refers to those instances where the rule was the only one to do so. For comparison, we also include the numbers of instances solved to optimality. Even the best deterministic rule, SLK, 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 even did so alone. It stands to reason that this kind of result extends to other problems as well.

Rule	*Instances Solved*		
	Optimally	Best	Best Alone
SLK	139	242	24
LST	126	203	0
LFT	120	179	42
MTS	93	131	23
SPT	17	23	4
DRD	21	24	1
TRD	19	24	0
TRS	19	24	2

Table 3: On the Mutual Dominance of Priority Rules

We therefore advocate selecting such values by some more flexible device, capable of exploiting the outcome of previous instantiations 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* 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.

Control Scheme	Fixed in Advance	Fixed for all Instances
Fixed	yes	yes
Class-Based	yes	no
Adaptive	no	no

Table 4: Classification of Control Schemes

4.2.2. Randomized Cooling-Based Control

To our first approach we refer as a *randomized cooling control scheme* (RCCS). It is an iterative control scheme which determines the values for each parameter as a function of three quantities, namely the so far best-performing value of that parameter, a random value, and a measure indicating how successful the last value was. The scheme terminates after a specified number Z of iterations. The attraction of this approach lies in the ease with which a - albeit restricted - learning capability can be added to a parameterized algorithmic scheme.

For each α_i, let denote α_i^r a random value from its domain and α^*_{iz} that value of α_i which produced the best solution in the z ($1 \le z \le Z$) iterations performed so far. Then the value α_{iz} of α_i in iteration z is determined as a convex combination of these two values, i.e.

$$\alpha_{iz} \leftarrow (1 - \beta_{iz})\, \alpha^*_{i,z-1} + \beta_{iz} \cdot \alpha_i^r \qquad (1 \le i \le I;\ 1 \le z \le Z) \quad (19)$$

where $\beta_{iz} \in [0,1]$. Clearly, at the endpoints of the one-dimensional parameter space of β_{iz}, we select purely random values α_i^r if $\beta_{iz} = 1$ and locally optimal values α^*_{iz} if $\beta_{iz} = 0$. Initially, of course, we set $\alpha_{i1} \leftarrow \alpha_i^r$ by setting $\beta_{i1} \leftarrow 1$.

The construction of this combination intends to eventually steer the process towards a steady state, i.e. to make it converge for $Z \rightarrow \infty$ to a local optimum. Thus, β is kept constant as long as this continues to produce better solutions; otherwise β is decreased in order to favor the selection of values close to the so far bestperforming ones. Essentially, this is done by counting the number of iterations, denoted by Ψ, which found a solution bettering the incumbent

best one, and taking its reciprocal. Now, using FT^*_{JZ} to denote the best project makespan constructed in all z iterations and setting $\beta_{i1} \leftarrow 1$, β_{iZ} can be calculated from

$$\beta_{iz} \leftarrow \begin{cases} 1/(\gamma \cdot \Psi) & \text{iff } FT^*_{J,z-1} = FT^*_{J,z-2} \wedge \gamma \cdot \Psi \geq 1 \\ \beta_{i,z-1} & \text{iff } FT^*_{J,z-1} < FT^*_{J,z-2} \end{cases} \qquad (1 \leq i \leq I;\ 3 \leq z \leq Z) \quad (20)$$

where $\gamma \in R_{>0}$. As the quotient in (20) decreases monotonically, the values of β_{iZ} converge to zero which provides a trajectory along which each α_{iZ} moves monotonically closer to some local optimum α^*_Z. The parameter γ allows to adjust the rate of convergence: $0 < \gamma < 1$ implies a slower, $\gamma > 1$ a faster convergence than the unadjusted rate $1 / \Psi$. The condition $\gamma \cdot \Psi \geq 1$ serves to ensure that $\alpha \in [0,1]$. The reference to cooling in the naming of this control scheme is motivated, of course, by its kinship to schemes used to control the performance of simulated annealing algorithms (Kirkpatrick et al. 1983)[1].

We should emphasize at this point that we need to distinguish between two kinds of control parameters: On one hand, those of the solution method applied (here the α_i of the composite rule-based scheduling approach), on the other hand those of the control scheme identifying appropriate values for the former ones (here the parameter γ of the RCCS). For the sake of distinction, one might refer to the latter as control scheme parameters. While replacing one kind of control parameter by another might seem a futile exercise, let us point out that a meaningful control scheme will of course use less parameters than the actual solution method; e.g., the RCCS requires only one parameter to control the instantiation of an arbitrary number I of parameters. Good control schemes, thus, will be intelligently devised to free the user from managing a multitude of parameters while - possibly - allowing him some kind of high-level or aggregate control via one or two control scheme parameters.

Let *I* denote an instance of the RCPSP. Also, let the function call SchedulingAlgorithm(I, $\vec{\alpha}_z$) refer to an implementation of a deterministic scheduling heuristic which, based upon the vector $\vec{\alpha}_z$ in iteration z, employs a composite rule as defined in (8) and returns the makespan FT_J of the schedule constructed for *I*. Finally, let FT* denote the best makespan found so far. Now, an algorithmic description of the scheme can be given as done in Table 5.

