OR Spectrum (2016) 38:305–334
DOI 10.1007/s00291-015-0402-2

REGULAR ARTICLE

A flow-based tabu search algorithm for the RCPSP with transfer times

Jens Poppenborg[1] · **Sigrid Knust**[2]

Received: 25 September 2014 / Accepted: 25 April 2015 / Published online: 20 May 2015

Abstract In this paper, we propose a tabu search algorithm for the resource-constrained project scheduling problem with transfer times. Solutions are represented by resource flows extending the disjunctive graph model for shop scheduling problems. Neighborhoods are defined by parallel and serial modifications rerouting or reversing flow on certain arcs. This approach is evaluated from a theoretical and experimental point of view. Besides studying the connectivity of different neighborhoods, computational results are presented for benchmark instances with and without transfer times.

Keywords RCPSP · Transfer times · Tabu search · Resource flow

1 Introduction

In the classical resource-constrained project scheduling problem (RCPSP), activities have to be scheduled under precedence and resource constraints such that a given objective function is minimized. For the execution of these activities, renewable resources with limited capacities are available. This problem has been extensively studied in scientific literature since the early 1960s and several books as well as surveys have been published dealing with the RCPSP and its extensions (cf. Demeulemeester and Herroelen 2002; Brucker and Knust 2011, or Hartmann and Briskorn 2010).

✉ Sigrid Knust
sigrid@informatik.uni-osnabrueck.de

Jens Poppenborg
jens.poppenborg@alumni.tu-clausthal.de

1 Institute of Applied Stochastics and Operations Research, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany

2 Institute of Computer Science, University of Osnabrück, 49069 Osnabrück, Germany

To solve the classical RCPSP, various exact as well as heuristic solution approaches have been proposed. While branch-and-bound algorithms can only solve smaller instances to optimality, heuristics obtain quite good results. For an evaluation of different heuristic approaches, see Hartmann and Kolisch (2000) as well as Kolisch and Hartmann (2006).

In this paper, we study an extension of the RCPSP where additionally transfer times (also called setup times or changeover times) are taken into account. Such times occur in practice when a resource is physically moved from one location to another (e.g., when a crane has to be transported between construction sites) or when a resource has to be adjusted between different activities (e.g., when a machine has to be cleaned between producing different products). In contrast to the classical RCPSP, only few papers deal with this extension. Mika et al. (2006) introduced a classification of setup times in the context of the RCPSP. This classification was extended by Krüger (2009) in her Ph.D. thesis as well as in a paper by Krüger and Scholl (2010) to include higher-tier resource transfers (i.e., transfers of resources that are required to support the transfer of other resources).

Apart from this, for example, Vanhoucke (2008) considered the RCPSP with preemption and sequence-independent setup times where a setup becomes necessary whenever a preempted activity is resumed. Schwindt and Trautmann (2000) dealt with batch scheduling problems in process industries and modeled these as a RCPSP with sequence- and resource-dependent changeover times as well as various other constraints. Similarly, Schwindt and Trautmann (2003) presented a model based on the RCPSP with sequence- and resource-dependent changeover times for a real-world production planning problem in the aluminum industry while Neumann et al. (2003) tackled the RCPSP with sequence- and resource-dependent changeover times and time windows. Here, both Schwindt and Trautmann (2003) as well as Neumann et al. (2003) used branch-and-bound algorithms to solve the respective problems.

The RCPSP with sequence- and resource-dependent transfer times has also been considered in the context of multi-project scheduling by Krüger and Scholl (2009) and Krüger (2009). Besides a mixed-integer linear programming formulation of this problem, Krüger (2009) introduced adaptions of the serial as well as the parallel schedule generation scheme (cf. Kolisch 1996) that select resource transfers based on priority rules. Additionally, she presented a genetic algorithm that represents solutions as activity lists to solve this problem.

While in most approaches for the RCPSP solutions are represented by activity lists or generated by schedule generation schemes, in this paper we represent solutions as resource flows. Such a representation has been suggested by Artigues et al. (2003) who proposed a tabu search algorithm where resource flows are modified by removing and reinserting an activity in an optimal position in relation to the current resource flow. We have opted to use this solution representation because it represents from which activities resource units are transferred to other activities. In contrast to this, if activity lists or schedule generation schemes are used, resource transfers are not immediately represented, but have to be selected separately (e.g., based on a priority rule). As a consequence, such an approach has only limited influence on how resource transfers are chosen (limited to, for example, selecting one out of several available priority

rules). Furthermore, we will show in Sect. 3 that with such an approach an optimal schedule may be excluded from the search space.

Based on the resource flow representation, we introduce neighborhoods for the RCPSP with transfer times. These neighborhoods are defined by parallel and serial modifications as suggested by Fortemps and Hapke (1997). However, in Fortemps and Hapke (1997), the ideas are only sketched and no computational results are reported. One purpose of our paper is to study these neighborhoods in more detail from a theoretical and practical point of view and to evaluate their efficiency in some computational experiments.

An advantage of the resource flow representation and the neighborhoods is that they can be extended to the RCPSP with first- and second-tier resource transfers, where the transfer of some resources has to be supported by other resources. For this more complex problem, no solution approaches have been developed so far. Based on the definitions and results reported in this paper, the necessary extensions are introduced by Poppenborg (2014).

The remainder of this paper is structured as follows. In Sect. 2, we give a formal description of the RCPSP with transfer times. Then, in Sect. 3, the solution representation based on resource flows is introduced and it is shown that the set of schedules represented by resource flows always contains an optimal solution. In Sect. 4, we define neighborhoods based on this solution representation. Additionally, we consider the connectivity of these neighborhoods. Finally, a tabu search algorithm is proposed in Sect. 5, while computational results are reported in Sect. 6. We conclude this paper with some remarks in Sect. 7.

2 Problem formulation

In this section, we describe the studied problem more formally. In the RCPSP, a set $\mathcal{R}$ consisting of r renewable resources $k = 1, \ldots, r$ with limited capacities R_k as well as a set V consisting of n activities $i = 1, \ldots, n$ with processing times $p_i > 0$ and resource requirements $r_{ik} \geq 0$ for $k \in \mathcal{R}$ are given. Furthermore, a dummy source activity 0 as well as a dummy sink activity $n + 1$ with processing times $p_0 = p_{n+1} = 0$ and resource requirements $r_{0k} = r_{n+1,k} = R_k$ for all $k \in \mathcal{R}$ are introduced. It is assumed that all resource units are initially located at the dummy source activity 0 and have to be collected by the dummy sink activity $n + 1$ at the end of the project. In the following, the set $V_{\text{all}} = \{0, 1, \ldots, n, n + 1\}$ contains all real activities from the set V as well as the two dummy activities, while the two sets $V_0 = \{0, 1, \ldots, n\}$ and $V_* = \{1, \ldots, n, n + 1\}$ contain subsets of these activities.

