European Journal of Operational Research 197 (2009) 492–508

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times

Doreen Krüger, Armin Scholl *

Friedrich-Schiller-Universität Jena, Fakultät für Wirtschaftswissenschaften, Lehrstuhl für Betriebswirtschaftliche Entscheidungsanalyse, Carl-Zeiß-Straße 3, D-07743 Jena, Germany

ARTICLE INFO

Article history:
Received 5 September 2007
Accepted 16 July 2008
Available online 5 August 2008

Keywords:
Project scheduling
Combinatorial optimisation
Mathematical model
Transfer times
Setup

ABSTRACT

We consider the problem of scheduling multiple projects subject to joint resource constraints. Most approaches proposed in the literature so far are based on the unrealistic assumption that resources can be transferred from one project to the other without any expense in time or cost. In order to contribute to closing this gap to reality, we generalise the multi-project scheduling problem by additionally including sequence- and resource-dependent transfer times, which represent setup activities necessary when a resource is removed from one project and reassigned to another (or from one job to another within the same project). In this paper, we define the modified resource constrained multi-project scheduling problem with transfer times (called RCMPSPTT), which aims at minimising the multi-project duration for the single-project approach or the mean project duration for the multi-project approach. We formulate both perspectives as an integer linear program, propose priority rule based solution procedures and present results of comprehensive computational experiments. Provided that the combination of scheduling scheme and priority rules is chosen appropriately, the procedures obtain good results. In particular, resource oriented priority rules are identified to be successful.

1. Introduction

1.1. Multi-project scheduling

Project scheduling has been playing a vital role in literature for some decades now. The common *resource constrained project scheduling problem (RCPSP)* has been studied extensively. However, single-project settings are rare in business today. Usually, companies run more than one project simultaneously. According to Payne (1995) up to 90% of all projects (measured by their value) worldwide are executed in a multi-project environment. This finding goes along with Lova and Tormos (2001), who questioned 202 Spanish companies and found that 84% of them run multiple projects in parallel. Nonetheless, single-project management concepts are by no means irrelevant as they provide a solid basis for multi-project concepts. For excellent introductions to resource constrained single-project scheduling, see Kolisch (1995), Klein (2000), Demeulemeester and Herroelen (2002), Neumann et al. (2003). For surveys of solution approaches see e.g. Brucker et al. (1999), Kolisch and Padman (2001), Kolisch and Hartmann (1999, 2006).

The *resource constrained multi-project scheduling problem* (*RCMPSP*) as an extension of the RCPSP is considered as the simultaneous scheduling of two or more projects which demand the same scarce resources. Precedence constraints are usually defined only within projects. However, precedence relations between projects are also possible which would result in a programme of interdependent projects as a special form of a sheer multi-project (Lycett et al., 2004, p. 289; Ireland, 2002, p. 23). Projects are linked by the usage of the same restricted resources of the company. An objective function on company level often has to be considered although objectives of single-projects may also be regarded (Kurtulus and Davis, 1982, p. 161). The company objective as e.g. maximising profit is aimed at by managing the whole project portfolio or multi-project of the company by a resource manager, whereas project targets are set by single-project managers. The latter aim to minimise project delay, project cost, etc.

* Corresponding author. Tel.: +49 3641943170.
E-mail addresses: d.krueger@wiwi.uni-jena.de (D. Krüger), a.scholl@wiwi.uni-jena.de (A. Scholl).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.07.036

Multi-project scheduling as considered here has been a research topic since the late 1960s. However, it has been studied not nearly as comprehensively as single-project scheduling. One may distinguish two main research fields in multi-project scheduling – the static and the dynamic project environment (Dumond and Mabert, 1988, p. 102). The *static environment* view assumes a closed project portfolio. All projects of the company are summarised to a super-project (portfolio) and scheduled once. The multi-project is unequivocal and no rescheduling necessary. After the last project of a multi-project has been completed, a new multi-project may start. On the contrary, the *dynamic environment* view considers an open project portfolio. While scheduled projects are executed, new projects arrive to the system and have to be integrated by rescheduling.

Research mainly focuses on the static environment. Scheduling in such a static environment has been researched amongst others by Fendley (1968), who was the first discussing the modelling of a complete multi-project scheduling system and proposing methods for assigning due dates to incoming projects and priority rules for sequencing individual jobs. Pritsker et al. (1969) present a zero–one integer program for the problem considering various possible constraints, e.g. job splitting or resource substitutability and objectives like, e.g., total throughput time, makespan or total lateness. Kurtulus and Davis (1982) introduce the single-project approach, whereas Kurtulus and Narula (1985) add the multi-project approach for the multi-project scheduling problem (see below). In both papers, special priority rule based solution procedures are developed and tested. Lawrence and Morton (1993) test resource price based priority rules to minimise weighted tardiness costs within a multi-project. Lova and Tormos (2001) analyse existing priority rules for the single- and multi-project approach. Moreover, they present new two-phase rules for the multi-project approach. Vercellis (1994) suggests a decomposition technique. However, these are only some examples for research on static multi-project scheduling, further references are, amongst others, Patterson (1973), Mohanty and Siddiq (1989), Wiley et al. (1998), and Lova et al. (2000).

Dynamic environments are researched by Dumond and Mabert (1988). Their study is based on priority rules for static environments including due date assignment rules. Dumond (1992) as well as Dumond and Dumond (1993) introduce different resource availability levels. Bock and Patterson (1990) allow resource preemption in the multi-project. Yang and Sum (1993, 1997) consider dynamic project environments with a dual-level management structure for assigning resources to projects on a higher level and scheduling projects on a lower level. Ash and Smith-Daniels (1999) put emphasis on the learning, forgetting and relearning cycle in dynamic multi-project environments while Anavi-Isakow and Golany (2003) apply queuing theory and adapt the production management concept of CONWIP (constant work in progress) to the multi-project environment.

This paper extends static multi-project scheduling which consists of a single- and a multi-project approach. The *single-project approach* merges all projects of the multi-project to an artificial super-project with a dummy start and end job for time and resource scheduling (Kurtulus and Davis, 1982, p. 162). Hence, in this case multi-project scheduling is identical to single-project scheduling of large projects. The multi-project duration, which is given by the realised finishing time of the last job of the latest project, is minimised. The *multi-project approach* keeps the projects separate for time scheduling to identify a critical path for each project. Afterwards, the projects are merged for resource scheduling. The objective is to minimise the mean project delay, i.e., average lateness with respect to due dates or, as a surrogate, critical path times.

The only scheduling procedures applied to both approaches so far are heuristics using priority rules. According to Lova and Tormos (2001), a multi-project consists of about 120 up to 480 jobs. Exact procedures cannot handle problems of this size so far since already the basic RCPSP is NP-hard. Hence, heuristics are the only realistic possibility for solving the RCMPSP.

Even though the static project environment is emphasised in literature, it still fails to pay attention to some important aspects of multi-projects. Virtually all papers neglect resource transfers between projects or assume them having zero duration. Time delays and costs caused by these transfers are not taken into account. In reality, transfers may take time

- when a resource is physically moved from one location to another, e.g. heavy machines, specialists that fly around the world, and/or
- when a resource has to be adjusted in respect to content, e.g. setups of machines or human resources that have to get acquainted with new projects.

1.2. Setup times in project scheduling

Setup times which are a variant of transfer times have already been investigated in production scheduling and lot sizing extensively. In single-project scheduling some research on the field of setup times has been done. Kaplan (1991) introduces sequence-independent setup times when a job is restarted after preemption and presents a dynamic programming procedure to solve the problem. However, Demeulemeester (1992) shows that Kaplan's algorithm may fail to find optimal solutions. Kolisch (1995) develops a zero–one integer program for a restricted version of RCPSP with sequence-independent setup times. Vanhoucke (2008) considers sequence-independent setups necessary to continue a preempted job. Neumann et al. (2003, chapter 2.14) extend the approach of Trautman (2001) and present the RCPSP with time windows and sequence-dependent changeover times. Unlike Trautmann, who assumes that resource requirements can only take values of 0 or 1, Neumann et al. allow for arbitrary resource capacities and resource requirements of jobs. They split the problem into two interdependent subproblems. In a first step, a precedence-feasible schedule is determined. In a second step it is checked whether this schedule is changeover feasible. Changeover feasibility is given when all resource constraints are met while setup times are considered. They also present a branch-and-bound algorithm that enumerates time- and changeover feasible schedules. Mika et al. (2006) give a more extensive literature review on setup times in project scheduling. They classify setups into several types from a job perspective but do not focus on resources which have to changeover to other jobs or projects.

In multi-project scheduling, setup or transfer times are rarely discussed in literature so far. Yang and Sum (1993, 1997) consider resource transfers in a dynamic multi-project environment with a dual-level management structure. A central resource pool manager assigns resources to projects, whereas each project manager schedules jobs within his project using the allocated resources. Resource changeovers can only be handled via a central pool. Dodin and Elimam (1997) present an audit scheduling problem considering sequence-dependent

travel (setup) cost when auditors change auditing projects. Neumann (2003) points out that the problem formulation presented in Neumann et al. (2003) can be applied to multi-project scheduling with distributed locations, too.

Generally, the terms setup, changeover and transfer times are interchangeable. Yet, we prefer using transfer times as Yang and Sum (1993, 1997) did. Setup times are often linked with the setup of e.g. a machine. Since we suppose that resources (e.g. workers or machines) are moved physically from one place to another or have to be adjusted in respect to content (e.g. setup of machines, skill adaptation of workers), we suppose that the term transfer time comprises all aspects of resource transfers in a better way and includes setup times as a special case.

In the following, we consider general resource transfers with sequence- and resource-dependent transfer times with emphasis on the resources which changeover between jobs in the same projects or between jobs in different projects. It is to be emphasised that transfer times exist in pure single-project environments as well. Hence, our models and procedures can be equally applied for single-project problems. However, we assume that the impact of resource transfers is higher in a multi-project context because the distance in place and content is larger than in single-projects. Thus, we mostly refer to the multi-project case.

In Section 2, the new resource constrained multi-project scheduling problem with transfer times (RCMPSPTT), which is a generalisation of the common RCMPSP, is defined.[1] Section 3 presents a mathematical model. In Sections 4 and 5, heuristic solution frameworks are described. Computational experiments are presented in Section 6. Conclusions are given in Section 7.

2. Problem description

In the new problem RCMPSPTT, we consider a multi-project made up of a set of single-projects $P = \{1,\ldots,m\}$. Each project $p \in P$ consists of a set J_p of real jobs as well as a dummy start job s_p and a dummy end job e_p. The multi-project comprises all project jobs as well as a global super source s_0 and sink e_0, which are all summarised in set $J' = \{1,\ldots,n\}$.

Jobs and precedence constraints of the multi-project are represented by a finish-to-start activity-on-node network without time lags. A set of direct predecessors A_j is given for each job $j \in J'$. The global source s_0 has no predecessor, whereas each local project source is successor of s_0 and the global sink e_0 succeeds each local project sink. Except for this global linkage of all projects, precedence constraints exist only between jobs of the same project but not between different projects. To ease presentation, we (re-)number all jobs $j \in J'$ topologically, i.e., $j > i$ for each pair $j \in J'$ and $i \in A_j$. Thus, job 1 is the global source node s_0 and job n the global sink node e_0.