---

[1] A commonly used cooling scheme for simulated annealing algorithms is $\alpha_z \leftarrow \beta \cdot \alpha_{z-1}$ $(1 \leq z \leq Z)$ where $\beta < 1$ (cp. e.g. Kirkpatrick et al. 1983; Wilhelm, Ward 1987).

Initialization

for each[2] $(1 \le i \le I)$
  $\beta_{i1} \leftarrow 1$
  $\beta_{i2} \leftarrow 1$
  draw α_{i1} **randomly from** [0,1]
  $\alpha^*_{i1} \leftarrow \alpha_{i1}$
endfor
$FT^*_1 \leftarrow$ SchedulingAlgorithm(I, $\vec{\alpha}_1$)
$\Theta \leftarrow 1$

Execution

for each $(1 \le i \le I)$
  draw $\alpha_{i2}{}^r$ **randomly from** [0,1]
  calculate α_{i2} **according to** (19)
endfor
$FT_J \leftarrow$ SchedulingAlgorithm(I, $\vec{\alpha}_2$)
if $(FT_J < FT^*)$
  $FT^* \leftarrow FT_J$
  $\vec{\alpha}^* \leftarrow \vec{\alpha}$
  $\Theta \leftarrow \Theta+1$
endif
for $z \leftarrow 3$ **to** Z
  for each $(1 \le i \le I)$
    calculate β_{iz} **according to** (20)
    draw $\alpha_{iz}{}^r$ **randomly from** [0,1]
    calculate α_{iz} **according to** (19)
  endfor
  $FT_J \leftarrow$ SchedulingAlgorithm(I, $\vec{\alpha}_z$)
  if $(FT_J < FT^*)$
    $FT^* \leftarrow FT_J$
    $\vec{\alpha}^* \leftarrow \vec{\alpha}$
    $\Theta \leftarrow \Theta+1$
  endif
endfor

Result

For each priority rule i $(1 \le i \le I)$, α^*_i denotes that control parameter value which produced the best solution; FT^* denotes the corresponding objective function value. ■

Table 5: Randomized Cooling-Based Control Scheme

We illustrate the procedure for I = 2 priority rules in Figure 1, after 5, 10, and 20 iterations. Each symbol "·" represents an evaluated α-vector and thus a solution for the instance at hand; the symbol "•" marks the respective best vector in whose vicinity the search for good vectors (and solutions) is intensified.

[2] We use the construct "**for each** ..." whenever the order in which the respective variable is instantiated is irrelevant for the algorithmic outcome, and the construct "**for** ... **to** ..." otherwise.

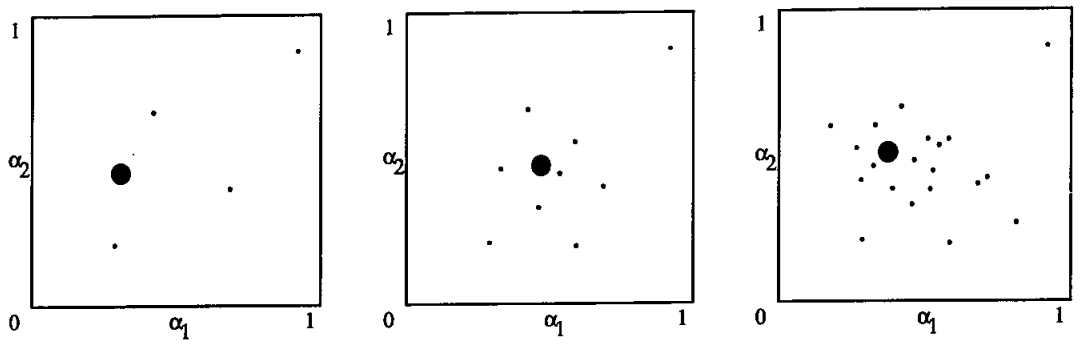

Figure 1: Exemplary Application of Randomized Cooling-Based Control Scheme

4.2.3. Local Search-Based Control

The second approach we propose is a *local search control scheme* (LSCS), which systematically applies and appraises different control parameter values (Haase 1996; Haase et al. 1998) in order to identify promising ones, hopefully guiding the computational effort to promising regions of the parameter and thus the solution space. 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 spans 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 space $[0,1]^I$, is refined to ever smaller subspaces. The number of gridpoints, which remains constant, is determined by an integral control parameter, the *grid granularity* σ, in the following way. Let for each iteration[3] 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* Δ_i of rule i is defined as

$$\Delta_i \leftarrow (U\alpha_i - L\alpha_i) / \sigma \qquad (1 \le i \le I) \quad (21)$$

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 α_1 by its grid width Δ_1 and constructs another solution, and so forth, until eventually $\alpha_1 = U\alpha_1$. Then α_1 is reset to $L\alpha_1$, and α_2 is incre-

[3] As the iteration index is not necessary in this section, we shall suppress it for brevity of presentation.

mented by Δ_2. When also α_2 has reached its upper bound, it is reset to its lower bound while α_3 is incremented by Δ_3 and so forth, until eventually all α_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 α^*_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

$$L\alpha_i \leftarrow \max\{0, \alpha^*_i - \Delta_i \cdot (\sigma-1) / \sigma\} \qquad (1 \le i \le I) \quad (22)$$

$$U\alpha_i \leftarrow \min\{1, \alpha^*_i + \Delta_i \cdot (\sigma-1) / \sigma\} \qquad (1 \le i \le I) \quad (23)$$

the grid widths Δ_i of all rules are updated according to (21), and the next iteration starts.