Furthermore, precedence constraints $(i, j) \in A$ may exist between pairs of activities $i, j \in V_{\text{all}}$, $i \neq j$ requiring that activity j can only start after activity i has been completed. We assume that the set A contains precedence constraints $0 \rightarrow j$ for all activities $j \in V$ without any predecessor activity (i.e., the dummy activity 0 is the first activity to be processed in any feasible schedule) and precedences $i \rightarrow n + 1$ for all activities $i \in V$ without any successor (i.e., $n + 1$ is the last activity to be processed). In the studied extension of the RCPSP, additionally sequence- and resource-dependent transfer times $\Delta_{ijk} \geq 0$ are given for all $i, j \in V_{\text{all}}$ and $k \in \mathcal{R}$ denoting the amount

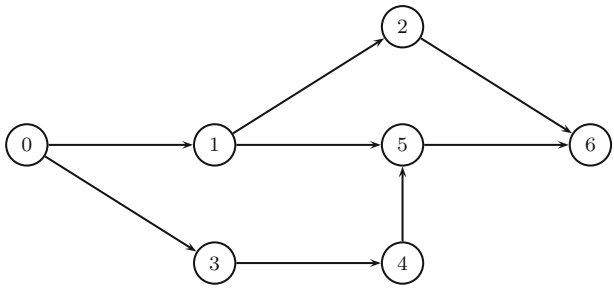


Fig. 1 Precedence constraints for the problem in Example 1

of time required to transfer resource k from activity i to activity j. The special case $\Delta_{ijk} = 0$ for all $i, j \in V_{\text{all}}$ and all $k \in \mathcal{R}$ corresponds to the classical RCPSP. It should be noted that during the transfer of a resource unit from one activity to another activity, this resource unit cannot be used to process other activities. Usually, it is assumed that the triangle inequality $\Delta_{ihk} + \Delta_{hjk} \geq \Delta_{ijk}$ for all $i, j \in V_{\text{all}}$ and all $k \in \mathcal{R}$ holds. All parameters are assumed to be integer.

The problem consists in determining starting times S_j for all activities $j \in V_*$ (the dummy activity 0 is assumed to start at time $S_0 = 0$) respecting the given precedence constraints (i.e., activity j can only start after all its predecessors $i \in V_0$ with $(i, j) \in A$ have been completed) and the resource constraints (i.e., all resource requirements r_{jk} of resources $k \in \mathcal{R}$ have to be satisfied simultaneously for p_j time periods to process activity j while no more than R_k units of resource k can be used in any time period). Preemption of the activities is not allowed. Additionally, if some units of resource $k \in \mathcal{R}$ are transferred from activity $i \in V_0$ to activity j, the transfer time Δ_{ijk} has to be observed between the completion time $C_i = S_i + p_i$ of activity i and the starting time S_j of activity j, i.e., $S_j \geq C_i + \Delta_{ijk}$ has to hold. The objective is to minimize the makespan $C_{\max}$, i.e., the time required to complete all activities, which is determined by the completion time C_{n+1} of activity $n + 1$.

Example 1 We consider a small project consisting of $n = 5$ activities with precedence constraints $1 \to 2$, $1 \to 5$, $3 \to 4$, and $4 \to 5$ (cf. Fig. 1). Furthermore, $r = 2$ renewable resources with capacities $R_1 = 4$ and $R_2 = 3$ are available. The processing times p_i and resource requirements r_{ik} of the activities as well as the transfer times Δ_{ijk} are given in Table 1. It is assumed that $\Delta_{0jk} = \Delta_{i,n+1,k} = 0$ holds for all transfer times involving the dummy activities. A feasible schedule for this project with makespan $C_{\max} = 15$ is displayed in Fig. 2.

3 Solution representation

In this section, we discuss how solutions for the RCPSP with transfer times can be represented. As already explained in Sect. 1, it cannot be guaranteed that always an optimal schedule can be found by an algorithm that selects resource transfers based on fixed priority rules. This is the case, for example, if a schedule generation scheme is used to generate schedules (e.g., based on an activity list). Such generation schemes

Table 1 Data for the problem in Example 1

i	p_i	r_{i1}	r_{i2}		
1	2	2	2		
2	3	1	2		
3	1	1	0		
4	1	2	1		
5	2	0	3		
$i \backslash j$	1	2	3	4	5
Transfer times Δ_{ij1}					
1	0	2	2	3	2
2	2	0	3	2	2
3	2	3	0	2	2
4	3	2	2	0	2
5	2	2	2	2	0
Transfer times Δ_{ij2}					
1	0	2	1	2	3
2	2	0	2	3	2
3	1	2	0	2	2
4	2	3	2	0	2
5	3	2	2	2	0

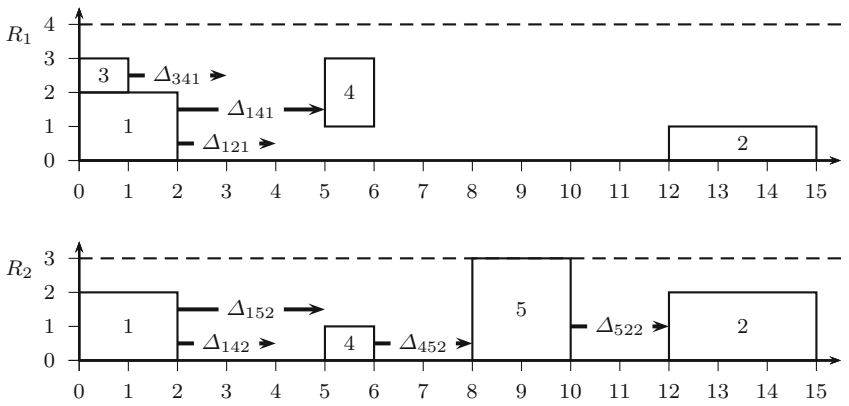


Fig. 2 A feasible schedule for the project in Example 1

usually start each activity as early as possible and hence produce semi-active (or even active) schedules where no activity can be shifted locally (or globally) to the left without violating feasibility (cf. Sprecher et al. 1995). As already observed for the parallel machine problem with precedence constraints and setup times (cf. Hurink and

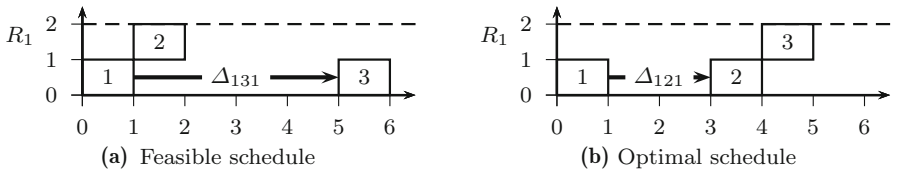


Fig. 3 Schedules for Example 2

Knust 2001), an optimal solution might be excluded from the search space if activities are scheduled as early as possible:

Example 2 We consider a project consisting of $n = 3$ activities as well as $r = 1$ renewable resource with a capacity of $R_1 = 2$. The processing times of activities $i = 1, 2, 3$ are given by $p_i = 1$ while the resource requirements are $r_{i1} = 1$. Furthermore, we assume transfer times $\Delta_{121} = \Delta_{211} = 2$, $\Delta_{131} = \Delta_{311} = 4$, $\Delta_{231} = \Delta_{321} = 4$ and $\Delta_{0j1} = \Delta_{i,n+1,1} = 0$ for the transfer times involving the dummy activities. Finally, the precedence constraints $1 \rightarrow 2$ and $2 \rightarrow 3$ are given between the activities.

For this project, the only precedence-feasible activity list is given by $(1, 2, 3)$. Now, a schedule generation scheme (i.e., either the serial or the parallel scheme) tries to schedule these activities as early as possible (cf. the adapted schedule generation schemes described by Krüger and Scholl 2009). As a result of this, both schedule generation schemes would generate the schedule displayed in Fig. 3a in which activity 2 is scheduled to start at time $S_2 = 1$. While this schedule has a makespan of $C_{\max} = 6$, the unique optimal schedule displayed in Fig. 3b in which activity 2 is scheduled to start at time $S_2 = 3$ has a makespan of $C_{\max} = 5$. □

To avoid that an optimal schedule may be excluded from the search space, in our algorithm we represent solutions as resource flows. A disadvantage of this representation is a larger solution space. However, with this representation, it can be ensured that the solution space always contains an optimal schedule.

