ELSEVIER

European Journal of Operational Research 127 (2000) 394–407

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem

Sönke Hartmann [a,☆], Rainer Kolisch [b,*]

[a] *Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, D-24098 Kiel, Germany*
[b] *Institut für Betriebswirtschaftslehre, Technische Universität Darmstadt, Hochschulstr. 1, D-64289 Darmstadt, Germany*

Abstract

We consider heuristic algorithms for the resource-constrained project scheduling problem. Starting with a literature survey, we summarize the basic components of heuristic approaches. We briefly describe so-called *X*-pass methods which are based on priority rules as well as metaheuristic algorithms. Subsequently, we present the results of our in-depth computational study. Here, we evaluate the performance of several state-of-the-art heuristics from the literature on the basis of a standard set of test instances and point out to the most promising procedures. Moreover, we analyze the behavior of the heuristics with respect to their components such as priority rules and metaheuristic strategy. Finally, we examine the impact of problem characteristics such as project size and resource scarceness on the performance.

Keywords: Resource-constrained project scheduling; Heuristics; Experimental evaluation

1. Introduction

The resource-constrained project scheduling problem (RCPSP) can be stated as follows. A single project consists of $n+1$ activities where each activity has to be processed in order to complete the project. The 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 have been finished. Second, performing the activities requires resources with limited capacities. Altogether we have a set of $\mathscr{K}$ resource types, the number of resource types is K. While being processed, activity j requires $r_{j,k}$ units of resource type $k \in \mathscr{K}$ during every time instant 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 non-negative and deterministic; for the project start and end activities we have $p_j = 0$ and $r_{j,k} = 0$ for

* Corresponding author.
☆ Present address: LOGAS Gesellschaft für Logistische Anwendungssysteme mbH, Gutenberg-Hans, Steckelhörn 5, 20457 Hamburg, Germany.
E-mail addresses: hartmann@logas.de (S. Hartmann), kolisch@bwl.tu-darmstadt.de (R. Kolisch).

0377-2217/00/$ - see front matter
PII: S0377-2217(99)00485-3

all $k \in \mathscr{K}$. 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.

Since its advent the RCPSP has been of interest for practitioners and researchers. Recent years have witnessed a tremendous increase in research for the RCPSP, both in terms of heuristic and optimal procedures. We refer to the surveys provided by Icmeli et al. [13], Özdamar and Ulusoy [29], Herroelen et al. [12], Kolisch and Padman [20] and Brucker et al. [6]. Recently, Kolisch and Hartmann [19] have given a specific overview of heuristics for the RCPSP. The paper focuses on the building blocks (schedule generation schemes, priority rules, schedule representations, operators, and search strategies) and the way these building blocks are combined to methods such as X-pass methods (single pass methods, multi-pass methods, sampling procedures) and different types of metaheuristics (simulated annealing, genetic algorithms, and tabu search). This paper is a follow-up study which provides an in-depth investigation of the performance of recent RCPSP heuristics as well as explanations for the observed results. Based on these observations, we subsequently give recommendations about prosperous directions for further research efforts.

2. Proposed heuristics

This section gives a short survey of the tested heuristics. We start with the description of schedule generation schemes which are (with the exception of the method of Baar et al. [3]) a core building block of all procedures.

2.1. Schedule generation schemes

Schedule generation schemes (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 which can be distinguished w.r.t. activity- and time-incrementation. The so-called *serial* SGS performs *activity-incrementation* while the so-called *parallel* SGS performs *time-incrementation*. We give a brief non-technical description of both SGS. For details, cf. [19].

Serial schedule generation scheme and list scheduling. The serial SGS consists of $g = 1, \ldots, n$ stages, in each of which one non-dummy activity is selected from the eligible set and scheduled at the earliest precedence- and resource-feasible completion time. The eligible set comprises all unscheduled activities which are eligible for scheduling because all predecessor activities have already been scheduled. The time complexity of the serial SGS is $O(n^2 \cdot K)$ (cf. [31]). The serial SGS generates active (and thus feasible) schedules which are for the resource-unconstrained scheduling problem optimal (cf. [17]). Active schedules have the property that none of the activities can be started earlier without delaying some other activity (cf. [35]). For scheduling problems with a regular performance measure (for a definition of the latter, cf. [35]) such as makespan minimization, the optimal solution will always be in the set of active schedules. Hence, the serial SGS does not exclude optimal schedules a priori.

A variant of the serial SGS is list scheduling. Here, the activities of the project are first ordered within a list $\lambda = \langle j_1, j_2, \ldots, j_n]$ where j_g denotes the activity at list position g. This list has to be precedence feasible, i.e., each activity has all its network predecessors as list predecessors (cf. [11]). Given λ, the activities can be scheduled in the order of the list at the earliest precedence- and resource-feasible start time. As a special case of the serial SGS, list scheduling has the same properties as the serial SGS. That is, it generates active schedules and hence there is always a list λ^* for which list scheduling will generate an optimal schedule when a regular measure of performance is considered.