An algorithmic description of the scheme is given in Table 6. We illustrate the procedure for I = 2 priority rules and a grid granularity of $\sigma = 3$. Figure 2 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.

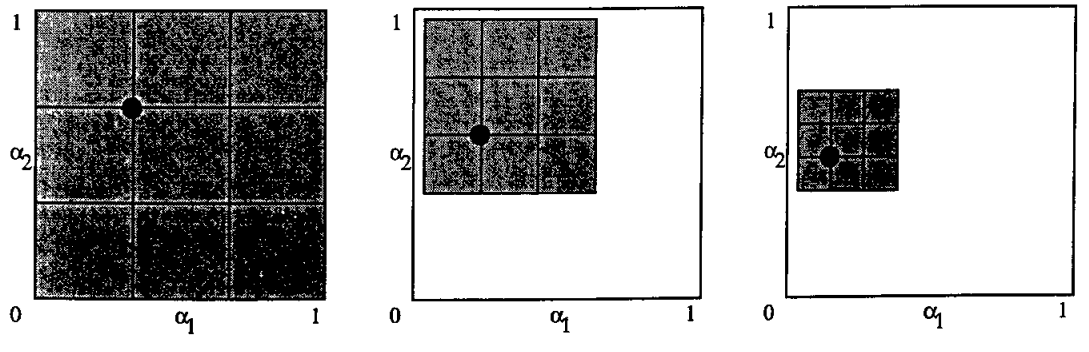

Figure 2: Exemplary Application of Local Search-Based Control Scheme

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 Δ_i **according to** (21)
  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** (22)
    calculate $U\alpha_i$ **according to** (23)
  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

For each priority rule i $(1 \le i \le I)$, α^*_i denotes that control parameter value which produced the best solution; FT^* denotes the corresponding objective function value. ■

Table 6: Local Search-Based Control Scheme

5. Experimental Analysis

5.1. Experimental Setting

5.1.1. Test Instances and Reference Solutions

As a test bed, we used the KSD-instance set J30 generated with ProGen (Kolisch, Sprecher 1997). Each instance comprises four renewable resources and 30 non-dummy activities, having a nonpreemptable duration of between one and ten periods and between one and three successors and predecessors each. Systematically varied design parameters for these instances are the network complexity (NC), the resource factor (RF), and the resource strength (RS). NC is defined as the average number of non-redundant arcs per activity, RF determines the number of resources that are requested by each activity, and RS expresses resource scarcity measured between minimum and maximum demand. A more comprehensive characterization is given in Kolisch et al. (1995). The respective parameter levels used are NC $\in$ {1.5, 1.8, 2.1}, RF $\in$ {0.25, 0.5, 0.75, 1.0}, and RS $\in$ {0.2, 0.5, 0.7, 1.0}. For each instance cluster defined by a combination of these parameters, J30 contains ten instances, for a total of 480 instances. Note that those 120 instances where RS = 1.0 are trivially solvable by the earliest start time schedule. The optimal reference solutions were computed with the exact algorithm of Demeulemeester, Herroelen (1992, 1997).

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 with large and small sets, we defined several such rule sets. Our classification of rules as good or bad draws on several studies (Boctor 1990; Valls et al. 1992; Kolisch 1996b; Schirmer 1997; Schirmer, Riesenberg 1997), measuring effectiveness in terms of average deviation from optimum when applying them deterministically under the SSS; exemplary results taken from Schirmer, Riesenberg (1997) are shown in Table 7.

Good Rules				*Bad Rules*			
SLK	LST	LFT	MTS	SPT	DRD	TRD	TRS
5.58%	6.56%	7.44%	8.74%	22.80%	23.13%	23.38%	24.01%

Table 7: Effectiveness and Classification of Priority Rules

Using this classification, we defined seven rule sets which are blended as shown in Table 8.

Rule Sets	I	Characterization	Priority Rules
1	8	4 good, 4 bad rules	SLK, LST, LFT, MTS, SPT, DRD, TRD, TRS
2	6	3 good, 3 bad rules	SLK, LST, LFT, DRD, TRD, TRS
3	4	2 good, 2 bad rules	SLK, LST, TRD, TRS
4	2	1 good, 1 bad rules	SLK, TRS
5	4	3 good, 1 bad rules	SLK, LST, LFT, TRS
6	4	1 good, 3 bad rules	SLK, DRD, TRD, TRS
7	4	4 good rules	SLK, LST, LFT, MTS
8	3	3 good rules	SLK, LST, LFT
9	2	2 good rules	SLK, LST
10	4	4 bad rules	SPT, DRD, TRD, TRS
11	3	3 bad rules	DRD, TRD, TRS
12	2	2 bad rules	TRD, TRS

Table 8: 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, $\vec{\alpha}_Z$), we always use the serial scheduling scheme as has been covered earlier.

