Open access • Journal Article • DOI:10.1002/(SICI)1520-6750(200004)47:3<201::AID-NAV2>3.0.CO;2-L

Case-based reasoning and improved adaptive search for project scheduling

Andreas Schirmer, Andreas Schirmer

Institutions: University of Kiel, Lufthansa

Published on: 01 Apr 2000 - Naval Research Logistics (John Wiley & Sons, Inc.)

Topics: Heuristics, Dynamic priority scheduling, Case-based reasoning and Applications of artificial intelligence

Schirmer, Andreas

Case-based reasoning and improved adaptive search for project scheduling

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 472

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

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 472

Case-Based Reasoning and Improved Adaptive Search for Project Scheduling

Schirmer

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 472

Case-Based Reasoning and Improved Adaptive Search for Project Scheduling

Schirmer

April 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 +49-431-880-15 31
E-Mail schirmer@bwl.uni-kiel.de
http://www.wiso.uni-kiel.de/bwlinstitute/prod
ftp://www.wiso.uni-kiel.de/pub/operations-research

Contents

Figures

Tables

Abstract: Most scheduling problems are notoriously intractable, so the majority of algorithms for them are heuristic in nature. Priority rule-based methods still constitute the most important class of these heuristics. Of these, in turn, parameterized biased random sampling methods have attracted particular interest, due to the fact that they outperform all other priority rule-based methods known. Yet, even the 'best' such algorithms are unable to relate to the full range of instances of a problem: usually there will exist instances on which other algorithms do better. We maintain that asking for the one best algorithm for a problem may be asking too much. The recently proposed concept of *control schemes*, which refers to algorithmic schemes allowing to steer parameterized algorithms, opens up ways to refine existing algorithms in this regard and improve their effectiveness considerably. We extend this approach by integrating heuristics and case-based reasoning (CBR), an approach that has been successfully used in artificial intelligence applications. Using the resource-constrained project scheduling problem as a vehicle, we describe how to devise such a CBR system, systematically analyzing the effect of several criteria on algorithmic performance. Extensive computational results validate the efficacy of our approach and reveal a performance similar or close to state-of-the-art heuristics. In addition, the analysis undertaken provides new insight into the behaviour of a wide class of scheduling heuristics.

Keywords: PROJECT SCHEDULING; HEURISTICS; RANDOM SAMPLING; CASED-BASED REASONING

1. Introduction

Most combinatorial scheduling problems belong to the class of strongly **NP**-equivalent problems (Garey, Johnson 1979) for which the state of the art still has only exponential answers to offer when it comes to the question of exact algorithms. Hence, the majority of scheduling algorithms known are heuristic in nature. Despite their age, priority rule-based methods are still the most important ones of these. Kolisch (1996b) gives several arguments for their popularity. First, they are straightforward which makes them easy to implement; indeed, most commercial scheduling programs rely on them (De Wit, Herroelen 1990; Kolisch 1997). Second, they are computationally inexpensive in terms of computer time and memory required, making them amenable even for large instances. Third, their very inexpensiveness allows to integrate them as fast "subroutines" into more complex metaheuristic algorithms (Storer et al. 1992; Bean 1994; Leon, Ramamoorthy 1995; Lee, Kim 1996; Özdamar 1996; Naphade et al. 1997; Hartmann 1997). Of the priority rule-based heuristics, parameterized sampling methods have attracted particular interest, since they currently outperform all other priority rule-based methods known (Kolisch 1996b); parameterized here indicates that such algorithms possess (one or more) control parameters which allow to steer their processing. Recent work on the resource-constrained project scheduling problem (RCPSP) has shown that appropriately instantiating these parameters can significantly improve algorithmic effectiveness (Kolisch, Drexl 1996; Schirmer, Riesenberg 1998). Indeed, the adaptive search procedure of Kolisch, Drexl (1996) may be seen as a first step into this direction.

The problem solving approach of case-based reasoning (CBR), which originated in the field of artificial intelligence, uses experience gleaned from past cases (here: instances) to solve new ones. Although this approach was not originally intended for mathematical problems, some

work has recently appeared on the application of CBR to operations research (OR) problems. In this paper, we consider ways to integrate CBR and scheduling heuristics. Our primordial goal is to demonstrate the benefits of using CBR to select appropriate heuristics for different classes of problem instances. Although we will embark on parameterized sampling methods for the resource-constrained project scheduling problem (RCPSP) as our vehicle, the ideas discussed easily apply to other applications as well. In developing a CBR system for the RCPSP, we concentrate on the effect of several instance characteristics (resource factor, resource strength, instance size) as well as the number of iterations on the effectiveness of different algorithms. We recapitulate relevant results from other studies and extend them by new experimentation. Although some of these factors have been studied before (Kolisch, Drexl 1996; Schirmer, Riesenberg 1998), a thorough experimental analysis of all four factors has been lacking. In addition, we will analyze the algorithmic designs arising and discuss the implications for sampling methods for the RCPSP. Doing so will demonstrate that a problem can be systematically analyzed by charting the appropriateness of different methods for different classes of instances. Also, the insight gained in this process will lead to a substantial improvement of the most effective sampling method in the open literature, viz. the adaptive search procedure of Kolisch, Drexl (1996). We thus hope to contribute in two ways to the current body of knowledge. First, for the practitioner we show a way to design more effective scheduling heuristics, without having to compromise efficiency. Second, for the researcher we provide deeper insight into the behaviour of sampling heuristics.

The remainder of this paper is organized as follows. The RCPSP as well as the fundamentals of parameterized sampling methods for this problem are covered in Section 2. Section 3 describes how CBR can be applied to OR problems in general. Section 4 explicates the development of a CBR system in more detail, using the RCPSP as example. From the insight gained therein, Section 5 extracts a number of implications which apply to all sampling methods for the RCPSP in general. Section 6 details computational results and provides a comparison with state-of-the-art heuristics for the RCPSP. A summary and conclusions are given in Section 7.

2. Project Scheduling

2.1. Problem Setting

The resource-constrained project scheduling problem can be characterized as follows: The single project consists of a number J of activities of known duration d_j; all activities have to be executed to complete the project. There are a number R of renewable resources, where the amount available per period is limited by a constant capacity K_r. During each period of its nonpreemptable execution an activity j uses k_{jr} units of resources r. A partial order $\angle$, representing precedence relations between activities, stipulates that some activities must be fin-

ished before others may be started. W.l.o.g. J, R, d_j, K_r (k_{jr}) are assumed to be positive (nonnegative) integers. Let us also assume w.l.o.g. that the activities 1 and J are dummy activities, with durations and resource requirements of zero, and that activity 1 (J) is the unique first (last) activity w.r.t. $\angle$. 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.

2.2. *Priority Rule-Based Scheduling Methods*

Priority rule-based scheduling methods consist of at least two components. A *scheduling scheme* determines how a schedule is constructed by augmenting partial schedules in a stage-wise manner. On each stage, the scheme determines the set of all activities which are currently eligible for scheduling. Usually a serial and a parallel variant are distinguished; for detailed algorithmic descriptions we refer the reader to Schirmer (1997).

