ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Kolisch, Rainer; Hartmann, Sönke

Working Paper — Digitized Version

Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 469

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

Suggested Citation: Kolisch, Rainer; Hartmann, Sönke (1998) : Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 469, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/147576

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 469

Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis

R. Kolisch and S. Hartmann

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 469

Heuristic Algorithms for Solving the Resource–Constrained Project Scheduling Problem: Classification and Computational Analysis

R. Kolisch and S. Hartmann

February 1998

Invited contribution to the
"Handbook on Recent Advances in Project Scheduling"
edited by J. Weglarz and to be published by Kluwer.

Dr. Rainer Kolisch and Sönke Hartmann

Lehrstuhl für Produktion und Logistik,
Institut für Betriebswirtschaftslehre,
Christian–Albrechts–Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
email: kolisch@bwl.uni-kiel.de, hartmann@bwl.uni-kiel.de
URL: http://www.wiso.uni-kiel.de/bwlinstitute/Prod/kolisch.html
http://www.wiso.uni-kiel.de/bwlinstitute/Prod/hartmann.html
ftp://ftp.wiso.uni-kiel.de/pub/operations-research

1 Introduction

The resource constrained project scheduling problem (RCPSP) can be given as follows. A single project consists of a set $\mathcal{J} = \{0, 1, \ldots, n, n+1\}$ of activities which have to be processed. Fictitious activities 0 and $n+1$ correspond to the "project start" and to the "project end", respectively. The activities are interrelated by two kinds of constraints. First, precedence constraints force activity j not to be started before all its immediate predecessor activities comprised in the set $\mathcal{P}_j$ have been finished. Second, performing the activities requires resources with limited capacities. We have K resource types, given by the set $\mathcal{K} = \{1, \ldots, K\}$. While being processed, activity j requires $r_{j,k}$ units of resource type $k \in \mathcal{K}$ during every period of its non–preemptable duration p_j. Resource type k has a limited capacity of R_k at any point in time. The parameters p_j, $r_{j,k}$, and R_k are assumed to be deterministic; for the project start and end activities we have $p_j = 0$ and $r_{j,k} = 0$ for all $k \in \mathcal{K}$. The objective of the RCPSP is to find precedence and resource feasible completion times for all activities such that the makespan of the project is minimized. Figure 1 gives an example of a project comprising $n = 6$ activities which have to be scheduled subject to $K = 1$ renewable resource type with a capacity of 4 units. A feasible schedule with an optimal makespan of 13 periods is represented in Figure 2.

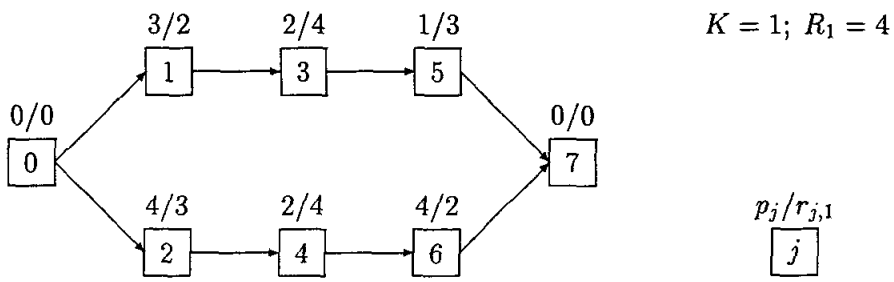

Figure 1: Project instance

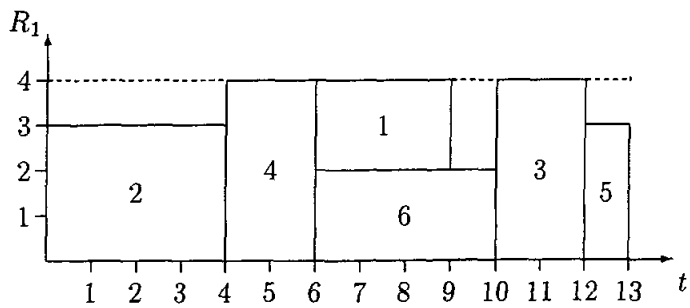

Figure 2: Example schedule

Let F_j denote the finish time of activity j. A vector of finish times $(F_1, F_2, \ldots, F_n)$ is called a schedule S. Let $\mathcal{A}(t) = \{j \in \mathcal{J} \mid F_j - p_j \leq t < F_j\}$ be the set of activities which are being processed (active) at time instant t. We now can provide the conceptual decision model (1) – (4) (cf. Christofides et al. [12]).

$$\text{Min } F_{n+1} \tag{1}$$

$$F_h \leq F_j - p_j \qquad j = 1, \ldots, n+1;\ h \in \mathcal{P}_j \tag{2}$$

$$\sum_{j \in \mathcal{A}(t)} r_{j,k} \leq R_k \qquad k \in \mathcal{K};\ t \geq 0 \tag{3}$$

$$F_j \geq 0 \qquad j = 1, \ldots, n+1 \tag{4}$$

The objective function (1) minimizes the finish time of the project end activity and thus the makespan of the project. Constraints (2) enforce the precedence constraints between activities, and constraints (3) limit for each resource type k and each time instant t that the resource demand of the activities which are currently processed does not exceed the capacity. Finally, (4) define the decision variables. (1) – (4) is a conceptual model since the sets $\mathcal{A}(t)$ are a function of the decision variables. Hence, the model cannot be solved with mixed integer programming (MIP) techniques. In order to solve the RCPSP with MIP–solvers such as CPLEX [6], one has to employ the 0–1 problem formulation of Pritsker et al. [59].

The RCPSP is denoted in Chapter 1 of this book (cf. Herroelen et al. [29]) as $m, 1/\text{cpm}/C_{\max}$ where a number of activities which are precedence related by finish–start relationship with zero time lags have to be scheduled on m renewable resource types such that the maximal completion time of all activities $C_{\max}$ is minimized.

It has been shown by Błażewicz et al. [7] that the RCPSP as a generalization of the classical job shop scheduling problem belongs to the class of $\mathcal{NP}$–hard optimization problems. Therefore, heuristic solution procedures are indispensable when solving large problem instances as they usually appear in practical cases. Since 1963 when Kelley [31] introduced a schedule generation scheme, a large number of different heuristics algorithms have been suggested in the literature.

The great number of optimal approaches (for a survey cf. Kolisch and Padman [39]) are mainly for generating benchmark solutions. Currently, the most competitive exact algorithms seem to be the ones of Brucker et al. [10], Demeulemeester and Herroelen [18], Mingozzi et al. [48] and Sprecher [65].

In what follows we will give an appraising survey of heuristic approaches for the RCPSP. We start in Section 2 with schedule generation schemes which are essential to construct feasible schedules. In Section 3 we show how these schemes are employed in priority rule based methods. Section 4 is devoted to metaheuristic algorithms such as simulated annealing, tabu search, and genetic algorithms. Heuristics which do neither belong to the class of priority rule based methods nor to metaheuristic approaches are treated in Section 5. In Section 6 we will report about a comparison of priority rule based and metaheuristic heuristics for the RCPSP. We end with a summary and an outlook on research opportunities in Section 7.

2 Schedule Generation Schemes

Schedule generation schemes (SGS) are the core of most heuristic solution procedures for the RCPSP. SGS start from scratch and build a feasible schedule by stepwise extension of a partial schedule. A partial schedule is a schedule where only a subset of the $n+2$ activities have been scheduled. There are two different SGS available. They can be distinguished w.r.t the incrementation into activity– and time–incrementation. The so–called *serial* SGS performs *activity–incrementation* and the so–called *parallel* SGS performs *time–incrementation.*

2.1 Serial Schedule Generation Scheme

We begin with a description of the serial SGS. It consists of $g = 1, \ldots, n$ stages, in each of which one activity is selected and scheduled at the earliest precedence– and resource–feasible completion time. Associated with each stage g are two disjoint activity sets. The scheduled set $\mathcal{S}_g$ comprises the activities which have been already scheduled, the eligible set $\mathcal{D}_g$ comprises all activities which are eligible for scheduling. Note that the conjunction of $\mathcal{S}_g$ and $\mathcal{D}_g$ does not give the set of all activities $\mathcal{J}$ because, generally, there are so–called ineligible activities, i.e. activities which have not been scheduled and can not be scheduled at stage g because not all of their predecessors have been scheduled. Let $\tilde{R}_k(t) = R_k - \sum_{j \in \mathcal{A}(t)} r_{j,k}$ be the remaining capacity of resource type k at time instant t and let $\mathcal{F}_g = \{F_j \mid j \in \mathcal{S}_g\}$ be the set of all finish times. Let further be $\mathcal{D}_g = \{j \in \mathcal{J} \setminus \mathcal{S}_g \mid \mathcal{P}_j \subseteq \mathcal{S}_g\}$ the set of eligible activities. We can now give the following description of the serial SGS.