The resource flow representation for the RCPSP was first introduced in Fortemps and Hapke (1997) as well as Artigues and Roubellat (2000) as an extension of the disjunctive graph model (cf. Roy and Sussmann 1964) which is frequently used as a solution representation in shop scheduling problems. While a disjunctive graph represents the sequence in which disjunctive machines are used to process operations in a shop scheduling problem, this representation is extended for the RCPSP to also denote the amounts of resources $k \in \mathcal{R}$ transferred from an activity $i \in V_0$ to another activity $j \in V_*$.

In the following, $f_{ijk} \geq 0$ denotes the amount of resource $k \in \mathcal{R}$ transferred from activity $i \in V_0$ to activity $j \in V_*$. All resource transfers f_{ijk} of a resource $k \in \mathcal{R}$ between activities $i \in V_0$ and $j \in V_*$ are then collected in a resource flow $\mathcal{F}_k$, while a set of resource flows $\mathcal{F}_k$ for all resources $k \in \mathcal{R}$ is denoted by $\mathcal{F}$. A feasible resource flow $\mathcal{F}$ is defined as a flow that satisfies the following conditions:

$$\sum_{j \in V_*} f_{0jk} = \sum_{j \in V_0} f_{j,n+1,k} = R_k \quad (k \in \mathcal{R}) \tag{1}$$

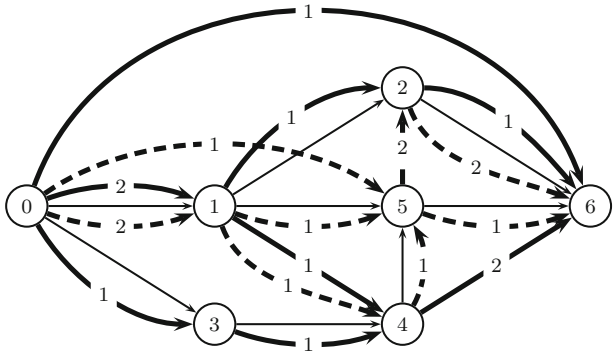


Fig. 4 AON-flow network corresponding to the feasible schedule from Fig. 2

$$\sum_{j \in V_*} f_{ijk} = \sum_{j \in V_0} f_{jik} = r_{ik} \qquad (i \in V,\ k \in \mathcal{R}) \tag{2}$$

First of all, all resource units are initially located at the dummy source activity 0 and have to be collected by the dummy sink activity $n + 1$ at the end of the project [cf. (1)]. Furthermore, for each activity $i \in V$ and each resource $k \in \mathcal{R}$, the amount of all incoming resource units as well as the amount of all outgoing resource units have to be equal to the resource requirement r_{ik} [flow conservation constraints (2)]. Finally, the resource flow has to be acyclic, i.e., no activity sends (directly or indirectly) resource units to itself and the flow must observe the given precedence constraints (i.e., for $(i, j) \in A$ activity j may not send (directly or indirectly) resource units to activity i).

A feasible resource flow $\mathcal{F}$ can be represented by a graph where each activity $i \in V_{\text{all}}$ is modeled as a node and each resource transfer $f_{ijk} > 0$ of a resource $k \in \mathcal{R}$ from activity $i \in V_0$ to activity $j \in V_*$ is represented by a directed arc $(i, j)_k$ from node i to node j. It should be noted that the resulting graph for a given resource flow $\mathcal{F}$ is a multigraph if more than one resource $k \in \mathcal{R}$ is transferred from activity i to activity j, i.e., there may be a total of up to r arcs $(i, j)_k$ connecting two activities i and j.

Similar to the disjunctive graph model for shop scheduling problems, an AON-flow network (cf. Artigues et al. 2003) incorporates the activity-on-node network representing the precedence constraints $(i, j) \in A$ as well as the graph representing a resource flow $\mathcal{F}$. In this graph, each arc (i, j) representing a precedence constraint $(i, j) \in A$ is weighted with the processing time p_i of activity i. To incorporate transfer times, we weight all flow arcs $(i, j)_k$ satisfying $f_{ijk} > 0$ with the value $p_i + \Delta_{ijk}$.

Example 3 We again consider the project from Example 1. A corresponding AON-flow network is shown in Fig. 4. Thin arcs represent precedence constraints, solid thick arcs represent resource transfers of resource 1, and dashed thick arcs represent resource transfers of resource 2. The arcs are weighted with the f_{ijk}-values. The earliest start schedule corresponding to this AON-flow network is the same as the original schedule from Fig. 2.

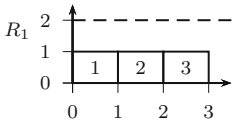


Fig. 5 An earliest start schedule which is not semi-active

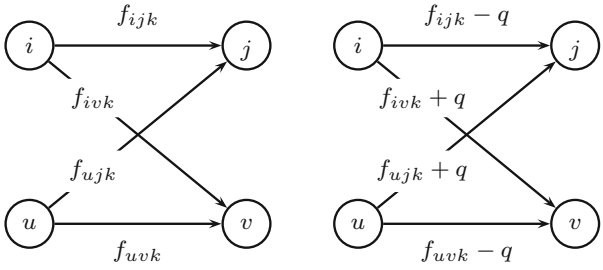
In the same way as for the classical RCPSP (cf. Artigues 2010), for each AON-flow network with transfer times incorporated, it is possible to generate an earliest start schedule S (i.e., a schedule in which all activities start as early as possible with respect to the given precedence constraints $(i, j) \in A$ and the resource transfers f_{ijk} from the given resource flow $\mathcal{F}$). This can be done by calculating the lengths l_{ij} of the longest paths between all activities $i, j \in V_{\text{all}}$ in the AON-flow network (e.g., using the Floyd–Warshall algorithm, cf. Floyd 1962). Here, the length of a path is defined as the sum of the weights of its arcs where only the arc with the largest weight between two activities is considered. Then, in a corresponding earliest start schedule each activity $i \in V_{\text{all}}$ is started at time $S_i = l_{0i}$ (corresponding to the length of a longest path from 0 to i). The makespan $C_{\max}$ is given by the length $l_{0,n+1}$. A corresponding longest path is a critical path.

Conversely, the problem of generating a resource flow $\mathcal{F}$ corresponding to a feasible schedule S can be modeled as a feasible flow problem (cf. Poppenborg 2014). Based on these results, similar to the situation of the classical RCPSP (cf. Artigues 2010), it can be shown that the set of schedules represented by resource flows always contains an optimal schedule (cf. Poppenborg 2014 for a proof of this theorem):

Theorem 1 *For the RCPSP with transfer times, the set of schedules represented by resource flows always contains an optimal schedule.*

Note that the set of schedules represented by resource flows contains the set of all semi-active (and active) schedules as a proper subset. This is due to the fact that each semi-active schedule can be represented as an earliest start schedule of a corresponding flow (cf. Poppenborg 2014). On the other hand, there are schedules represented by resource flows which are not semi-active. This can be seen from the following example with $n = 3$ activities and a single renewable resource with capacity $R_1 = 2$. The processing times of the activities are $p_1 = p_2 = p_3 = 1$, the resource requirements are $r_{11} = r_{21} = r_{31} = 1$. There are no precedence constraints, all transfer times Δ_{ij1} between the activities are assumed to be zero. A feasible resource flow for this project is given by $f_{011} = f_{041} = f_{121} = f_{231} = f_{341} = 1$, the corresponding earliest start schedule is shown in Fig. 5. Obviously, this schedule is not semi-active, since either activity 2 or activity 3 can be shifted to the left. Hence, the set of schedules represented by resource flows is a proper superset of the set of semi-active schedules.