The *serial scheduling scheme* (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. On stage n, we refer to the set of scheduled activities as S_n and to the set of eligible activities as *decision set* D_n. Let P_j ($2 \le j \le J$) the set of all immediate predecessors of activity j w.r.t. $\angle$. Then D_n is determined dynamically from

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

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 $EFST_{j*}$. 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. For a formal description let RK_{rtn} ($1 \le r \le R$; $1 \le t \le \overline{T}$; $1 \le n \le N$) denote the remaining capacity of resource r in period t on stage n and LST_j the latest start time of activity j ($1 \le j \le J$). Then, using $EPST_{j*}$ to denote the earliest precedence-feasible start time of j*, i.e.

$$EPST_{j*} \leftarrow \max \{FT_j \mid j \angle j*\} \quad (2)$$

$EFST_{j*}$ can be computed from

$$EFST_{j*} \leftarrow \min \{\tau \mid EPST_{j*} \le \tau \le LST_{j*} \wedge k_{j*r} \le RK_{rtn} \ (1 \le r \le R;\ \tau+1 \le t \le \tau+d_{j*})\} \quad (3)$$

The *parallel scheduling scheme* (PSS) proceeds in $N \ge J$ stages, indexed by n, each of which is associated with a schedule time t_n. The scheme divides the set of activities into four disjoint

subsets or states: *active, finished, eligible*, and *ineligible*. A scheduled activity is *active* during its execution, afterwards it becomes *finished*. In contrast to the serial scheme, an activity that is neither active nor finished is called *eligible* if it could be scheduled w.r.t. precedence *and* resource constraints, *ineligible* otherwise. Consequentially, the partial schedule consists of all active and finished activities. We refer to the set of active (finished, eligible) activities on stage n, i.e. in period $t_n + 1$, as A_n (F_n, D_n). On each stage, one of two things happens. Either one eligible activity is selected, using a priority rule if more than one activity is eligible, scheduled to begin at the current schedule time, and moved from D_n to A_n; this stage is repeated as long as D_n is nonempty. Or, if the decision set D_n is empty, the schedule time t_n is incremented. This is done by setting t_n to the minimum of the finish times of all activities in A_{n-1}, then activities with a finish time equal to t_n are moved from A_n to F_n which in turn may make some formerly ineligible activities eligible, transferring them to D_n. This stage is repeated until the decision set is indeed nonempty. The scheme terminates when each activity is scheduled, i.e. is either active or finished. For a formal description of the PSS, let denote RK_{rn} ($1 \le r \le R$; $1 \le n \le N$) the remaining capacity of resource r at the schedule time t_n. D_n is derived dynamically from

$$D_n \leftarrow \{j \mid j \notin A_n \cup F_n \wedge P_j \subseteq F_n \wedge k_{jr} \le RK_{rn}\ (1 \le r \le R)\} \qquad (1 \le n \le N) \quad (4)$$

Priority rules serve to resolve selection conflicts whenever the decision set contains more than one candidate. Formally, any priority rule can be cast in terms of a mapping which calculates a numerical measure or priority value for each candidate and a dichotomical parameter extremum $\in$ {max, min} which specifies whether high or low priorities are to be favored. Ties can be broken arbitrarily, e.g. by smallest activity index (as done in the sequel) or randomly. We briefly introduce several well-known priority rules. Let for each activity j ($1 \le j \le J$) denote LFT_j the latest finish time, calculated from an upper bound $\overline{T}$ on the total project length, and $EFST_j$ the - dynamically updated - earliest feasible start time w.r.t. *all* constraints. Concerning the measures $IRSM_j$ and WCS_j, AP_n denotes the set of all pairs of nonidentical activities i and j in the decision set; E_{ij} denotes the earliest time to schedule activity j if activity i is started at t_n (for details cf. Kolisch 1996a). Then the rules can be defined as done in Table 1 where also a classification in terms of several straightforward criteria (Cooper 1976; Kolisch 1995, pp. 85-86) is given; the last criterion refers to whether the rule is applicable in both scheduling schemes or only in the parallel one. These rules were selected because they have been found to be the best-performing ones in several studies (Alvarez-Valdés, Tamarit 1989; Boctor 1990; Kolisch 1996a; Schirmer, Riesenberg 1997).

Extremum	Measure	Definition	Static vs. Dynamic	Local vs. Global	Serial vs. Parallel
MAX	MTS_j	$\leftarrow \lvert\{j' \mid j \angle j'\}\rvert$	S	L	S, P
MIN	LST_j	$\leftarrow LFT_j - d_j$	S	G	S, P
MIN	LFT_j	$\leftarrow LFT_j$	S	L	S, P
MIN	RSM_j	$\leftarrow \max\{0, t_n + d_j - LST_i \mid (i,j) \in AP_n\}$	D	G	P
MIN	$IRSM_j$	$\leftarrow \max\{0, E_{ji} - LST_j \mid (i,j) \in AP_n\}$	D	G	P
MIN	WCS_j	$\leftarrow LST_j - \max\{E_{ij} \mid (i,j) \in AP_n\}$	D	G	P

Table 1: Priority Rules - Definition and Classification

2.3. Parameterized Random Sampling Methods

Deterministic heuristics return one sole solution for an instance, even if applied several times; considering that this solution may be arbitrarily bad, determinism may be a major deficiency for heuristics. Hence, *random sampling schemes* resolve selection conflicts according to probabilities which are proportional to priority values; in other words, the probabilities are biased by the priorities. Thus, in each scheduling step any eligible job *may* be chosen but those sharing higher priorities will have a higher probability of being selected. Note also that then tie-breaking rules become obsolete as ties cannot occur. Evidence gathered in several computational studies confirms that such methods outperform traditional, deterministic approaches (Cooper 1976; Hart, Shogan 1987; Laguna et al. 1994; Drexl, Grünewald 1993; Schirmer 1997). Many of the so-arising *biased random sampling* methods are *parameterized*, possessing control parameters allowing to influence the way in which they proceed.

In the sequel, we briefly introduce two such schemes which in recent studies have been found to be the most effective ones available (Kolisch, Drexl 1996; Schirmer, Riesenberg 1997). The regret-based biased random sampling scheme (RBRS) was introduced by Drexl (1991) and Drexl, Grünewald (1993). The regret value of a candidate j measures the worst-case consequence that might arise from selecting another candidate. Let denote v(j) the priority of candidate j and $V(D_n)$ the set of priority values of all candidates in a decision set D_n. Then, the regrets are computed as

$$v'(j) \leftarrow \begin{cases} \max V(D_n) - v(j) & \text{iff extr} = \min \\ v(j) - \min V(D_n) & \text{iff extr} = \max \end{cases} \qquad (j \in D_n) \quad (5)$$

and modified by

$$v''(j) \leftarrow (v'(j) + \varepsilon)^{\alpha} \qquad (j \in D_n) \quad (6)$$

where $\varepsilon \in R_{>0}$ and $\alpha \in R_{\geq 0}$. Having determined these values, the selection probabilities are derived from

$$p(j) \leftarrow \frac{v''(j)}{\sum_{j' \in D_n} v''(j')} \qquad (j \in D_n) \quad (7)$$

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 $\alpha \to \infty$ 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.

An alteration of the RBRS leads to the modified regret-based biased random sampling scheme (MRBRS). It computes regrets from the original priorities according to (5) and modifies them according to (6); yet, rather than using a constant value, ε is determined dynamically from the candidates' modified priorities. Let denote $V'(D_n)^+$ the set of all positive transformed priority values of the candidates in D_n, i.e.

$$V'(D_n)^+ \leftarrow \{v'(j) \mid j \in D_n \wedge v'(j) > 0\} \quad (8)$$

Then ε is calculated from

$$\varepsilon \leftarrow \begin{cases} \min V'(D_n)^+ / \delta & \text{iff } (\exists j \in D_n)\ v'(j) > 0 \\ 1 & \text{otherwise} \end{cases} \quad (9)$$

where δ is a positive integer. Selection probabilities are again derived from (7). Thus, the above actually defines a family MRBRS/δ of sampling schemes (Schirmer 1997).

2.4. *Control Schemes*

Parameterized algorithms possess (one or more) *control parameters* which allow to direct the way in which they proceed. While such parameters may be only numerical in nature, we adopt a broader view and include also the choice of algorithmic components. Thus, control parameters in the scheduling algorithms examined here include the scheduling scheme, priority rule, and random sampling scheme employed. To algorithms which govern the instantiation of the control parameters we refer as *control schemes*. Two classes of control schemes will be of concern in the sequel.