Serial SGS

Initialization: $F_0 = 0, \mathcal{S}_0 = \{0\}$,
For $g = 1$ to n do

Calculate $\mathcal{D}_g$, $\mathcal{F}_g$, $\tilde{R}_k(t)$ $(k \in \mathcal{K};\ t \in \mathcal{F}_g)$
Select one $j \in \mathcal{D}_g$
$EF_j = \max_{h \in \mathcal{P}_j} \{F_h\} + p_j$
$F_j = \min \left\{ t \in [EF_j - p_j, LF_j - p_j] \cap \mathcal{F}_g \mid r_{j,k} \leq \tilde{R}_k(\tau), k \in \mathcal{K},\ \tau \in [t, t + p_j[\cap \mathcal{F}_g \right\} + p_j$
$\mathcal{S}_g = \mathcal{S}_{g-1} \cup \{j\}$

$F_{n+1} = \max_{h \in \mathcal{P}_{n+1}} \{F_h\}$

The initilization assigns the dummy source activity $j = 0$ a completion time of 0 and puts it into the partial schedule. At the beginning of each step g, the decision set $\mathcal{D}_g$, the set of finish times $\mathcal{F}_g$, and the remaining capacities $\tilde{R}_k(t)$ at the finish times $t \in \mathcal{F}_g$ are calculated. Afterwards, one activity j is selected from the decision set. The finish time of j is calculated by first determining the earliest precedence feasible finish time EF_j and then calculating the earliest (precedence– and) resource–feasible finish time F_j within $[EF_j, LF_j]$. LF_j denotes the latest finish time as calculated by backward recursion (cf. Elmaghraby [21]) from an upper bound of the project's finish time T. Table 1 reports the serial SGS when generating the schedule given in Figure 2.

g	1	2	3	4	5	6
$\mathcal{D}_g$	{1,2}	{1,4}	{1,6}	{3,6}	{3}	{5}
j	2	4	1	6	3	5

Table 1: Example for serial SGS

The serial SGS generates always feasible schedules which are for the resource–unconstrained scheduling problem (1), (2), and (4) optimal. Kolisch [37] has shown that the serial SGS generates active schedules, that is schedules $S = (F_1, F_2, \ldots, F_n)$ where none of the activities can be started earlier without delaying some other activity. For scheduling problems with regular performance measure (for a definition of the latter cf. to Sprecher et al. [66]) such as makespan minimization,

the optimal solution will always be in the set of active schedules. The time complexity of the serial SGS as given above is $\mathcal{O}(n^2 \cdot K)$ (cf. Pinson et al. [57]).

Let j_g denote the activity which is selected in iteration g. Then, an execution of the serial SGS can be recorded by a list $\lambda = \langle j_1, j_2, \ldots, j_n]$ which prescribes that activity j_g has been scheduled in iteration g. Note, that this list is precedence feasible, i.e., we have $\mathcal{P}_{j_g} \subseteq \{j_1, \ldots, j_{g-1}\}$ (cf. Hartmann [26]). The list for the above given example is $\lambda = \langle 2, 4, 1, 3, 5, 6]$. Given a list λ, we can now give a special case of the serial SGS, namely the serial SGS for activity lists.

Serial SGS for Activity Lists

Initialization: $F_0 = 0, S_0 = \{0\}$,
For $g = 1$ to n do

Calculate $\mathcal{F}_g$, $\tilde{R}_k(t)$ $(k \in \mathcal{K};\ t \in \mathcal{F}_g)$
$j = j_g$
$EF_j = \max_{h \in \mathcal{P}_j} \{F_h\} + p_j$
$F_j = \min \left\{ t \in [EF_j - p_j, LF_j - p_j] \cap \mathcal{F}_g \mid r_{j,k} \leq \tilde{R}_k(\tau), k \in \mathcal{K},\ \tau \in [t, t + p_j[\cap \mathcal{F}_g \right\} + p_j$
$S_g = S_{g-1} \cup \{j\}$

$F_{n+1} = \max_{h \in \mathcal{P}_{n+1}} \{F_h\}$

The serial SGS for activity lists plays an important role in classical machine scheduling where it is referred to as list scheduling (cf. Kim [32] and Schutten [63]). Since the serial SGS for activity lists is a special case of the serial SGS, it generates active schedules. Hence, there is always a list λ^* for which list scheduling will generate an optimal schedule when a regular measure of performance is considered.

2.2 Parallel Schedule Generation Scheme

The parallel scheduling scheme does time incrementation. For each iteration g there is a schedule time t_g. Activities which have been scheduled up to g are either element of the complete set $\mathcal{C}_g$ or of the active set $\mathcal{A}_g$. The complete set comprises all activities which have been completed up to t_g, i.e., $\mathcal{C}_g = \{j \in \mathcal{J} \mid F_j \leq t_g\}$ and the active set comprises all activities which are active at t_g, i.e., $\mathcal{A}_g = \mathcal{A}(t_g) = \{j \in \mathcal{J} \mid F_j - p_j \leq t < F_j\}$. The eligible set $\mathcal{D}_g$ comprises all activities which can be precedence– and resource–feasibly started at t_g, i.e., $\mathcal{D}_g = \left\{ j \in \mathcal{J} \setminus (\mathcal{C}_g \cup \mathcal{A}_g) \mid \mathcal{P}_j \subseteq \mathcal{C}_g \wedge r_{j,k} \leq \tilde{R}_k(t_g)\ (k \in \mathcal{K}) \right\}$. The remaining capacity at t_g is $\tilde{R}_k(t_g) = R_k - \sum_{j \in \mathcal{A}_g} r_{j,k}$. An algorithmic description of the parallel SGS can be given as follows:

Parallel SGS

Initialization: $g = 0, t_g = 0, \mathcal{A}_0 = \{0\}, \mathcal{C}_0 = \{0\}, \tilde{R}_k(0) = R_k$
While $\mid \mathcal{A}_g \cup \mathcal{C}_g \mid \leq n$ do

(1) $g := g + 1$
$t_g = \min_{j \in \mathcal{A}_g} \{F_j\}$
Calculate $\mathcal{C}_g$, $\mathcal{A}_g$, $\tilde{R}_k(t_g)$, $\mathcal{D}_g$

(2) While $\mathcal{D}_g \neq \emptyset$ do

Select one $j \in \mathcal{D}_g$
$F_j = t_g + p_j$
Calculate $\tilde{R}_k(t_g)$, $\mathcal{A}_g$, $\mathcal{D}_g$

$F_{n+1} = \max_{h \in \mathcal{P}_{n+1}} \{F_h\}$

The initilization sets the schedule time to 0, assigns the project start activity to the active and the complete set and sets the available capacity. Each iteration consists of two steps. (1) determines the next schedule time t_g, the associated activity sets $\mathcal{C}_g$, $\mathcal{A}_g$, $\mathcal{D}_g$ and the available capacity $\tilde{R}_k(t_g)$. (2) schedules a non–proper subset of the eligible activities to start at t_g. Table 2 reports the parallel SGS when generating the schedule given in Figure 2. Note that the parallel SGS might have less than n stages but that there are exactly n selection decisions which have to be made.

g	1	2	3	3	4	5	6
t_g	0	4	6	6	9	10	12
$\mathcal{D}_g$	{1,2}	{1,4}	{1,6}	{6}	{}	{3}	{5}
j	2	4	1	6		3	5

Table 2: Example for parallel SGS

As the serial, so does the parallel SGS always generate feasible schedules which are optimal for the resource–unconstrained case. It has been shown by Kolisch [37] that the parallel SGS constructs non–delay schedules. A non–delay schedule is a schedule where, even if activity preemption is allowed, none of the activities can be started earlier without delaying some other activity. The set of non–delay schedules is a non–proper subset of the set of active schedules. It thus has, on average, a smaller cardinality. But it has the severe drawback that it might not contain an optimal schedule with a regular performance measure. E.g., in Kolisch [37] is is shown that out of 298 problem instances from the set j30sm (cf. Kolisch et al. [40]) only 175, i.e. 59.73 % have an optimal solution which is in the set of non–delay schedules. The time complexity of the parallel SGS is $\mathcal{O}(n^2 \cdot K)$.

3 Priority Rule Based Heuristics

Priority rule based heuristics employ one or both of the SGS in order to construct one or more schedules. The priority rule itself is used in order to select an activity j from the decision set $\mathcal{D}_g$. We first give a survey of different priority rules. Thereafter we show how scheduling schemes and priority rules can be combined in order to obtain different priority rule based heuristics.