Based on the definitions in Neumann et al. (2003), the schedules represented by resource flows can be classified as quasiactive schedules. In particular, each resource flow $\mathcal{F}$ defines a strict order such that $S_i + p_i + \Delta_{ijk} \leq S_j$ holds for each arc $(i, j)_k$ in the corresponding graph and the activities in a corresponding earliest start schedule start as early as possible with respect to these arcs.



(a) Before modification **(b)** After modification

Fig. 6 Resource flow before and after a reroute move has been applied

4 Neighborhoods

Based on the resource flow solution representation for the RCPSP with transfer times described in Sect. 3, we now introduce neighborhoods. These neighborhoods are defined by parallel and serial modifications as suggested in Fortemps and Hapke (1997) extending neighborhoods developed for the job-shop problem with only disjunctive resources. However, in Fortemps and Hapke (1997), the ideas are only sketched and no further results (neither experimental not theoretical) are reported. To the best of our knowledge, also no additional literature has been published dealing with these neighborhoods any further.

First of all, neighborhood $\mathcal{N}_{\text{reroute}}$ is introduced based on the parallel modification described in Fortemps and Hapke (1997). For a modification in this neighborhood, two arcs $(i, j)_k$ and $(u, v)_k$ between activities $i, u \in V_0$ and $j, v \in V_*$ representing resource transfers $f_{ijk} > 0$ and $f_{uvk} > 0$ of a resource $k \in \mathcal{R}$ are selected in a resource flow $\mathcal{F}$ such that in the AON-flow network no directed path exists from activity j to activity u or from activity v to activity i. Then, an amount of $q \in \{1, \ldots, \min\{f_{ijk}, f_{uvk}\}\}$ units of resource k is rerouted from activity i to activity v as well as from activity u to activity j. This results in a resource flow $\mathcal{F}'$ with the modified resource transfers $f'_{ijk} = f_{ijk} - q$, $f'_{uvk} = f_{uvk} - q$, $f'_{ivk} = f_{ivk} + q$, and $f'_{vjk} = f_{vjk} + q$ as displayed in Fig. 6.

Next, neighborhood $\mathcal{N}_{\text{reverse}}$ is introduced based on the serial modification described in Fortemps and Hapke (1997). For a modification in this neighborhood, two activities $i, j \in V$ with $f_{ijk} > 0$ for at least one resource $k \in \mathcal{R}$ are selected in a resource flow $\mathcal{F}$ such that $(i, j) \notin A$ and no other directed path exists from activity i to activity j via other activities $h \in V$ in the AON-flow network consisting of precedence or flow arcs. Additionally, sets U_k of activities $u \in V_0$ are selected for each resource $k \in \mathcal{R}$ with $f_{ijk} > 0$ based on a priority rule (e.g., based on the numbering of the activities) such that an arc $(u, i)_k$ with $f_{uik} > 0$ exists between each activity $u \in U_k$ as well as activity i and

$$\sum_{u \in U_k} f_{uik} \geq f_{ijk}$$

holds for the amount of resource k transferred between these activities. These sets are chosen such that no activity $u \in U_k$ can be removed from the set U_k without violating

this inequality (i.e., the inequality does not hold for any proper subset of the set U_k). Similarly, sets V_k of activities $v \in V_*$ are selected for each resource $k \in \mathcal{R}$ with $f_{ijk} > 0$ based on a priority rule such that an arc $(j, v)_k$ with $f_{jvk} > 0$ exists between activity j as well as each activity $v \in V_k$ and

$$\sum_{v \in V_k} f_{jvk} \geq f_{ijk}$$

holds for the amount of resource k transferred between these activities. Again, these sets V_k are chosen such that no activity $v \in V_k$ can be removed from the set V_k without violating this inequality.

It should be noted that in Fortemps and Hapke (1997) it is not described how the sets U_k and V_k are selected and how the resource units are redirected. Instead, only a trivial case is treated where a single activity $u \in V_0$ with $f_{uik} \geq f_{ijk}$ as well as a single activity $v \in V_*$ with $f_{jvk} \geq f_{ijk}$ can be selected for each resource $k \in \mathcal{R}$ with $f_{ijk} > 0$. In the following, we assume that for resource $k \in \mathcal{R}$ the subset U_k contains a total of a_k activities $u_1, \ldots, u_{a_k}$ with $f_{ijk} > 0$, while the subset V_k contains a total of b_k activities $v_1, \ldots, v_{b_k}$.

Now, a reverse move (cf. Fig. 7) first reverses the direction of all arcs $(i, j)_k$ for resources $k \in \mathcal{R}$ with $f_{ijk} > 0$ between activities i and j such that an amount of $f'_{jik} = f_{ijk}$ units of resource $k \in \mathcal{R}$ is transferred from activity j to activity i in the resulting resource flow $\mathcal{F}'$. Additionally, for each resource $k \in \mathcal{R}$ with $f_{ijk} > 0$, the resource transfers for all activities $u \in U_k$ as well as for all activities $v \in V_k$ are adapted. Here, all $f_{u_\lambda ik}$ units of resource k are redirected from activities $u_\lambda \in U_k$ with $\lambda = 1, \ldots, a_{k-1}$ to activity j. Additionally,

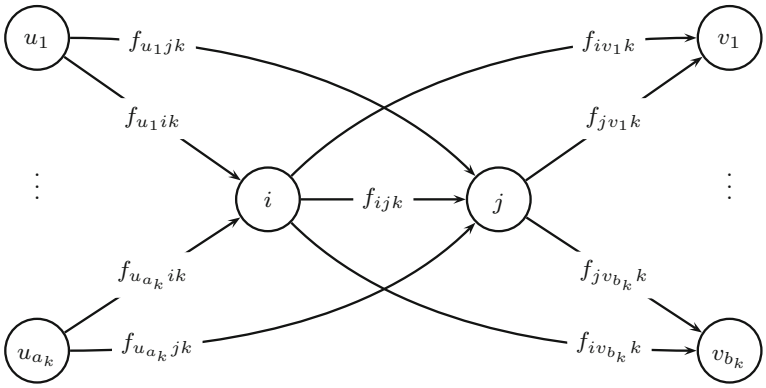
$$q_{a_k} = f_{ijk} - \sum_{\lambda=1}^{a_{k-1}} f_{u_\lambda ik}$$

units of resource k are redirected from the remaining activity u_{a_k} to activity j. Similarly, all $f_{jv_\nu k}$ units of resource k are redirected from activity i to activities $v_\nu \in V_k$ with $\nu = 1, \ldots, b_{k-1}$ while

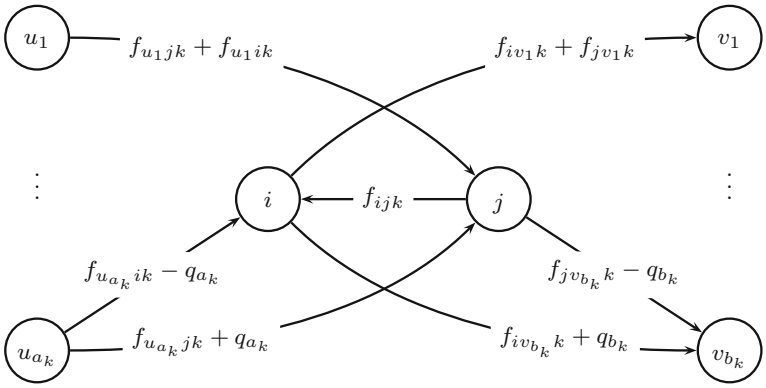
$$q_{b_k} = f_{ijk} - \sum_{\nu=1}^{b_{k-1}} f_{jv_\nu k}$$

units of resource k are redirected from activity i to the remaining activity v_{b_k}. These modifications ensure that flow conservation is maintained in the resulting resource flow $\mathcal{F}'$. It should be noted that the last activities u_{a_k} and v_{b_k} to be considered are also the last activities that have been selected by the corresponding priority rule.