Fixed control schemes (FCS) constitute the most simple class of control schemes. The attribute 'fixed' indicates that the way in which the control parameters are instantiated is prescribed in advance and for all instances alike. In particular, any control scheme which simply applies one and the same algorithm over all instances falls in this class of control schemes. Based upon the results of some previous experimentation, these precepts will reflect those control parameter values which produced the best results on average over all test instances used. FCS are e.g. used by Kolisch, Drexl (1996) and Böttcher et al. (1996) to instantiate the bias parameter α in the RBRS.

Another approach is motivated by the observation that for most computationally intractable problems there simply exists no algorithm which performs best on 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 e.g. been made 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. The adaptive search procedure of Kolisch, Drexl (1996) is built around a CCS which selects a scheduling scheme and a priority rule according to two quantities: the number of iterations to be performed and the resource strength of the instance. Schirmer, Riesenberg (1998) rely on resource factor and resource strength to determine scheduling scheme, sampling scheme, and priority rule; we will review these studies in more detail later. Also class-based control schemes require extensive experimentation to develop sufficient insight into the influence of the parameters on the algorithms' performance.

The advantage gained by refining a fixed to a class-based control scheme is obvious: doing so cannot deteriorate algorithmic effectiveness but will improve it in most cases. Additional attractiveness lies in the fact that this refinement may be easily accomplished by appropriately exploiting the results used to derive a FCS in the first place (for details cf. Schirmer, Riesenberg 1998). Two caveats are in order, though. Combinatorially speaking, all random sampling schemes currently used in scheduling heuristics are implemented as "sampling with replacement" which has a high result variability, especially for small samples. Therefore, the actual count of times a specific candidate is selected may well differ substantially from the corresponding expected value (Reeves 1997). This effect may be compounded if the set of test instances employed is relatively small. Failing to address this effect may produce what Reeves (1997) graphically calls "horses for courses", i.e. control schemes fine-tuned to the specifics of a non-representative experimental design. For the sake of robustness, therefore, the test instances should be as representative of the expressive power of the problem as possible, as well in quality as in quantity. Also, selecting a specific algorithm A to achieve minimal gains in terms of effectiveness, while another algorithm B consistently dominates on all adjacent in-

stance classes should be avoided; most likely, the good performance of algorithm A would be due to the above sampling error.

3. Integrating Case-Based Reasoning and Operations Research

After reviewing the basic ideas of CBR, we discuss why CBR should be applied to OR problems and and how to go about it. In particular, we sketch the general design of such a system.

3.1. Fundamental Ideas

Analogical or case-based reasoning (CBR) uses - or even reuses - experience gleaned from past cases to solve new ones. Past cases, along with their solutions, are stored in a data base or library, forming an extensional representation of the experience available. Now, CBR starts by identifying relevant past cases. If one of these cases is sufficiently similar to the one at hand, the solution stored with the most similar of these is used. Otherwise, the case may either be solved by from-the-scratch reasoning, or by modifying the solutions of the most similar cases (Pomerol 1997). The inclusion of such newly acquired cases provides a learning mechanism that allows to extend and refine past experience. The CBR approach originated in the field of artificial intelligence, to better represent human problem solving. Since people mostly reason on whole cases rather than on isolated facts (Pomerol 1997), it is hoped that CBR systems will facilitate insight into how humans arrive at solutions and will also facilitate interactive solution processes (Kraay, Harker 1996). An additional, more tangible motivation is provided by the fact that there are applications where it is easier or faster to modify past solutions than to develop new ones from the scratch. More detailed introductions to the subject matter are given by Schank (1982) and Kolodner et al. (1985).

3.2. Applicability and Motivation

Although the CBR approach was originally used to tackle ill-structured problems which expose little structure to be exploited when solving them, recently some authors have begun to apply CBR to OR problems. Kraay, Harker (1996) develop a CBR approach for what they call repetitive combinatorial optimization problems, i.e. problems where closely related instances are to be solved on a regular basis. Applications may be found e.g. in production environments where planning is done with a rolling horizon. With increasing size and complexity of the instances and with rising frequency of having to solve these it becomes more advantageous to modify existing solutions for similar instances, rather than running time-consuming solution procedures time and again.

Considering that combinatorial optimization problems are usually well structured, that they often do not have to be solved recurrently, and that for many of them an abundance of algo-

rithms exists, they seem to pose an unsuitable domain for CBR. But another, more natural approach to applying CBR in this domain evolves when the goal of deriving good solutions is replaced by the goal of selecting methods able to compute good solutions; recall that given an instance any algorithm solving it can be seen as encoding one of its solutions. Applications following this road are presented by Grolimund, Ganascia (1997), who use CBR to control operator selection in tabu search algorithms for several problems, and by Kimms (1997), who demonstrates the selection of appropriate algorithms for lotsizing and scheduling. When employed in this manner, the CBR-specific notions case and solution translate into the terminology of OR as problem instance and solution method.

The motivation to use CBR to identify suited algorithms arises from the recognition that w.r.t. their solvability not all instances are created equal. From a theoretical point of view, well-behaved (polynomially solvable) special cases of otherwise intractable (strongly **NP**-hard) problems exist. From a practical standpoint, one algorithm may perform better on some instances of a problem, a second algorithm on others. We will demonstrate later that this insight indeed applies to the RCPSP. When facing such problems, it would be worthwhile to know which algorithm produces good solutions for a given instance, rather than having to resort to try numerous possible methods. Therefore, we suggest to employ CBR systems as control schemes. Since we define control parameters as general as to encompass the selection of any algorithmic component, the appropriate selection of an algorithm translates to the appropriate instantiation of the control parameters. In this role, a CBR system extends the CCS approach in two directions: First, in addition to the instance characteristics used by CCS, other criteria can be called upon to devise control schemes. Second, the presence of a learning capability allows also to validate and refine already stored control schemes. Note that thus the recommendation to apply a particular algorithm cannot be fixed in advance but is generated dynamically for each instance tackled (cf. Table 2). Both these features will be shown to pave the way towards more flexible and thus more suited algorithms.

Control Scheme	Fixed in Advance	Fixed for all Instances
FCS	yes	yes
CCS	yes	no
CBR	no	no

Table 2: Control Schemes - Classification

3.3. Implementation

Several ingredients are required for the implementation of a CBR system for the purpose outlined. We will cover these in the sequel.

3.3.1. Case Library Initialization

One ingredient is the case library itself, which in principle could consist of a number of instances, each along with a reference to an appropriate algorithm. The decision for a particular algorithm will usually reflect which algorithm proved the most effective on average over all instances. Other criteria are conceivable, though; more complex criteria may include e.g. measures of efficiency, robustness, or reliability of algorithms (Crowder et al 1980; Badiru 1988; Jackson et al. 1991). We follow this practice, choosing the algorithm most effective on average, and of these the most efficient one in case of ties.

Rather than to store each instance exhaustively, we propose to divide the instances into equivalence classes, relying on characteristics which influence whether a particular instance is harder or easier to solve. As these characteristics represent problem-specific knowledge, they have to be identified either from theoretical insight (cf. e.g. Kolisch 1996b; Nübel, Schwindt 1997 for the field of project scheduling) or from preliminary experiments. For the RCPSP, two characteristic parameters widely used are the resource factor (RF) and the resource strength (RS): RF determines the number of resources that are requested by each activity, and RS expresses the scarcity of the resources present (Kolisch et al. 1995). Although these are not the only such characteristics, we will demonstrate later that they allow to gauge instance tractability to a large extent. Note that such characteristics should be easily computable for any given instance. Now, instead of each previously solved instance merely one vector per class needs to be memorized, each component the value of one such parameter. Such a compact encoding minimizes the requirements for storage space as well as the computation times needed to identify relevant cases.