3.1 Priority Rules

A priority rule is a mapping which assigns each activity j in the decision set $\mathcal{D}_g$ a value $v(j)$ and an objective stating whether the activity with the minimum or the maximum value is selected. In case of ties, one or several tie breaking rules have to be employed. The easiest ways to resolve ties is to choose the activity with the smallest activity label. There has been an overwhelming

Acronym	Priority Rule	Ref.	$v(j)$
GRPW	Greatest Rank Positional Weight	[2]	$p_j + \sum_{i \in \mathcal{S}_j} p_i$
LFT	Minimum Latest Finish Time	[16]	LF_j
LST	Minimum Latest Start Time	[35]	$LF_j - p_j$
MSLK	Minimum Slack	[16]	$LF_j - EF_j$
MTS	Most Total Successors	[2]	$\mid \overline{\mathcal{S}}_j \mid$
RSM	Resource Scheduling Method	[64]	$\max_{(i,j) \in \mathcal{AP}} \{0, t_g + p_j - (LF_i - p_i)\}$
SPT	Shortest Processing Time	[2]	p_j
WCS	Worst Case Slack	[36]	$LF_j - p_j - \max_{(i,j) \in \mathcal{AP}} \{E(i,j)\}$

Table 3: Priority Rules

amount of research on priority rules for the RCPSP; cf. Alvarez–Valdés and Tamarit [2], Boctor [8], Cooper [13, 14], Davies [15], Davis and Patterson [16], Elsayed [22], Kolisch [36, 37], Lawrence [43], Özdamar and Ulusoy [52, 53, 71], Patterson [55, 56], Shaffer et al. [64], Thesen [69], Thomas and Salhi [70], Valls et al. [73], and Whitehouse and Brown [74].

Priority rules can be classified according to different criteria. W.r.t. to the type of information employed to calculate $v(j)$, we can distinguish network, time, and resource based rules (cf. Alvarez–Valdés and Tamarit [2] and Lawrence [43]) as well as lower and upper bound rules. Lower bound rules calculate for each activity a lower bound of the objective function value, upper bound rules calculate for each activity an upper bound of the objective function value. W.r.t. the amount of information employed it can be distinguished in local or global rules. Local rules employ only information from the activity under consideration such as the processing time while global rules make use of a wider range of information. A distinction into static and dynamic rules is made w.r.t. the fact if the value $v(j)$ remains constant or changes during the iterations of the SGS. W.r.t. the SGS we can distinguish in rules which can be employed in the serial, the parallel or both SGS. Table 3 gives an overview of some well known priority rules. The MTS rule employs $\overline{\mathcal{S}}_j$, the set of all (direct and indirect) successors of activity j. The WCS and RSM rules employ $\mathcal{AP} = \{(i,j) \in \mathcal{D}_g \times \mathcal{D}_g \mid i \neq j\}$, the set of all activity pairs (i,j) which are in the decision set $\mathcal{D}_g$. Finally, the WCS rule uses $E(i,j)$, the earliest precedence– and resource–feasible start time of activity j if activity i is started at the schedule time t_g. Note that the RSM rule selects the activity with the lowest priority value.

3.2 Proposed Methods

Priority rule based heuristics combine priority rules and schedule generation schemes in order to construct a specific algorithm. If the heuristic generates a single schedule, it is called a single pass method, if it generates more than one schedule, is is referred to as multi pass method.

3.2.1 Single Pass Methods

The oldest heuristics are *single pass methods* which employ one SGS and one priority rule in order to obtain one feasible schedule. Examples are the heuristics of Alvarez–Valdés and Tamarit [2], Boctor [8], Cooper [13, 14], Davies [15], Davis and Patterson [16], Elsayed [22], Kolisch [36, 37], Lawrence [43], Patterson [55, 56], Thesen [69], Valls et al. [73], and Whitehouse and Brown [74].

Recently, more elaborate priority rules have been proposed by Kolisch [36] as well as Özdamar and Ulusoy [53, 72]. Kolisch [36] developed, amongst other priority rules, the so–called worst case slack (WCS) rule for the parallel SGS which is given in Table 3. Özdamar and Ulusoy [53, 72] introduced the local constraint based analysis (LCBA). LCBA employs the parallel SGS and decides via feasibility checks and so–called essential conditions which activities have to be selected and which activities have to be delayed at the schedule time.

3.2.2 Multi Pass Methods

There are many possibilities to combine SGS and priority rules to a multi pass method. The most common ones are multi priority rule methods, forward–backward scheduling methods, and sampling methods.

Multi priority rule methods employ the SGS several times. Each time a different priority rule is used. Often, the rules are used in the order of descending solution quality. Boctor [8] employed 7 different rules in his experimental study. Instead of using m priority rules in order to generate m schedules, a virtually unlimited number of schedules can be generated by using convex combinations of I priority rules $v(j) = \sum_{i=1}^{I} w_i \cdot v_i(j)$ with $w_i \geq 0$ for all i and $\sum_{i=1}^{I} w_i = 1$. Examples of such approaches are given by Ulusoy and Özdamar [71] and Thomas and Salhi [70]. Ulusoy and Özdamar employed a convex combination of $I = 2$ rules in order to generate 10 different schedules.

Forward–backward scheduling methods employ an SGS in order to iteratively schedule the project by alternating between forward and backward scheduling. Forward scheduling is as outlined in Section 2. Backward scheduling applies one of the SGS to the reversed precedence network where the former end activity $n+1$ has become the new start activity. The priority values are usually obtained from the start or completion times of the lastly generated schedule. Forward–backward scheduling methods have been proposed by, e.g., Li and Willis [46] and Özdamar and Ulusoy [54, 53].

Sampling methods make generally use of one SGS and one priority rule. Different schedules are obtained by biasing the selection of the priority rule through a random device. Instead of a priority value $v(j)$ a selection probability $p(j)$ is computed. At selection decision g of the SGS, $p(j)$ is the probability that activity j from the decision set $\mathcal{D}_g$ will be selected. Dependent on how the probabilities are computed, one can distinguish random sampling, biased random sampling, and regret based biased random sampling [37]. *Random sampling (RS)* assigns each activity in the decision set the same probability $p(j) = 1/|\mathcal{D}_g|$. *Biased random sampling (BRS)* employs the priority values directly in order to obtain the selection probabilities. If the objective of the priority rule is to select the activity with the highest priority value, then the probability is calculated to $p(j) = v(j)/\left(\sum_{i\in\mathcal{D}_g} v(i)\right)$. Biased random sampling methods have been applied by Alvarez–Valdés and Tamarit [1] and Cooper [13]. Schirmer and Riesenberg [62] propose a modification called normalized biased random sampling (NBRS) which essentially ensures that the selection probability of the activity with the smallest (highest) priority value is the same when seeking for the activity with the highest (smallest) priority value. *Regret based biased random sampling (RBRS)* uses the priority values indirectly via regret values. If the objective is again to select the activity with the highest priority value, the regret value $r(j)$ is the absolute difference between the priority value $v(j)$ of the activity under consideration and the worst priority value of all activities in the decision set, i.e. $r(j) = v(j) - \min_{i\in\mathcal{D}_g}\{v(i)\}$. Before calculating the selection probabilities based on the regret values, the latter can be modified by $r'(j) = (r(j)+\epsilon)^{\alpha}$ (cf. Drexl [20]).

$j \in \mathcal{D}_g$	1	2	3
$v(j)$	11	13	20
Random Sampling	0.33	0.33	0.33
Biased Random Sampling	0.25	0.30	0.45
Regret Based Biased Random Sampling	0.07	0.21	0.72

Table 4: Selection Probabilities $p(j)$ for Different Sampling Methods

Adding the constant $\epsilon > 0$ to the regret value assures that the selection probability for each activity in the decision set is greater than zero and thus every schedule of the population can be generated. With the choice of the parameter α the amount of bias can be controlled. A high α will cause no bias and thus deterministic activity selection while an α of zero will cause maximum bias and hence random activity selection. Kolisch [35] found out that, in general, $\epsilon = \alpha = 1$ will provide good results. Drexl [20] uses $\epsilon = \min_{i \in \mathcal{D}_g} v(i)$. Schirmer and Riesenberg [62] propose a modified regret based biased random sampling (MRBRS) where ϵ is determined dynamically. Experimental comparisons performed by Kolisch [35] as well as Schirmer and Riesenberg [62] revealed (modified) regret based biased random sampling as the best sampling approach. Table 4 gives examplary different selection probabilities for $\mid \mathcal{D}_g \mid = 3$, $\epsilon = 1$, and $\alpha = 1$.

A hybrid multi pass approach has been proposed by Kolisch and Drexl [38]. The heuristic applies the serial SGS with the LFT–priority rule and the parallel SGS with the WCS–priority rule while employing deterministic and regret based sampling activity selection. The decision on the specific method is based on an analysis of the problem at hand and the number of iterations already performed. Partial schedules are discarded by the use of lower bounds. Schirmer and Riesenberg [61] have extended this approach by employing both schedule generation schemes together with four different priority rules (MTS,LFT,LST,WCS) and two different sampling schemes (MRBRS,RBRS).