A job j with duration d_j, which is assumed to be integer, must not be interrupted once it has been started. Several types of renewable resources $r \in R$ with constant capacity a_r are available for project execution in every period. Executing job $j \in J'$ requires a constant integer number of resource units u_{jr} of resource type $r \in R$ per period. The required amount must be transferred from other jobs to job j and, thus, often from other projects to the job that is to be executed. This transfer takes time Δ_{ijr} depending on the originating job $i \in J'$ and receiving job $j \in J' - \{i \cup A_i\}$ as well as the resource type $r \in R$ irrespective of the number of resource units transferred. Transfer times may occur within projects but also and, more importantly, among projects. All transfer times are assumed to fulfil the triangular inequality. The global source and sink can be considered to represent a global resource pool where all resources are stored before the multi-project starts and to which all resources must return after having terminated the multi-project, which may also require some time. However, transfers which flow from the global source s_0 to the global sink e_0 directly take no time because these flows absorb redundant resource units and are not executed in reality.

Dummy jobs are characterised by a duration and resource usage of zero for each resource type with exception of the global source s_0 and sink e_0. These two dummy jobs take no time but their resource usage is defined by $u_{s_0r} = u_{e_0r} = a_r$ for $r \in R$ because the global source needs to provide all resources to the multi-project while the global sink collects them.

The RCMPSPTT is constituted by determining finishing times F_j for all jobs $j \in J'$ and corresponding resource transfers x_{ijr} such that a multitude of constraints is met. All precedence and resource constraints must be observed while sequence- and resource type-dependent transfer times for resources changing to other jobs are considered.

In the single-project approach, the multi-project duration MPD, i.e., the finishing time F_{e_0} of the global sink e_0, or its relative increase MPDI is minimised. Both objectives are equivalent. MPDI is measured as the relative deviation of MPD from the time of the multi-project's critical path from s_0 to e_0. The critical path time CP is used to evaluate the delay of the multi-project caused by resource restrictions. This approach is equivalent to scheduling single-projects aiming at minimal project duration.

In the multi-project approach, the objective consists in minimising mean project delay MD defined as average relative deviation of the realised finishing time F_{e_p} from the critical path time CP_p over all projects $p \in P$. CP_p is used as a surrogate for a due date of each project to evaluate their delays, because due dates are not pre-determined in this problem version. This approach is not equal with single-project scheduling since the objective differs.

The presented objectives are commonly used in existing literature on the basic multi-project scheduling problem (Pritsker et al., 1969; Kurtulus and Davis, 1982; Kurtulus and Narula, 1985 as well as Lova and Tormos, 2001). However, these objectives for scheduling multiple projects should not be taken for granted. This paper uses them for a first analysis of the new problem but will point out other possibilities in the conclusion.

3. Mathematical model

In Krüger and Scholl (2008, chapter 3.2) a mixed-integer linear program for the RCMPSPTT is developed. It is based on combining the traditional model for RCPSP by Pritsker et al. (1969) with a network flow based formulation of the single-project scheduling problem with sequence-dependent setup times as proposed by Neumann et al. (2003, chapter 2.14) as well as the flow formulation of Artigue et al.

[1] In supplementary document (sect. A), the new approach is contrasted to former ones presented in literature.

(2003). The model includes the typical assumptions of the RCPSP and extends it by considering transfer times between jobs of different or the same project(s).

The following *parameters* are used:

P set of projects; index: p
J_p set of real jobs of project $p \in P$; index: j
J set of real jobs within all projects; $J = \cup_{p\in P} J_p$
J'_p jobs of project $p \in P$ including dummy start job s_p and dummy end job e_p; $J'_p = J_p \cup \{s_p, e_p\}$
J' set of jobs within all projects plus single global source s_0 and global sink e_0, i.e., $J' = \cup_{p\in P} J'_p \cup \{s_0, e_0\}$
n number of jobs; $n = |J'|$
d_j duration of job $j \in J'$ (with $d_{s_p} = d_{e_p} = 0$ for $p \in P \cup \{0\}$)
T upper bound on the project duration
t index for periods; $t = 0,\ldots,T$
A_j set of direct predecessors of job $j \in J'$; $A_{s_0} = \{\}$, $A_{e_0} = \cup_{p\in P}\{e_p\}$, $A_{s_p} = \{s_0\}$ for $p \in P$; for $j \in J_p$ with $p \in P$: $A_j \subseteq J_p - \{j\}$ (precedence constraints only within a project)
A^*_j set of direct and indirect predecessors of job $j \in J'$ (predecessors in the transitive closure of the graph)
S_j set of direct successors of job $j \in J'$; $S_{e_0} = \{\}$, $S_{s_0} = \cup_{p\in P}\{s_p\}$, $S_{e_p} = \{e_0\}$ for $p \in P$ for $j \in J_p$ with $p \in P$: $S_j = \{i | i \in J_p \cup \{e_p\} \wedge j \in A_i\}$
S^*_j set of direct and indirect successors of job $j \in J'$
EF_j earliest finishing time of job $j \in J'$; $\mathrm{EF}_{s_p} = 0$ for $p \in P \cup \{0\}$ (forwards path)
LF_j latest finishing time of job $j \in J'$ (backwards path)
TI_j time window for finishing job $j \in J'$; $\mathrm{TI}_j = [\mathrm{EF}_j\ \mathrm{LF}_j]$
CP critical path time of multi-project
CP_p individual critical path time of project $p \in P$
R set of resources; index: r
a_r number of units of resource $r \in R$ available per period
u_{jr} number of units of resource $r \in R$ required for performing job $j \in J'$ per period, $u_{s_0 r} = u_{e_0 r} = a_r \forall r \in R$
Jr_j set of real jobs (including global sink) to which resources might be transferred after having performed; $j \in J$; $Jr_j := J \cup \{e_0\} - \{j\} - A^*_j$
Δ_{ijr} time for transferring units of resource $r \in R$ from task $i \in J \cup \{s_0\}$ to task $j \in Jr_i \cup \{e_0\}$, $\Delta_{s_0 e_0 r} = 0\ \forall r \in R$

The following *variables* are used:

$$f_{jt} = \begin{cases} 1 & \text{if activity } j \text{ is terminated at the end of period } t \text{ for } j \in J' \text{ and } t \in \mathrm{TI}_j \\ 0 & \text{otherwise} \end{cases}$$

F_j realised finishing time of job $j \in J'$

$$z_{ijr} = \begin{cases} 1 & \text{if resources } r \text{ is transferred from activity } i \text{ to } j \text{ for } r \in R,\ i \in J \cup \{s_0\},\ j \in Jr_i \\ 0 & \text{otherwise} \end{cases}$$

x_{ijr} number of units of resource $r \in R$ transferred from job $i \in J \cup \{s_0\}$ to job $j \in Jr_i$

The *model* is given by the objective function (1) and the set of constraints (2)–(12).

(*I*) Time scheduling

$$\text{Minimise } \Phi(\mathbf{F}, \mathbf{f}, \mathbf{x}, \mathbf{z}) \text{ s.t.} \tag{1}$$

$$\sum_{t\in \mathrm{TI}_j} f_{jt} = 1 \quad \text{for all } j \in J' \tag{2}$$

$$F_j = \sum_{t\in \mathrm{TI}_j} t - f_{jt} \quad \text{for all } j \in J' \tag{3}$$

$$F_j - F_i \geqslant d_j \quad \text{for } j \in J' \text{ and } i \in A_j \tag{4}$$

Linking (I) and (II)

$$F_i + \Delta_{ijr} + d_j \leqslant F_j + T \cdot (1 - z_{ijr}) \text{ for } i \in J \cup \{s_0\},\ j \in Jr_i \text{ and } r \in R \tag{5}$$

$$x_{ijr} \leqslant z_{ijr} \cdot \min\{u_{ir}; u_{jr}\} \text{ for } i \in J \cup \{s_0\},\ j \in Jr_i \text{ and } r \in R \tag{6}$$

$$z_{ijr} \leqslant x_{ijr} \text{ for } i \in J \cup \{s_0\},\ j \in Jr_i \text{ and } r \in R \tag{7}$$

(II) Resource scheduling

$$\sum_{h\in J - \{i\} - S^*_i} x_{hir} = u_{ir} \quad \text{for } i \in J \cup \{e_0\}, \text{ and } r \in R \tag{8}$$

$$\sum_{j\in Jr_i} x_{ijr} = u_{ir} \text{ for } i \in J \cup \{s_0\} \text{ and } r \in R \tag{9}$$

Variables

$$f_{jt} \in \{0,1\} \quad \text{for } j \in J' \text{ and } t \in \mathrm{TI}_j \tag{10}$$

$$F_j \geqslant 0 \quad \text{for } j \in J' \tag{11}$$

$$z_{ijr} \in \{0,1\},\ x_{ijr} \geqslant 0 \text{ for } r \in R,\ i \in J \cup \{s_0\},\ j \in Jr_i \tag{12}$$

The objective function (1) depends on the approach applied. In the single-project perspective, the multi-project duration increase MPDI is minimised, which is equivalent to minimising the multi-project duration MPD:

$$\text{Minimise MPDI } (\mathbf{F},\mathbf{f},\mathbf{x},\mathbf{z}) = \frac{F_{e_0} - \mathrm{CP}}{\mathrm{CP}} \cdot 100\%. \tag{13}$$

In the multi-project approach, the mean project delay MD is used as performance measure:

$$\text{Minimise MD}(\mathbf{F},\mathbf{f},\mathbf{x},\mathbf{z}) = \frac{1}{|P|} \cdot \sum_{p \in P} (F_{e_p} - \mathrm{CP}_p). \tag{14}$$

Time scheduling constraints (2)–(4) are well-known from the RCPSP formulation. Constraints (8) and (9) represent resource flows in the multi-project. Inequalities (5)–(7) link the time scheduling and the resource flow parts of the model. Moreover, they handle resource transfers. Eventually, (10)–(12) define decision variables of the problem. The constraints are explained in more detail in supplementary document (sect. B).

Since RCMPSPTT is a generalisation of RCPSP, the problem is NP-hard. As expected, preliminary experiments indicate that modelling and solving the problem with modern optimisation software like XPress-MP or CPlex is not sufficient to solve problems of realistic size. Therefore, we develop heuristic solution frameworks which combine elements for time and resource scheduling. Basics for time scheduling are the two common schedule generation schemes and priority rules well-known for RCPSP and RCMPSP (cf., e.g., Kurtulus and Davis, 1982; Klein, 2000; Lova and Tormos, 2001) which have to be adapted adequately. For resource and transfer scheduling new scheduling rules are presented. In Section 4, we define a solution framework based on the parallel scheduling scheme. In Section 5, the framework is adapted to the serial scheme. The presented framework may also be used as a basis for metaheuristics and branch-and-bound procedures.

4. Parallel scheduling framework

4.1. Algorithm

The time oriented parallel scheduling scheme creates a feasible schedule by considering increasing decision times t. At each decision time t as many jobs as possible are started such that precedence and resource feasibility is given (Lova and Tormos, 2001, p. 267). The sequence in which jobs are considered is determined by a priority rule. This common scheduling scheme is adapted to the case of sequence- and resource type-dependent transfer times.