Establishing the initial case library in a systematic way involves the generation of a representative set of test instances. This can be done by taking several possible values for each characteristic parameter and constructing a number of instances for each of the induced combinations; for the RCPSP we will describe instance sets obtained in this way later. To each of these instances all those available algorithms are applied which are considered likely to produce good results; note that this implies that algorithms which are already known to be unsuitable need not be applied. The initial partition is then defined by letting the borderline between adjacent classes run through the midpoints between the tested values. If we assume e.g. a problem that is characterized by only one parameter from [0, 1], then the values 0.3, 0.5, and 0.7 would induce the intervals [0, 0.4], (0.4, 0.6], and (0.6, 1], so each of the three corresponding equivalence classes would consist of all instances whose parameter values lie in the same interval. Doing so essentially lays an n-dimensional grid with n the number of factors distinguished over the parameter space, where each equivalence class is represented by an n-dimensional hypercube around the point defined by its parameter values.

Let us emphasize that while such an - explicitly or implicitly - full factorial design may seem excessive in terms of experimental effort, it is not overly so. First, recall that sampling methods are among the fastest scheduling heuristics available, bettered only by their deterministic counterparts. With current computers, sampling methods run several hundred iterations in fractions of seconds such that computing the initial experiments may be done in several hours. Second, while this effort might seem particular to devising CBR systems, the same experimentation is involved in the traditional approach of identifying an FCS-based algorithm well-suited for a problem, viz. running several alternative algorithms on an instance set and recommending as best for the problem that algorithm with the highest average effectiveness over all instances.

3.3.2. Case Selection

A second ingredient is an algorithmic means to identify relevant cases, i.e. similar instances, once a new instance comes up. Here, recognizing the applicable case merely requires to identify the matching interval for each parameter, and thus the appropriate equivalence class. More complex pattern recognition methods which do not rely upon the above interval structure are discussed in Kimms (1997).

3.3.3. Case Library Updating

A third ingredient, which may be referred to as a learning component, decides how to validate and when to update the case library. Clearly such features have the potential to enhance the quality of the recommendations given and are therefore a sensible part of any CBR system (Kolodner 1983). In its most simple form, this may prescribe, whenever a new case has been processed, the storing of the case and whatever solution is adopted for it. However, using equivalence classes, we do not differentiate between instances of the same class, so updates are not mandated by formal reasons; notice that updates would be required if the instances were memorized in an extensional manner. We therefore propose two more evolved approaches. The first one stipulates an update whenever a new case differs from those in the library by more than a specified degree; this could be measured in terms of the distances from the incumbent interval midpoints, taking e.g. the average or the maximum value over all matching intervals. To such an instance, all seemingly worthwhile algorithms should be applied, as done for the initial setup of the library. If the results favour another than the currently prescribed algorithm, a refinement of the interval structure is in place. This allows, so to speak, to chart the boundaries between the classes more precisely. Another approach would be to apply promising algorithms to each newly acquired instance, including those instances which are not covered by the first approach. For these instances, the approach would be used ex post, the intention being to verify that the recommendation given by the CBR system was indeed fitting. In this way, a validation of the knowledge base can be achieved on a system load permitting basis.

4. Case-Based Reasoning for Project Scheduling

In this section, we apply the general ideas just outlined to project scheduling and develop the design of a CBR system selecting parameterized random sampling methods for the RCPSP. We describe the initial setup of the case library, drawing upon the results from extensive computational experimentation, and discuss details of the case selection. As we confine ourselves to the initial library setup, we do not elaborate on case library updating; if such a feature is to be added, any of the approaches just outlined can be used.

4.1. Case Library Initialization

4.1.1. Test Instances

The factors examined in the experiments conducted for the library setup are sampling algorithm, specified in more detail by scheduling scheme, random sampling scheme, and priority rule used, number of iterations, and instance size. We should emphasize that the control parameters α and δ are held constant throughout; in preliminary tests the values $\alpha = 1$ and $\delta = 10$ gave the best results for the algorithms discussed here, and these are the values used. We thus omit them from any algorithmic descriptions.

Specifying a set of values for each factor describes over which levels it is varied during an *experiment*, while one value for each factor determines a *run* of an experiment. For each instance and algorithm, the outcome of each run can be assessed in terms of effectiveness and efficiency. Effectiveness is measured as the percentage deviation of the best schedule found for the instance in that run from a reference solution. Efficiency captures the CPU-time required for the run, measured in terms of seconds.

As a test bed, we used the instance sets J30, J60, and J120 generated by the generator ProGen (Kolisch et al. 1995). Each instance is made up of 30, 60, or 120 non-dummy activities, having a nonpreemptable duration of between one and ten periods and requiring one or several of four 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 characteristics for these instances are the resource factor (RF) and the resource strength (RS), explained above, as well as the network complexity (NC), defined as the average number of non-redundant arcs per activity. A more comprehensive characterization is given in Kolisch et al. (1998). The respective parameter levels used are shown in Table 3.

Parameter	Levels for J30 and J60				Levels for J120				
NC	1.5	1.8	2.1		1.5	1.8	2.1		
RF	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0	
RS	0.2	0.5	0.7	1.0	0.1	0.2	0.3	0.4	0.5

Table 3: Instance Sets - Varied Instance Characteristics

For each cluster defined by a combination of these parameters, each instance set contains ten instances, for a total of 480 instances for J30 and J60, and 600 for J120. Note that of J30 and J60, those 120 instances where RS = 1.0 are trivially solvable by the earliest start time schedule.

For the most part, our experimentation used a full factorial design on the above factors. Several experiments of limited scope that were conducted in addition will be covered in the subsequent text, where appropriate.

4.1.2. Effect of Instance Characteristics

Merely precedence-constrained instances of the RCPSP, i.e. those where RS = 1.0, can be optimally solved in linear time, so only the presence of scarce resources induces the strong **NP**-equivalence of the problem. This insight is easily illustrated by disaggregating the effectiveness of pure random sampling (RAS) over RF and RS, as shown in Figure 1 (where we use RAS under the SSS; the picture is essentially the same for the PSS). (The height of the surface represents the average deviation from optimum. The differently shaded stripes reflect different levels of the surface, measured in terms of 1% increments.) Figure 1 indeed demonstrates that less capacitated instances are more tractable than instances with rather scarce resources. It thus seems straightforward that, even if focussing on non-trivial instances only, certain algorithms should perform better on some instances than others.

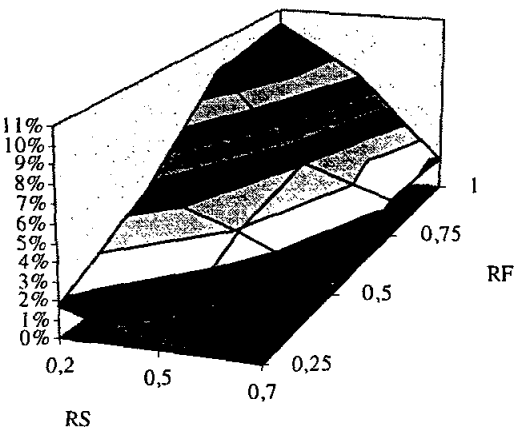
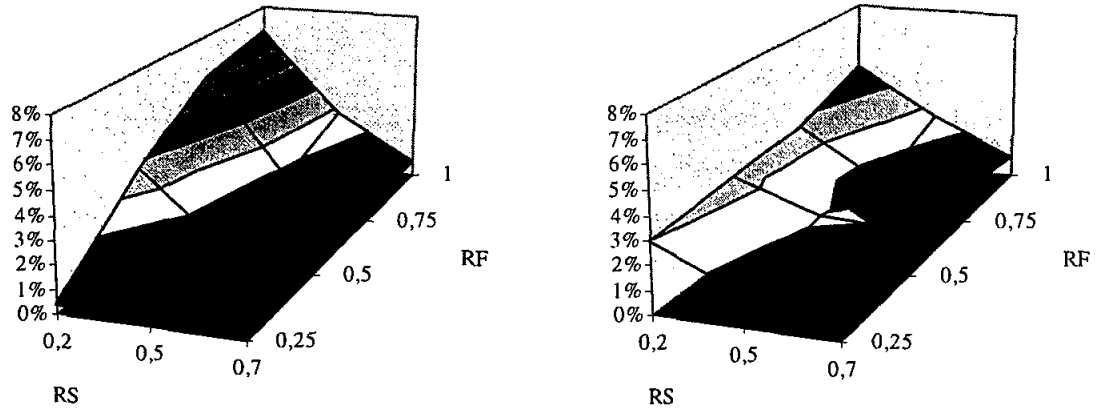