Table 5 gives a survey of priority rule based heuristics for the RCPSP.

4 Metaheuristic Approaches

4.1 General Metaheuristic Strategies

Several metaheuristic strategies have been developed to solve hard optimization problems. The following summary briefly describes those general aproaches that have been used to solve the RCPSP.

4.1.1 Simulated Annealing

Simulated Annealing (SA), introduced by Kirkpatrick et al. [33], originates from the physical annealing process in which a melted solid is cooled down to a low–energy state. Starting with some initial solution, a so–called neighbor solution is generated by slightly perturbing the current one. If this new solution is better than the current one, it is accepted, and the search proceeds from this new solution. Otherwise, if it is worse, the new solution is only accepted with a probability that depends on the magnitude of the deterioration as well as on a parameter called temperature.

Author(s)	SGS	Priority Rule	Sampling	Passes
Alvarez-Valdés/Tamarit [2]	par.	several rules	—	single
Alvarez-Valdés/Tamarit [1]	par.	several rules	BRS	multi
Boctor [8]	par./ser.	several rules	—	multi
Cooper [13]	ser.	several rules	BRS	multi
Davis/Patterson [16]	par.	LFT and other	—	single
Kolisch [37]	par./ser.	several rules	RBRS	multi
Kolisch [36]	par.	WCS and other	—	single
Kolisch [35]	par./ser.	several rules	RS, BRS, RBRS	multi
Kolisch/Drexl [38]	par./ser.	LFT/WCS	RBRS	multi
Li/Willis [46]	par.	start & finish times	—	multi
Özdamar/Ulusoy [54, 53]	par.	LCBA	—	multi
Özdamar/Ulusoy [52]	par.	LCBA	—	single
Shaffer et al. [64]	par.	RSM	—	single
Schirmer/Riesenberg [61]	par./ser.	several rules	RBRS,MRBRS	multi
Schirmer/Riesenberg [62]	par./ser.	several rules	diff. sampling methods	multi
Thomas/Salhi [70]	par.	convex combination	—	multi
Ulusoy/Özdamar [71]	par.	convex combination	—	multi

Table 5: Survey of priority rule based heuristics for the RCPSP

As the algorithm proceeds, this temperature is reduced in order to lower the probability to accept worse neighbors.

Clearly, SA can be viewed as an extension of a simple greedy procedure, sometimes called *First Fit Strategy (FFS)*, which immediately accepts a better neighbor solution but rejects any deterioration.

4.1.2 Tabu Seach

Tabu Search (TS), developed by Glover [23, 24], is essentially a steepest descent/mildest ascent method. That is, it evaluates all solutions of the neighborhood and chooses the best one, from which it proceeds further. This concept, however, bears the possibility of cycling, that is, one may always move back to the same local optimum one has just left. In order to avoid this problem, a tabu list is set up as a form of memory for the search process. Usually, the tabu list is used to forbid those neighborhood moves that might cancel the effect of recently performed moves and might thus lead back to a recently visited solution. Typically, such a tabu status is overrun if the corresponding neighborhood move would lead to a new overall best solution (aspiration criterion).

It is obvious that TS extends the simple steepest descent search, often called *Best Fit Strategy (BFS)*, which scans the neighborhood and then accepts the best neighbor solution, until none of the neighbors improves the current objective function value.

4.1.3 Genetic Algorithms

Genetic Algorithms (GA), inspired by the process of biological evolution, have been introduced by Holland [30]. In contrast to the local search strategies above, a GA simultaneously considers a set or population of solutions instead of only one. Having generated an initial population, new solutions are produced by mating two existing ones (crossover) and/or by altering an existing

one (mutation). After producing new solutions, the fittest solutions "survive" and make up the next generation while the others are deleted. The fitness value measures the quality of a solution, usually based on the objective function value of the optimization problem to be solved.

4.2 Representations

Once a metaheuristic strategy has been chosen to attack a given optimization problem, one has to select a suitable representation for solutions. Usually, metaheuristic approaches for the RCPSP rather operate on representations of schedules than on schedules themselves. Then an appropriate decoding procedure must be selected to transform the representation into a schedule. Finally, operators are needed to produce new solutions w.r.t. the selected representation. A unary operator constructs a new solution from an existing one, that is, it makes up the neighborhood move in local search procedure such as SA and TS as well as the mutation in a GA. A binary operator constructs a new solution from two existing ones, as done by crossover in a GA.

This subsection summarizes five representations reported in the literature that have been used within metaheuristic approaches to solve the RCPSP. For each representation, we give the related decoding procedures and operators. In order to keep the description short, we will restrict the definition of binary operators to the one–point crossover type for the different representations. Roughly speaking, the one–point crossover splits two existing solutions and takes one part from the first and one part from the second solution in order to form a new one. Other general crossover types that are well known from the GA literature (such as two–point and uniform crossover) can be easily obtained from extending the one–point definitions given here, see e.g. Hartmann [25].

4.2.1 Activity List Representation

In the activity list representation, a precedence feasible activity list

$$\lambda = \langle j_1, j_2, \ldots, j_n]$$

is given, in which each activity must have a higher index than each of its predecessors. As shown in Section 2, the serial SGS can be used as a decoding procedure to obtain a schedule from an activity list. Note, however, that the parallel scheme cannot be applied without modification. As is easily verified, the serial SGS transforms example activity list

$$\lambda^E = \langle 2, 4, 6, 1, 3, 5]$$

for the project of Figure 1 into the schedule of Figure 2. The initial solution(s) can be generated by randomly selecting an activity from the decision set in each step of the SGS. To obtain better solution quality, one can also use a priority rule or priority rule based sampling scheme for choosing an eligible activity. In either case, recording the activities in the order of their selection results in a (precedence feasible) activity list.

Several unary operators have been proposed for the activity list representation, see e.g. Della Croce [17]. The so-called pairwise interchange is defined as swapping two activities j_q and j_s, $q, s \in \{1, \ldots, n\}$ with $q \neq s$, if the resulting activity list is precedence feasible. As a special case, the adjacent pairwise interchange swaps two activities j_q and j_{q+1}, $q \in \{1, \ldots, n-1\}$, that are adjacent in λ but not precedence related. Considering again the example project of Figure 1, we could apply the adjacent pairwise interchange for $q = 3$ to λ^E as given above and obtain

$$\lambda^N = \langle 2, 4, 1, 6, 3, 5].$$

Furthermore, the simple shift operator selects some activity j_q and inserts it immediately after some other activity j_s, if the precedence constraints are not violated. In our example, shifting activity 6 immediately after activity 3 in λ^E results in neighbor activity list

$$\lambda^{N'} = \langle 2, 4, 1, 3, 6, 5]\,.$$

More sophisticated shift operators have been proposed by Baar et al. [3] for the RCPSP. They make use of the schedule $S(\lambda)$ that is represented by the current activity list λ. The operators are based on the notion of a critical arc which is defined as a pair of successively scheduled activities (i, j), that is, $F_i + p_j = F_j$ in $S(\lambda)$. The underlying idea is that at least one critical arc must become non–critical to improve the current schedule. Hence, Baar et al. define three shift operators that may cancel a critical arc. They extend the simple shift by allowing more than one activity to be shifted. Without giving the formal definitions here, we illustrate such a shift operator on the critical arc (4,1) in the schedule $S(\lambda^E)$ shown in Figure 2: Shifting activity 4 and its successor activity 6 immediately after activity 1 leads to neighbor activity list

$$\lambda^{N''} = \langle 2, 1, 4, 6, 3, 5]\,.$$

As a binary operator, i.e. as crossover for a GA, Hartmann [25] used the following technique: Given two "parent" activity lists, a "mother" $\lambda^M = \left\langle j_1^M, \ldots, j_n^M \right]$ and a "father" $\lambda^F = \left\langle j_1^F, \ldots, j_n^F \right]$, the "child" activity list $\lambda^C = \left\langle j_1^C, \ldots, j_n^C \right]$ is defined as follows: After drawing a random integer q with $1 \leq q < n$, the positions $i = 1, \ldots, q$ are taken from the "mother" λ^M by setting $j_i^C := j_i^M$. The activity list of positions $i = q+1, \ldots, n$ is taken from the "father" λ^F. However, the jobs that have already been taken from the mother may not be considered again. We obtain $j_i^C := j_k^F$ where k is the lowest index such that $j_k^F \notin \{j_1^C, \ldots, j_{i-1}^C\}$. Choosing $q = 3$, this definition is illustrated for our project example by

$$\begin{aligned} \lambda^M &= \langle 1, 3, 2, 5, 4, 6], \\ \lambda^F &= \langle 2, 4, 6, 1, 3, 5], \\ \lambda^C &= \langle 1, 3, 2, 4, 6, 5]\,. \end{aligned}$$

As shown by Hartmann [25], the "child" activity list λ^C resulting from two precedence feasible "parent" activity lists λ^M and λ^F is also precedence feasible.