In order to ease presentation, the dummy source nodes s_p and end nodes e_p of all single-projects are eliminated by directly linking the projects to the global source node s_0 and the global sink node e_0. Finally, the starting and finishing times of those nodes can simply be set as follows:

$$s_p = \min\{F_j - d_j \mid j \in J_p\}, \quad e_p = \max\{F_j \mid j \in J_p\} \quad \text{for } p = 1,\ldots,P.$$

The algorithmic representation is based on that of Kolisch and Hartmann (1999). It uses the following additional parameters complementing the notation of Section 3:

J'	due to temporarily deleting the dummy sources and sinks of the projects, we get $J' = J \cup \{s_0,e_0\}$; $n = \lvert J' \rvert$
$\mathcal{C}$	set of completed jobs at scheduling time t; $\mathcal{C} = \{j \in J' \mid F_j \leqslant t\}$
$\mathcal{A}$	set of jobs active in period t; $\mathcal{A} = \{j \in J' \mid F_j - d_j \leqslant t < F_j\}$
$\mathcal{S}$	set of jobs scheduled up to time t; $\mathcal{S} = \mathcal{A} \cup \mathcal{C}$
$\tilde{a}_r(t)$	available units of resource type $r \in R$ at scheduling time t with $\tilde{a}_r(t) = a_r - \sum_{j \in \mathcal{A}} u_{jr}$
$\mathcal{D}$	set of eligible jobs at scheduling time t with $\mathcal{D} = \{j \in J' - \mathcal{S} \mid A_j \subseteq \mathcal{C} \wedge u_{jk} \leqslant \tilde{a}_r(t) \forall r \in R\}$
y_{jr}	number of units of resource r transferred to job j and still awaiting to be delivered to a another job
TJ^r	set of transfer-feasible jobs able to deliver units of resource type r to job j up to time t; $\mathrm{TJ}^r = \{i \in \mathcal{S} \mid F_i + \Delta_{ijr} \leqslant t \wedge y_{ir} > 0\}$
s^r	total supply of resource type $r \in R$ by transfer-feasible jobs at time t; $s^r = \sum_{i \in \mathrm{TJ}^r} y_{ir}$

Algorithm 1 starts out with an initialisation of the problem instance. All quantities of resource transfers x_{ijr} and activity resource stocks y_{jr} are initialised to zero. The dummy global source $s_0 = 1$ is scheduled at time 0 and put into the set of completed jobs. Afterwards, the modified parallel scheduling scheme is applied.

For each trial scheduling time t, as many eligible jobs as possible are scheduled provided resource and precedence constraints including transfer times are not hurt. From the set $\mathcal{D}$ of eligible jobs, the jobs j to be examined are chosen one after another in a sequence defined by an appropriate *job rule* (see Section 4.2). For each considered j, the sets TJ^r, which contain all jobs that can deliver at least one unit of resource type r to job j up to time t, are determined. If these jobs supply sufficient resource units of each resource type $r \in R$, job j is scheduled and delivering jobs i are chosen from TJ^r by a *resource transfer rule* until the demand of job j is satisfied (see Section 4.3). Simultaneously, transfer quantities x_{ijr} from i to j are calculated for each r. Should any determined set TJ^r offer insufficient resource amounts to the selected job j, this job is removed from the set $\mathcal{D}$ of eligible jobs. If there does not remain any jobs in the set of eligible jobs $\mathcal{D}$ for which feasible resource transfers can be determined, scheduling time t is increased step by step until a non-empty set $\mathcal{D}$ is determined and at least one job j is contained for which all required resources can be provided in time. When all jobs have been scheduled, the algorithm stops.

Algorithm 1. Modified parallel scheduling scheme

Initialise $t := 0$; $\mathcal{A} := \emptyset$; $\mathcal{S} := \mathcal{C} := \{1\}$; $F_1 := 0$; $\tilde{a}_r(t) := a_r$ for all $r \in R$ and $t = 0, \ldots, T$;
$y_{jr} := 0$ for all $j \in J'$ and $r \in R$; $x_{ijr} := 0$ for all $i, j \in J'$ and $r \in R$
$\mathcal{D} := \{j \in J' \mid A_j = \{1\}\}$; /* set of eligible jobs at t=0
While $|\mathcal{S}| < n$ do /* iterate on increasing t until all jobs are assigned
While $\mathcal{D} \neq \emptyset$ do /* assign as many jobs as possible to current time t
Select best job $j \in \mathcal{D}$; /* apply *job rule* (Section 4.2)
Resource scheduling
For $r \in R$ do /* for each resource type ...
$TJ^r = \{ i \in \mathcal{S} \mid F_i + \Delta_{ijr} \leq t \ \wedge \ y_{ir} > 0 \}$; /* ... compute transfer-feasible set
$s^r := \sum_{i \in TJ^r} y_{ir}$; /* ... compute total supply of resource r for j in t
If ($s^r \geq u_{jr}$ for all $r \in R$) do /* if supply is sufficient for all resources, then ...
For $r \in R$ do /* ... find delivering jobs for all r as follows:
While $y_{jr} < u_{jr}$ do /* repeat until resource demand of j for r is satisfied
Select and remove best job $i \in TJ^r$; /* apply *resource transfer rule* (Section 4.3)
$x_{ijr} := \min\{y_{ir}, u_{jr} - y_{jr}\}$; /* compute transferable amount of resource r
$y_{jr} := y_{jr} + x_{ijr}$; $y_{ir} := y_{ir} - x_{ijr}$ /* update resource stocks
$F_j := t + d_j$; $\mathcal{S} := \mathcal{S} \cup \{j\}$; $\mathcal{A} := \mathcal{A} \cup \{j\}$ /* ... schedule job j starting in t and finishing in $t + d_j$
For $\tau \in [t, t + d_j[$ and $r \in R$ do
$\tilde{a}_r(\tau) := \tilde{a}_r(\tau) - u_{jr}$; /* ... reduce available capacity of resources
$\mathcal{D} := \mathcal{D} - \{j\}$; /* remove j from the set of (unexamined) eligible jobs
$t := t + 1$; /* increase current scheduling time t
$\mathcal{C}' = \{i \in \mathcal{A} \mid F_i \leq t\}$; /* set of tasks just completed in current t
$\mathcal{A} := \mathcal{A} - \mathcal{C}'$; $\mathcal{C} := \mathcal{C} \cup \mathcal{C}'$; /* update set of active and completed jobs
$\mathcal{D} := \{j \in J' - \mathcal{S} \mid A_j \subseteq \mathcal{C} \wedge u_{jk} \leq \tilde{a}_r(t) \forall r \in R\}$ /* compute set of eligible tasks for current t

4.2. Job rules

Job (selection) rules are the original priority rules known for the RCPSP or the RCMPSP. A job rule assigns priority values to all jobs and orders them according to non-increasing or non-decreasing values. Within a scheduling scheme, the jobs are examined and, if possible, scheduled in this order. A plethora of such rules are known for the RCPSP. Overviews can be found in, e.g., Kolisch and Hartmann (1999), Kolisch and Padman (2001) and Klein (2000, chapter 5.2+7.4).

Yet, in the context of multi-project scheduling specialised job rules have been developed and applied both in the parallel and the serial scheme. The rules can be classified according to the approach they are designed for. There are special rules for the single- and the multi-project approach as well as rules that are independent of these approaches. The most common rules for the multi-project scheduling problem are summarised in Table 1 (see Lova and Tormos (2001); Pritsker et al. (1969); Kurtulus and Davis (1982); Kurtulus and Narula (1985)).

Table 1
Job rules for parallel and serial scheme

	Job rule (job j)	Extremum	Priority value π_j
Multi-project approach	minSASP	min	$\pi_j = CP_{p(j)} + d_j$ with $CP_{p(j)}$ as critical path of project $p(j)$
	minSLK_MP	min	$\pi_j = LF_j(CP_{p(j)}) - EF_j(CP_{p(j)})$ with $CP_{p(j)}$ as critical path of project $p(j)$
	minLFT_MP	min	$\pi_j = LF_j(CP_{p(j)})$ with $CP_{p(j)}$ as critical path of project $p(j)$
	maxTWK(dyn)	max	$\pi_j = d_j \cdot \sum_{r=1}^{\lvert R \rvert} u_{jr} + \sum_{k \in S \cap J_{p(j)}} d_k \cdot \sum_{r=1}^{\lvert R \rvert} u_{kr}$
	maxRD_MP	max	$\pi_j = d_j \cdot \sum_{r=1}^{\lvert R \rvert} u_{jr} + \sum_{k=1}^{\lvert J_{p(j)} \rvert} (d_k \cdot \sum_{r=1}^{\lvert R \rvert} u_{kr})$
Single-project approach	minSLK_SP	min	$\pi_j = LF_j(CP) - EF_j(CP)$ with CP as critical path of the super-project
	minSLK_SP(dyn)	min	$\pi_j = LF_j(PS) - EF_j(PS)$ with PS as partial schedule of the super-project
	minLFT_SP	min	$\pi_j = LF_j(CP)$ with CP as critical path of the super-project
	maxRD_SP	max	$\pi_j = d_j \cdot \sum_{r=1}^{\lvert R \rvert} u_{jr}$
Independent	FCFS(dyn)	min	$\pi_j = \max\{EF_i \mid i \in A_j \cap S\}$
	Random	min	$\pi_j = \text{random}()$

Column 1 classifies the presented rules regarding the approach they are used with. For the notation used see Section 3; additionally $p(j)$ indicates the project to which a job j belongs, i.e., $j \in J_{p(j)}$. Shortest activity from shortest project (minSASP), minimum slack (minSLK), minimum latest finishing time (minLFT), maximum total work content (maxTWK) as well as maximum resource demand (maxRD) are rules for the multi-project approach because they rely on information about each project within the multi-project (extension "MP"). Apart from minSASP and maxTWK, the rules can be used for the single-project approach as well (extension "SP"). They use information of the super (multi-)project for priority calculation. In this context, dynamic rules ("dyn") are possible, too. Priority values are updated in each iteration depending on the already scheduled jobs (partial schedule PS) and their fixed starting and finishing times instead of being computed only once at the beginning of the whole procedure (static rules). The rules first come first served (FCFS) and Random are independent of the approach to multi-project scheduling.

4.3. Resource transfer rules

When a job j is selected to be scheduled at time t and all sets TJ^r supply sufficient resource units, it must be decided which of the jobs in each TJ^r are chosen to deliver resources to job j. This is obviously only necessary if there is a surplus of provided resource units.

To find the delivering jobs for each resource r required by job j, all jobs in TJ^r are sorted by a priority rule which we call *resource transfer rule* (to be more specific, it is a transfer-from rule, cf. Section 5.2). Even if the same rule is applied to each r, different priority lists will result.

We suggest three rules, divided into time and resource oriented ones (Table 2). A time oriented rule is minTT, which selects a job i with lowest transfer time from i to job j first, regarding the considered resource type r. The rule minGAP prefers jobs with minimal gap between the earliest delivery time (=finishing time F_i + transfer time Δ_{ijr}) and the start time t of j. The resource oriented rule RS sorts the jobs of TJ^r according to their free resource stock after task execution, i.e., the amount of resource units that are not transferred to another job yet. minRS selects the job with minimum and maxRS with maximum stock first.