Figure 1: Tractability of Instances as Demonstrated by Effectiveness of RAS

Using a subset of J30 and allotting 100 iterations to each algorithm, Kolisch, Drexl (1996) found the priority rules LST and WCS to perform best under the SSS and the PSS, respectively. Also, they found the relative effectiveness of the algorithms SSS, RBRS, LST and PSS, RBRS, WCS to depend on the RF of the instance attempted: SSS, RBRS, LST outperforms its counterpart on instances where RF $\leq$ 0.75 while PSS, RBRS, WCS fares better if RF > 0.75. In a more recent study (Schirmer, Riesenberg 1998), encompassing the whole 360 non-trivial instances, 502 FCS-based algorithms were analysed separately on all 36 clusters. As already reported by De Reyck, Herroelen (1996), the network complexity has no significant bearing on algorithmic effectiveness, yet both RF and RS turned out to affect the effectiveness of algorithms. Figure 2 exhibits the respective results of the best serial and parallel algorithms in that study, viz. SSS, MRBRS/10, LST and PSS, RBRS, WCS, disaggregated over RF and RS. The above hypothesis is supported, as indeed different algorithms produce better results on different instance clusters.



(SSS, MRBRS/10, LST) (PSS, RBRS, WCS)

Figure 2: Effect of RF, RS - Deviations

	SSS, MRBRS/10, LST			*PSS, RBRS, WCS*		
		RS			*RS*	
RF	0.2	0.5	0.7	0.2	0.5	0.7
0.25	0.43%	0.00%	0.00%	3.00%	1.36%	1.42%
0.50	3.78%	0.66%	0.08%	3.44%	2.12%	1.91%
0.75	6.07%	1.70%	0.39%	3.70%	1.32%	1.51%
1.00	6.74%	2.81%	0.74%	4.91%	2.67%	0.94%

Table 4: Effect of RF, RS - Deviations

The corresponding numerical results are detailed in Table 4, revealing a clear cut dividing line between those instance clusters on which the best serial FCS-based algorithm dominates and

those on which the parallel does (dominated areas are put in shading); for ease of presentation we will refer to the former as serial and to the latter as parallel clusters. Thus, the recommendation is straightforward to combine these two algorithms by selecting the better of both for each cluster defined by RF and RS (cf. Table 5 below, marked J30, $Z \leq 400$).

4.1.3. Effect of Sample Size

Since this composite algorithm is based upon the results from running several FCS-based algorithms for a sample size of $Z = 100$ iterations each, naturally the question arises whether the sample size has any measurable impact on the effectiveness of these algorithms relative to each other. If so, the case library of a CBR system would have to include entries for different iteration numbers in order to adjust for these effects. Indeed it is shown in Schirmer (1997) that the ranking of priority rules may change with the number of iterations. In Figure 3, we summarize the results of an experiment in which all priority rules were employed with the MRBRS/10 under both scheduling schemes, running them for up to 500 iterations on the set J30. The results are representative of those for the RBRS and other MRBRS-variants.

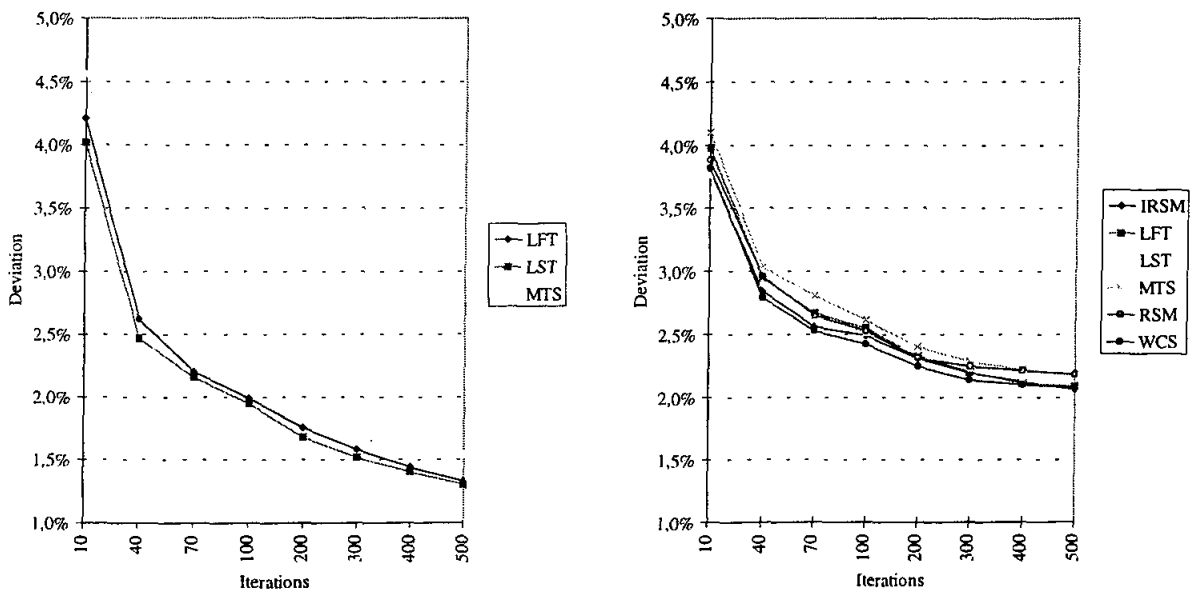


Figure 3: Effect of Iterations and Priority Rules - Deviations

Although the ranking of the rules remains the same under the SSS over all iteration numbers, it changes under the PSS: the rule WCS emerges as the best one for all iteration numbers smaller than 400 whereas for larger iteration numbers the rule LFT is the best one; for 400 iterations, both rules perform identically. Note that using PSS, LFT instead of PSS, WCS for

larger samples has the additional benefit of being faster, since on average the latter combination requires about 25% more computation time (Schirmer 1997). Also, the dividing line between the "serial" and the "parallel" clusters moves with increasing iteration numbers; so the best composition is also affected by the size of the sample taken.

We extend these results by a similar experiment, taking measurements after 1000 and 5000 iterations. This experiment serves several purposes. First, it enables us to check whether the algorithms SSS, MRBRS/10, LST and PSS, RBRS, LFT remain the most effective ones on the serial and parallel clusters, respectively, when the sample size is increased markedly. Second, it allows to verify whether the dividing line between serial and parallel clusters remains unaffected by the size of the sample taken. An additional benefit is that it allows to compare the effectiveness of sampling methods with that of more complex metaheuristic algorithms which usually are taking much larger samples. We will come back to this last issue later. Without reproducing the numerical results here, the outcome can be summarized as follows. LFT continues to dominate WCS on the parallel clusters, except for Z = 5000 where WCS regains a slight advantage - less than 0.03% - on J30 and J60. Due to the higher efficiency of LFT, we chose to disregard this result. SSS, MRBRS/10, LST and PSS, RBRS, LFT are still the most effective algorithms on the serial and parallel clusters, respectively. Yet, a more interesting finding is that the dividing line between serial and parallel clusters indeed varies with different sample sizes. As an example, consider the instance set J30: Whereas for 100 iterations 15 clusters are better solved by the best parallel algorithm, this number reduces to 9 clusters for larger samples. Hence, for samples sizes above 400 a modified algorithm is recommended (cf. Table 5 below, marked J30, Z > 400).