4.2.2 Random Key Representation

Several researchers employed a representation which makes use of an array

$$\rho = (r_1, r_2, \ldots, r_n)$$

that assigns a (typically real-valued) number r_j to each activity j. Following Bean [4], we will call such an encoding random key representation. It is similar to the priority value representation of Lee and Kim [44] and Cho and Kim [11] and the problem-space based representation of Storer et al. [68], Leon and Ramamoorthy [45], and Naphade et al. [49], such that we will discuss these three approaches in a unified framework here.

For an initial solution, the random keys are usually chosen randomly (see e.g. Lee and Kim [44]) or computed by some priority rule (see e.g. Leon and Ramamoorthy [45]).

Both the parallel and the serial SGS can be used to derive a schedule from ρ: On each stage g, we can select activity j with the highest random key $r_j = \max\{r_i \mid i \in \mathcal{D}_g\}$ from the decision set $\mathcal{D}_g$ (clearly, if initialized with e.g. the latest finish time, one would select the activity with the minimum random key). In other words, the random keys play the role of priority values. Considering again our example project, we obtain the schedule of Figure 2 from applying either the parallel or the serial SGS to

$$\rho^E = (0.58, 0.64, 0.31, 0.87, 0.09, 0.34).$$

An alternative approach is proposed by Naphade et al. [49]. Here, the random keys are used to perturb activity slacks which serve as priority values.

While both the parallel and the serial SGS as decoding procedures guarantee that only feasible schedules are found, only the serial one ensures the existence of at least one optimal schedule in the solution space, as discussed in Section 2. In order to overcome the drawback of possible exclusion of all optimal solutions by the parallel SGS, several researchers (see Cho and Kim [11], Naphade et al. [49], and Leon and Ramamoorthy [45]) introduced different modifications of the parallel SGS as decoding procedures for the random key representation. These essentially allow to delay a schedulable activity such that the search is not restricted to non–delay schedules.

As a unary operator, any pairwise interchange of r_j and r_i can be employed, including the adjacent pairwise interchange of r_j and r_{j+1}. Considering a pairwise interchange with $j = 2$ and $i = 4$, an example neighbor of ρ^E is

$$\rho^N = (0.58, 0.87, 0.31, 0.64, 0.09, 0.34).$$

In an approach for the job shop problem, Storer et al. [68] proposed the so–called problem–space based neighborhood which randomly reselects $r_j^{new} \in [r_j^{old} - \epsilon \cdot r_j^{old}, r_j^{old} + \epsilon \cdot r_j^{old}]$ from a uniform distribution, where ϵ is a real-valued constant. For this neighborhood definition with $\epsilon = 0.1$ an example neighbor of ρ^E is given by

$$\rho^{N'} = (0.59, 0.62, 0.34, 0.89, 0.09, 0.33).$$

The random key representation allows the application of the standard one-point crossover as binary operator: Given a random integer q with $1 \leq q < n$, a new random key array $\rho^C = (r_1^C, \ldots, r_n^C)$ is derived by taking the first q random keys from a "mother" array $\rho^M = (r_1^M, \ldots, r_n^M)$ and the remaining ones from a "father" array $\rho^F = (r_1^F, \ldots, r_n^F)$. We obtain $r_i^C = r_i^M$ for $i = 1, \ldots, q$ and $r_i^C = r_i^F$ for $i = q+1, \ldots, n$. An example for $q = 3$ is

$$\begin{aligned} \rho^M &= (0.58, 0.64, 0.31, 0.87, 0.09, 0.34), \\ \rho^F &= (0.12, 0.43, 0.99, 0.65, 0.19, 0.22), \\ \rho^C &= (0.58, 0.64, 0.31, 0.65, 0.19, 0.22). \end{aligned}$$

4.2.3 Priority Rule Representation

The priority rule representation, used by e.g. Dorndorf and Pesch [19] for the job shop problem and adapted by Hartmann [25] to the RCPSP, is based on a list of priority rules

$$\pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle,$$

where each π_i is a priority rule. As decoding procedures, both the parallel and the serial SGS can be used by selecting the i-th activity to be scheduled according to priority rule π_i. For our example project, the schedule of Figure 2 can be obtained from e.g.

$$\pi^E = \langle \text{LST, GRPW, MTS, LST, MSLK, LFT} \rangle .$$

The commonly used unary operator randomly selects a new priority rule for some π_i. For the example priority list π^E as given above, a possible neighbor is

$$\pi^N = \langle \text{MTS, GRPW, MTS, LST, MSLK, LFT} \rangle .$$

The binary operator, i.e. the one-point crossover, follows again the standard definition. Given a random integer q with $1 \leq q < n$, a new priority rule list π^C is derived by taking the first q priority rules from a "mother" list π^M and the remaining ones from a "father" list π^F. Consider as an example $q = 3$ and

$$\begin{aligned} \pi^M &= \langle \text{LST, GRPW, MTS, LST, MSLK, LFT} \rangle , \\ \pi^F &= \langle \text{LFT, GRPW, MSLK, SPT, LFT, GRPW} \rangle , \\ \pi^C &= \langle \text{LST, GRPW, MTS, SPT, LFT, GRPW} \rangle . \end{aligned}$$

4.2.4 Shift Vector Representation

The shift vector representation has been proposed by Sampson and Weiss [60] for the RCPSP. A solution is represented by a shift vector

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n),$$

where σ_j is a nonnegative integer. As a decoding procedure, an extension of the classical forward recursion (cf. Elmaghraby [21]) is used, in which the start time S_j of an activity j is calculated as the maximum of the finish times of its predecessors plus the shift σ_j of activity j, that is, $S_0 = 0$ and $S_j = \max\{S_h + p_h \mid h \in \mathcal{P}_j\} + \sigma_j$ for $j = 1, \ldots, n+1$. The following shift vector for our example project leads to the schedule of Figure 2:

$$\sigma^E = (6, 0, 1, 0, 0, 0)$$

As this decoding procedure does not consider the resource constraints, a schedule derived from a shift vector may be infeasible. This is illustrated by the following shift vector which forces activities 1 and 4 to be simultaneously in process, thus exceeding the resource capacity by 2 units:

$$\sigma^I = (4, 0, 1, 0, 0, 0)$$

Consequently, the objective function is extended by penalizing the violation of the resource constraints.

The neighborhood of a shift vector σ is given by those vectors that differ from σ in exactly one position and do not lead to a project makespan that exceeds some given upper bound. An example neighbor for shift vector σ^E above is

$$\sigma^N = (6, 0, 1, 0, 1, 0) .$$

4.2.5 Schedule Scheme Representation

The schedule scheme representation has been introduced by Brucker et al. [10] for a branch–and–bound algorithm for the RCPSP. In what follows we give a brief and rather informal description. A schedule scheme (C, D, N, F) consists of four disjoint relations. $(i, j) \in C$ implies that activity i must be finished before activity j can be started (conjunctions). $(i, j) \in D$ implies that activities i and j may not overlap (disjunctions). $(i, j) \in N$ implies that activities i and j must be processed in parallel in at least one period (parallelity relations). For activities i and j with $(i, j) \in F$ there are no restrictions (flexibility relations). A schedule scheme represents those (not necessarily feasible) schedules in which the related relations are maintained. As a decoding procedure, Baar et al. [3] develop a heuristic that constructs a feasible schedule in which all relations of C and D and a "large" number of parallelity relations N are satisfied.

Baar et al. [3] introduce a neighborhood definition which basically consists of moves that transform flexibility relations into parallelity relations and parallelity relations into flexibility relations. The neighborhood size is reduced by a critical path calculation and impact estimations for the moves.

We do not discuss this representation and its neighborhood definition in more detail here, referring the reader to Chapter ? (cf. Brucker et al. []).

4.3 Proposed Methods

In this subsection, we briefly describe the metaheuristic approaches which are documented for the RCPSP. The references are listed in alphabetical order. We have, however, restricted the list to those papers that consider the RCPSP, while papers that cover its extensions (such as the multi-mode RCPSP) are not included. A summarizing overview is given in Table 6. For each reference, we denote the employed metaheuristic strategies together with the underlying representation, the method employed to generate initial solutions, the SGS, and the used operator. In case of a GA, we only mention the crossover operator.

Baar et al. [3] develop two TS algorithms. The first one is based on the activity list representation in accordance with the serial SGS. The neighborhood is defined by three kinds of critical path based moves. Their second TS approach employs the schedule scheme representation with the related decoding procedure and neighborhood definition. Both TS algorithms use dynamic tabu lists as well as priority based start heuristics.