Parallel schedule generation scheme. The parallel SGS does time incrementation. For each iteration g there is a schedule time t_g and a set of eligible activities. An activity is eligible if it can be precedence- and resource-feasibly started at the schedule time. Activities are chosen from the eligible set and started at the schedule time until there are no more eligible activities left. After-

wards, the scheduling scheme steps to the next schedule time which is given by the earliest finish time of the activities in process. The time complexity of the parallel SGS is $O(n^2 \cdot K)$ (cf. [19]). The parallel SGS generates (precedence- and resource-feasible) non-delay schedules which are optimal for the resource-unconstrained case (cf. [17]). 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 (cf. [35]). The set of non-delay schedules is a 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 for a regular performance measure. For example, in [17] it is shown that in a well-known instance set 40.3% of the instances have optimal solutions which are not in the set of non-delay schedules. Hence, the parallel SGS might exclude all optimal solutions a priori.

2.2. X-pass methods

X-pass methods, also known as priority rule based heuristics, employ one or both of the SGS in order to construct one or more schedules. Depending on how many schedules are generated, we distinguish single pass methods ($X = 1$) and multi-pass methods ($X > 1$). Each time a schedule is generated, X-pass methods start from scratch without considering any knowledge from previously generated solutions. In order to select at each stage of the generation procedure one activity to be scheduled, a priority rule is employed. The latter is defined by a mapping which assigns each activity j in the eligible set a value $v(j)$ and an objective stating whether an activity with a large or a small $v(j)$-value is desired.

Single pass methods. Single pass heuristics select in each iteration the activity which maximizes or minimizes the $v(j)$-value. In case of ties, one or several tie-breaking rules have to be employed. A simple way to resolve ties is to choose the activity with the smallest activity label. There has been an overwhelming amount of research on priority rules for the RCPSP; an extensive survey is given in [19]. For our computational analysis we have selected the two priority rules which have shown favourable results in the experimental studies of Alvarez-Valdés and Tamarit [2], Davis and Patterson [10] and Kolisch [15,17]: LFT (latest finish time) and WCS (worst case slack). LFT is a well-known priority rule. WCS has been introduced by Kolisch [16] for the parallel scheduling scheme only. The rule calculates for an activity j the slack which remains in the worst case if j is not selected in the current iteration.

Multi-pass methods. There are many possibilities to combine SGS and priority rules to a multi-pass method. The most common ones are according to Kolisch and Hartmann [19] multi-priority rule methods (cf. [4,37,39],), forward–backward scheduling methods (cf. [26,30]), and sampling methods (cf. [1,9,17]).

Multi-priority rule methods employ the SGS several times. Each time a different priority rule is used. Forward–backward scheduling methods employ an SGS in order to iteratively schedule the project by alternating between forward and backward scheduling. For reasons to be given in Section 3.1, both approaches are not considered in this study.

Sampling methods generally make use of one SGS and one priority rule. Different schedules are obtained by biasing the selection of the priority rule through a random device. In addition to a priority value $v(j)$, a selection probability $p(j)$ is computed. Depending on how the probabilities are computed, it is distinguished between random sampling, biased random sampling, and regret based biased random sampling (cf. [15]). Random sampling (RS) assigns each activity in the eligible set the same probability. Biased random sampling (BRS) (cf. [1,9]) employs the priority values directly in order to obtain the selection probabilities; i.e., if the objective of the priority rule is to select the activity with the largest priority value, then the probability is calculated by dividing the priority value of the activity under consideration by the sum of the priority values of all eligible activities. Regret based biased random sampling (RBRS) uses the priority values indirectly via regret values; i.e., if, again, the objective is to select the activity with the largest 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 eligible set. Before calculating the selection probabilities based on the regret values, the latter are modified by adding $\epsilon > 0$. This assures that each activity in the eligible set has a selection probability greater than 0 and thus every schedule of the population can be generated. Schirmer and Riesenberg [34] propose a variant of RBRS where ϵ is determined dynamically.

So-called adaptive RBRS have been proposed by Kolisch and Drexl [18] as well as Schirmer [33]. The essence of adaptive sampling is to select the SGS, the priority rule, and the way the selection probabilities are calculated based on characteristics of the problem instance at hand. We refer to these characteristics, e.g., the number of activities, the resource strength, and the resource factor, henceforth as problem parameters (cf. Section 3.1). The method of Kolisch and Drexl [18] 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. Schirmer [33] has extended this approach by employing both schedule generation schemes together with four different priority rules and two different sampling schemes (RBRS and a variant of RBRS).

2.3. Metaheuristic approaches

Many metaheuristic strategies such as genetic algorithms, simulated annealing, and tabu search are applied to solve hard combinatorial optimization problems. In this section, we briefly describe those metaheuristic approaches for the RCPSP that are used in our computational study. For each heuristic, we mention the underlying representation, the method to produce an initial solution, and the operator to generate new solutions from existing ones. For a more detailed introduction into the representations and operators, we refer to [19]. This reference also includes a description of those metaheuristic algorithms for the RCPSP that could not be considered for the following experimental analysis.

Baar et al. [3] develop two tabu search (TS) algorithms. For our analysis, the approach based on the so-called schedule scheme representation has been tested. A schedule scheme consists of four disjoint relations. These relations specify whether two activities are precedence related (conjunctions), may not be processed in parallel (disjunctions), must be processed in parallel (parallelity relations), or are not subject to any restrictions (flexibility relations). A schedule scheme represents those (not necessarily feasible) schedules in which the relations are maintained. In order to transform the current schedule scheme into a schedule, Baar et al. [3] employ a decoding procedure that constructs a feasible schedule in which all conjunctions and all disjunctions and a 'large' number of parallelity relations of the schedule scheme are satisfied. The neighborhood definition is made up by 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. The tabu search procedure uses a dynamic tabu list as well as priority rule based start heuristics.