From these results one might assume that for samples smaller than 100 even more clusters are solved better by a parallel algorithm. In order to check this, we conducted another experiment with the same algorithms, taking measurements after 10, 40, and 70 iterations for each instance set. It turns out for J30 that some minor changes occur when only 10 iterations are performed, adding the clusters with RF ≥ 0.75 and RS = 0.7 to the parallel ones; for sample sizes between 10 and 100 as well as for the other instance sets, the picture remains the same as for 100 iterations. If we regard any iteration number of 10 or below as unreasonable, due to the random error immanent to sampling (Hancock 1994), we can conclude that even for very small samples the best serial algorithm remains the better choice for the easier instances, i.e. those with small RF and large RS.

4.1.4. Effect of Instance Size

A related question is whether also the size of the attempted instances influences the best composition of CCS-based algorithms, i.e. the position of the dividing line between serial and parallel clusters. Note that, although several problem parameters contribute to the size of RCPSP-instances as formalized in complexity theory, we will here consider only the number of

activities to be scheduled. To assess the impact of the instance size, we extend the above experiments to the larger sets J60 and J120. The best control schemes found are again reported in Table 5 (marked J60 and J120). The results show that increasing the number of activities may indeed produce a similar, if less pronounced effect as decreasing the number of iterations performed. As an example, compare the best algorithms on J30 and J60 for a sample size of 500 iterations: Here the increase in the number of activities reduces the number of clusters on which the best serial algorithm performs better from 27 to 21. Another interesting finding is that, although the effectiveness of the rules LFT and WCS under the PSS is always rather close, WCS performs better only on J30 and only for samples of 400 iterations or less; on the larger instance sets, LFT consistently outperforms WCS for all sample sizes tested.

In the earlier literature on priority rule-based scheduling methods, the conjecture was widely accepted that the best rules on smaller instances would also be the best ones on larger instances (Davis, Patterson 1975; Badiru 1988; Alvarez-Valdés, Tamarit 1989). On the basis of our results, this conjecture, which is probably due to the unavailability of computers fast enough to allow large scale experimentation, has to be refuted, at least in its generality.

4.2. Case Selection

As a basis for case selection, we may summarize the outcome of our experiments, i.e. our initial knowledge base, in terms of algorithmic recommendations for the RCPSP. These are depicted in Table 5 where for each instance class the most effective, and in case of ties the most efficient, sampling algorithm identified is specified.

J30, Z≤400	*RS*		
RF	0.2	0.5	0.7
0.25			SSS, LST
0.50			
0.75			
1.00	PSS, WCS		

J30, Z>400	*RS*		
RF	0.2	0.5	0.7
0.25			SSS, LST
0.50			
0.75			
1.00	PSS, LFT		

J60	*RS*		
RF	0.2	0.5	0.7
0.25			SSS, LST
0.50			
0.75			
1.00	PSS, LFT		

J120	*RS*				
RF	0.1	0.2	0.3	0.4	0.5
0.25				SSS,	LST
0.50					
0.75					
1.00	PSS,	LFT			

Table 5: Algorithm CBR-SAR - Composition

Following the ideas outlined before, we partition the instance space using the midpoints between the tested values of J, Z, RF, and RS. Case selection under this initial knowledge base can then be managed by the algorithm listed in the Appendix (Table 11).

5. Implications for Sampling Methods

It is known from theoretical analysis that the space **AS** of active solutions, which is sampled by the SSS, always contains at least one optimum whereas the space **NDS** of non-delay schedules, which is sampled by the PSS, may contain no optimal solution at all (Kolisch 1996). The above experimental results for the case library initialization allow to complement these theoretical results by several interesting conclusions which provide deeper insight into the behaviour of both scheduling schemes.

Detailing the results over the instance clusters revealed the number of parallel ones, i.e. those clusters on which the PSS is more effective than the SSS, to be the larger, the larger the instances attempted and the smaller the samples taken are (Table 5). Both effects can be traced back to the same cause. As the PSS searches a smaller space than the SSS - **NDS** is a subspace of **AS** (Sprecher et al. 1995) - it will tend to find better schedules than the SSS in the first iterations. In this sense, the PSS could be said to search in a more focussed way. This advantage wears off in later iterations, and thus for larger samples, since then the SSS can more fully search its solution space which contains better, even optimal solutions. In other words, the smaller the portion of an instance's solution space is that will be searched in an experiment, the more beneficial it is to employ the PSS. In much the same way, large instances mean that their solution space is too large to be searched exhaustively in only a few iterations; again the PSS has a competitive advantage that is the greater, the larger the instance attempted is.

Further, the PSS works better on hard instances and the SSS on easy ones. Here, the reason is more involved, it arises from an interesting relationship between RF and RS on one side and the size of certain solution subspaces on the other. Given an instance of the RCPSP, the space **FS** of feasible solutions will become smaller when additional constraints are adjoined (such as by increasing RF) or existing ones are tightened (such as by decreasing RS). This would seem

to contradict the insight that the PSS outperforms the SSS on larger solution spaces only. However, both schemes do not sample **FS** exhaustively but, by excluding numerous schedules from the generation process, search substantially smaller subspaces of **FS**. And the spaces **NDS** of non-delay and **AS** of active schedules which are searched by the PSS and the SSS, respectively (Kolisch 1996), become larger for harder instances. Both schemes can be said to try and schedule activities as long as this is resource-feasible. So, some activities that could be scheduled to run in parallel if only precedence constraints were to be considered have to be relegated to later start times when they compete for the allocation of scarce resources. Assume now an instance where RF = 0 or RS = 1 such that all activities can be run in parallel, then **NDS** and **AS** comprise only one solution each. After increasing resource scarcity to a point where two activities must be processed sequentially, both spaces comprise two schedules, each delaying one of the activities. In this way, tightening resource scarcity is putting more solutions in the spaces sampled by the scheduling schemes. Therefore, for any fixed sample size, the searched portion of the solution space is smaller for hard instances and larger for easy ones, such that the more focussed PSS can outrank the SSS on the former.

6. Computational Results

In order to demonstrate the validity of our approach, we provide effectiveness and efficiency results derived on the test instances employed. In addition, we compare our approach to a number of state-of-the-art heuristics for the RCPSP.

6.1. Effectiveness

For the sake of brevity, we detail the results for 500 iterations only; the results for the other sample sizes can be obtained from the author upon request. As reference solutions for the instances in J30 we use the respective optima as published in the project scheduling problem library PSPLIB (Kolisch, Sprecher 1997). Since not all optima are available for J60 and J120, the best heuristic solutions found in the study of Kolisch, Hartmann (1998) are used. To facilitate comparisons, we also employ the precedence-based lower bound, i.e. the length of the critical path, for each instance, providing a datum that is easily reproduced to make results comparable. The results for the different instance sets are summarized in Table 6, where Deviation Opt (BF, LB) stands for the deviation from optimum (best solution found, precedence-based lower bound). For convenience, we have added the dividing line between serial and parallel clusters.

J30	Deviation Opt		
	RS		
RF	0.2	0.5	0.7
0.25	0.14%	0.00%	0.00%
0.50	2.43%	0.24%	0.08%
0.75	2.81%	0.85%	0.17%
1.00	3.57%	1.67%	0.55%

J60	Deviation BF			Deviation LB		
	RS			RS		
RF	0.2	0.5	0.7	0.2	0.5	0.7
0.25	1.30%	0.13%	0.00%	10.63%	1.89%	1.53%
0.50	5.46%	1.34%	0.00%	38.61%	2.19%	0.17%
0.75	6.31%	2.04%	0.04%	57.97%	3.46%	0.09%
1.00	5.40%	2.62%	0.05%	78.97%	5.91%	0.00%

J120	Deviation BF					Deviation LB				
	RS					RS				
RF	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
0.25	3.55%	5.36%	1.82%	0.59%	0.19%	31.28%	20.38%	4.29%	2.19%	0.50%
0.50	6.69%	4.84%	5.06%	4.38%	1.85%	97.97%	48.44%	20.04%	9.53%	2.76%
0.75	5.33%	3.26%	3.89%	4.47%	3.19%	144.20%	65.45%	29.58%	15.35%	4.35%
1.00	4.07%	2.40%	2.58%	3.16%	2.65%	170.23%	78.19%	37.86%	18.18%	6.04%

Table 6: Effectiveness of Algorithm CBR-SAR

6.2. *Efficiency*

To control for the effect of different priority rules (which are documented in Schirmer, Riesenberg 1997), in Table 7 the results are shown for the rule LFT under both the SSS and the PSS, run for 500 iterations. Measurements were taken using an implementation in Pascal and run on a Pentium II/266 personal computer with 32 MB RAM under Windows 95. The results are representative for those of other algorithms.