Bouleimen and Lecocq [9] propose an SA procedure based on the activity list representation together with the serial SGS. For neighborhood moves, the shift operator is used.

Cho and Kim [11] modify the SA algorithm of Lee and Kim [44] (see below) by extending the random key representation in order to allow the delay of schedulable activities within an adapted parallel SGS.

Hartmann [25] proposes a GA based on the activity list representation and compares it to GAs which make use of the random key and priority rule representations, respectively. All three approaches employ the serial SGS as well as two–point crossover operators related to the respective representation. In the activity list based GA, the regret based biased random sampling method with the serial SGS and the LFT–rule (see Section 3) is used to determine the initial generation.

Kohlmorgen et al. [34] develop a GA which employs the random key representation with standard two–point crossover. They test their approach on a massively parallel computer.

Lee and Kim [44] propose an SA algorithm, a TS procedure and a GA. These three approaches

Paper	Metah.	Repres.	Initial Sol.	SGS	Operator
Baar et al. [3]	TS	activity list	prio. rule	serial	crit. path shifts
	TS	sched. scheme	prio. rule	relat. heur.	related moves
Bouleimen, Lecocq [9]	SA	activity list	prio. rule	serial	shift
Cho, Kim [11]	SA	random key	random	mod. par.	pairw. int.
Hartmann [25]	GA	activity list	prio. rule	serial	two–point cr.
	GA	random key	random	serial	two–point cr.
	GA	prio. rule	random	serial	two–point cr.
Kohlmorgen et al. [34]	GA	random key	prio. rule	serial	two–point cr.
Lee, Kim [44]	SA	random key	random	par.	pairw. int.
	TS	random key	random	par.	pairw. int.
	GA	random key	random	par.	one–point cr.
Leon, Ramamoorthy [45]	FFS	random key	prio. rule	mod. par.	problem–space
	BFS	random key	prio. rule	mod. par.	problem–space
	GA	random key	prio. rule	mod. par.	one–point cr.
Naphade et al. [49]	BFS	random key	prio. rule	mod. par.	problem–space
Pinson et al. [57]	TS	activity list	prio. rule	serial	adj. pairw. int.
	TS	activity list	prio. rule	serial	pairw. int.
	TS	activity list	prio. rule	serial	shift
Sampson, Weiss [60]	SA-var.	shift vector	null-vector	ext. recurs.	related move

Table 6: Survey of metaheuristic strategies for the RCPSP

are based on the random key representation with the parallel SGS as decoding procedure. While SA and TS make use of a restricted version of the pairwise interchange move, the GA employs the standard one–point crossover.

Leon and Ramamoorthy [45] test an FFS and a BFS approach as well as a GA. They employ the problem–space based version of the random key representation. The random keys are initialized with values computed by a priority rule. A modified variant of the parallel SGS serves as decoding procedure. The unary operator is defined by the problem–space based neighborhood, and the binary one is the standard one–point crossover.

Naphade et al. [49] use the BFS concept with the problem–space based variant of the random key representation. The random keys are initialized with the latest finish times of the activities and then modified according to the problem-space based neighborhood. As decoding procedure, they employ a modified parallel SGS, where the random keys are used to calculate slack based priority values.

Pinson et al. [57] suggest several variants of a TS approach based on the activity list representation, the serial SGS, and a priority rule procedure for computing a start solution. The variants differ in the neighborhood definitions, using the adjacent pairwise interchange, the (general) pairwise interchange, and the shift move, respectively.

Sampson and Weiss [60] suggest a local search procedure which can be viewed as a variant of TA. Their approach is based on the shift vector representation and the related neighborhood definition.

5 Other Heuristics

5.1 Truncated Branch and Bound Methods

Pollack–Johnson [58] uses a so–called depth–first, jumptracking branch and bound search of a partial solution tree. The algorithm is essentially a parallel scheduling heuristic. Instead of scheduling the activity with the highest priority value it branches on certain occasions such that one branch has the activity with the highest priority value and the other branch has the activity with the second highest priority value, which is scheduled next. Note, due to use of the parallel SGS optimal solution might be excluded from the search space.

Sprecher [65] employs his depth–first search branch and bound procedure as a heuristic by imposing a time limit. The enumeration process is guided by the so–called precedence tree which essentially branches on the activities in the decision set of the serial SGS. Via backtracking, all precedence feasible activity lists are (implicitly) enumerated. In order to obtain good solutions early in the search process (and thus within the time limit), priority rules are applied to select the most promising activity from the decision set for branching first.

5.2 Disjunctive Arc Based Methods

The basic idea of the disjunctive–arc–based approaches is to extend the precedence relations (the set of conjunctive arcs) by adding additional arcs (the disjunctive arcs) such that the minimal forbidden sets, i.e. sets of technologically independent activities which cannot be scheduled simultaneously due to resource constraints, are destroyed and thus the earliest finish schedule is feasible with respect to (precedence and) resource constraints.

Shaffer et al. [64] restrict the scope, within their "resource scheduling method", to those forbidden sets for which all activities in the earliest finish schedule are processed at the same time. The disjunctive arc which produces the smallest increase in the earliest finish time of the unique sink is introduced and the earliest finish schedule is recalculated. The algorithm terminates as soon as a (precedence– and) resource–feasible earliest finish schedule is found. Note that this approach can be transformed into a single-pass priority rule method based on the parallel SGS, cf. Tables 3 and 5.

Alvarez–Valdés and Tamarit [2] propose four different ways of destroying the minimal forbidden sets. The best results were achieved by applying the following strategy: Beginning with the minimal forbidden sets of lowest cardinality, one set is arbitrarily chosen and destroyed by adding the disjunctive arc for which the earliest finish time of the unique dummy sink is minimal.

Bell and Han [5] present a two–phase algorithm for this problem. The first phase is very similar to the approach of Shaffer et al. However, phase 2 tries to improve the feasible solution obtained by phase one as follows: after removing redundant arcs, each disjunctive arc that is part of the critical path(s) is temporarily cancelled and the phase 1 procedure is applied again.

5.3 Further Approaches

Integer programming based heuristics have been used by Oğuz and Bala [51]. The method employs the integer programming formulation originally proposed by Pritsker et al. [59]. The planning horizon is divided in T periods of equal length and the processing times p_j have to be given as discrete multiples of one period. The binary decision variable is $x_{j,t} = 1$ if activity j is finished

at the end of period t.

Mausser and Lawrence [47] use block structures to improve the makespan of projects. They start by generating a feasible solution with a parallel scheduling scheme. Following this, they identify blocks which represent contiguous time spans that completely contain all activities processed within it. Each such block can be considered independent of the other blocks. The method essentially reschedules individual blocks in order to shorten the overall project length.

6 Computational Analysis

6.1 Test Design

This section reports on a computational comparison of several of the heuristics summarized above. As test instances we have employed the standard sets j30, j60, and j120 for the RCPSP which have been generated using ProGen (cf. Kolisch et al. [42]). The sets j30 and j60 consist of 480 projects with $n = 30$ and $n = 60$ activities, respectively. The set j120 consists of 600 projects, each with $n = 120$ activities. Details of these problem instances are given in Chapter ? of this book (cf. Kolisch et al. [40]).

Each algorithm was tested by its author(s) using the original implementation. This allowed the authors to adjust the parameters in order to obtain good results. As a consequence, however, the tests were performed on different computer architectures and operating systems. Therefore, we could not impose a bound on the computation time to provide a basis for the comparison. Instead, we have chosen to limit the number of generated and evaluated schedules to 1000 and 5000, respectively. This decision is based on the assumption that the effort needed for generating one schedule is similar in the tested heuristics. With the exception of the schedule scheme representation based TS approach of Baar et al. [3] all algorithms considered for our investigation make use of an SGS as described in Section 2. Hence, we found this assumption justified.

6.2 Results

Tables 7–11 display the results of the computational comparison. The heuristics are sorted according to descending performance with respect to 5000 iterations. Table 7 summarizes the percentage deviations from the optimal makespan for the instance set j30. As for the other two instance sets some of the optimal solutions are not known, we measured for these sets the average percentage deviation from an upper and a lower bound, respectively. The upper bound was set to the lowest makespan found by any of the tested heuristics while the lower bound was selected to be the critical path based lower bound (cf. Stinson et al. [67]). We employed the lower bound in order to allow researchers to compare their results with the ones obtained in this study. All lower and upper bounds can be obtained from the authors upon request. For the j60 set, the percentage deviations from the upper and lower bounds are reported in Tables 8 and 10, respectively. Note, that the schedule scheme based TS heuristic of Baar et al. [3] was additionally run by allowing 2 trials where each trial was terminated after no improved solution was found after 250 iterations. This way, the deviation from the upper bound was lowered to 1.14 %. Finally, Tables 9 and 11 provide the respective deviations for the j120 set.