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 instance and algorithm, the outcome of each run in terms of both effectiveness and efficiency. Effectiveness is determined for the best schedule found in a run, measuring its deviation from the optimum as a percentage. Efficiency usually 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 Pascal, running on a Pentium 266 personal computer with 32 MB RAM under Windows 95.

5.2. *Randomized Cooling-Based Control*

5.2.1. *Experimental Design*

The factors examined here are control parameter γ, rule set cardinality I, rule set composition Θ, and number of iterations Z.

5.2.2. *Effect of Gamma*

In order to evaluate the effect of the control parameter γ on effectiveness, we conducted an experiment in which each rule set was applied with different levels of γ, with the sample size Z fixed to 100 iterations. Results are shown in Table 9 where the best value of γ for each rule set is underlined.

Rule Set	I	Θ	*Gamma*				
			0.25	0.5	0.75	1	2
1	8	50%	<u>3.62%</u>	3.83%	5.19%	6.11%	8.26%
2	6	50%	5.26%	<u>5.04%</u>	6.78%	10.34%	14.26%
3	4	50%	5.74%	<u>5.68%</u>	6.90%	8.71%	12.45%
4	2	50%	<u>6.88%</u>	7.00%	7.03%	8.35%	10.32%
5	4	75%	4.35%	3.90%	<u>3.89%</u>	4.59%	6.71%
6	4	25%	<u>6.87%</u>	7.52%	10.14%	13.81%	16.87%
7	4	100%	<u>3.91%</u>	4.78%	5.23%	5.75%	6.04%
8	3	100%	2.91%	2.87%	<u>2.84%</u>	2.96%	3.32%
9	2	100%	3.39%	<u>3.26%</u>	3.47%	3.54%	3.82%
10	4	0%	<u>7.48%</u>	8.77%	13.66%	18.06%	20.68%
11	3	0%	<u>8.86%</u>	10.62%	18.08%	21.90%	22.28%
12	2	0%	<u>10.85%</u>	12.02%	17.45%	23.02%	23.26%

Table 9: Effect of Gamma - Deviations (RCCS)

The only recommendation to be derived is that for bad rule sets γ should be set to a small value, such as 0.25 (see the results for rule sets 10 through 12 with Θ = 0%, and rule set 6 with Θ = 25%). Recalling that smaller values of γ imply a slower convergence to a steady state, the interpretation of this finding is obvious: the smaller the proportion of good rules, the more important a slower convergence, as too fast a convergence tends to trap the search process in bad local optima early. Vice versa, one is lead to conjecture that large values of Θ favor larger settings of γ. While this conjecture is mostly corroborated, the results of rule set 7 are not exactly in line with this; we tend to attribute this aberration to the fact that here the effect of γ is the smallest over all sets, and to the high variability of random sampling in general. The cardinality of the rule sets employed does not show any effect of the best choice of γ.

Further, we have analyzed the interaction between γ and the sample size Z. Clearly, we would like the RCCS to keep the middle ground between two extremes, viz. getting trapped too early in local optima on one hand, and spending too many iterations on a purely random search on the other. It therefore seems reasonable that for small iteration numbers smaller values of γ should be preferable: as these imply a more random search, they would allow to sample a larger part of the solution space, instead of concentrating the search in the vicinity of local optima. In Table 10, we detail the results accordingly, underlined entries reflect the best γ-value per rule set and iteration number. For brevity, we restrict the presentation to $\gamma \in \{0.25, 0.5, 0.75\}$, as the picture would not change if we included the other values.

Rule Set	Gamma	*Sample Size* 40	70	100	Rule Set	Gamma	*Sample Size* 40	70	100
1	0.25	4.30%	3.82%	3.62%	7	0.25	4.58%	4.14%	3.91%
	0.5	4.63%	4.07%	3.83%		0.5	5.18%	4.94%	4.78%
	0.75	6.05%	5.53%	5.19%		0.75	5.69%	5.42%	5.23%
2	0.25	6.57%	5.61%	5.26%	8	0.25	3.40%	3.08%	2.91%
	0.5	6.80%	5.53%	5.04%		0.5	3.40%	3.03%	2.87%
	0.75	8.82%	7.39%	6.78%		0.75	3.38%	3.03%	2.84%
3	0.25	7.53%	6.34%	5.74%	9	0.25	3.87%	3.54%	3.39%
	0.5	7.48%	6.24%	5.68%		0.5	3.92%	3.46%	3.26%
	0.75	8.99%	7.77%	6.90%		0.75	3.92%	3.68%	3.47%
4	0.25	8.22%	7.36%	6.88%	10	0.25	9.42%	8.09%	7.48%
	0.5	8.47%	7.38%	7.00%		0.5	11.53%	9.67%	8.77%
	0.75	8.58%	7.49%	7.03%		0.75	16.51%	14.56%	13.66%
5	0.25	5.40%	4.76%	4.35%	11	0.25	10.77%	9.26%	8.86%
	0.5	5.29%	4.30%	3.90%		0.5	14.25%	11.81%	10.62%
	0.75	5.04%	4.26%	3.89%		0.75	20.08%	18.71%	18.08%
6	0.25	8.41%	7.38%	6.87%	12	0.25	13.31%	11.90%	10.85%
	0.5	9.62%	8.06%	7.52%		0.5	15.60%	12.88%	12.02%
	0.75	13.12%	10.97%	10.14%		0.75	20.08%	18.32%	17.45%