Bouleimen and Lecocq [5] propose a simulated annealing (SA) procedure. A solution is represented by an activity list which is assumed to be precedence-feasible, i.e., each activity must appear later in the list than all of its predecessors (cf. Section 2.1). As a decoding procedure, the serial SGS is used. From the current activity list, a neighbor activity list (and thus a neighbor solution) is obtained by applying the so-called shift operator. This operator selects some activity from the list and inserts it at some other position, such that the resulting activity list is again precedence feasible. The initial solution is constructed using a priority rule based heuristic.

Hartmann [11] suggests a genetic algorithm (GA) based on the activity list representation described above and compares it to genetic algorithms which make use of the random key and priority rule representations, respectively. For all three representations, the serial SGS is used as decoding procedure. The random key (or priority

value) representation encodes a solution as a vector of n (real) numbers where the jth number relates to activity j. The random key array is transformed into a schedule by successively scheduling the activity with the highest random key among the eligible activities. Thus the random keys play the role of priority values. In the priority rule representation, a solution is given by a list of priority rules. A schedule is obtained by successively selecting an activity from the set of the eligible activities. Thereby, the first activity to be scheduled is selected using the first priority rule in the list and so on. All three approaches employ 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 is used to determine the initial generation.

Leon and Ramamoorthy [25] introduce two local search approaches and a genetic algorithm. As the latter outperforms the former two heuristics, only the genetic algorithm is used in our computational investigation. It employs the so-called problem-space based representation which is similar to the random key representation described above. As decoding procedure, an extended version of the parallel SGS is used which allows an activity to start later than at the schedule time t_g (which is the earliest possible start time in the current partial schedule). This way, the search space is not restricted to the set of the non-delay schedules. The standard one-point crossover is used to determine the next generation in the artificial evolution. The initial random keys are computed by a priority rule.

Further metaheuristic procedures which are not included in our study have been proposed by Cho and Kim [7], Kohlmorgen et al. [14], Lee and Kim [24], Naphade et al. [27], Pinson et al. [31], Sampson and Weiss [32] and Thomas and Salhi [38].

3. Computational results

3.1. Test design

As test instances we have employed the standard sets j30, j60, and j120 for the RCPSP presented in [21]. The sets j30 and j60 consist of 480 projects with four resource types as well as $n = 30$ and $n = 60$ activities, respectively. The levels of the three independent problem parameters network complexity, resource factor, and resource strength are systematically altered to define a full factorial experimental design.

The network complexity (NC) defines the average number of non-redundant precedence relations per activity. The resource factor (RF) gives the average percent of different resource types for which a non-dummy activity has a non-zero resource demand. Finally, the resource strength (RS) defines how scarce resource are. An RS-value of 0 defines for a specific resource k its capacity as the maximum demand for resource k by any of the activities. An RS-value of 1 defines the capacity of resource k to be equal to the maximum demand imposed by the project when performed according to the earliest start time schedule. A formal definition of the three parameters can be found in [22] as well as [23].

For the sets j30 and j60 with 480 instances each, the levels of the parameters are set as follows: $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$, and $RS \in \{0.2, 0.5, 0.7, 1.0\}$. The set j120 consists of 600 projects, each with $n = 120$ activities and four resource types. Again, a full factorial design with the three independent problem parameters NC, RF, and RS is used. The levels of the parameters are $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$, and $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

The following heuristics are included in our experimental investigation (cf. Sections 2.2 and 2.3): Within (deterministic) single pass heuristics we employed WCS with the parallel SGS and LFT with the parallel and the serial SGS (cf. [16,17]). These three methods are also employed as RBRS variants (cf. [17]). Additionally, pure random sampling with the parallel and the serial SGS is used (cf. [15]). Adaptive sampling approaches included are due to Kolisch and Drexl [18] as well as Schirmer [33]. Metaheuristics tested in this study include the schedule scheme based TS method of Baar et al. [3] as well as the activity list based SA procedure of Bouleimen and Lecocq [5]. Furthermore, three GA methods based on the activity list, random key, and priority rule representations of

Hartmann [11] as well as the problem space GA of Leon and Ramamoorthy [25] are considered. This gives altogether 16 heuristics which have been tested.

Each algorithm was tested by its author(s) using the original implementation. This allowed the authors to adjust their methods 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, within all tested non-single pass heuristics, we limited 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 2.1. Hence, we found this assumption justified. At the end of Section 3.2 we will give some observations regarding the computation time.

As mentioned above, we did neither include multi-priority rule methods nor forward–backward scheduling methods. This is due to the fact that these two types of heuristics cannot be easily adapted to construct any specified number of schedules for an instance (in our case, 1, 1000, and 5000). Generally, the number of schedules produced by multi-priority rule methods is determined by the number of priority rules employed. Forward–backward scheduling methods terminate when in two consecutive forward and backward passes no further changes occur and hence no better schedule will be obtained.

3.2. Influence of the project size

Tables 1–5 display the results of the computational comparison obtained from the evaluation of 1000 and 5000 schedules, respectively. In each table, the heuristics are sorted according to descending performance with respect to 5000 iterations. Table 1 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. [36]). 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.