As described above, some conditions have to apply for the selected activities for a reroute or reverse modification to ensure that the resulting resource flow is acyclic. Here, ensuring that no directed path exists from activity $j \in V_*$ to activity $u \in V_0$ and from activity $v \in V_*$ to activity $i \in V_0$ for a modification in the neighborhood $\mathcal{N}_{\text{reroute}}$ can be done in $\mathcal{O}(1)$ time based on the matrix containing the longest path



(a) Before modification



(b) After modification

Fig. 7 Resource flow before and after a reverse move has been applied

lengths between the activities. Similarly, it can be ensured in $\mathcal{O}(1)$ time that activity $i \in V$ is no direct or indirect predecessor of activity $j \in V$ for a modification in the neighborhood $\mathcal{N}_{\text{reverse}}$. However, on the other hand, for a reverse modification, additional computational time is required to ensure that no other directed path from activity $i \in V$ to activity $j \in V$ exists in the AON-flow network. This can, for example, be determined by a depth-first search in $\mathcal{O}(n^2)$ time if an adjacency matrix is used to represent which activities are connected by arcs.

Example 4 We again consider the project introduced in Example 1 as well as the resource flow $\mathcal{F}$ for this project as it is displayed in Fig. 4. Now, we use a reroute modification in the neighborhood $\mathcal{N}_{\text{reroute}}$ to reroute 1 unit of resource 1 on the arcs $(0, 6)_1$ and $(1, 4)_1$. The AON-flow network incorporating the resulting resource flow $\mathcal{F}'$ is displayed in Fig. 8, while the corresponding earliest start schedule with makespan $C'_{\max} = 14$ is shown in Fig. 9.

Next, we use a reverse modification in the neighborhood $\mathcal{N}_{\text{reverse}}$ to reverse the only arc $(5, 2)_2$ between activities 5 and 2 in resource flow $\mathcal{F}'$. Then, the incoming resource

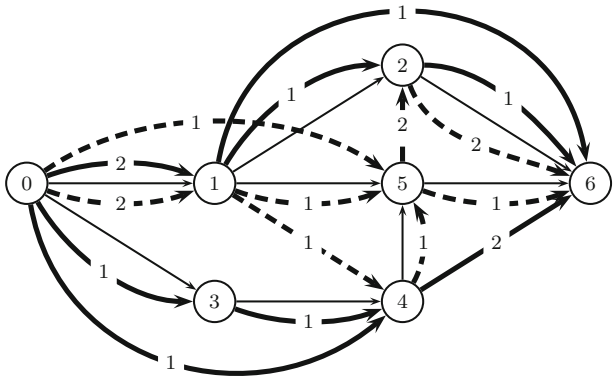


Fig. 8 AON-flow network based on the resource flow $\mathcal{F}'$ after applying a reroute modification to resource flow $\mathcal{F}$ from Fig. 4

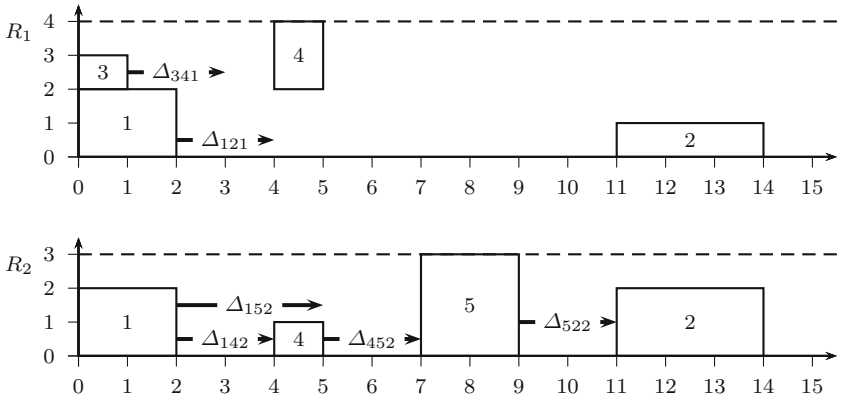


Fig. 9 Earliest start schedule corresponding to the AON-flow network from Fig. 8

transfers $f_{052} = 1$, $f_{152} = 1$, and $f_{452} = 1$ can be redirected to activity 2, while only the outgoing resource transfer $f_{262} = 2$ can be redirected to originate from activity 5. For example, using a priority rule that selects resource transfers based on increasing activity numbers, 1 unit of resource 2 from activity 0 as well as 1 unit of resource 2 from activity 1 are redirected to activity 2, and 2 units of resource 2 are redirected from activity 5 to activity 6 in the resulting resource flow $\mathcal{F}''$. The AON-flow network visualizing this resource flow is shown in Fig. 10, while a corresponding earliest start schedule with makespan $C''_{\max} = 11$ is displayed in Fig. 11. □

It should be noted that the neighborhood $\mathcal{N}_{\text{reverse}}$ closely resembles the neighborhood based on disjunctive graphs introduced in van Laarhoven et al. (1992) for the job-shop scheduling problem. While modifications in this neighborhood change the sequence in which activities are processed by the same resource units, the neighborhood $\mathcal{N}_{\text{reroute}}$ changes the resource plan for a selected resource $k \in \mathcal{R}$.

Now, we consider the size of the neighborhoods $\mathcal{N}_{\text{reroute}}$ and $\mathcal{N}_{\text{reverse}}$. Here, because an acyclic directed graph representing a resource flow $\mathcal{F}_k$ for a resource $k \in \mathcal{R}$ consists

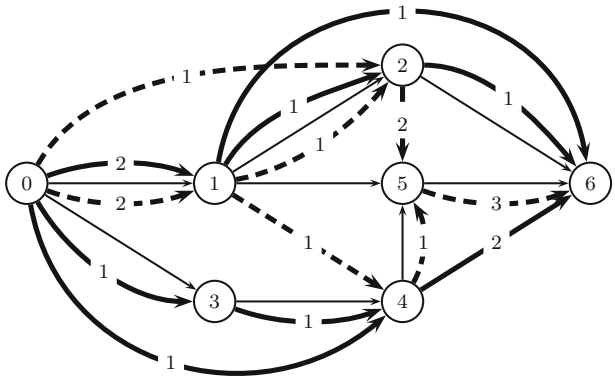


Fig. 10 AON-flow network based on the resource flow $\mathcal{F}''$ after applying a reverse modification to resource flow $\mathcal{F}'$ from Fig. 8