Table 10: Effect of Gamma and Sample Size - Deviations (RCCS)

Yet, for most rule sets the above results reveal no effect of the sample size on the best choice of γ. While we still believe the effect to exist, it can at best be a marginal one because for most rule sets the best values of γ observed are from [0.25, 0.5], an interval of which we only measure the extremes (Table 9); thus more fine-grained experimentation, using more levels of γ, would be necessary to detect it. Still, due to the small effect of γ on algorithmic effectiveness, we deemed additional experimentation not to be worthwhile.

Finally, notice that good rule sets are relatively immune against adverse effects from selecting unsuited γ-values: if most rules are good, then their compositions are likely to be good as well; conversely, the greater the proportion of bad rules in a set, the greater its susceptability to unsuited γ-values. Therefore, if one is looking for a general recommendation for γ, a setting of γ

= 0.25 should be used for good and bad rule sets alike. In the sequel, we will thus concentrate on the results from employing this setting.

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 11. The results were obtained from setting $\gamma = 0.25$ and $Z = 100$. Note that for the sake of compactness also the results of the other algorithms are included, which we will explain and discuss later.

Rule Set	I	Θ	RCCS	LSCS	RLSCS
7	4	100%	3.91%	5.23%	2.95%
8	3	100%	2.91%	2.35%	1.99%
9	2	100%	3.39%	2.91%	2.32%
1	8	50%	3.62%	4.25%	2.74%
2	6	50%	5.26%	3.86%	2.75%
3	4	50%	5.74%	3.40%	2.55%
4	2	50%	6.88%	6.33%	3.39%
10	4	0%	7.48%	9.53%	3.53%
11	3	0%	8.86%	10.99%	3.64%
12	2	0%	10.85%	11.23%	3.68%

Table 11: Effect of Rule Set Cardinality

For most rule sets 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. The only exception to this finding is set 7, presumably due to the same reasons outlined earlier.

Similar to the above, we arrange in Table 12 the results pertaining to the influence of the rule set composition, given a fixed set cardinality. Again, we used settings of $\gamma = 0.25$ and $Z = 100$. Unsurprisingly, we find that increasing (decreasing) the proportion of good rules in a constant number of rules produces better (worse) results since the number of good rules is larger from which an appropriate rule can be selected for each instance.

Rule Set	I	Θ	RCCS	LSCS	RLSCS
7	4	100%	3.91%	5.23%	2.95%
5	4	75%	4.35%	2.68%	2.11%
3	4	50%	5.74%	3.40%	2.55%
6	4	25%	6.87%	6.50%	3.39%
10	4	0%	7.48%	9.53%	3.53%
9	2	100%	3.39%	2.91%	2.32%
4	2	50%	6.88%	6.33%	3.39%
12	2	0%	10.85%	11.23%	3.68%

Table 12: Effect of Rule Set Composition - Deviations

5.2.4. *Effect of Beta*

Another question engaging us was whether the idea of adapting the β_{iz} according to the number of iterations that improved the incumbent best solution is actually beneficial. We thus performed an experiment in which the value of β_{iz} was fixed to 0.5 for all iterations. The results are summarized in Table 13, juxtaposing them with those from varying β_{iz} with $\gamma = 0.25$.

Rule Set	Adapting Beta	Fixed Beta
1	3.62%	9.02%
2	5.26%	15.47%
3	5.74%	13.76%
4	6.88%	11.48%
5	4.35%	7.67%
6	6.87%	17.51%
7	3.91%	6.21%
8	2.91%	3.74%
9	3.39%	4.12%
10	7.48%	21.79%
11	8.86%	22.75%
12	10.85%	23.27%

Table 13: Effect of Adapting vs. Fixed Beta - Deviations (RCCS)

Fixing the β_{iz} makes RCCS markedly less effective. Although the results shown above were derived with a setting of $\gamma = 0.5$, this was found to hold true for any choice of γ tested. Deviations for the fixed variant generally turned out to be between one-and-a-half and two-and-a-half times larger than those observed for the adaptive one. Notice that the differences are larger for worse rule sets; in other words, the better the rule sets, the less harm can be done by unsuited values of β_{iz}. Without providing the numerical results, let us also mention that increasing the iterations from 10 to 100 cuts deviations to about half when adapting the β_{iz}, yet this ten-fold increase in sample size barely improves effectiveness when β_{iz} is kept fixed.

5.2.5. *Efficiency*

Computation times required by the RCCS approach in our experimentation are shown in Table 14; again the results from Z = 100 and $\gamma = 0.25$ are detailed over the different rule sets.