Table 1
Average deviations from optimal solution – $n = 30$

Algorithm	SGS	Reference	Iterations		
			1	1000	5000
SA – activity list	ser.	[5]	–	0.38	0.23
GA – activity list	ser.	[11]	–	0.54	0.25*
Sampling – adaptive	ser./par.	[33]	–	0.65	0.44
TS – schedule scheme	spec.	[3]	–	0.86	0.44*
Sampling – adaptive	ser./par.	[18]	–	0.74	0.52
Single pass/sampling – LFT	ser.	[17]	5.58	0.83	0.53
GA – random key	ser.	[11]	–	1.03	0.56*
Sampling – random	ser.	[15]	–	1.44	1.00
GA – priority rule	ser.	[11]	–	1.38	1.12
Single pass/sampling – WCS	par.	[16,17]	3.88	1.40	1.28
Single pass/sampling – LFT	par.	[17]	4.39	1.40	1.29*
Sampling – random	par.	[15]	–	1.77	1.48*
GA – problem space	ext. par.	[25]	–	2.08	1.59

Table 2
Average deviations from best solution – $n = 60$

Algorithm	SGS	Reference	Iterations		
			1	1000	5000
GA – activity list	ser.	[11]	–	0.88	0.33
SA – activity list	ser.	[5]	–	1.05	0.37*
Sampling – adaptive	ser./par.	[33]	–	1.09	0.82
GA – priority rule	ser.	[11]	–	1.32	0.94*
Sampling – adaptive	ser./par.	[18]	–	1.48	1.14*
GA – random key	ser.	[11]	–	2.36	1.34
TS – schedule scheme	spec.	[3]	–	1.68	1.42
Single pass/sampling – LFT	ser.	[17]	4.88	1.76	1.43*
single pass/sampling – WCS	par.	[16,17]	4.30	1.76	1.44
Single pass/sampling – LFT	par.	[17]	4.78	1.71	1.45*
GA – problem space	ext. par.	[25]	–	2.36	1.71*
Sampling – random	par.	[15]	–	2.71	2.28*
Sampling – random	ser.	[15]	–	3.34	2.73

Table 3
Average deviations from best solution – $n = 120$

Algorithm	SGS	Reference	Iterations		
			1	1000	5000
GA – activity list	ser.	[11]	–	2.36	0.67*
SA – activity list	ser.	[5]	–	5.50	1.64*
GA – priority rule	ser.	[11]	–	2.58	1.66
Sampling – adaptive	ser./par.	[33]	–	2.87	2.04
Single pass/sampling – LFT	par.	[17]	5.76	2.69	2.10
Single pass/sampling – WCS	par.	[16,17]	5.54	2.71	2.11*
Sampling – adaptive	ser./par.	[18]	–	3.71	3.10*
GA – problem space	ext. par.	[25]	–	5.09	3.53*
Single pass/sampling – LFT	ser.	[17]	8.02	4.55	3.87*
GA – random key	ser.	[11]	–	6.85	4.31*
Sampling – random	par.	[15]	–	6.21	5.22*
Sampling – random	ser.	[15]	–	9.39	8.19

For the j60 set, the percentage deviations from the upper and lower bounds are reported in Tables 2 and 4, respectively. [2] Finally, Tables 3 and 5 provide the respective deviations for the j120 set.

In order to detect significant differences between the heuristic performance for 5000 iterations we did for each heuristic and the next best one a pairwise comparison with the Wilcoxon signed-rank test using SPSS [28]. A significant difference in performance at the 5% level of confidence is marked with a star (*) in the last column of Tables 1–3.

Best heuristics. The heuristics that performed best in our study are the SA of Bouleimen and Lecocq [5] and the activity list based GA of Hartmann [11]. 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. Only on the j120 set, however, the difference between both procedures is significant. On the other hand, these two procedures significantly outperform all other heuristics on each instance set.

[2] Note that the schedule scheme based TS heuristic of Baar et al. [3] was additionally run using the original termination criterion, which allows two trials each of which was terminated after no improved solution was found after 250 iterations. This way, the deviation from the upper bound was lowered to 1.14%.

Table 4
Average deviations from critical path lower bound – $n = 60$

Algorithm	SGS	Reference	Iterations		
			1	1000	5000
GA – activity list	ser.	[11]	–	12.68	11.89
SA – activity list	ser.	[5]	–	12.75	11.90
Sampling – adaptive	ser./par.	[33]	–	12.94	12.59
GA – priority rule	ser.	[11]	–	13.30	12.74
Sampling – adaptive	ser./par.	[18]	–	13.51	13.06
GA – random key	ser.	[11]	–	14.68	13.32
TS – schedule scheme	spec.	[3]	–	13.80	13.48
Single pass/sampling – LFT	ser.	[17]	18.13	13.96	13.53
Single pass/sampling – WCS	par.	[16,17]	16.87	13.66	13.21
Single pass/sampling – LFT	par.	[17]	17.46	13.59	13.23
GA – problem space	ext. par.	[25]	–	14.33	13.49
Sampling – random	par.	[15]	–	14.89	14.30
Sampling – random	ser.	[15]	–	15.94	15.17