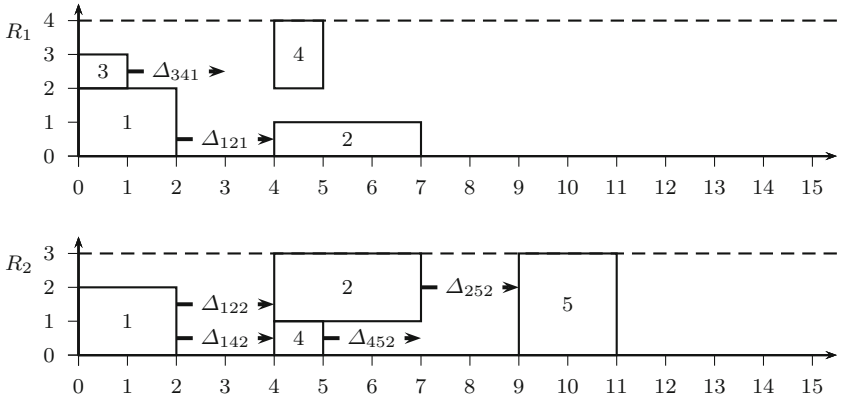


Fig. 11 Earliest start schedule corresponding to the AON-flow network from Fig. 10

of at most $\frac{n(n-1)}{2}$ directed arcs, the size of the neighborhood $\mathcal{N}_{\text{reverse}}$ is bounded by $\mathcal{O}(n^2)$. It should be noted that the size of this neighborhood does not depend on the number of arcs connecting two activities $i, j \in V$ because all arcs between these activities are reversed simultaneously by a modification in the neighborhood $\mathcal{N}_{\text{reverse}}$.

For the neighborhood $\mathcal{N}_{\text{reroute}}$, two arbitrary arcs $(i, j)_k$ and $(u, v)_k$ for a resource $k \in \mathcal{R}$ as well as an amount $q \in \{1, \ldots, \min\{f_{ijk}, f_{uvk}\}\}$ of resource units to be rerouted have to be selected. Thus, the size of this neighborhood is bounded by $\mathcal{O}(rn^4 R_{\max})$ where $R_{\max} = \max_{k \in \mathcal{R}}\{R_k\}$ is the maximal amount of available resource units of a resource $k \in \mathcal{R}$. For this neighborhood, if multiple arcs for resources $k \in \mathcal{R}$ connect two activities, each of these arcs has to be considered separately.

Due to the size of these neighborhoods (in particular, the pseudo-polynomial neighborhood $\mathcal{N}_{\text{reroute}}$ may be quite large), we now reduce the neighborhoods. First of all, similar to Fortemps and Hapke (1997), we introduce the neighborhood $\mathcal{N}^{\text{ca}}_{\text{reverse}}$ in which reverse moves are limited to critical activities $i, j \in V$ with $f_{ijk} > 0$ for at least one resource $k \in \mathcal{R}$. As before, to reverse the arcs $(i, j)_k$ for all resources $k \in \mathcal{R}$ with $f_{ijk} > 0$ between these activities, i must not be a direct or indirect predecessor

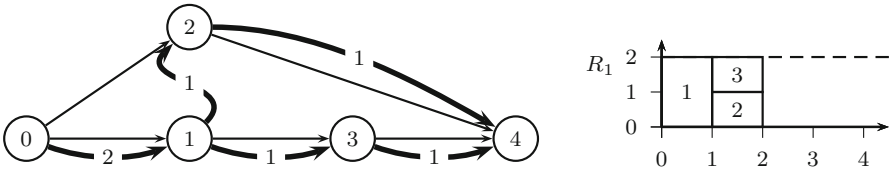
of j according to the precedence constraints. For the classical RCPSP (i.e., the situation without transfer times) similar to the job-shop problem, it can be shown that if consecutive activities on a critical path are chosen, reversing these arcs always results in a feasible (acyclic) resource flow (see Poppenborg 2014). However, for the RCPSP with transfer times, this property is lost (see Example 6). The size of the reduced neighborhood $\mathcal{N}^{\text{ca}}_{\text{reverse}}$ is limited to $\mathcal{O}(n)$.

Next, we introduce a neighborhood $\mathcal{N}^{\text{max,ca}}_{\text{reroute}}$ in which one critical arc $(i, j)_k$ as well as an arbitrary arc $(u, v)_k$ representing resource transfers $f_{ijk} > 0$ and $f_{uvk} > 0$ are selected instead of two arbitrary arcs. Additionally, similar to Fortemps and Hapke (1997), always the maximal amount of $q = \min\{f_{ijk}, f_{uvk}\}$ units of resource k is rerouted. The size of this neighborhood is then limited to $\mathcal{O}(rn^3)$. This neighborhood can be further reduced to a neighborhood $\mathcal{N}^{\text{max,ca},\Delta}_{\text{reroute}}$ of size $\mathcal{O}(n^3)$ by only considering one arc $(i, j)_k$ with the largest transfer time Δ_{ijk} for each critical arc (i, j) (in case of a tie, one of the possible arcs is randomly selected).

In the following, we deal with the question whether the defined neighborhoods are connected (i.e., every solution can be reached from any other solution by a finite number of modifications in the neighborhood) or at least opt-connected (i.e., from each solution, an optimal solution can be reached). We show that neither the neighborhood $\mathcal{N}_{\text{reroute}}$ nor the neighborhood $\mathcal{N}_{\text{reverse}}$ alone is opt-connected (and hence also not connected) already for the classical RCPSP without transfer times.

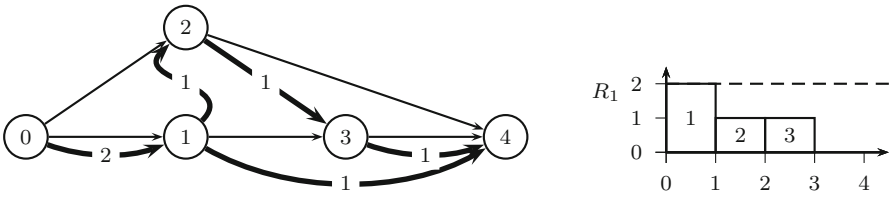
Example 5 We consider a project consisting of $n = 3$ real activities as well as $r = 1$ renewable resource with a capacity of $R_1 = 2$. The processing times of the activities are $p_i = 1$ for $i = 1, 2, 3$, while the resource requirements are $r_{11} = 2$ and $r_{21} = r_{31} = 1$. All transfer times Δ_{ij1} are assumed to be zero. Finally, only the precedence constraint $1 \to 3$ exists. The AON-flow network incorporating the unique optimal resource flow $\mathcal{F}^*$ for this project as well as the corresponding earliest start schedule with makespan $C^*_{\max} = 2$ are displayed in Fig. 12.

Now, we consider the feasible resource flow $\mathcal{F}$ shown in Fig. 13. If only the neighborhood $\mathcal{N}_{\text{reverse}}$ is available, it is not possible to improve the makespan of the corresponding schedule by reversing all arcs $(i, j)_1$ between two activities $i, j \in V$ because this only changes the sequence in which the activities are processed but not the resource plan. As a result of this, it is not possible to transform this resource flow $\mathcal{F}$ into the unique optimal resource flow $\mathcal{F}^*$ using only the neighborhood $\mathcal{N}_{\text{reverse}}$, i.e., the neighborhood $\mathcal{N}_{\text{reverse}}$ is not opt-connected. Instead, to transform this resource flow into the optimal resource flow $\mathcal{F}^*$, the arcs $(2, 3)_1$ and $(1, 4)_1$ would have to be modified based on the neighborhood $\mathcal{N}_{\text{reroute}}$.