J30	SSS, LFT			PSS, LFT		
	RS			RS		
RF	0.2	0.5	0.7	0.2	0.5	0.7
0.25	1.31	1.29	1.28	1.04	0.98	1.00
0.50	1.38	1.31	1.28	1.14	1.06	1.01
0.75	1.43	1.33	1.30	1.24	1.11	1.07
1.00	1.46	1.32	1.30	1.30	1.17	1.07

J60	*SSS, LFT*			*PSS, LFT*		
		RS			*RS*	
RF	0.2	0.5	0.7	0.2	0.5	0.7
0.25	3.86	3.63	3.46	3.14	2.97	2.91
0.50	3.69	3.56	3.45	3.39	3.09	2.96
0.75	3.83	3.58	3.48	3.73	3.27	3.06
1.00	5.02	4.12	4.08	4.10	3.45	3.14

J120	*SSS, LFT*					*PSS, LFT*				
			RS					*RS*		
RF	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
0.25	13.51	12.52	11.88	11.21	10.74	13.32	12.40	11.87	11.35	11.14
0.50	13.64	11.33	12.05	10.79	11.70	11.86	11.36	10.55	12.72	12.06
0.75	12.98	12.59	13.65	10.68	10.95	13.84	12.48	13.80	11.21	10.00
1.00	14.87	16.33	14.24	11.43	10.41	13.88	13.78	13.41	12.55	10.59

Table 7: Efficiency of Algorithm CBR-SAR

In general, the PSS is more efficient than the SSS, a fact that has already been noted by Kolisch (1995, p. 106). Clearly, computation times increase with increasing RF and decreasing RS. While the influence of RF can be easily explained for both scheduling schemes alike since more resources need to be taken into account when RF increases, the influence of RS calls for two explanations. Under the PSS, the decision set is built anew whenever a new schedule time has been determined, which in turn happens when all elements of the previous decision set have been scheduled. With decreasing RS, resource capacities become scarcer such that less activities can be processed in parallel, the decision sets become smaller and thus need to be rebuilt more often. Under the SSS, scarcer resources mean that the earliest feasible start times of some activities are later ones, so more candidate start times (which are traversed in chronological order) have to be checked for resource feasibility.

6.3. Comparison With Other Algorithms

In order to further demonstrate the efficacy of the CBR approach, we compare its effectiveness to that of several other algorithms for the RCPSP. These comprise several simple FCS-based scheduling heuristics, which all are based upon using one priority rule each. We have choosen the two best serial and parallel algorithms known. We complement these with several recent metaheuristics; results for these are gleaned from the comparative study of Kolisch, Hartmann (1998). Note that, although the results cited there encompass all 480 instances of J30 and J60, we exclude the trivial 120 instances with RS = 1.0. Note also that for the tabu search algorithm of Baar et al (1997) no results on J120 are available, as this instance set made excessive demands on computer resources.

For metaheuristics the definition of an iteration may substantially differ from the one used in sampling or other simple heuristics. For genetic algorithms, e.g., one iteration may involve the

construction or modification of a whole generation of individuals, possibly comprising dozens or even hundreds of solutions, whereas in less involved heuristics one iteration is understood to construct one solution at most. To keep the ensuing comparisons impartial, all results are related to the best solution found after having generated and evaluated 1,000 and 5,000 (not necessarily different) solutions. The results are juxtaposed in Tables 8 - 10.

Algorithm	*Reference*	*Type*	*Deviation Opt Solutions* 1000	5000
Simulated annealing	Bouleimen, Lecocq (1998)	Metaheuristic	0.51%	0.31%
Genetic algorithm	Hartmann (1997)	Metaheuristic	0.72%	0.33%
CBR	-	Sampling	0.87%	0.59%
Tabu search	Baar et al. (1997)	Metaheuristic	1.15%	0.59%
Adaptive search	Kolisch, Drexl (1996)	Sampling	0.99%	0.69%
SSS, MRBRS/10, LST	Schirmer, Riesenberg (1997)	Sampling	1.09%	0.73%
SSS, MRBRS/10, LFT	Schirmer, Riesenberg (1997)	Sampling	1.11%	0.75%
PSS, RBRS, WCS	Kolisch (1996a)	Sampling	1.89%	1.70%
PSS, RBRS, LFT	Kolisch (1996b)	Sampling	1.86%	1.71%
Genetic algorithm	Leon, Ramamoorthy (1995)	Metaheuristic	2.77%	2.12%

Table 8: Comparison of Several Algorithms (J30)

Algorithm	*Reference*	*Type*	*Deviation BF Solutions* 1000	5000	*Deviation LB Solutions* 1000	5000
Genetic algorithm	Hartmann (1997)	Metaheuristic	1.32%	0.60%	16.64%	15.80%
Simulated annealing	Bouleimen, Lecocq (1998)	Metaheuristic	1.56%	0.65%	17.00%	15.87%
CBR	-	Sampling	1.61%	1.25%	17.26%	16.78%
Adaptive search	Kolisch, Drexl (1996)	Sampling	2.13%	1.68%	18.01%	17.41%
SSS, MRBRS/10, LST	Schirmer, Riesenberg (1997)	Sampling	2.46%	1.95%	18.55%	17.87%
SSS, MRBRS/10, LFT	Schirmer, Riesenberg (1997)	Sampling	2.53%	2.03%	18.64%	17.96%
Tabu search	Baar et al. (1997)	Metaheuristic	2.39%	2.05%	18.40%	17.97%
PSS, RBRS, WCS	Kolisch (1996a)	Sampling	2.50%	2.06%	18.20%	17.60%
PSS, RBRS, LFT	Kolisch (1996b)	Sampling	2.45%	2.09%	18.12%	17.65%
Genetic algorithm	Leon, Ramamoorthy (1995)	Metaheuristic	3.31%	2.43%	19.11%	17.99%

Table 9: Comparison of Several Algorithms (J60)

Algorithm	*Reference*	*Type*	*Deviation BF Solutions* 1000	5000	*Deviation LB Solutions* 1000	5000
Genetic algorithm	Hartmann (1997)	Metaheuristic	2.59	0.89	39.37	36.74
Simulated annealing	Bouleimen, Lecocq (1998)	Metaheuristic	5.73	1.86	42.81	37.68
CBR	-	Sampling	3.10	2.27	39.85	38.70
PSS, RBRS, LFT	Kolisch (1996b)	Sampling	3.19	2.47	39.91	38.90
PSS, RBRS, WCS	Kolisch (1996a)	Sampling	3.28	2.57	40.04	39.05
Adaptive search	Kolisch, Drexl (1996)	Sampling	3.95	3.33	41.37	40.45
Genetic algorithm	Leon, Ramamoorthy (1995)	Metaheuristic	5.33	3.76	42.91	40.69
SSS, MRBRS/10, LST	Schirmer, Riesenberg (1997)	Sampling	4.81	4.08	42.77	41.71
SSS, MRBRS/10, LFT	Schirmer, Riesenberg (1997)	Sampling	4.85	4.08	42.92	41.79

Table 10: Comparison of Several Algorithms (J120)

7. Summary and Conclusions

In this paper, we proposed ways to integrate CBR and algorithms for OR problems, our intention being to demonstrate the benefits of selecting algorithms according to several influential criteria. Although we focussed on sampling methods for the RCPSP, the ideas discussed are easily transferred to other applications. Along the way, we have analyzed the influence of several instance characteristics (resource factor, resource strength, instance size) as well as the number of iterations on the effectiveness of a broad range of sampling algorithms. This revealed some previously unknown effects and allowed to substantially improve the most effective sampling method in the open literature, viz. the adaptive search approach of Kolisch, Drexl (1996). Also, it extends the class-based approach of Schirmer, Riesenberg (1998) to larger instance classes. A comparison with state-of-the-art heuristics for the RCPSP corroborated the effectiveness of our approach. We thus hope to contribute in two ways to the current body of knowledge. First, for the practitioner by showing how to design more effective algorithms without compromising efficiency. Second, for the researcher by providing deeper insight into the behaviour of sampling heuristics.