The heuristics that performed best in our study are the SA of Bouleimen and Lecocq [9] and the GA of Hartmann [25]. While the procedure of Bouleimen and Lecocq performs best on the j30 set, the approach of Hartmann dominates on the instance sets with larger projects.

Algorithm	SGS	Reference	Iterations 1000	5000
SA – activity list	serial	Bouleimen, Lecocq [9]	0.38	0.23
GA – activity list	serial	Hartmann [25]	0.54	0.25
TS – schedule scheme	special heuristic	Baar et al. [3]	0.86	0.44
sampling – adaptive	serial/parallel	Kolisch, Drexl [38]	0.74	0.52
sampling – LFT	serial	Kolisch [37]	0.83	0.53
sampling – adaptive	serial/parallel	Schirmer, Riesenberg [61]	0.71	0.59
sampling – WCS	parallel	Kolisch [36]	1.40	1.28
sampling – LFT	parallel	Kolisch [37]	1.40	1.29
GA – random key	mod. parallel	Leon, Ramamoorthy [45]	2.08	1.59

Table 7: Average deviations from optimal solution — $J = 30$

Algorithm	SGS	Reference	Iterations 1000	5000
GA – activity list	serial	Hartmann [25]	0.99	0.45
SA – activity list	serial	Bouleimen, Lecocq [9]	1.17	0.49
sampling – adaptive	serial/parallel	Schirmer, Riesenberg [61]	1.26	0.97
sampling – adaptive	serial/parallel	Kolisch, Drexl [38]	1.60	1.26
TS – schedule scheme	special heuristic	Baar et al. [3]	1.79	1.54
sampling – LFT	serial	Kolisch [37]	1.88	1.55
sampling – LFT	parallel	Kolisch [37]	1.83	1.56
sampling – WCS	parallel	Kolisch [36]	1.88	1.56
GA – random key	mod. parallel	Leon, Ramamoorthy [45]	2.48	1.82

Table 8: Average deviations from best solution — $J = 60$

Algorithm	SGS	Reference	Iterations 1000	5000
GA – activity list	serial	Hartmann [25]	2.59	0.89
SA – activity list	serial	Bouleimen, Lecocq [9]	5.73	1.86
sampling – LFT	parallel	Kolisch [37]	2.92	2.32
sampling – WCS	parallel	Kolisch [36]	2.94	2.34
sampling – adaptive	serial/parallel	Schirmer, Riesenberg [61]	3.28	2.55
sampling – adaptive	serial/parallel	Kolisch, Drexl [38]	3.95	3.33
GA – random key	mod. parallel	Leon, Ramamoorthy [45]	5.33	3.76
sampling – LFT	serial	Kolisch [37]	4.78	4.10

Table 9: Average deviations from best solution — $J = 120$

Algorithm	SGS	Reference	Iterations 1000	5000
GA – activity list	serial	Hartmann [25]	12.68	11.89
SA – activity list	serial	Bouleimen, Lecocq [9]	12.75	11.90
sampling – adaptive	serial/parallel	Schirmer, Riesenberg [61]	13.02	12.62
sampling – adaptive	serial/parallel	Kolisch, Drexl [38]	13.51	13.06
sampling – WCS	parallel	Kolisch [36]	13.66	13.21
sampling – LFT	parallel	Kolisch [37]	13.59	13.23
TS – schedule scheme	special heuristic	Baar et al. [3]	13.80	13.48
GA – random key	mod. parallel	Leon, Ramamoorthy [45]	14.33	13.49
sampling – LFT	serial	Kolisch [37]	13.96	13.53

Table 10: Average deviations from critical path based lower bound — $J = 60$

Algorithm	SGS	Reference	Iterations 1000	5000
GA – activity list	serial	Hartmann [25]	39.37	36.74
SA – activity list	serial	Bouleimen, Lecocq [9]	42.81	37.68
sampling – LFT	parallel	Kolisch [37]	39.60	38.75
sampling – WCS	parallel	Kolisch [36]	39.65	38.77
sampling – adaptive	serial/parallel	Schirmer, Riesenberg [61]	40.08	39.08
sampling – adaptive	serial/parallel	Kolisch, Drexl [38]	41.37	40.45
GA – random key	mod. parallel	Leon, Ramamoorthy [45]	42.91	40.69
sampling – LFT	serial	Kolisch [37]	42.84	41.84

Table 11: Average deviations from critical path based lower bound — $J = 120$

Generally, the results show that the best metaheuristic strategies outperform the best priority rule based sampling approaches. Increasing the number of schedules allowed to be computed further increases the superiority of the metaheuristics. This is mainly because sampling procedures generate each schedule anew without considering any information given by already visited solutions while metaheuristic algorithms typically exploit the knowledge gained from the previously evaluated schedule(s).

A comparison of the results obtained from the metaheuristics shows that the choice of the underlying representation is crucial. The two best procedures make use of different metaheuristic paradigms while they both employ the activity list representation. The use of one metaheuristic paradigm itself does not necessarily lead to consistently good solutions. This can be seen by the results of the two GA's of Hartmann [25] and Leon and Ramamoorthy [45]. The activity list based GA excells the problem space based GA.

Analyzing the priority rule based sampling procedures, we observe a strong influence of the SGS when used together with the LFT rule. While the serial SGS leads to better results on the j30 instance set, the parallel one is superior on the j60 and j120 instance sets, respectively. The two rules WCS and LFT give almost identical results when employed within the parallel SGS. The adaptive sampling strategies do not consistently dominate the simple sampling procedures. Compared to each other, we observe that the adaptive sampling approach of Schirmer and Riesenberg [61] outperforms the one of Kolisch and Drexl [38] on the j60 and j120 instance sets while the latter yields better results on the j30 set.

We finally remark that the metaheuristic algorithms which make use of the activity list representation can be assumed to be the fastest approaches. This is due to the fact that the underlying serial SGS for activity lists (cf. Section 2) does not compute the eligible set or select an activity on the basis of priority values.

7 Outlook on Research Opportunities

Our computational results indicate that the best heuristics currently available are metaheuristic strategies which make use of activity lists. Further investigations which are currently under way (cf. Hartmann [27]) indicate that the exploitation of problem-specific knowledge is crucial when designing good metaheuristic strategies. This fact is well known from other classical scheduling problems such as the job shop problem (cf. Nowicki and Smutnicki [50]). Further investigations (cf. Hartmann and Kolisch [28]) are headed towards a deeper insight of the functioning of different heuristics subject to different problem characteristics as given in Kolisch et al. [40, 42] and Kolisch and Sprecher [41].

Although priority rule based methods do not give the best results, they are important for several reasons. First, they are indispensable when solving large problem instances in a short amount of time. Second, good priority rule based methods are needed to determine the initial solution(s) for metaheuristic procedures. Hence, further efforts in this area are still justified.

Summarizing, recent years have brought a considerable progress in designing more efficient heuristics for the RCPSP, and we believe that this will remain a fruitful field of research in the future. In addition to the development of even better heuristics, extending the current approaches to more general project scheduling problems are of special interest.

Acknowledgement. We are indebted to Tonius Baar, Peter Brucker, and Sigrid Knust (University of Osnabrück), Kamel Bouleimen and Henri Lecocq (University of Liège), Jorge Leon

and Balakrishnan Ramamoorthy (Texas A&M University) as well as Andreas Schirmer and Sven Riesenberg (University of Kiel) for their help in this research. Furthermore, we would like to thank Andreas Drexl for his continuous support.

References

[1] R. Alvarez-Valdés and J.M. Tamarit. Algoritmos heurísticos deterministas y aleatorios en secuenciacón de proyectos con recursos limitados. *Qüestiió*, 13:173–191, 1989.

[2] R. Alvarez-Valdés and J.M. Tamarit. Heuristic algorithms for resource–constrained project scheduling: A review and an empirical analysis. In R. Słowiński and J. Węglarz, editors, *Advances in project scheduling*, pages 113–134. Elsevier, Amsterdam, 1996.

[3] T. Baar, P. Brucker, and S. Knust. Tabu–search algorithms for the resource–constrained project scheduling problem. Technical report, Osnabrücker Schriften zur Mathematik, Fachbereich Mathematik/Informatik, Osnabrück.

[4] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[5] C.E. Bell and J. Han. A new heuristic solution method in resource–constrained project scheduling. *Naval Research Logistics*, 38:315–331, 1991.

[6] N. Bixby and E. Boyed. *Using the CPLEX callable library.* CPLEX Optimization Inc., Houston, 1996.

[7] J. Błażewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[8] F.F. Boctor. Some efficient multi–heuristic procedures for resource–constrained project scheduling. *European Journal of Operational Research*, 49:3–13, 1990.