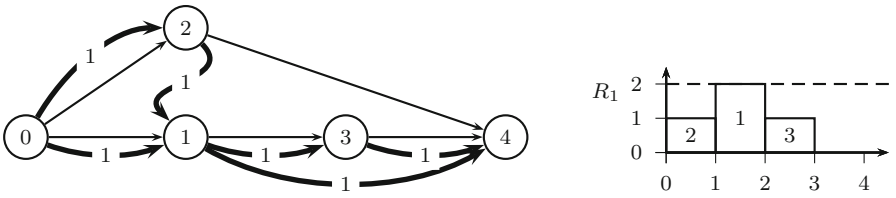
(a) AON-flow network **(b)** Schedule

Fig. 12 Optimal resource flow $\mathcal{F}^*$ and the corresponding earliest start schedule for Example 5



(a) AON-flow network **(b)** Schedule

Fig. 13 Resource flow $\mathcal{F}$ and the corresponding earliest start schedule for Example 5



(a) AON-flow network **(b)** Schedule

Fig. 14 Resource flow $\mathcal{F}'$ and the corresponding earliest start schedule for Example 5

Finally, we consider another feasible resource flow $\mathcal{F}'$ shown in Fig. 14. In this case, if only the neighborhood $\mathcal{N}_{\text{reroute}}$ is available, it is not possible to improve the makespan of the corresponding schedule by only rerouting arcs. In particular, it is not possible to select an arc from the left side of the graph (i.e., between activities 0, 1, and 2) as well as one arc from the right side of the graph (i.e., between activities 1, 3, and 4) for a modification because these arcs are always connected by a path via activity 1. For this reason, it is not possible to transform this resource flow into the optimal resource flow $\mathcal{F}^*$ using only the neighborhood $\mathcal{N}_{\text{reroute}}$, i.e., the neighborhood $\mathcal{N}_{\text{reroute}}$ is not opt-connected. Instead, to transform this resource flow into the optimal resource flow $\mathcal{F}^*$, it is necessary to reverse the critical arc $(2, 1)_1$ between activities 2 and 1 based on the neighborhood $\mathcal{N}_{\text{reverse}}$. Using a priority rule that selects activities according to increasing numbers, this results in resource flow $\mathcal{F}$ which can then be transformed into the optimal resource flow $\mathcal{F}^*$ as described above. □

As visualized in this example, neither the neighborhood $\mathcal{N}_{\text{reroute}}$ nor the neighborhood $\mathcal{N}_{\text{reverse}}$ alone is opt-connected. We now consider the larger neighborhood $\mathcal{N}_1 = \mathcal{N}_{\text{reroute}} \cup \mathcal{N}_{\text{reverse}}$. As shown in the appendix, this neighborhood is even connected:

Theorem 2 *For the RCPSP with transfer times, the neighborhood $\mathcal{N}_1$ is connected.*

Next, we consider the connectivity of the reduced neighborhood $\mathcal{N}_2 = \mathcal{N}_{\text{reroute}}^{\max,\text{ca},\Delta} \cup \mathcal{N}_{\text{reverse}}^{\text{ca}}$. Here, it can be shown that this neighborhood is not connected for the RCPSP with transfer times:

Example 6 We consider a project consisting of $r = 3$ renewable resources with capacities $R_1 = R_2 = R_3 = 1$ as well as $n = 4$ activities with the precedence

Table 2 Data for the problem in Example 6

i	p_i	r_{i1}	r_{i2}	r_{i3}
1	1	1	1	1
2	5	0	0	1
3	1	1	1	0
4	1	0	1	0
$i \backslash j$	1	2	3	4
Transfer times Δ_{ij1}				
1	0	6	6	6
2	6	0	6	6
3	6	6	0	6
4	6	6	6	0
Transfer times Δ_{ij2}				
1	0	2	2	2
2	2	0	2	2
3	2	2	0	2
4	2	2	2	0

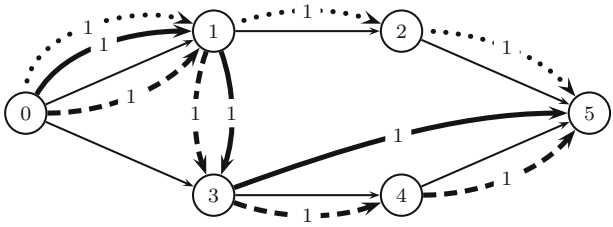


Fig. 15 Unique optimal resource flow $\mathcal{F}^*$

constraints $1 \to 2$ and $3 \to 4$. The processing times p_i and resource requirements r_{ik} of the activities as well as the transfer times Δ_{ijk} for resources $k = 1, 2$ are given in Table 2. It is assumed that $\Delta_{0jk} = \Delta_{i,n+1,k} = 0$ holds for all transfer times involving the dummy activities and $\Delta_{ij3} = 0$ holds for all transfer times involving resource 3.

The unique optimal resource flow for this project is displayed in Fig. 15 where thick arcs represent transfers of resource 1, dashed arcs represent transfers of resource 2, and dotted arcs represent transfers of resource 3. The corresponding earliest start schedule with makespan $C^*_{\max} = 11$ is shown in Fig. 16.

Now, we consider the feasible resource flow $\mathcal{F}$ from Fig. 17 where thick arcs graph represent transfers of resource 1, dashed arcs represent transfers of resource 2, and dotted arcs represent transfers of resource 3. The corresponding earliest start schedule with makespan $C_{\max} = 13$ is displayed in Fig. 18. The unique critical path of this schedule is $0 \to 3 \to 1 \to 2 \to 5$. This resource flow cannot be transformed

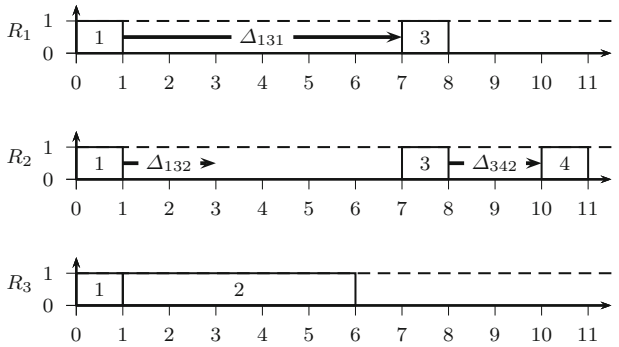


Fig. 16 Earliest start schedule corresponding to the optimal resource flow $\mathcal{F}^*$ from Fig. 15