Table 5
Average deviations from critical path lower bound – $n = 120$

Algorithm	SGS	Reference	Iterations		
			1	1000	5000
GA – activity list	ser.	[11]	–	39.37	36.74
SA – activity list	ser.	[5]	–	42.81	37.68
GA – priority rule	ser.	[11]	–	39.93	38.49
Sampling – adaptive	ser./par.	[33]	–	39.85	38.70
Single pass/sampling – LFT	par.	[17]	43.86	39.60	38.75
Single pass/sampling – WCS	par.	[16,17]	43.57	39.65	38.77
Sampling – adaptive	ser./par.	[18]	–	41.37	40.45
GA – problem space	ext. par.	[25]	–	42.91	40.69
Single pass/sampling – LFT	ser.	[17]	48.11	42.84	41.84
GA – random key	ser.	[11]	–	45.82	42.25
Sampling – random	par.	[15]	–	44.46	43.05
Sampling – random	ser.	[15]	–	49.25	47.61

Comparison of metaheuristics and X-pass methods. Generally, the results show that the best metaheuristics outperform the best sampling approaches. Increasing the number of generated schedules from 1000 to 5000 results in a stronger superiority of the metaheuristic results. 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).

Random sampling as benchmark. We have included two simple sampling procedures in which the eligible activity to be scheduled next is selected by pure random choice. Here, the only project scheduling knowledge is contained in the SGS (due to this knowledge, the results are not as bad as one may expect). These random procedures serve as benchmarks as they allow us to evaluate how much the results can be improved by incorporating more project scheduling knowledge. Employing random sampling methods as benchmark solutions is common for the evaluation of scheduling heuristics (cf., e.g., [8]). Generally, any procedure should perform considerably better than a pure random procedure, especially if the same SGS is used. The results for the j30 set show, however,

that two of the genetic algorithms yield a higher average deviation from the optimal makespan than the respective random sampling procedure with the same SGS. For the two other instance sets the two random sampling procedures perform worst among all heuristics if 5000 iterations are considered.

Performance of metaheuristics. 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 strategies 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 three GAs of Hartmann [11] and the GA of Leon and Ramamoorthy [25]. While the activity list based GA yields the best results for all project sizes, the outcome of the random key based GA is of moderate quality on the j30 set and much worse on the j120 set. The performance of the priority rule based GA, on the other hand, improves with increasing project size. This is because combining good priority rules is a good strategy in huge search spaces. But, as shown by Hartmann [11], it might exclude all optimal solutions from the search space which is a severe disadvantage for small project sizes. Note, however, that Hartmann [11] reports that the priority rule based GA hardly exploits the benefits of genetic inheritance.

Performance of deterministic single pass heuristics. For all three instance sets we see that the WCS priority rule with the parallel SGS performs best followed by the LFT priority rule with the parallel SGS and the LFT priority rule with the serial SGS. This is in line with the results documented in [16].

Performance of sampling methods. Analyzing the priority rule based sampling procedures, we observe that the two rules WCS and LFT give almost identical results when employed within the parallel SGS. On none of the three instance sets, a significant difference could be found. On the other hand, there is a strong influence of the SGS, as will be explained below. It is evident that sampling is capable of improving the results of the associated deterministic single pass approaches.

Performance of adaptive sampling methods. Considering the adaptive sampling strategies, we see that the recent approach of Schirmer [33] outperforms the one of Kolisch and Drexl [18]. This is due to the fact that the former procedure is based on a more accurate partitioning of the solution space in terms of problem parameters. The approach of Schirmer [33] makes use of the problem parameters number of activities (n), resource factor (RF), and resource strength (RS) while the heuristic of Kolisch and Drexl [18] only employs the resource factor. Furthermore, the adaptive method of Kolisch and Drexl [18] was not tested on instances with more than 30 activities during the design phase. This also explains why even some of the simple (non-adaptive) sampling procedures outperform the approach of Kolisch and Drexl [18] on the j120 set.

Impact of SGS. Both the LFT based and the pure random sampling approach were tested with the serial as well as the parallel SGS. We observe a strong influence of the SGS which depends on the problem size: While the serial SGS is the better choice for these sampling approaches on the j30 instance set, the parallel one takes the lead on the j120 set. This result can be explained as follows: Using the parallel SGS implies that only a subset of the active schedules, namely the non-delay schedules, are searched. This is a promising strategy if the search space is huge, as it is the case for the j120 set. On the j30 set, however, the much smaller search space makes it possible to find good (or even optimal) solutions within small time limits. There, the restriction to the non-delay schedules is disadvantageous as one may exclude all optimal solutions from the search space, hence the serial one becomes the SGS of choice. Similar findings are reported by Kolisch [15].

Computation times. The metaheuristic algorithms which make use of the activity list representation are the fastest approaches. This is due to the fact that the underlying serial SGS for activity lists (cf. Section 2.1) neither computes the eligible set nor selects an activity on the basis of priority values. This, however, has to be done by all other approaches. Additional effort may arise for the priority rule based methods, i.e., X-pass methods as well as metaheuristics with priority rule repre-

sentation. There, in each stage and for each eligible activity the priority value has to be calculated if a dynamic priority rule (cf. [19]) is employed. Biased random sampling approaches spend additional computational effort with the priority value based computation of selection probabilities and the random selection. Consequently, the heuristics show different computation times when constructing the same number of schedules.