Rule Set	1	2	3	4	5	6
Avg CPU	4.74	2.98	1.75	0.93	1.92	2.04
Rule Set	7	8	9	10	11	12
Avg CPU	2.75	1.71	1.12	1.99	1.63	0.88

Table 14: Effect of Rule Sets and Gamma - CPU Times (RCCS)

5.3. *Local Search-Based Control*

5.3.1. *Experimental Design*

The factors examined here are control parameter σ, rule set cardinality I, and rule set composition Θ.

In order to demonstrate the influence of I and σ on the number of gridpoints, Table 15 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 15. Indeed, if not specified otherwise, σ is set - for each rule set cardinality I - to the maximum value lying in the non-shaded area.

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 15: Exemplary Numbers of Gridpoints

5.3.2. *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 Table 16, the effect on efficiency in Table 17. The results are representative for those obtained from varying σ on other rule sets. We might add that also the deviations for the rule sets 11 and 12 vary with varying σ, but as they decrease only in the third decimal the effect remains invisible in Table 16.

Rule Set	*Sigma*									
	1	2	3	4	5	6	7	8	9	10
7	5.47%	5.24%	5.23%							
8	4.11%	2.89%	2.64%	2.43%	2.35%					
9	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	2.91%
10	9.95%	9.71%	9.53%							
11	10.99%	10.99%	10.99%	10.99%	10.99%					
12	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%	11.23%

Table 16: Effect of Sigma - Deviations (deterministic LSCS)

Rule Set	*Sigma*									
	1	2	3	4	5	6	7	8	9	10
7	0.35	1.92	6.22							
8	0.11	0.39	0.92	1.68	2.80					
9	0.04	0.09	0.16	0.24	0.36	0.48	0.63	0.80	0.99	1.10
10	0.37	1.74	5.51							
11	0.16	0.49	1.16	2.26	3.94					
12	0.04	0.09	0.16	0.25	0.36	0.49	0.64	0.80	0.99	1.20

Table 17: Effect of Sigma - CPU Times (deterministic LSCS)

5.3.3. *Effect of Rule Set Cardinality and Composition*

For the corresponding results refer to Tables 11 and 12. In contrast to the findings for the RCCS, for the deterministic LSCS the relationship between rule set cardinality and effectiveness is not simply a linear one; rather the best results are obtained when two (rule sets 9, 3) or three (rule sets 8, 2) good rules are part of the rule set. W.r.t. the composition of the rule sets, including good rules in a rule set is clearly important, as the presence of bad rules only cannot be compensated by the LSCS. Yet we find that even rule sets with $\Theta = 100\%$ are not guaranteed to be the most effective ones, as again those rule sets which contain two or three good rules have the highest effectivity.

5.3.4. *Efficiency*

The computation times observed for the LSCS approach are reported in Table 18, for each rule set the maximum value of σ as defined in Table 15 was used.

Rule Set	1	2	3	4	5	6
Avg CPU	11.28	1.63	5.36	1.13	5.87	5.41
Rule Set	7	8	9	10	11	12
Avg CPU	6.26	3.62	1.62	9.36	5.11	2.03

Table 18: Effect of Rule Sets - CPU Times (deterministic LSCS)

Note that due to the use of 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 19.

Rule Set	1	2	3	4	5	6
Avg Schedules	321	74	334	179	307	382
Rule Set	7	8	9	10	11	12
Avg Schedules	344	231	136	413	359	202

Table 19: Effect of Rule Sets - Generated Schedules (deterministic LSCS)

5.4. *Randomized Local Search-Based Control*

5.4.1. *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 1997). In this section, we follow up on the question whether the same improvements can also be reaped for adaptive control schemes. Let us mention here that we deliberately decided to add a second, randomized stage to the LSCS only, as the RCCS was found to hold less promise for improvement; we refer to the two-stage LSCS as RLSCS. The factors examined in the following are control parameter σ, rule set cardinality I, and rule set composition Θ.

5.4.2. *Effect of Sigma*

In Tables 20 and Table 21 we summarize the results for rule sets 7 - 12, using all applicable σ-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	3.12%	2.96%	2.95%							
8	2.81%	2.22%	2.11%	2.02%	1.94%					
9	3.61%	3.61%	3.61%	3.60%	3.60%	3.61%	3.62%	3.62%	3.67%	2.32%
10	3.65%	3.66%	3.53%							
11	3.60%	3.62%	3.62%	3.64%	3.64%					
12	3.61%	3.61%	3.61%	3.60%	3.60%	3.61%	3.62%	3.62%	3.67%	3.68%

Table 20: Effect of Sigma - Deviations (randomized LSCS)

Rule Set	*Sigma*									
	1	2	3	4	5	6	7	8	9	10
7	1.36	2.94	7.25							
8	1.02	1.33	1.87	2.64	3.77					
9	0.68	0.73	0.80	0.88	0.99	1.12	1.27	1.44	1.63	1.83
10	1.75	3.12	6.90							
11	1.32	1.65	2.32	3.42	5.39					
12	0.68	0.73	0.80	0.89	1.00	1.12	1.27	1.44	1.63	1.84

Table 21: 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.