After the transfer-feasible jobs have been sorted, the actually delivering tasks are selected. We propose two different ways, forward and backward selection.

Forward selection uses the original priority list. The job with highest priority is chosen and delivers as many free resource units to job j as possible or necessary, whatever is the lower. Jobs from the top of the list are selected until the resource demand of job j is satisfied. *Backward selection* sums up the provided free resources along the priority list, starting with the highest priority job. The first job in the list which causes equality or surplus of resource units when added is used as the starting point for backward transfer scheduling. From this starting point on, the resource demand of job j is satisfied by the jobs along the list downwards to the task with the highest priority. In each step as many free resources are provided to job j as required and available, whatever comes first.

Table 2
Resource transfer rules for parallel scheme

	Transfer-from rule	Extremum	Priority value π_i^{jr} for $i \in TJ^r$
Time oriented	minTT	min	$\pi_i^{jr} = \Delta_{ijr}$
	minGAP	min	$\pi_i^{jr} = \text{gap}_{ijr} = t - (F_i + \Delta_{ijr})$
Resource oriented	minRS	min	$\pi_i^{jr} = y_{ir} = u_{ir} - \sum_{k \in S} x_{ikr}$
	maxRS	max	$\pi_i^{jr} = y_{ir} = u_{ir} - \sum_{k \in S} x_{ikr}$

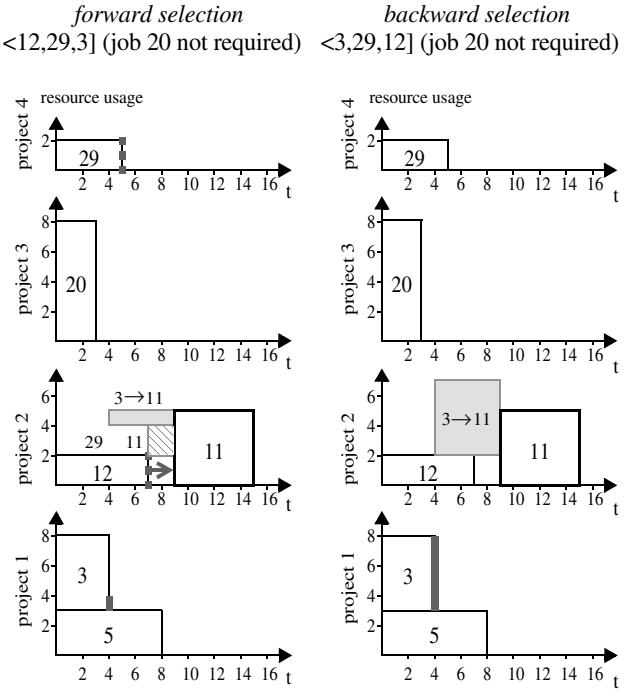

Fig. 1. Forward and backward selection.

Fig. 1 illustrates both selection principles by means of an example with $|P| = 4$ projects and a single resource with capacity $a_1 = 20$. Each job is visualised as a rectangle with its duration defining the horizontal and its resource usage the vertical extent. Job $j = 11$ out of project 2 with a resource demand of $u_{11,1} = 5$ is selected by some job rule for the current scheduling time $t = 9$. All jobs in $\mathcal{S} = \{3, 5, 12, 20, 29\}$ are already scheduled. As transfer times we consider $\Delta_{3,11} = 5$, $\Delta_{5,11} = 5$, $\Delta_{12,11} = 0$, $\Delta_{20,11} = 6$, and $\Delta_{29,11} = 2$ (index for $r = 1$ omitted). Due to $F_5 + \Delta_{5,11} = 8 + 5 = 13 > 9$, we obtain $TJ^1 = \{3, 12, 20, 29\}$. The minTT rule determines the priority list $\langle 12, 29, 3, 20]$. To fulfil the resource needs of job 11, the first three jobs of the list, which can provide nine resource units, are sufficient. The actual quantities sent by those jobs are different depending on the selection principle used. Forward selection results in transfers from all three jobs as illustrated in Fig. 1 while for backward selection only job 3 delivers resource units to task 11.

Assuming that transfer times within the same project are smaller than between different projects, the forward selection principle together with the minTT rule ensures that resources are kept and used within the same project if possible. However, as the example shows, this may lead to splitting up transfers (job 3 delivers only a single unit). It could be of advantage if job 3 provided all its free resource units to task 11 since a resource transfer from project 1 to project 2 takes place anyway. Then the resource supply of jobs 12 and 29 could be fully used for other transfers as achieved by backward selection here.

5. Serial scheduling framework

5.1. Basic algorithm

Contrary to the parallel scheduling scheme, the serial one is job oriented and builds the schedule in n stages with n jobs to be scheduled. At each stage g, a job is selected and scheduled as early as possible such that precedence and resource feasibility is given (Lova and Tormos, 2001, p. 267).

Algorithm 2. Modified serial scheduling scheme

Initialise $\tilde{a}_r(t) = a_r$ for $r \in R$, $t = 0, \ldots, T$; $F_1 = 0$; $\mathcal{S} = \{1\}$;
$y_{jr} := 0$ for $j \in J'$, $r \in R$; $x_{ijr} := 0$ for $i, j \in J'$, $r \in R$;
$\mathcal{D} := \{j \in J' - \{1\} \mid A_j = \{1\}\}$; /* set of eligible jobs

For g := 1 to n do
  Select the best job $j \in \mathcal{D}$ /* apply *job rule*
  $EF_j := \max\{F_h \mid h \in A_j\} + d_j$
  $ERF_j := \min\{t \geq EF_j \mid u_{jr} \leq \tilde{a}_r(\tau)\ \forall r \in R, \tau \in [t - d_j; t[\}$;
  $ETF_j := \text{CalcResTrans}(j, ERF_j)$;
  $\mathcal{S} := \mathcal{S} \cup \{j\}$; $F_j := ETF_j$; /* schedule job j
  For $\tau \in [F_j - d_j, F_j[$ and $r \in R$ do
    $\tilde{a}_r(\tau) := \tilde{a}_r(\tau) - u_{jr}$; /* reduce available capacity
  $\mathcal{D} := \{j \in J' - \mathcal{S} \mid A_j \subseteq \mathcal{S}\}$; /* update eligible set

Algorithm 2 illustrates the time scheduling frame of the adapted serial scheduling scheme based on the representation of Kolisch and Hartmann (1999). After initialisation, one job j, which is selected by *a job rule*, is scheduled at each stage g. For this purpose,the same rules as in case of the parallel scheme can be used (cf. Section 4.2). However, the set of eligible jobs $\mathcal{D}$ is now defined by $\mathcal{D} = \{j \in J' - \mathcal{S} \mid A_j \subseteq \mathcal{S}\}$.

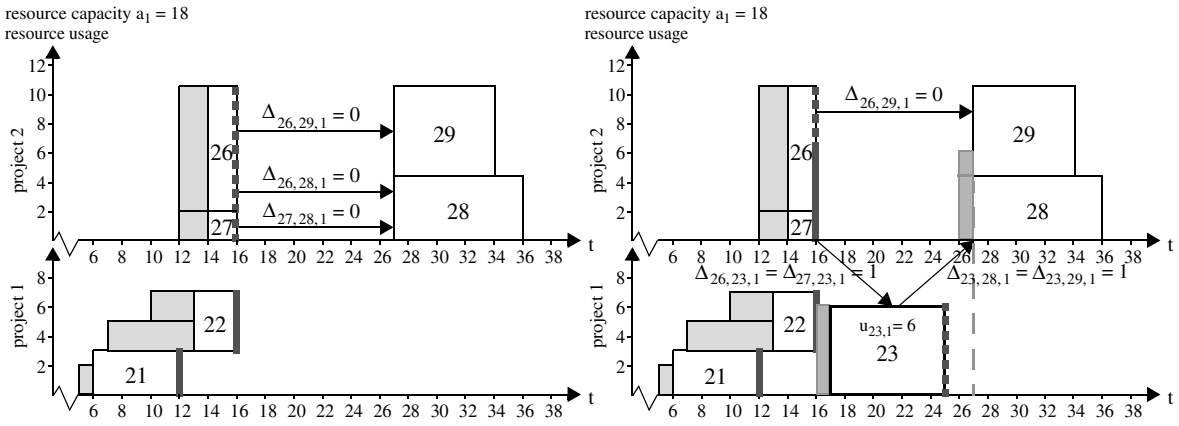

Fig. 2. Activity insertion (serial schedule generation scheme).

For the selected job j the earliest precedence-feasible finishing time EF_j is determined by usual time calculation. The earliest resource feasible finishing time ERF_j of job j considers all resource capacities additionally but still ignores transfer times Δ_{ijr}. Finally, the earliest transfer-feasible finishing time ETF_j is calculated. It is the earliest time at which job j can be finished when precedence, resource and transfer constraints are taken into account. Thus, it is used as realised finishing time F_j in the solution schedule. Obviously, $EF_j < ERF_j < ETF_j$ holds. When a job j has been scheduled, it is inserted in the set of scheduled jobs $\mathcal{S}$ while available resources and the set $\mathcal{D}$ of eligible jobs are updated.

Algorithm 3. Function *CalcResTrans*(j, ERF_j) for feasible starting time of selected job j and resource flows

Initialise $z_r := 0\ \forall r \in R$; $t := ERF_j - d_j$; /* z_r adds up the resource units delivered to j

Repeat /* repeat until feasible starting time t is found

For each $r \in R$ do /* consider each resource r separately

For each $i \in \mathcal{S}$ do /* set of receiving jobs in breakable transfers

$K_{ir} := \{s \in \mathcal{S} - \{i\} \mid x_{isr} > 0 \wedge F_i + \Delta_{ijr} \le t \le F_s - d_s - \Delta_{jsr} - d_j\}$;

$TJ^r := \{i \in \mathcal{S} \mid (y_{ir} > 0 \wedge F_i + \Delta_{ijr} \le t) \vee K_{ir} \neq \emptyset\}$; /* set of potentially delivering jobs

$s^r := \sum_{i \in TJ^r} \left(y_{ir} + \sum_{k \in K_{ir}} x_{ikr} \right)$ /* total supply of resource r for j

If ($s^r \ge u_{jr}$ for each $r \in R$) then /* if supply is sufficient for all resources then ...

resource scheduling

For each $r \in R$ do /* ... generate deliveries for all resource types r

While $z_r < u_{jr}$ do /* repeat as long as units of r are missing

Select and remove best delivering job $i \in TJ^r$ /* apply *transfer-from rule*

While ($K_{ir} \neq \emptyset$) and ($z_r < u_{jr}$) do /* repeat for all breakable transfers

Select the best receiving job $k \in K_{ir}$ /* apply *transfer-to rule*

$x' := \min\{x_{ikr}, u_{jr} - y_{jr}\}$; $K_{ir} := K_{ir} - \{k\}$; /* set transferable quantity, reduce K_{ir}

$x_{ijr} := x_{ijr} + x'$; $x_{ikr} := x_{ikr} - x'$; $x_{jkr} := x'$; $z_r := z_r + x'$ /* modify transfers

$x' := \min\{y_{ir}, u_{jr} - y_{jr}\}$; /* transfer remaining supply of i to j

$x_{ijr} := x_{ijr} + x'$; $y_{ir} := y_{ir} - x'$; $y_{jr} := y_{jr} + x'$; $z_r := z_r + x'$ /* modify transfers

Return $ETF_j = t + d_j$; /* ... terminate function, return feasible time for j

Else t := t+1; /* increase trial time t and repeat

The function CalcResTrans(j,ERF_j) for determining a transfer-feasible finishing time ETF_j and corresponding resource flows is given in Algorithm 3. It begins with trial starting time $t = ERF_j - d_j$ of the selected job j and increases t incrementally until transfer-feasibility is achieved for all resource types. As in case of the parallel scheme, it has to be ensured that each resource unit required by job j arrives in t or earlier. Moreover, if job j is inserted between already scheduled jobs, it must be ensured that the broken resource flows can be repaired feasibly. Therefore, for each already scheduled i and each resource r, any transfer from i to an also scheduled k is examined. If it is possible to deliver resource units from i to j at time t or earlier and to transfer them to k after finishing j without delaying job k, a *breakable transfer* is found and k is added to the set K_{ir}. The set TJ^r of jobs potentially delivering resource r to j in time consists of all jobs which have still undelivered resource units ($y_{ir} > 0$) and of those from which at least one breakable transfer starts ($K_{ir} \neq \emptyset$). All those possible deliveries are summed up in s^r.