In order to give an impression of the computation times that occurred in this study, we mention some computation times that we obtained on a Pentium based computer with 133 MHz and Linux operating system. There, activity list based GA needs on the average 0.54 seconds when computing 1000 schedules for the j30 set. The LFT based sampling method with serial SGS needs 0.78 seconds while the priority rule based GA requires 0.91 seconds. Considering the j120 set, the time needed by the activity list based GA to compute 1000 schedules increases to 3.04 seconds. Note that there is no linear relationship between the number of activities and the computation times (cf. the complexity statements given in Section 2.1). For each instance set and each heuristic, computing 5000 schedules takes approximately five times longer than computing 1000. The latter observation is reasoned by the fact that schedule construction requires far more computational effort than further components of the heuristics such as, e.g., the selection operator in a GA.

3.3. Influence of further problem parameters

Multi-variate linear regression analysis. In order to assess the influence of the problem parameters resource strength, resource factor, and network complexity we have performed for each tested heuristic a multi-variate linear regression analysis. We employed the average deviation from the optimal solution given 5000 iterations for the benchmark set j30 as dependent variable and the three problem parameters *RS*, *RF*, and *NC* as independent variables. Table 6 provides for each method the resulting R^2-value, the constant (Const), and the coefficient for each problem parameter. A value of '0' indicates that the coefficient is not significant at the 5% level of confidence, while a star (*) indicates that the coefficient is significant at the 1% level of confidence. The resulting R^2-values range between 0.14 and 0.38. This indicates that the three problem parameters do not explain much of the variance, and hence that the parameters do not allow to predict the performance of a heuristic on a specific project instance. Nevertheless, some interesting observations can be made.

All Const-values are significant at the 1% level of confidence. A low Const-value goes along with a good average heuristic performance while a high Const-value indicates, on average, a poor performance. The two best Const-values are obtained for the activity list based approaches of Bouleimen and Lecocq [5] and Hartmann [11] with the latter having a slightly better Const-value. The worst

Table 6
Results of the multivariate regression – $n = 30$

Algorithm	SGS	Reference	R^2	Const	*NC*	*RS*	*RF*
SA – activity list	ser.	[5]	0.14	0.74*	0	−0.99*	0.31*
GA – activity list	ser.	[11]	0.16	0.67*	0	−0.87*	0.64*
TS – schedule scheme	spec.	[3]	0.17	1.44*	−0.56*	−1.02*	1.01*
Sampling – adaptive	ser./par.	[18]	0.28	1.26*	0	−1.83*	1.51*
Sampling – LFT	ser.	[17]	0.30	1.71*	−0.54*	−1.98*	1.58*
GA – random key	ser.	[11]	0.30	1.49*	−0.53*	−1.72*	1.70*
Sampling – random	ser.	[15]	0.38	3.01*	−1.03*	−3.29*	2.92*
GA – priority rule	ser.	[11]	0.28	3.93*	−0.72	−4.02*	1.46*
Sampling – WCS	par.	[16,17]	0.16	4.31*	0	−2.96*	−0.71
Sampling – LFT	par.	[17]	0.16	4.22*	0	−2.98*	−0.65
Sampling – random	par.	[15]	0.20	5.17*	−0.86	−3.68*	0
GA – problem space	ext. par.	[25]	0.21	5.09*	−0.75	−3.75*	0

Const-value is due to random sampling with the parallel SGS.

All significant *NC* coefficients are negative, indicating that with increasing *NC*-level the average performance of the heuristics increases. This confirms the findings of Kolisch et al. [23]. The explanation is that more precedence relations between activities lower the number of possible activity sequences and thus the size of the solution space. The influence of the network complexity is rather low. The highest coefficient is obtained for random sampling with serial SGS. This is because the narrower the shape of the project network becomes, the smaller is the set of eligible activities and thus the risk of (randomly) selecting an activity which leads to a poor schedule.

The *RS* coefficients are all highly significant and range between −0.87 (activity list based GA of Hartmann [11]) and −4.02 (priority rule based GA of Hartmann [11]). The negative sign of the coefficients is reasoned by the fact that higher *RS*-levels assign more resource capacity to the problems and hence make them easier. The coefficient indicates how sensitive the performance of the heuristics is in terms of the resource scarceness of the problem. Again, the most insensitive heuristics are the activity list approaches of Hartmann [11] and Bouleimen and Lecocq [5].

The analysis of the *RF* coefficients has to be separated. All coefficients for the serial based SGS approaches are positive and highly significant, while the ones for the parallel SGS based approaches are either non-significant or negative at the 5% level of confidence. That is, the performance of serial SGS based heuristics is negatively affected by an increasing density of the resource demand array while the performance of parallel SGS based heuristics are either slightly positive or not affected. This observation has been originally made for *X*-pass heuristics in [15]. An explanation is that for high resource density arrays the serial SGS is not capable of building 'compact schedules' where activities are put close to one another. Contrary, the parallel SGS, generating non-delay schedules, does inherently build 'compact schedules'.