5.4.3. *Effect of Rule Set Cardinality and Composition*

For exploring the effect of the control parameters α and δ, we first of all ran nine experiments where $\alpha \in \{10, 100, 1000\}$ and $\delta \in \{10, 100, 1000\}$, values which are in line with similar experimentation in the context of sampling methods using simple priority rules (Schirmer 1997). It turned out, however, that here the influence of α and δ is negligible: the standard deviation of the average deviations, computed over all nine combinations of α and δ, is never larger than 0.15%, and for 11 of the 12 rule sets ranges between 0.02 and 0.04%. We thus confine ourselves in the sequel to the results from using $\alpha = 1$ and $\delta = 10$.

For the results refer again to Tables 11 and 12. Evidently, 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 two or three good rules sport the highest effectiveness.

Another interesting question is how often the additional randomization stage actually improves effectiveness. Table 22 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 α and δ.

Rule Set	1	2	3	4	5	6
Instances	108	97	80	178	60	170
Rule Set	7	8	9	10	11	12
Instances	136	41	59	257	277	277

Table 22: Effect of Randomization - Improved Instances

5.4.4. *Efficiency*

The computation times measured for the randomized LSCS approach are given in Table 23.

Rule Set	1	2	3	4	5	6
Avg CPU	13.58	3.63	7.07	2.00	7.78	7.03
Rule Set	7	8	9	10	11	12
Avg CPU	7.46	5.08	3.01	12.06	6.83	3.39

Table 23: Effect of Rule Sets - CPU Times (randomized LSCS)

5.5. *Comparison with Other Algorithms*

In order to assess the merits of the algorithmic approaches dicussed in this paper, we compare them to the most effective sampling-based construction methods currently available for the RCPSP (Kolisch, Drexl 1996; Schirmer, Riesenberg 1997; Schirmer 1998a). The results for the adaptive search procedure of Kolisch, Drexl (1996), which were not reported for all 360 instances in the original paper, were derived from reimplementing the procedure. For comparison, we also include a number of simple priority rule-based construction algorithms. Let us mention that we deliberately exclude more evolved metaheuristic algorithms, using concepts such as tabu search or population genetics, from the comparison, for two reasons. One, such methods are more complicated by design and thus less efficient than the algorithms considered here. Two, the benefits of tabu search or genetic algorithms usually come to fully bear only when larger samples of solutions are drawn; recent studies thus examine samples of be-

tween 1000 and 5000 schedules (cf. Kolisch, Hartmann 1998; Hartmann, Kolisch 1998). Yet, as we look for fast methods we intend to use much smaller samples. Also, exclusion of such metaheuristics is not too severe a restriction, since the best construction method listed below outperforms several recent metaheuristic search algorithms even for large sample sizes (Schirmer 1998a). For the sake of simplicity, we restrict the presentation of results for the adaptive control algorithms to rule set 8. As the first stage of the LSCS-algorithms usually performs between 100 and 500 iterations (cf. Table 19), we provide the results from both these sizes in Table 24.

Algorithm	Reference	*Sample Size*	
		100	500
Case-based reasoning	Schirmer (1998a)	1.53%	1.04%
Adaptive search	Kolisch, Drexl (1996)	1.85%	1.20%
RLSCS, rule set 8	—	1.94%	
SSS, MRBRS/10, LST	Schirmer, Riesenberg (1997)	1.95%	1.31%
SSS, MRBRS/10, LFT	Schirmer, Riesenberg (1997)	1.99%	1.33%
LSCS, rule set 8	—	2.35%	
PSS, RBRS, WCS	Kolisch (1996a)	2.36%	2.08%
PSS, RBRS, LFT	Kolisch (1996b)	2.40%	2.09%
RCCS, rule set 8	—	2.91%	2.76%

Table 24: Comparison of Several Construction Methods - Deviations

We find the RCCS approach to be the least effective of all algorithms considered, even with the best rule set found in our experimentation. The LSCS is much more effective, especially when augmented by a randomization stage; its performance is comparable to that of other construction methods, which however will tend to require less iterations to attain the same performance. State-of-the-art sampling-based methods still outperform the algorithms considered here.

6. Summary and Conclusions

Similar to other computationally intractable problems, findings suggest that there is no single heuristic which performs best on all possible instances of the RCPSP. Therefore, adaptive control schemes aim at identifying good algorithms from an unsorted collection of good and bad algorithms, using specific knowledge on the particular instance at hand which is acquired dynamically as the schemes proceed. Such schemes may be seen as lending a learning capability to parameterized algorithms; the promise associated with them is that heuristics tailored to each instance anew will prove effective even without the extensive experimentation required by more conventional approaches of identifying heuristics which perform well over a wide range of instances. We thus aim at closing the performance gap between construction

and search methods - measured i.t.o. both effectiveness and efficiency - by developing algorithms which use some not too complicated search-based mechanism to select well-suited construction methods for each instance tackled. As an outcome, we hope for methods which are slightly less efficient than pure construction algorithms but more so than search ones, while at the same time being more effective than pure construction and not too much less so than evolved metaheuristics.