When enough units are available for all resources, j can be scheduled to start at time t. The delivering jobs i are selected from TJ^r one after another according to a *transfer-from rule* (cf. Section 5.2.1). The receiving jobs $k \in K_{ir}$ of breakable flows are chosen one after another according to a *transfer-to rule* (cf. Section 5.2.2). The current flow x_{ikr} from i to k is broken up and job j receives this very quantity from job i while job j delivers the same amounts back to job k. If still necessary, the remaining resource stocks of job i (y_{ir}) are used to satisfy the demands of job j. This process is repeated until all resource needs of job j are satisfied.

Fig. 2 represents a cutout of a multi-project schedule with two projects and a capacity of $a_1 = 18$ units of the single resource 1. Activity $j = 23$ as part of project 1 is to be scheduled. It requires 6 units of the resource. Regarding all precedence relations, it may start after job 22 is finished. Jobs 21 and 22 deliver all their resource units to already scheduled jobs, which are not part of the cutout (indicated by the solid bar at the end of those jobs). Moreover, it has already been decided that jobs 26 and 27 provide all their resource supply to jobs 28 and 29, which for some reasons cannot start earlier. Since transfer times within a project are supposed to be zero, the resources supplied by jobs 26 and 27 are waiting for their usage by 28 and 29 (this slack is indicated by a dotted bar). The right Gantt chart shows that a part of these waiting resources can be transferred to job 23, used to perform it and return to 28 and 29 in time. If this would not be possible without delaying already scheduled jobs, insertion must be forbidden due to the logic of the serial scheme to schedule each task finally. Notice that job insertion cannot occur in the parallel scheduling scheme, because the starting time t of jobs is increased monotonic from iteration to iteration in this scheme.

Table 3
Additional transfer-from rules for resource transfers (serial scheduling scheme)

	Transfer-from rule	Extremum	Priority value π_i^{tr} for $i \in \mathrm{TJ}^r$
Time oriented	minES	min	$\pi_i^{tr} = F_i + \Delta_{ijr}$
Resource oriented	minTRS, maxTRS	min, max	$\pi_i^{tr} = y_{ir} + \sum_{k \in K_{ir}} x_{ikr}$
Compound	maxCV	max	$\pi_i^{tr} = \lambda_1 \cdot \left(\frac{\bar{\Delta}_{jr} - \Delta_{ijr}}{\bar{\Delta}_{jr} - \underline{\Delta}_{jr}}\right) + \lambda_2 \cdot \left(\frac{\overline{FA}_{jr} - (F_i + \Delta_{ijr})}{\overline{FA}_{jr} - \underline{FA}_{jr}}\right) + \lambda_3 \cdot \left(\frac{y_{ir} - \bar{y}_r}{\underline{y}_r - \bar{y}_r}\right)$ with $\underline{\Delta}_{jr} = \min\{\Delta_{hjr} \mid h \in \mathrm{TJ}^r\}$, $\bar{\Delta}_{jr} = \max\{\Delta_{hjr} \mid h \in \mathrm{TJ}^r\}$ $\underline{FA}_{jr} = \min\{F_h + \Delta_{hjr} \mid h \in \mathrm{TJ}^r\}$, $\overline{FA}_{jr} = \max\{F_h + \Delta_{hjr} \mid h \in \mathrm{TJ}^r\}$ $\underline{y}_r = \min\{y_{hr} \mid h \in \mathrm{TJ}^r\}$, $\bar{y}_r = \max\{y_{hr} \mid h \in \mathrm{TJ}^r\}$

5.2. Resource transfer rules

In contrast to the parallel scheme, two types of transfer rules are necessary in the serial scheme because resource transfers to be broken up must be priorised as well.

5.2.1. Transfer-from rules

Like for the parallel scheme, it must be determined by a priority rule which of the transfer-feasible jobs $i \in \mathrm{TJ}^r$ deliver resources to the selected job j. Again, this is to relevant only, if there is a surplus of available resources. The rules that are established in Table 2 for the parallel scheme can also be applied for the serial scheme. Yet, additional rules are possible and summarised in Table 3.

Besides minTT and minGAP, minES is another time oriented rule. It selects that job i which leads to the earliest starting time of job j. The group of resource oriented rules is extended by the total resource supply (TRS) rule, which considers not only still freely available resource stocks of a job but also the resources that can be provided by breaking up existing resource flows feasibly. For the parallel scheme, this rule does not differ from the resource stock (minRS, maxRS) rule since breaking up resource flows (job insertion) is impossible.

Finally, a compound priority rule was built. It joins time and resource oriented aspects by considering relative transfer times, relative earliest starting times and relative resource stocks. In each category, the relative values are obtained by normalising the absolute values to the interval [0,1] with the best job in this category set to 1 and the worst one set to 0. Finally, the ratings of the three categories are weighted and summed up. The weights can be set according to the preferences of the decision maker, whatever he assumes most important. For the parallel scheme, the relative earliest starting time component is irrelevant but can be replaced by the relative gap (with the biggest gap as best value) to form a compound rule for this scheme, too.

The delivering jobs for the transfer can be selected from the priority list *forwards* or *backwards* just as described in Section 4.3 for the parallel scheduling scheme until the resource demand of job j is fulfilled.

5.2.2. Transfer-to rules

If a job i, which has been selected by a transfer-from rule as described in the preceding section, delivers more units of a resource r to already scheduled jobs than still needed by job j, i.e., $\sum_{k \in K_{ir}} x_{ikr} > u_{jr} - z_r$ (for z_r see Algorithm 3), the sequence of breaking up resource flows becomes relevant. Thus, an additional priority rule (called *transfer-to rule)* is required to determine which of the existing resource flows from j to other jobs $k \in K_{ir}$ are broken up to satisfy the demand of job j for resource r. Table 4 presents two transfer-to rules.

Fig. 3 illustrates the effect of the rules. Transfer times are indicated by grey areas. Job 7, with a resource demand of $u_{7,1} = 7$ for the only resource $r = 1$, is to be inserted between jobs 3, 4 and 5, 6. When job 3 is selected first as delivering job, the insertion $3 \to 7 \to 5$ does not need a priority rule because all resource units of the flow $3 \xrightarrow{4} 5$ must be redirected via job 7. When job 7 breaks the flows from task 4 afterwards, an additional rule is necessary since these flows provide one more resource unit than needed.

A simple rule based on increasing job numbers breaks flow $4 \xrightarrow{1} 5$ first and then $4 \xrightarrow{2} 6$ (Fig. 3a). The remaining flow $4 \xrightarrow{1} 6$ is not disturbed. The minTT rule breaks flows which lead to a minimal transfer time from job j to k primarily. Due to $\Delta_{7,6} = 2 < 3 = \Delta_{7,5}$, $4 \xrightarrow{3} 6$ is broken and $4 \xrightarrow{1} 5$ kept. The advantage of the minTT rule is that the gap between the inserted job j and the receiving job k will be relatively large. Thus, other jobs may be inserted between j and k in later scheduling steps. In case of the rule maxFlow large flows are broken first in order to break as few flows as possible when job j is inserted. For the example, this rule would also result in schedule (b).

5.3. Resource based version of the serial scheme

The serial scheme as well as our adaptation to RCMPSPTT described above rely on minimising the starting time for each job j selected to be scheduled (called *time based serial scheme).* This might lead to arranging resource flows in a disadvantageous manner. In order to overcome this problem, a resource based approach may be used which focuses primarily on resource flows.

The potential advantage of the resource based view is illustrated in Fig. 4. Job 6 is to be scheduled after task 4 and 5. The time based serial scheme (see Algorithm 3) schedules job 6 at the earliest time $t = 14$ by providing it with resources from job 4 which is the only one allowing this minimal starting time ($\mathrm{TJ}^1 = \{4\}$). In a next step, job 7 is selected by the job rule. Its earliest possible starting time is

Table 4
Transfer-to rules for breaking up flows

	Transfer-to rule	Extremum	Priority value π_k^{tr}
Time oriented	minTT	min	$\pi_k'' = \Delta_{jkr}$
Resource oriented	maxFlow	max	$\pi_k'' = \begin{cases} x_{ikr} & \text{if } F_j + \Delta_{jkr} \leqslant F_k - d_k \\ 0 & \text{else} \end{cases}$

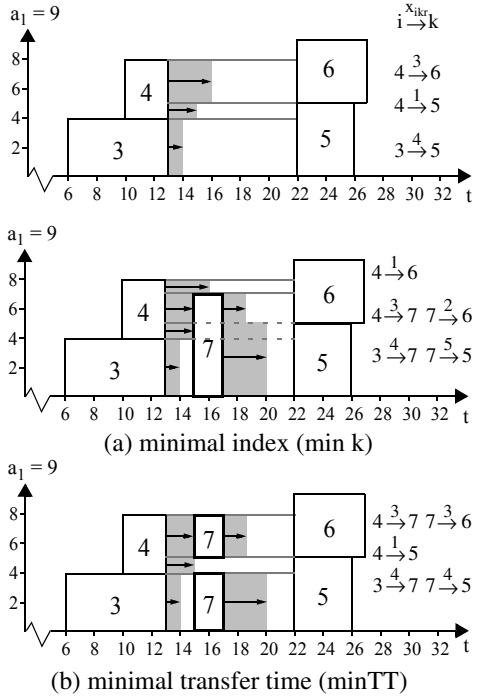

Fig. 3. Example for breaking up flows.

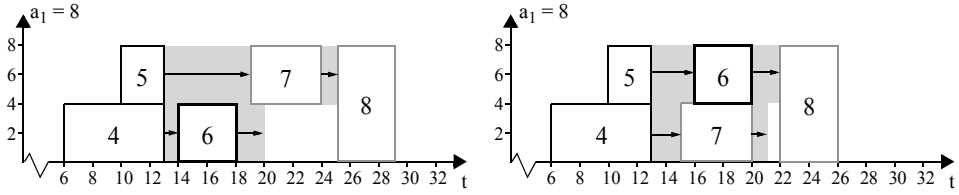

Fig. 4. Time-based versus resource-based serial scheme.