We were not able to perform an analogous analysis for the j60 and j120 sets because for these sets there are no optimal solutions available. When employing lower or upper bounds instead, the results are biased. Using as the dependent variable the difference between the upper bound found by the heuristic under consideration and a lower bound, the regression coefficients explain the deviation of the lower bound from the unknown optimal solution, rather than the deviation of the upper bound from the unknown optimal solution. This is because the lower bounds currently available are not tight (especially for the j120 set). When defining the dependent variable as the difference between the best upper bound found by all heuristics and the upper bound of the heuristic under consideration, the regression coefficients reflect not only the performance of the heuristic under consideration but implicitly also the per-

Table 7
Average deviations from optimal solution w.r.t. resource strength – $n = 30$

Algorithm	SGS	Reference	*RS*			
			0.25	0.50	0.75	1.00
SA – activity list	ser.	[5]	0.81	0.09	0.00	0.00
GA – activity list	ser.	[11]	0.68	0.24	0.04	0.00
TS – schedule scheme	spec.	[3]	0.78	0.66	0.31	0.00
Sampling – adaptive	ser./par.	[18]	1.51	0.37	1.19	0.00
Sampling – LFT	ser.	[17]	1.60	0.41	0.10	0.00
GA – random key	ser.	[11]	1.31	0.76	0.15	0.00
Sampling – random	ser.	[15]	2.58	1.15	0.26	0.00
GA – priority rule	ser.	[11]	3.18	1.11	0.18	0.00
Sampling – WCS	par.	[16,17]	2.52	1.28	1.29	0.00
Sampling – LFT	par.	[17]	2.53	1.31	1.29	0.00
Sampling – random	par.	[15]	2.98	1.62	1.31	0.00
GA – problem space	ext. par.	[25]	3.08	1.85	1.42	0.00

Table 8
Average deviations from optimal solution w.r.t. resource factor – $n = 30$

Algorithm	SGS	Reference	*RF*			
			0.25	0.50	0.75	1.00
SA – activity list	ser.	[5]	0.12	0.09	0.40	0.28
GA – activity list	ser.	[11]	0.02	0.11	0.37	0.47
TS – schedule scheme	spec.	[3]	0.03	0.30	0.67	0.75
Sampling – adaptive	ser./par.	[18]	0.00	0.18	0.83	1.04
Sampling – LFT	ser.	[17]	0.00	0.18	0.83	1.10
GA – random key	ser.	[11]	0.01	0.13	0.89	1.18
Sampling – random	ser.	[15]	0.01	0.38	1.54	2.06
GA – priority rule	ser.	[11]	0.26	1.33	1.45	1.43
Sampling – WCS	par.	[16,17]	1.44	1.60	1.00	1.04
Sampling – LFT	par.	[17]	1.44	1.59	1.01	1.09
Sampling – random	par.	[15]	1.44	1.63	1.21	1.63
GA – problem space	ext. par.	[25]	1.44	1.79	1.40	1.71

formance of the other heuristics. This leads to regression coefficients which cannot be interpreted as those for the j30 set.

Means for resource parameter levels. In the remainder of this computational analysis, we focus on the separate effects of the *RS* and the *RF*. Tables 7 and 8 display the average deviations from the optimal makespan after 5000 iterations for each parameter setting of the j30 set, respectively. For all procedures, the average deviation decreases monotonically with increasing resource strength, i.e., with declining resource availability, confirming the results obtained from the multi-variate regression. For $RS = 1$, the problems are resource-unconstrained, allowing each heuristic to find an optimal solution. Observe also that the activity list based GA of Hartmann [11] yields better results than the SA procedure of Bouleimen and Lecocq [5] on the (hard) instances with a low *RS* value while the latter performs better on the moderately resource-constrained problems. Analyzing the performance of the heuristics w.r.t. the resource factor, we can see that for a low *RF* value, procedures based on the serial SGS perform much better than those based on the parallel one. Again, this is in line with the multi-variate regression results.

4. Summary and guidance for future research

Based on our findings we can give the following recommendations towards the development of further improvements for project scheduling heuristics.

The most successful approaches in our numerical evaluation are metaheuristics, namely the simulated annealing procedure of Bouleimen and Lecocq [5] and the genetic algorithm of Hartmann [11]. Nevertheless, priority rule based sampling methods are indispensable to construct initial solutions for metaheuristics. Hence, research in both directions – sampling and metaheuristics – will remain a promising field of research in the future.

Among the metaheuristics, the procedures which make use of the activity list representation performed best. The underlying metaheuristic strategy does not seem to be as influential as the representation; a genetic algorithm and a simulated annealing approach share the top ranks in our analysis. This indicates that the development of new representations may be more interesting than the incorporation of existing representations into other metaheuristic strategies.

As already mentioned, the best heuristics in our computational analysis are metaheuristics based on the activity list representation and the serial SGS. The experiences with sampling methods show that the parallel SGS outperforms the serial one on the larger problems. Thus, the question arises if it would pay to develop a variant of the parallel SGS for activity lists. Possibly, this might lead to improved metaheuristics for very large problems.

Adaptive sampling methods may outperform simple sampling approaches. The key to success,

however, is an appropriate partitioning of the solution space in terms of problem parameters, such as number of activities, resource factor, and resource strength. Equally important is the consideration of all possibly relevant parameter settings when designing an adaptive method. One should keep in mind, however, that the proposed adaptive sampling procedures, once adapted to the test instances by their designer, are fixed and incapable of learning from previously evaluated schedules. This makes them inferior to metaheuristics which have the inherent property of learning, i.e., self-adaptation.

Acknowledgements

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 (University of Kiel) for their help in this research. Furthermore, we would like to thank Andreas Drexl (University of Kiel) for his support.