One interesting direction for future research is to analyze effectiveness and efficiency of metaheuristic algorithms in a more detailed manner, as done here for sampling methods. A conceivable outcome might be to also integrate such algorithms into a CBR system. Such a system could employ an effective metaheuristic to tackle the hardest instances on which the effectiveness of sampling leaves to be desired, and apply - more efficient and still highly effective - sampling methods to easier instances. Another promising idea is to refine the scheduling schemes that are currently incorporated in many metaheuristics, drawing on the implications outlined. Notice that both of the currently best metaheuristic algorithms for the RCPSP (Hartmann 1997; Bouleimen, Lecocq 1998) utilize such methods to generate schedules. We there-

fore augur that further improving these methods will pave the way also for further improvements in the best known metaheuristics.

Acknowledgements

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; the author is most grateful for the support, in particular of Peter Kandzia, that made this work possible. We also wish to thank Andreas Drexl for his helpful comments and Rainer Kolisch for providing us with the reference solutions of the instance sets J60 and J120. Partial funding was granted by the Deutsche Forschungsgemeinschaft (German National Science Foundation) under Grant Dr 170/4-1.

Appendix

Using the midpoints between the tested values of the instance characteristics to define the required partition, a formal description of the algorithm CBR-SAR defined in Table 5 can be given as in Table 11; we omit the pseudo-code for some subroutines which are obvious to complete from those given.

Initialization

determine J
determine Z
calculate RF
calculate RS

Execution

if (J ≤ 45)
  if (Z ≤ 400)
    call CBR_J30_Z≤400
  else
    call CBR_J30_Z>400
else if (J ≤ 90)
  call CBR_J60
else
  call CBR_J120
endif

Subroutine CBR_J30_Z≤400

```
if (( RF ≤ 0.375 )
or ( 0.375 < RF ≤ 0.625 and RS ≥ 0.35 )
or ( RS ≥ 0.6 ))
    for z ← 1 to Z
        call SSS(MRBRS/10, LST, α = 1)
else
    for z ← 1 to Z
        call PSS(RBRS, WCS, α = 1)
endif
```

Subroutine CBR_J120

```
if (( RS ≤ 0.25 )
or ( 0.375 < RF and 0.25 < RS ≤ 0.45 )
or ( 0.625 < RF and 0.45 < RS ≤ 0.55 ))
    for z ← 1 to Z
        call PSS(RBRS, LFT, α = 1)
else
    for z ← 1 to Z
        call SSS(MRBRS/10, LST, α = 1)
```

■

Table 11: Algorithm CBR-SAR

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.

BAAR, T., P. BRUCKER, AND S. KNUST (1997), "Tabu search algorithms for the resource-constrained project scheduling problem", Technical Report, University of Osnabrück.

BADIRU, A.B. (1988), "Towards the standardization of performance measures for project scheduling heuristics", IEEE Transactions on Engineering Management 35, pp. 82-89.

BEAN, J.C. (1994), "Genetic algorithms and random keys for sequencing and optimization", ORSA Journal on Computing 6, pp. 154-160.

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

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.

BOULEIMEN, K. AND H. LECOCQ (1998), "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem", Technical Report, Service de Robotique et Automatisation, Université de Liège, France.

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

CROWDER, H., R. DEMBO, AND J. MULVEY (1980), "On reporting computational experiments with mathematical software", ACM Transactions on Mathematical Software 5, pp. 193-203.

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.

DE REYCK, B. AND W.S. HERROELEN (1996), "On the use of the complexity index as a measure of network complexity in activity networks", European Journal of Operational Research 91, pp. 347-366.

DE WIT, J. AND W.S. HERROELEN (1990), "An evaluation of microcomputer-based software packages for project management", European Journal of Operational Research 49, pp. 102-139.

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.

GAREY, M.R. AND D.S. JOHNSON (1979), *Computers and intractability - A guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, CA.

GROLIMUND, S. AND J.-G. GANASCIA (1997), "Driving tabu search with case-based reasoning", European Journal of Operational Research 103, pp. 326-338.

HANCOCK, P.J.B. (1994), "An empirical comparison of selection methods in evolutionary algorithms", in: *Evolutionary Computing: AISB Workshop, Leeds, UK*, T.C. Fogerty (ed.), Springer, Berlin.

HART, J.P. AND A.W. SHOGAN (1987), "Semi-greedy heuristics: An empirical study", Operations Research Letters 6, pp. 107-114.

HARTMANN, S. (1997), "A competitive genetic algorithm for resource-constrained scheduling", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 451, to appear in Naval Research Logistics.

JACKSON, R.H.F., P.T. BOGGS, S.G. NASH, AND S. POWELL (1991), "Guidelines for reporting results of computational experiments - Report of the ad hoc Committee", Mathematical Programming 49, pp. 413-425.

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

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. (1997), "Resource allocation capabilities of commercial project management packages", Research Report, Universität Kiel, Germany.

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., C. SCHWINDT, AND A. SPRECHER (1998), "Benchmark instances for project scheduling problems", 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.

KOLODNER, J.-L. (1983), "Maintaining organization in a dynamic long term memory", Cognitive Science 7, pp. 243-279.

KOLODNER, J.-L., R.-L. SIMPSON, AND K. SYCARA-CYRANSKI (1985), "A process model of case-based reasoning in problem solving", *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 284-290.

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.

LAGUNA, M., T.A. FEO AND H.C. ELROD (1994), "A greedy randomized adaptive search procedure for the two-partition problem", Operations Research 42, pp. 677-687.

LEE, J.-K. AND Y.-D. KIM (1996), "Search heuristics for resource constrained project scheduling", Journal of the Operational Research Society 47, 678-689.

LEON, V.J. AND B. RAMAMOORTHY (1995), "Strength and adaptability of problem-space based neighborhoods for resource constrained scheduling", Operations Research Spektrum 17, pp. 173-182.

NAPHADE, K.S., S.D. WU, AND R.H. STORER (1997), "Problem space search algorithms for resource-constrained project scheduling ", *Annals of Operations Research 70*, pp. 307-326.

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

ÖZDAMAR, L. (1996), "A genetic algorithm approach to a general category project scheduling problem", Research Report, Marmara University, Istanbul, Turkey.

POMEROL, J.-C. (1997), "Artificial intelligence and human decision making", European Journal of Operational Research 99, pp. 3-25.

REEVES, C.R. (1993), "Improving the efficiency of tabu search for machine sequencing problems", Journal of the Operational Research Society 44, pp. 167-178.

SCHANK, R.-C. (1982), *Dynamic memory*, Cambridge, University Press.

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

SCHIRMER, A., S. RIESENBERG (1997), "Parameterized heuristics for project scheduling - Biased random sampling methods", Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel 456.

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

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

STORER, R.H., S.D. WU, AND R. VACCARI (1992), "New search spaces for sequencing problems with application to job shop scheduling", Management Science 38, pp. 1495-1509.