$t = 19$ after having received its resources from task 5. Finally, job 8 is scheduled at $t = 25$, which results in a project duration of 29 time units. Using the resource based serial scheme, we define a different set $ETJ^1 = \{4,5\}$ for $j = 6$, which contains all scheduled jobs. Thus, the transfer $5 \rightarrow 6$ may be chosen by an appropriate transfer-from rule instead of $4 \rightarrow 6$. Job 6 does not start the earliest but the project duration is, nevertheless, shortened by three time units since job 7 can be scheduled in $t = 15$ due to a shorter transfer time when it receives its resources from task 4 instead of 5. Using ETJ^r obviously allows a greater range of possible schedules, because there are more possibilities for resource flows than with TJ^r.

Generally, this *resource based serial scheme* works as follows: Starting with Algorithm 2 and choosing job j to be scheduled leads to a resource feasible finishing time ERF_j. Function *CalcResTrans*(j, ERF_j) (see Algorithm 3) must be adapted due to a different definition of the set of potentially delivering jobs. We refrain from presenting the new algorithm in detail but focus on describing the modification. For each job j, resource transfers from all scheduled jobs are considered possible, defining the extended set $ETJ^r = \{i \in S | u_{ir} > 0\}$ of all potentially delivering jobs i instead of TJ^r. From this set, jobs are selected and added to a subset I^r one after another using a transfer-from rule until the supplied quantity of all resources is just sufficient for job j. For this minimal subset I^r of the most promising potentially delivering jobs, the earliest feasible starting time of job j is computed using a transfer-to rule for sorting the flows to be broken as described in Algorithm 3. If not enough flows are breakable, a feasible schedule cannot be generated. For resources r that cause infeasibility, a further potentially delivering job i is added to I^r according to the transfer-from rule used. Finding a feasible (earliest) starting time for j is tried again. This process stops with a feasible starting time t and the corresponding end time for $ETF_j = t + d_j$ job j as in case of the time based version of the serial scheme (see Algorithm 3).

Within the resource based serial scheme, the same transfer-from rules as collected in Tables 2 and 3 can be applied. However, for the minGap rule the question of a trial time t which serves as reference point for gap computation arises: An initial trial time is given by $t := ERF_j - d_j$. For this trial time, $gap_{ijr} := t - (F_i + \Delta_{ijr})$ is calculated for each job in $ETJ^r = \{i \in \mathcal{S} \mid u_{ir} > 0\}$ for the first resource r to be examined. Now, as many jobs as necessary for satisfying the resource demand of job j for the considered resource r are chosen from the priority list and collected in I^r. If only jobs with positive gaps are chosen, trial time t can be achieved for resource r. If jobs with negative gaps are also necessary to satisfy the demand of j for r, trial time t is not a feasible scheduling time.

The modified priority rule minGAP, therefore, prefers jobs which will increase the intended scheduling time least (see Table 5). If sufficient resources can be provided by free resource stocks and feasibly broken flows, i.e., I^r offers enough resources, job j is (temporarily) scheduled as early as possible at time $\tau := \max\{F_i + \Delta_{ijr} | r \in R \wedge i \in I^r\}$ for the examined resource r. To schedule transfers for another resource

Table 5
Modified transfer-from rules for resource for transfers (resource based serial scheduling scheme)

Transfer-from rule	Extremum	Priority value $\pi_i^{r'}$ for $i \in \mathrm{ETJ}^{r'}$
minGAP	min	$\pi_i^{r'} = \begin{cases} \mathrm{gap}_{ijr} & \text{if } \mathrm{gap}_{ijr} \geqslant 0 \\ \lvert\mathrm{gap}_{ijr}\rvert + T & \text{else} \end{cases}$
min|GAP|	min	$\pi_i^{r'} = \lvert\mathrm{gap}_{ijr}\rvert$

type r' the trial time t is set to $t := \tau$ and the above process is repeated. If not enough resources of r' can be delivered up to t, this trial time needs to be increased. This might result in unscheduling the resource flows of already examined resource types. If for any of these resource types r flows from i to k were broken, it must be calculated whether the insertion of job j for resource type r is still feasible for the new trial scheduling time t. If not, they must be rescheduled assuming this new trial time. This process is repeated until for all resource types feasible resource transfers can be scheduled.

An additional transfer-from rule min|GAP| is introduced for the resource based serial scheme, which is an adaptation of minGap and considers only the absolute deviations from the scheduling time t. It is assumed that resources should be delivered as closely to trial time t (before and after) as possible. Hence, positive and negative deviations from t are rated equally (see Table 5).

Notice that there is no difference between the time and resource based version of the serial scheduling scheme if all transfer times are zero in case of RCPSP and RCMPSP because each job is scheduled the earliest by both approaches.

6. Computational experiments

In order to examine the performance of the different heuristic procedures developed for the new problem RCMPSPTT, we carry out a number of computational experiments. On the one hand, we consider the effect of transfer times within single-projects as that has not been done before. On the other hand, we examine the consequences of transfer times in a multi-project environment. The heuristic frameworks were coded in the programming language C and the tests have been performed on an Intel Pentium 4 processor with 3.2 GHz and 1 GB RAM.

6.1. Classification scheme for tested procedures

The heuristic frameworks presented in Sections 4 and 5 consist of different generation schemes and priority rules which can be combined to a considerable number of concrete heuristic procedures. In order to reference these procedures in a comfortable manner, we use a classification tuple $(a|b|c|d|e)$ with the following meanings:

a generation scheme; $a \in \{\mathrm{par}, \mathrm{tbser}, \mathrm{rbser}\}$ for parallel scheme, time based and resource based serial scheme
b job rule (see Table 1)
c transfer-from rule (see Tables 2, 3, 5)
d planning direction; $d \in \{\mathrm{fwd}, \mathrm{bwd}\}$ for forward and backward selection of delivering jobs (cf. Section 4.3)
e transfer-to rule (see Table 4); left empty for the parallel scheme

For example, (tbser | minLFT_SP|minTT|bwd| maxFlow) specifies a procedure using the time based serial scheduling scheme, the job rule minLFT_SP, the transfer-from rule minTT, backward selection and the transfer-to rule maxFlow.

6.2. Experiments for single-projects

At first, we evaluate the heuristic frameworks for single-projects with and without transfer times. This experiment allows conclusions for real single-projects and combined multi-projects within the single-project approach of multi-project management (cf. Section 1.1).

6.2.1. Generating test data

The experiment is based on the standard benchmark data set J60 of the ProGen library (Kolisch et al., 1995). To get a stable and reliable basis of comparison, all 480 project instances of the J60 set have been solved by Scatter PROGRESS (Klein and Scholl, 1999, 2000). 400 of these instances could be solved optimally within 3600 s. These 400 instances and the corresponding optimal solutions were used to build a data set containing transfer times. Transfer times have been introduced for each resource type such that optimality of the solution found by Scatter PROGRESS is retained. This is done in order to provide optimal solutions for RCMPSPTT such that the quality of our heuristic solutions can be evaluated in the best possible manner by comparing optimal and heuristic values.

Moreover, the generated transfer times comply with triangular inequalities for all (h,i,j)-combinations with $h,i,j \in J$ and $h \neq i \neq j$. This means that the time for a transfer from h to j cannot be reduced by sending the resource via i. Under these conditions, symmetric transfer times $\Delta_{ijr} = \Delta_{jir}$ are generated systematically and randomly for each resource type $r \in R$. In a first step, the optimal time schedule is complemented by a corresponding feasible resource flow. This is done by solving the resource flow problem contained in the mathematical program given the optimal finishing times F_j of all jobs (cf. Section 3). When a job i delivers a resource r to another job j in this resource flow, feasible transfer times are bounded from above by $F_j - d_j - F_i$. In order not to get a very tight and unrealistic problem, the maximal value allowed is, however, set to $\overline{\Delta}_{ijr} := \alpha \cdot (F_j - d_j - F_i)$ with $\alpha = 0.5$. For relations (i,j) without an actual resource flow, a maximal transfer time $\overline{\Delta}_{ijr}$ is sampled uniformly from an interval $[0, \max_{ijr}]$. The upper bound $\max_{ijr}$ depends on the time span between the first and the last usage of resource r in the optimal schedule as well as the number of jobs that require r during this period. It can be interpreted as the average time a job that requires resource r could use this resource in the optimal solution until r is set free ultimately. This value is used to limit transfer times to realistic durations which may be lower or higher than real job durations but will not be extraordinary high. Transfers from

the global source and to the global sink node are set to zero because resources are assumed to be already available at the respective jobs when the project starts and need not be transferred elsewhere after the project has been finished ($\overline{\Delta}_{s_0ir} = \overline{\Delta}_{ie_0r} = 0\ \forall i \in J', r \in R$). Furthermore, $\overline{\Delta}_{iir} = 0\ \forall i \in J', r \in R$.

Given these maximal transfer times for each potential relation (i,j), a linear program is formulated and solved which sets all transfer times as large as possible such that the triangular conditions and the upper limits are fulfilled. Due to the triangular conditions, in many cases smaller transfer times than $\overline{\Delta}_{ijr}$ must be chosen such that no more than 80% non-zero elements are contained in the transfer time matrices of the 400 modified J60 projects within the data set on average.

Different data sets have been generated varying the parameter $\alpha \in \{0.5, 1\}$ and the mean percentage of non-zero elements of the transfer time matrices. Yet, the results and findings are similar for all data sets. Therefore, we summarise representative results for the described data set with $\alpha = 0.5$ and 80% non-zero elements. The results of an additional small data set, where transfer times are generated randomly irrespective of optimality, are added in the supplementary document (sect. C).

6.2.2. Selected results

As a reference point, we apply the heuristic frameworks to the 400 selected instances of the original J60 data set (all transfer times are 0). In this case, both serial schemes are identical (tbser = rbser) and only job rules are required.

Table 6 summarises the mean relative deviations from optimum for selected job rules. The performance obtained is consistent with test results by Klein (2000). Of course, many other exact and heuristic procedures exist for the RCPSP, which have been proved to perform better than priority rules; for an overview see Kolisch and Hartmann (2006). However, as we developed a first priority rule based heuristic for the RCMPSPTT, we used priority rule based procedures for the case of zero transfer times for a fair comparison. The experiments show that the parallel scheme often generates better results than the serial one. The only exception is minLFT_SP which proves to be the best rule of the test in combination with the serial scheme. An average deviation from optimum of 2.85% has been reached. For the parallel scheme minSLK(dyn) outperformed other rules with a deviation of 3.49%. However, minLFT is not far behind with 3.87%. Concluding one can remark that critical path based rules like minLFT or minSLK generally turn out to be more efficient than resource based rules like maxRD.

When resource transfer times are considered, the results get significantly worse. At first, several transfer-from rules were evaluated. Here, the results only for some of the tests are presented, a large number of further tests confirm the obtained statements. Since minLFT proved to be the best job rule for the serial scheduling scheme and one of the best ones for the parallel scheme in absence of transfer times, it is selected as a basis of this very first analysis. In order to make a ceteris paribus comparison of different transfer-from rules, we choose backward selection and the transfer-to rule maxFlow. Table 7 summarises the results for mean relative deviations from optimum (omitted values indicate the corresponding rule being inappropriate for that scheme).