References

[1] R. Alvarez-Valdés, J.M. Tamarit, Algoritmos heurísticos deterministas y aleatorios en secuenciacón de proyectos con recursos limitados, Qüestiió 13 (1989) 173–191.
[2] R. Alvarez-Valdés, J.M. Tamarit, Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, in: R. Słowiński, J. Węglarz (Eds.), Advances in Project Scheduling, Elsevier, Amsterdam, 1989, pp. 113–134.
[3] T. Baar, P. Brucker, S. Knust, Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem, in: S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1998, pp. 1–8.
[4] F.F. Boctor, Some efficient multi-heuristic procedures for resource-constrained project scheduling, European Journal of Operational Research 49 (1990) 3–13.
[5] K. Bouleimen, 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.
[6] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, European Journal of Operational Research 112 (1) (1999) 3–41.
[7] J.-H. Cho, Y.-D. Kim, A simulated annealing algorithm for resource constrained project scheduling problems, Journal of the Operational Research Society 48 (1997) 735–744.
[8] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of Scheduling, Addison-Wesley, Reading, MA, 1967.
[9] D.F. Cooper, Heuristics for scheduling resource-constrained projects: An experimental investigation, Management Science 22 (11) (1976) 1186–1194.
[10] E.W. Davis, J.H. Patterson, A comparison of heuristic and optimum solutions in resource-constrained project scheduling, Management Science 21 (1975) 944–955.
[11] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Naval Research Logistics 45 (1998) 733–750.
[12] W. Herroelen, E. Demeulemeester, B. De Reyck, Resource-constrained project scheduling – A survey of recent developments, Computers & Operations Research 25 (4) (1998) 279–302.
[13] O. Icmeli, S.S. Erenguc, C.J. Zappe, Project scheduling problems: A survey, International Journal of Operations & Production Management 13 (11) (1993) 80–91.
[14] U. Kohlmorgen, H. Schmeck, K. Haase, Experiences with fine-grained parallel genetic algorithms, Annals of Operations Research 90 (1999) 203–219.
[15] R. Kolisch, Project Scheduling under Resource Constraints – Efficient Heuristics for Several Problem Classes, Physica, Heidelberg, 1995.
[16] R. Kolisch, Efficient priority rules for the resource-constrained project scheduling problem, Journal of Operations Management 14 (3) (1996) 179–192.
[17] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research 90 (1996) 320–333.
[18] R. Kolisch, A. Drexl, Adaptive search for solving hard project scheduling problems, Naval Research Logistics 43 (1996) 23–40.
[19] R. Kolisch, S. Hartmann, Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis, in: J. Węglarz (Ed.), Project Scheduling – Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, 1999, pp. 147–178.
[20] R. Kolisch, R. Padman, An integrated survey of deterministic project scheduling, Technical Report 463, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.
[21] R. Kolisch, C. Schwindt, A. Sprecher, Benchmark instances for project scheduling problems, in: J. Węglarz (Ed.), Project Scheduling – Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, 1999, pp. 197–212.

[22] R. Kolisch, A. Sprecher, PSPLIB – a project scheduling problem library, European Journal of Operational Research 96 (1996) 205–216.
[23] R. Kolisch, A. Sprecher, A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, Management Science 41 (10) (1995) 1693–1703.
[24] J.-K. Lee, Y.-D. Kim, Search heuristics for resource constrained project scheduling, Journal of the Operational Research Society 47 (1996) 678–689.
[25] V.J. Leon, B. Ramamoorthy, Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling, OR Spektrum 17 (2/3) (1995) 173–182.
[26] R.K.-Y. Li, J. Willis, An iterative scheduling technique for resource-constrained project scheduling, European Journal of Operational Research 56 (1992) 370–379.
[27] K.S. Naphade, S.D. Wu, R.H. Storer, Problem space search algorithms for resource-constrained project scheduling, Annals of Operations Research 70 (1997) 307–326.
[28] M.J. Norušis, The SPSS Guide to Data Analysis, SPSS Inc, Chicago, 1990.
[29] L. Özdamar, G. Ulusoy, A survey on the resource-constrained project scheduling problem, IIE Transactions 27 (1995) 574–586.
[30] L. Özdamar, 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 (1996) 400–407.
[31] E. Pinson, C. Prins, F. Rullier, Using tabu search for solving the resource-constrained project scheduling problem, in: Proceedings of the Fourth International Workshop on Project Management and Scheduling, Leuven, 1994, pp. 102–106.
[32] S.E. Sampson, E.N. Weiss, Local search techniques for the generalized resource constrained project scheduling problem, Naval Research Logistics 40 (1993) 665–675.
[33] A. Schirmer, Case-based reasoning and improved adaptive search for project scheduling, Technical Report 472, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1998.
[34] A. Schirmer, S. Riesenberg, Parameterized heuristics for project scheduling – biased random sampling methods, Technical Report 456, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.
[35] A. Sprecher, R. Kolisch, A. Drexl, Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research 80 (1995) 94–102.
[36] J.P. Stinson, E.W. Davis, B.M. Khumawala, Multiple resource-constrained scheduling using branch and bound, AIIE Transactions 10 (1978) 252–259.
[37] P.R. Thomas, S. Salhi, An investigation into the relationship of heuristic performance with network-resource characteristics, Journal of the Operational Research Society 48 (1) (1997) 34–43.
[38] P.R. Thomas, S. Salhi, A tabu search approach for the resource constrained project scheduling problem, Journal of Heuristics 4 (2) (1998) 123–139.
[39] G. Ulusoy, L. Özdamar, Heuristic performance and network/resource characteristics in resource-constrained project scheduling, Journal of the Operational Research Society 40 (12) (1989) 1145–1152.