We have scrutinized two adaptive control schemes recently proposed in the literature. Applying these approaches to the RCPSP, we find the RCCS to be too simple to be of much value, even when furnished with the best rule set found in our experimentation. The RLSCS, i.e. the LSCS combined with a random sampling scheme, achieves an effectiveness comparable to that of other good construction methods; it is hampered by a lower efficiency, though, as it clearly requires more than 100 iterations with most rule sets to achieve this effectiveness (cf. Table 19).

Even if the RLSCS is not as effective as the most recent construction heuristics for the RCPSP, adaptive control schemes should not be disregarded prematurely, on the basis of effectiveness alone. Such a focus often 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). In particular evaluating algorithms on problems with such a long history of more and more efficient heuristics as the RCPSP clearly puts new algorithms at a disadvantage if the outcome is considered in terms of effectiveness alone. 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 previous knowledge on their effects otherwise required to devise good algorithms. Second, especially the RLSCS is relatively immune against the adverse effects of bad rule sets, thus diminishing the importance of knowledge on what constitutes good rules. This is evidenced by the rather good performance of the RLSCS even with bad rule sets 10 - 12 when compared to that of simple bad rules (cf. Tables 11 and 7). Third, they are simple and easy to implement.

While researchers eventually look for algorithms which are both robust and effective, robustness is often more desirable when having to solve newly arising problems where not much is known on which algorithms are suitable for which instance classes. Only later, after some substantial amount of experimental research has taken place, the search for better algorithms shifts towards a quest for more effectiveness. As adaptive control schemes do not presuppose any previous expertise, we therefore advocate using them to obtain good solutions for instances of previously untackled problems where experimental expertise, which would allow to obtain better solutions, is not yet available.

Future work, in order to further unlock the potential of adaptivity in control schemes, should be directed at 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 and to intensify the search around all of these in a breadth-first fashion, continuing with the best candidate then. 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, $\vec{\alpha}_Z$) by a number of randomized iterations, where the best makespan found is returned. While these refinements still keep the algorithm within the realm of local search-based control schemes, the basic idea of adaptivity easily allows for more evolved control schemes employing other 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. We also gratefully acknowledge the excellent research assistance of Carsten Scholz and the able programming of Bashir Krenawi and Sven Riesenberg. Partial 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.

BOCTOR, F.F. (1990), "Some efficient multi-heuristic procedures for resource-constrained project scheduling", European Journal of Operational Research 49, pp. 3-13.

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 (1996), "Project scheduling under partially renewable resource constraints", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 398.

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.

DEMEULEMEESTER, E. AND W.S. HERROELEN (1992), "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem", Management Science 38, pp. 1803-1818.

DEMEULEMEESTER, E. AND W.S. HERROELEN (1997), "New benchmark results for the resource-constrained project scheduling problem", Management Science 43, pp. 1485-1492.

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

GAREY, M.R. AND D.S. JOHNSON (1975), "Complexity results for multiprocessor scheduling under resource constraints", SIAM Journal on Computing 4, pp. 397-411.

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 (1998), "The course scheduling problem at Lufthansa Technical Training", European Journal of Operational Research 110, pp. 441-456.

HARTMANN, S. AND R. KOLISCH (1998), "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 476.

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. (1997), "Fallbasiertes Schließen auf Methoden zur Produktionsplanung", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 433 (in German).

KIRKPATRICK, S., C. GELATT, JR., AND M.P. VECCHI (1983), "Optimization by simulated annealing", Science 220, pp. 671-680.

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

KOLISCH, R. (1996a), "Efficient priority rules for the resource-constrained project scheduling problem", Journal of Operations Management 14, pp. 179-192.

KOLISCH, R. (1996b), "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.

KOLISCH, R. AND S. HARTMANN (1998), "Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis", to appear in: *Handbook on recent advances in project scheduling*, J. Weglarz (ed.), Kluwer, Amsterdam.

KOLISCH, R. AND A. SPRECHER (1997), "PSPLIB - A project scheduling problem library", European Journal of Operational Research 96, pp. 205-216.

KOLISCH, R., A. SPRECHER, AND A. DREXL (1995), "Characterization and generation of a general class of resource-constrained project scheduling problems", Management Science 41, pp. 1693-1703.

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. (1997), "Advanced biased random sampling in serial and parallel scheduling, Research Report, Universität Kiel.

SCHIRMER, A. (1998a), "Case-based reasoning and improved adaptive search for project scheduling", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 472, www.wiso.uni-kiel.de/pub/operations-research/wp472.ps.

SCHIRMER, A. (1998b), *Project scheduling with scarce resources - Models, methods, and applications*, unpublished Ph.D. thesis, Christian-Albrechts-Universität zu Kiel.

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, www.wiso.uni-kiel.de/pub/operations-research/wp456.ps.

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, www.wiso.uni-kiel.de/pub/operations-research/wp471.ps.

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.

VALLS, V., M.A. PEREZ, AND M.S. QUINTANILLA (1992), "Heuristic performance in large resource-constrained projects", Working Paper, Departament D'Estadistica I Investigacio Operativa, Universitat de Valencia, Spain.

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.

WILHELM, M.R. AND T.L. WARD (1987), "Solving quadratic assignment problems by simulated annealing", IIE Transactions 19, pp. 107-119.

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.