The results show huge differences for the resource based serial scheme. The mean deviation from optimum lies between 11.04% and 257.08% while the maximum deviation even reaches 601.85% for the maxTRS rule. The same is true for tests considering other combinations of job rules, transfer-to rules and planning directions. It becomes clear that for the resource based serial scheme only minGAP and minES are good transfer-from rules. The four resource based rules, however, show very poor performance as the extended set of potentially delivering jobs together with a resource oriented point of view leads to unnecessarily enlarged starting times.

The time based serial scheme avoids these large deviations and gets rather acceptable results for most of the rules and slightly better results even for minGAP and minES. Here, the different sets of potentially delivering jobs have only a minor effect due to preferring resource transfers that allow for an early starting of job j. Since minGAP is the rule which tries to avoid unproductive times of resources most

Table 6
Selected job rules applied to original J60 data set without transfer times (relative deviation from optimum in %)

Scheduling scheme	tbser			par		
Job rule	Avg	Min	Max	Avg	Min	Max
minSLK_SP	6.69	0	55.17	4.64	0	42.71
minSLK_SP (dyn)	6.92	0	44.83	**3.49**	0	**26.19**
minLFT_SP	**2.85**	0	**31.11**	3.87	0	28.99
maxRD_SP	11.14	0	51.72	8.11	0	45.33
FCFS	8.40	0	38.55	7.11	0	40.96
Random	13.15	0	60.94	8.71	0	52.38

Table 7
J60 test with (* |minLFT_SP| * |bwd| maxFlow)-combinations (relative deviation from optimum in %)

Scheduling scheme	rbser			tbser			par		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
minTT	141.65	0	375.00	13.91	0	93.20	**10.36**	0	**45.92**
minES	18.12	0	94.18	17.98	0	94.18	–	–	–
minGAP	**11.04**	0	**86.41**	**10.98**	0	**82.52**	10.47	0	56.48
min |GAP|	89.65	4.48	263.51	–	–	–	–	–	–
minRS	236.76	41.00	470.31	13.62	0	96.12	10.37	0	56.48
maxRS	241.59	21.21	583.33	15.19	0	93.20	10.40	0	54.63
minTRS	202.83	27.47	433.33	14.50	0	93.20	–	–	–
maxTRS	257.08	31.88	601.85	14.18	0	94.18	–	–	–
maxCV	126.40	0	322.03	14.59	0	93.20	10.42	0	51.61
Random	142.11	23.08	324.07	14.56	0	94.18	10.43	0	51.61

Table 8
J60 test with (* |*| minGAP |bwd| maxFlow)-combinations (relative deviation from optimum in %)

Scheduling scheme	rbser			tbser			par		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
minSLK_SP	32.56	0	111.63	32.23	0	124.00	**6.09**	0	49.07
minSLK_SP (dyn)	34.48	0	133.33	34.33	0	133.33	11.48	0	60.22
minLFT_SP	**11.04**	0	86.41	**10.98**	0	82.52	10.47	0	56.48
maxRD_SP	32.71	0	130.56	32.67	0	130.56	9.60	0	**46.94**
FCFS	13.50	0	**71.58**	13.63	0	**71.58**	13.77	0	59.14
Random	36.15	0	133.33	35.66	0	133.33	10.53	0	58.95

consequently, it performs best in both serial schemes. However, the parallel scheme turns out to be better whatever transfer-from rule is used. The differences between these rules are marginal. Even the random rule gets the same performance level. Obviously, the strict manner of using resources in each period as exhaustively as possible reduces the degree of freedom in making transfer decisions thereby reducing the potential of making wrong decisions.

As a first result, we can state that in case of single-projects with transfer times applying the parallel scheme is usually preferable to applying one of the serial schemes. Concerning the serial scheme, the time based version clearly outperforms the resource based one. In both cases, selecting a transfer-from rule is a sensitive decision.

In any case, the transfer-from rule minGAP is a good decision. Therefore, we fix this rule for a further test which examines job rules for all schemes. The results are presented in Table 8. It reveals that for both serial schemes only two rules are acceptable, namely minLFT_SP and FCFS. The latter has indeed a larger average deviation but a smaller maximal one. The performance of rbser is for most rules slightly worse than that of tbser, except for FCFS, where it is even better. Obviously, the chosen transfer-from rule minGAP makes the schemes almost identical as it prefers resource transfers that enable early starting times for the job j to be scheduled.

The overall best results are achieved by the parallel scheduling scheme again. The best job rule for this scheduling method is minSLK_SP with a mean deviation of only 6.09%. It must also be noticed that the resource based job rule maxRD_SP is second best rule for the parallel scheduling scheme, which is a completely different result to the evaluation without transfer times (RCPSP) where it was one of the worst. Moreover, the differences between all scheduling schemes diminish when FCFS is used as a job rule since it aims at scheduling jobs in order of increasing starting times even in the serial scheme.

In order to find the overall best scheme-rule combination, a further systematic test is performed using the job rule minSLK_SP and the parallel scheme by varying the transfer-from rules (c) and the planning direction (d) as summarised in Table 9. It turns out that only small differences exist. The best combination is (parallel |minSLK_SP| minGAP|fwd|$\varnothing$) with an average relative deviation of 5.81% closely followed by the combination (parallel |minSLK_SP|minTT|bwd|$\varnothing$). However, even the random selection of delivering jobs gives reasonable result.

To summarise, the presented results allow the following conclusions. Basically, the parallel scheme should be preferred. If a serial scheme should be applied the time based one is slightly preferable. In both cases, selecting an appropriate transfer-from rule is more essential than selecting the job rule. On the contrary, the job rule is more important for the parallel scheme.

Transfer-to rules and the planning direction have much less influence as further test series, which are not documented here, show. Perhaps, the following recommendation may be given: If the aim of project management is to keep resources in the project as long as possible, forward selection (fwd) should be applied together with the transfer-to rule minTT. If transferring as many resource units as possible when a resource must change projects anyway is the approach to project management, backward selection (bwd) together with the transfer-to rule maxFlow will be preferable.

6.3. Experiments in a multi-project environment

In the following, experiments concerning the performance of the heuristics within a multi-project environment are reported. Multi-project based priority rules and differing objective functions are tested. A corresponding data set is generated and selected results are given and interpreted.

6.3.1. Generating test data

The multi-project data set is based on the well-known Patterson data set (Patterson, 1984) as it contains different projects of different sizes which enables to construct realistic multi-projects. A total of 100 multi-project instances are generated with $|J| \in [93,204]$ and $|R| = 3$ by randomly choosing five Patterson instances, respectively, each forming a project within the multi-project, i.e., $|P| = 5$. The common capacity a_r of any resource $r \in R$ in the new multi-projects is sampled uniformly from $[\max\{\max\{u_{jr} | j \in J'\}, \min\{a_{pr} | p \in P\}\};$

Table 9
Results for combinations (par|minSLK_SP|*|*|ø)

Transfer-from rule (c)	Planning direction (d)	
	fwd	bwd
minTT	5.92	**5.82**
minGAP	**5.81**	6.09
minRS	5.99	5.99
maxRS	6.39	6.14
maxCV	6.15	6.11
Random	6.40	6.08

$\sum_{p=1}^{5} a_{pr} - \max\{a_{pr} \mid p \in P\}]$ with a_{pr} denoting the capacity of resource r in the original project p. To ensure existence of a feasible solution, the lower bound of the interval for resource type r is given by the maximal resource usage of any job within the multi-project or the minimal resource capacity of any original single-project whatever is greater. The upper bound ensures that the projects actually compete for the resources.

Transfer times are once again generated randomly depending on minimal and maximal job durations and considering triangular inequalities. In order to reflect realistic conditions in many multi-projects, it is assumed that non-zero times only occur for transfers between projects. Transfers from the sink and to the source are given zero times as well.

This data set is used to test the single- and the multi-project perspective based on the multi-project duration increase MPDI and the mean project delay MD, respectively. Optimal solutions are not known for this data set. Hence, we are only comparing different heuristic strategies according to the relevant objective function. An evaluation study of the heuristics on the basis of optimal solutions was presented in Section 6.2. Moreover, results of small multi-project instances, for which optimal values are known, are reported in supplementary document (sect. C).

6.3.2. Selected results

To examine the performance of the heuristic frameworks depending on job rules and scheduling schemes, we again fix the transfer-to rule to minGAP, the transfer-to rule to minFlow, and use the backwards selection. In order to judge the influence of transfer times, the same test is performed completely ignoring these times on the one hand and considering them on the other. The results are summarised in Table 10, the best rules in each group are highlighted.

The results for the transfer-less data set (columns 2–4) are largely consistent with those of Lova and Tormos (2001). For the objective MPDI of the single-project approach, the parallel scheme performs always better than the serial one. However, for the objective MD of the multi-project approach, it depends on the job rule which scheme is better. Moreover, for MPDI all rules developed for this very approach (lower part of Table 10) – apart from maxRD – perform better than their multi-project counterparts (upper part of Table 10). For MD, all rules specialised on the multi-project approach outperform the corresponding single-project rules.

The overall best combination for MPDI uses the parallel scheme together with the rule minSLK_SP(dyn), directly followed by minLFT_SP. The latter rule is the best one combined with the serial scheme. The rules maxTWK, maxRD_MP and minSASP, as rules for the multi-project approach, are among the worst job rules for this objective, whereas they turn out to be among the best ones for minimising MD.

When resource transfers are considered (columns 5–10), the parallel scheme is now generally superior to the serial ones irrespective of the objective confirming our results in Section 6.2. However, job rule performance is quite different to the transfer-less test. Likewise the results of Section 6.2.2, for both objectives, it is confirmed that the time based serial scheme is (slightly) better than the resource based one.

Moreover, it can be observed that for minimising MPDI for both serial schemes single-project time oriented rules are once again better than their multi-project counterparts. However, when it comes to resource oriented rules, the approach, for which the rules are designed, is irrelevant. MaxRD_MP and maxTWK, which perform poorly when no transfer times occurred, produce best results now. The other way around, minLFT_SP, FCFS and minLFT_MP, which are pretty good rules before, turn out to be some of the worst now. For the parallel scheme, minLFT_MP, which is a multi-project rule, appears to be the best rule. Thus, the preference of single-project rules for minimising MPDI with the parallel scheme cannot be supported when transfer times occur. Yet, minSLK_SP and minLFT_SP are only slightly worse. Consequently, independent of the considered approach the rules are intended for, one can say that minLFT_MP, minLFT_SP and minSLK_SP show a good performance again. Yet, minSLK_SP(dyn) which is best in the transfer-less test is one of the worst rules in presence of transfer times.

The conclusions for minimising MD are quite different. For both serial schemes and the parallel scheme, SASP and maxTWK (only for serial schemes), which already obtain good results without resource transfers outperform other rules again. The SASP rule is once again the overall best rule.

An additional experiment reported in supplementary document (sect. D) indicates again that the remaining components c, d and e of the heuristic framework, in particular the transfer-from rules, considerably influence the performance of the serial schemes only. Further extensive tests not reported here indicate that the transfer-from rule minGAP is the best transfer-from rule in (almost) all cases and that the influence of the transfer-to rule and the planning direction is negligible. Thus, the results presented in Table 10 indeed show the best available performance of the tested combinations of job rule and scheduling scheme.