[9] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource–constrained project scheduling problem. Technical report, Service de Robotique et Automatisation, Université de Liège, 1998.

[10] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch & bound algorithm for the resource–constrained project scheduling problem. *European Journal of Operational Research*, forthcoming.

[11] J.-H. Cho and Y.-D. Kim. A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, 48:735–744, 1997.

[12] N. Christofides, R. Alvarez-Valdés, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

[13] D.F. Cooper. Heuristics for scheduling resource–constrained projects: An experimental investigation. *Management Science*, 22(11):1186–1194, 1976.

[14] D.F. Cooper. A note on serial and parallel heuristics for resource–constrained project scheduling. *Foundations of Control Engineering*, 2(4):131–133, 1977.

[15] E.M. Davies. An experimental investigation of resource allocation in mulitactivity projects. *Operational Research Quarterly*, 24:587–591, 1973.

[16] E.W. Davis and J.H. Patterson. A comparison of heuristic and optimum solutions in resource–constrained project scheduling. *Management Science*, 21:944–955, 1975.

[17] F. Della Croce. Generalized pairwise interchanges and machine scheduling. *European Journal of Operational Research*, 83:310–319, 1995.

[18] E. Demeulemeester and W. Herroelen. New benchmark results for the resource–constrained project scheduling problem. *Management Science*, 43(11):1485–1492, 1997.

[19] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25–40, 1995.

[20] A. Drexl. Scheduling of project networks by job assignment. *Management Science*, 37(12):1590–1602, 1991.

[21] S.E. Elmaghraby. *Activity networks: Project planning and control by network models.* Wiley, New York, 1977.

[22] E.A. Elsayed. Algorithms for project scheduling with resource constraints. *International Journal of Production Research*, 20(1):95–103, 1982.

[23] F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[24] F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2:4–32, 1989.

[25] S. Hartmann. A competitive genetic algorithm for resource–constrained project scheduling. Technical Report 451, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[26] S. Hartmann. Project scheduling with multiple modes: A genetic algorithm. Technical Report 435, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[27] S. Hartmann. Acceptance criteria in local search heuristics with an application to project scheduling. Technical report, forthcoming.

[28] S. Hartmann and R. Kolisch. An experimental investigation of state–of–the–art heuristics for the resource–constrained project scheduling problem. Technical report, forthcoming.

[29] W. Herroelen, E. Demeulemeester, and B. De Reyck. A classification scheme for project scheduling. In J. Weglarz, editor, *Handbook on recent advances in project scheduling*, pages 1–26. Kluwer, 1998.

[30] H.J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[31] J.E. Kelley. The critical–path method: Resources planning and scheduling. In J.F. Muth and G.L. Thompson, editors, *Industrial scheduling*, pages 347–365. Prentice–Hall, New Jersey, 1963.

[32] Y.-D. Kim. A backward approach in list scheduling algorithms for multi–machine tardiness problems. *Computers & Operations Research*, 22(3):307–319, 1995.

[33] S. Kirkpatrick, C. D. Gelatt jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[34] U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine–grained parallel genetic algorithms. *Annals of Operations Research*, forthcoming.

[35] R. Kolisch. *Project scheduling under resource constraints — Efficient heuristics for several problem classes.* Physica, Heidelberg, 1995.

[36] R. Kolisch. Efficient priority rules for the resource–constrained project scheduling problem. *Journal of Operations Management*, 14(3):179–192, 1996.

[37] R. Kolisch. Serial and parallel resource–constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

[38] R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43:23–40, 1996.

[39] R. Kolisch and R. Padman. An integrated survey of project scheduling. Technical Report 463, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[40] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark instances for project scheduling problems. In J. Weglarz, editor, *Handbook on recent advances in project schedulin*, pages ??–?? Kluwer, Amsterdam.

[41] R. Kolisch and A. Sprecher. PSPLIB — a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[42] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource–constrained project scheduling problems. *Management Science*, 41(10):1693–1703, 1995.

[43] S.R. Lawrence. Resource constrained project scheduling - A computational comparison of heuristic scheduling techniques. Technical report, Graduate School of Industrial Administration, Carnegie–Mellon University, Pittsburgh, 1985.

[44] J.-K. Lee and Y.-D. Kim. Search heuristics for resource constrained project scheduling. *Journal of the Operational Research Society*, 47:678–689, 1996.

[45] V.J. Leon and B. Ramamoorthy. Strength and adaptability of problem–space based neighborhoods for resource–constrained scheduling. *OR Spektrum*, 17(2/3):173–182, 1995.

[46] R.K.-Y. Li and J. Willis. An iterative scheduling technique for resource–constrained project scheduling. *European Journal of Operational Research*, 56:370–379, 1992.

[47] H.E. Mausser and S.R. Lawrence. Exploiting block structure to improve resource–constrained project schedules. Technical report, University of Colorado, Graduate School of Business Administration, 1995.

[48] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, forthcoming.

[49] K.S. Naphade, S.D. Wu, and R.H. Storer. Problem space search algorithms for resource–constrained project scheduling. *Annals of Operations Research*, 70:307–326, 1997.

[50] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[51] O. Oğuz and H. Bala. A comparative study of computational procedures for the resource constrained project scheduling problem. *European Journal of Operational Research*, 72:406–416, 1994.

[52] L. Özdamar and G. Ulusoy. A local constraint based analysis approach to project scheduling under general resource constraints. *European Journal of Operational Research*, 79:287–298, 1994.

[53] L. Özdamar and G. Ulusoy. An iterative local constraint based analysis for solving the resource constrained project scheduling problem. *Journal of Operations Management*, 14(3):193–208, 1996.

[54] L. Özdamar and G. Ulusoy. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. *European Journal of Operational Research*, 89:400–407, 1996.

[55] J.H. Patterson. Alternate methods of project scheduing with limited resources. *Naval Research Logistics Quarterly*, 20:767–784, 1973.

[56] J.H. Patterson. Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly*, 20:95–123, 1976.

[57] E. Pinson, C. Prins, and F. Rullier. Using tabu search for solving the resource–constrained project scheduling problem. In *Proceedings of the 4. International Workshop on Project Management and Scheduling*, pages 102–106, Leuven, 1994.

[58] B. Pollack–Johnson. Hybrid structures and improving forecasting and scheduling in project management. *Journal of Operations Management*, 12:101–117, 1995.

[59] A.A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multiproject scheduling with limited resources: A zero–one programming approach. *Management Science*, 16:93–107, 1969.

[60] S.E. Sampson and E.N. Weiss. Local search techniques for the generalized resource constrained project scheduling problem. *Naval Research Logistics*, 40:665–675, 1993.

[61] A. Schirmer and S. Riesenberg. Case–based reasoning and parameterized random sampling for project scheduling. Technical report, forthcoming.

[62] A. Schirmer and S. Riesenberg. Parameterized heuristics for project scheduling — Biased random sampling methods. Technical Report 456, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[63] J.M.J. Schutten. List scheduling revisited. *Operations Research Letters*, 18:167–170, 1996.

[64] L.R. Shaffer, J.B. Ritter, and W.L. Meyer. *The critical–path method.* McGraw Hill, New York, 1965.

[65] A. Sprecher. Solving the RCPSP efficiently at modest memory requirements. Technical Report 425, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1996.

[66] A. Sprecher, R. Kolisch, and A. Drexl. Semi–active, active, and non–delay schedules for the resource–constrained project scheduling problem. *European Journal of Operational Research*, 80:94–102, 1995.

[67] J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource–constrained scheduling using branch and bound. *AIIE Transactions*, 10:252–259, 1978.

[68] R.H. Storer, S.D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[69] A. Thesen. Heuristic scheduling of activities under resource and precedence restrictions. *Management Science*, 23(4):412–422, 1976.

[70] P.R. Thomas and S. Salhi. An investigation into the relationship of heuristic performance with network–resource characteristics. *Journal of the Operational Research Society*, 48(1):34–43, 1997.

[71] G. Ulusoy and L. Özdamar. Heuristic performance and network/resource characteristics in resource–constrained project scheduling. *Journal of the Operational Research Society*, 40(12):1145–1152, 1989.

[72] G. Ulusoy and L. Özdamar. A constraint–based perspective in resource constrained project scheduling. *International Journal of Production Research*, 32(3):693–705, 1994.

[73] V. Valls, M.A. Pérez, and M.S. Quintanilla. Heuristic performance in large resource–constrained projects. Technical Report 92–2, Departament D'Estadistica I Invecigacio Operativa, Universitat de Valencia, 1992.

[74] G.E. Whitehouse and J.R. Brown. GENRES: An extension of Brooks algorithm for project scheduling with resource constraints. *Computers & Industrial Engineering*, 3:261–268, 1979.