Table 10
Results for (* |*| minGAP |bwd| maxFlow)-combinations

Job rule	No transfer times considered				Transfer times considered					
	MPDI (%)		MD (days)		MPDI (%)			MD (days)		
	Scheme tbser	par	Scheme tbser	par	Scheme rbser	tbser	par	Scheme rbser	tbser	par
minSASP	141.63	126.89	**31.41**	**30.05**	210.98	208.33	150.67	**47.81**	**47.36**	**39.78**
minSLK_MP	117.06	115.91	41.60	41.23	197.91	193.76	143.52	71.84	69.98	45.06
minLFT_MP	105.12	101.95	39.32	38.41	198.63	196.72	**135.04**	76.01	75.28	41.21
maxTWK	117.83	112.73	35.82	34.92	178.92	176.74	150.74	54.45	54.15	46.76
maxRD_MP	110.98	106.89	38.83	38.63	**177.30**	**174.46**	137.45	60.55	59.85	45.30
minSLK_SP	105.23	103.71	44.78	43.69	183.74	180.31	135.73	74.30	73.42	46.73
minSLK_SP(dyn)	110.41	**89.67**	41.49	49.19	182.40	180.33	147.21	65.50	64.75	52.33
minLFT_SP	**93.77**	90.40	50.35	48.47	193.79	192.28	138.77	96.29	95.51	52.43
maxRD_SP	111.50	110.91	45.98	43.18	191.51	186.73	139.39	84.23	81.77	44.20
FCFS	103.84	102.19	44.70	43.37	206.82	205.31	145.64	88.46	87.59	44.99
Random	126.65	116.39	43.79	42.61	199.06	195.50	144.67	62.33	60.58	45.93

To summarise the most important findings, transfer times should not be neglected when scheduling projects, because they have an important influence on objective values. They must be considered when scheduling a multi-project but the usage of special transfer priority rules is not vital for the parallel scheme, which is in marked contrast to the results for the serial scheme. Moreover, when transfer times occur, resource based information and gain in importance for priority calculation, especially in the serial scheduling scheme.

7. Conclusions and future research

In this paper, aspects of resource transfers in multi-project scheduling are considered for the first time. We present a mathematical model for the resulting new multi-project scheduling problem with resource transfers. Furthermore, a heuristic framework based on the well-known priority rule based heuristics for the RCPSP and RCMPSP has been developed and tested. The results illustrate that resource transfers should not be neglected since they increase the multi-project duration or the mean project delay whichever is the aim to be minimised by project managers. Yet, it has been shown that applying sensible transfer scheduling rules is of great importance for the serial scheduling scheme whereas for the parallel scheme emphasis must be put on selecting appropriate job rules. Especially resource based rules gain in importance. However, the obtained results are still not satisfying. Deviations from optimum of about 5% on average are still relatively high. It should be aimed at improving the results by developing new and improved solution procedures such as meta heuristics, e.g. genetic algorithms. The presented heuristic solution procedure can build an essential basis for this.

Finally, other aspects should be taken into account. For this paper, it is assumed that precedence constraints are defined only within the projects. This assumption holds for multi-project scheduling as researched here. However, the developed models and procedures can be transferred to programme scheduling as well, which requires that precedence constraints between projects become relevant. This can easily be integrated. Moreover, we only analyse time oriented objective functions in this paper. However, in practice cost aspects play an important role as well. Resource transfers increase cost measures. Thus, they should be integrated into the models and solution procedures in next research steps. In a wider context additional considerations should be remarked. The underlying static environment assumption often does not hold for multi-projects in practice. In a real multi-project environment, dynamic aspects as stochastically arriving projects must be considered as well. Furthermore, the changing of project and task portfolios is frequently caused by change requests. These requests are an even more important reason for the dynamic nature of projects. Change requests express a formalised wish to change the characteristics of the project content which the parties to a contract already agreed on. The aspect of projects arriving stochastically to the system are already considered in the literature on dynamic environments. But the way they are dealing with this problem is still not satisfying for applicable project scheduling in the real world. Finally, they are not considering the changing of the structure of projects that are already in the portfolio. In this case of changing project requisitions, the concept of resources being transferred between projects becomes even more important.

Last but not least, the absence of uncertainty in multi-project scheduling, e.g. in job durations or resource availability, up to now is a point of criticism that should not be neglected. In single-project scheduling this aspect is already part of research. In multi-project scheduling it has not been considered yet. However, one can assume that deterministic scheduling deals with the uncertainty problem by working with expected job durations or resource requirements as one-point estimators. Incorporating all aspects of uncertainty into models and solution procedures will certainly lead to very complex systems, especially since the problem becomes more complex in the context of multiple interdependent projects.

Despite all the mentioned open questions, a first step to resolve the criticism of the state-of-the-art research is being made in this paper by considering the sequence- and resource type-dependent resource transfers as an additional aspect to the RCMPSP in a static environment.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2008.07.036.

References

Anavi-Isakow, S., Golany, B., 2003. Managing multiple-project environments through constant work-in-progress. International Journal of Project Management 21 (1), 9–18.
Artigue, C., Michelon, P., Reusser, S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149 (2), 249–267.
Ash, R., Smith-Daniels, D.E., 1999. The effects of learning, forgetting, and relearning on decision rule performance in multiproject scheduling. Decision Sciences 30 (1), 47–82.
Bock, D.B., Patterson, J.H., 1990. A comparison of due date setting, resource assignment, and job preemption heuristics for the multiproject scheduling problem. Decision Sciences 21 (2), 387–402.
Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112 (1), 3–41.
Demeulemeester, E.L., 1992. Optimal algorithms for various classes of multiple resource-constrained project scheduling problems. Unpublished Dissertation, Katholieke Universiteit Leuven, Belgium.
Demeulemeester, E.L., Herroelen, W.S., 2002. Project Scheduling: A Research Handbook. Kluwer Academic Publishers, Boston.
Dodin, B., Elimam, A.A., 1997. Audit scheduling with overlapping activities and sequence-dependent setup costs. European Journal of Operational Research 97 (1), 22–33.
Dumond, J., 1992. In a multi-resource environment, how much is enough. International Journal of Production Research 30 (2), 395–410.
Dumond, E.J., Dumond, J., 1993. An examination of resourcing policies for the multi-resource problem. International Journal of Operations & Production Management 13 (5), 54–76.
Dumond, J., Mabert, V.A., 1988. Evaluating project scheduling and due date assignment procedures: An experimental analysis. Management Science 34 (1), 101–118.
Fendley, L.G., 1968. Towards the development of a complete multiproject scheduling system. Journal of Industrial Engineering 19 (10), 505–515.
Ireland, L.R., 2002. Managing multiple projects in the 21st century. In: Pennypacker, J.S., Dye, L.D. (Eds.), Managing multiple projects. Dekker, New York, pp. 21–34.
Kaplan, L., 1991. Resource-constrained project scheduling with setup times. Unpublished Working Paper, Department of Management Science, University of Tennessee, Knoxville, USA.
Klein, R., 2000. Scheduling of Resource-constrained Projects. Kluwer Academic, Boston.
Klein, R., Scholl, A., 1999. Scattered branch and bound. An adaptive search strategy applied to resource-constrained project scheduling. Central European Journal of Operations Research 7 (3), 177–201.
Klein, R., Scholl, A., 2000. PROGRESS: Optimally solving the generalized resource-constrained project scheduling problem. Mathematical Methods of Operations Research 52 (3), 467–488.

Kolisch, R., 1995. Project Scheduling Under Resource Constraints. Physica, Heidelberg.
Kolisch, R., Hartmann, S., 1999. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling: Recent Models, Algorithms, and Applications. Kluwer, Boston, pp. 147–178.
Kolisch, R., Hartmann, S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174 (1), 23–37.
Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 29 (3), 249–272.
Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41 (10), 1693–1704.
Krüger, D., Scholl, A., 2008. Managing and modelling general resource transfers in (multi-)project scheduling, OR Spectrum, doi:10.1007/s00291-008-0144-5.
Kurtulus, I.S., Davis, E.W., 1982. Multi-project scheduling: Categorization of heuristic rules performance. Management Science 28 (2), 161–172.
Kurtulus, I.S., Narula, S.C., 1985. Multi-project scheduling: Analysis of project performance. IIE Transactions 17 (1), 58–66.
Lawrence, S.R., Morton, T.E., 1993. Resource constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. European Journal of Operational Research 64 (2), 168–187.
Lova, A., Tormos, P., 2001. Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. Annals of Operations Research 102, 263–286.
Lova, A., Maroto, C., Tormos, P., 2000. A multicriteria heuristic method to improve resource allocation in multi-project scheduling. European Journal of Operational Research 127 (2), 408–424.
Lycett, M., Rassau, A., Danson, J., 2004. Programme management: A critical review. International Journal of Project Management 22 (4), 289–299.
Mika, M., Waligóra, G., Weglarz, J., 2006. Modelling setup times in project scheduling. In: Jósefowska, J., Weglarz, J. (Eds.), Perspectives in Modern Project Scheduling. Springer, New York, pp. 131–163.
Mohanty, R.P., Siddiq, M.K., 1989. Multiple projects-multiple resources-constrained scheduling: Some studies. International Journal of Production Research 27 (2), 261–280.
Neumann, 2003. Project scheduling with changeover times – modelling and applications. In: Proceedings of the International Conference on Industrial Engineering and Production Management, vol. 1, Porto/Portugal, May 26–28, pp. 30–36.
Neumann, K., Schwindt, C., Zimmermann, J., 2003. Project Scheduling With Time Windows and Scarce Resources. Springer, Berlin.
Patterson, J.H., 1973. Alternate methods of project scheduling with limited resources. Naval Research Logistics Quarterly 20 (4), 767–783.
Patterson, J.H., 1984. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. Management Science 30 (7), 854–867.
Payne, J.H., 1995. Management of multiple simultaneous projects: A state-of-the-art review. International Journal of Project Management 13 (3), 163–168.
Pritsker, L.J., Watters, A.A.B., Wolfe, P.M., 1969. Multiproject scheduling with limited resources: A zero–one programming approach. Management Science 16 (1), 93–108.
Trautman, 2001. Anlagenbelegungsplanung in der Prozessindustrie. Gabler, Wiesbaden.
Vanhoucke, M., 2008. Setup times and fast tracking in resource-constrained project scheduling. Computers and Industrial Engineering 54 (4), 1062–1070.
Vercellis, C., 1994. Constrained multi-project planning problems: A Lagrangian decomposition approach. European Journal of Operational Research 78 (2), 267–275.
Wiley, V.D., Deckro, R.F., Jackson, J.A., 1998. Optimization analysis for design and planning multi-project programs. European Journal of Operational Research 107 (2), 492–506.
Yang, K.K., Sum, C.C., 1993. A comparison of resource allocation and activity scheduling rules in a dynamic multi-project environment. Journal of Operations Management 11 (2), 207–218.
Yang, K.K., Sum, C.C., 1997. An evaluation of due date, resource allocation, project release, and activity scheduling rules in a multiproject environment. European Journal of Operational Research 103 (1), 139–154.