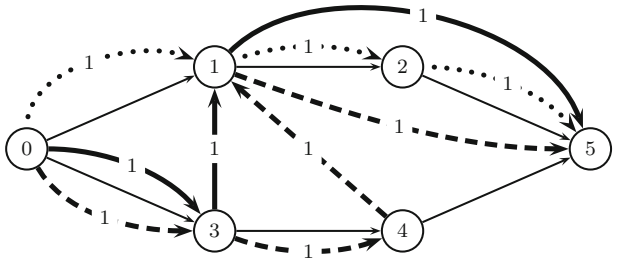


Fig. 17 Resource flow $\mathcal{F}$

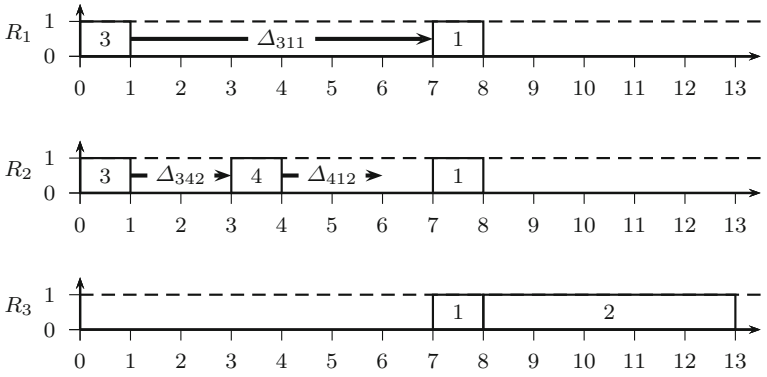
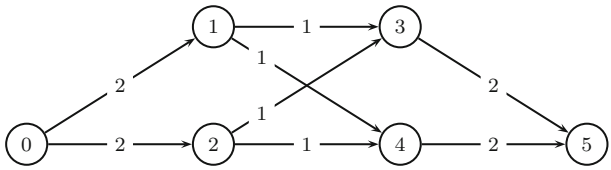
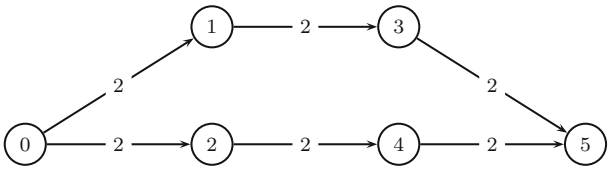


Fig. 18 Earliest start schedule corresponding to flow $\mathcal{F}$ from Fig. 17

into resource flow $\mathcal{F}^*$ for the following reasons. First of all, no modification in the neighborhood $\mathcal{N}_{\text{reroute}}^{\max,\text{ca},\Delta}$ (or $\mathcal{N}_{\text{reroute}}$) can be used because all resources have unit capacity (i.e., no resource units can be rerouted). Furthermore, on the critical path, only the order of activities 3 and 1 is not predetermined by precedence constraints. The arc $(3, 1)_1$ between these activities cannot be reversed based on a modification in the neighborhood $\mathcal{N}_{\text{reverse}}^{\text{ca}}$, however, because another directed path from activity 3 to



(a) Flow $\mathcal{F}$



(b) Flow $\mathcal{F}'$

Fig. 19 Feasible resource flows $\mathcal{F}, \mathcal{F}'$ for the project from Example 7

activity 1 via activity 4 exists (i.e., reversing the arc $(3, 1)_1$ would result in a cyclic and hence infeasible resource flow). Thus, the neighborhood $\mathcal{N}_2 = \mathcal{N}_{\text{reroute}}^{\max,\text{ca},\Delta} \cup \mathcal{N}_{\text{reverse}}^{\text{ca}}$ is not opt-connected (and hence also not connected). □

It should be noted that this result only holds for the RCPSP with transfer times. For the classical RCPSP without transfer times, the question whether the neighborhood $\mathcal{N}_2$ is opt-connected remains open.

Finally, we consider the neighborhood $\mathcal{N}_3 = \mathcal{N}_{\text{reroute}}^{\max,\text{ca},\Delta} \cup \mathcal{N}_{\text{reverse}}$. This neighborhood is neither connected for the RCPSP with transfer times nor for the classical RCPSP without transfer times. In particular, already the neighborhood $\mathcal{N}_4 = \mathcal{N}_{\text{reroute}}^{\max} \cup \mathcal{N}_{\text{reverse}}$ that restricts reroute moves to the maximal amount of $q = \min\{f_{ijk}, f_{uvk}\}$ resource units prevents the resulting neighborhood from being connected for the classical RCPSP and hence also for the RCPSP with transfer times. This can be seen by the following example.

Example 7 We consider a project consisting of $n = 4$ activities as well as $r = 1$ renewable resource with a capacity of $R_1 = 4$. The processing times of the activities are $p_i = 1$ for $i = 1, \ldots, 4$ while the resource requirements are $r_{i1} = 2$ for all activities $i = 1, \ldots, 4$. There are no precedence constraints between the real activities, all transfer times Δ_{ij1} are assumed to be zero. Two feasible resource flows $\mathcal{F}, \mathcal{F}'$ are displayed in Fig. 19.

Here, because modifications in both neighborhoods $\mathcal{N}_{\text{reroure}}^{\max}$ and $\mathcal{N}_{\text{reverse}}$ always redirect the maximal amount of $q = 2$ units of resource 1 in resource flow $\mathcal{F}'$ as well as in any resulting resource flow, it is impossible to transform resource flow $\mathcal{F}'$ into resource flow $\mathcal{F}$. □

This example only shows that the neighborhood $\mathcal{N}_3$ is not connected. It remains open whether the neighborhood is opt-connected or not.

5 A tabu search algorithm

To evaluate the solution representation as well as the neighborhoods described above, we implemented a tabu search algorithm in which solutions are represented by resource flows. An initial solution for a problem instance is generated by a parallel schedule generation scheme with forward–backward improvement as it has been described in Krüger and Scholl (2009). For this, the activity priority rule LFT is used that selects activities according to non-decreasing latest finish times. Furthermore, the transfer priority rule GAP is used that selects resource transfers according to non-decreasing differences between the earliest arrival time of the resource units at the activity and the starting time of the activity. It should be noted that the transfer priority rule is only used if more resource units than necessary can be transferred to the activity until its starting time.

In each iteration of the tabu search, a solution is selected from the neighborhood $\mathcal{N}_2 = \mathcal{N}_{\text{reroute}}^{\max,\text{ca},\Delta} \cup \mathcal{N}_{\text{reverse}}^{\text{ca}}$ as it has been introduced above. Here, to choose sets U_k for a reverse modification, the priority rule ES is used that selects activities according to non-decreasing earliest arrival times of the resource units at the receiving activity. To choose sets V_k, the priority rule TT is used that selects activities according to non-decreasing transfer times between the activities. Ties are always broken by the smallest index rule. To accelerate the search process, only non-tabu modifications are evaluated which implies that also no aspiration criterion is used. Furthermore, if a solution with a better objective function value than the current solution is found, this solution is immediately accepted. Otherwise, the best non-tabu solution from the neighborhood is chosen.

After a solution has been chosen, the tabu lists are updated based on the selected modification. Here, two tabu lists TL_{add} and TL_{drop} are used as described by Glover and Laguna (1998). If a reroute move has been applied to reroute resource units of resource $k \in \mathcal{R}$ on two selected arcs between activities $i \in V_0$ and $j \in V_*$ as well as between $u \in V_0$ and $v \in V_*$, the tabu list TL_{add} stores the triples (i, j, k) and (u, v, k) (indicating that less units of resource k are transferred from activity i to activity v as well as from u to j), while the tabu list TL_{drop} stores the triples (i, v, k) and (u, j, k) (indicating that more units of resource k are transferred from activity i to activity j as well as from u to v). Similarly, if a reverse move has been applied to reverse all arcs between two activities $i, j \in V$, the tabu list TL_{add} stores all triples (i, j, k) corresponding to resources $k \in \mathcal{R}$ for which at least one arc has been reversed while the tabu list TL_{drop} stores all triples (j, i, k).

A reroute move for two selected arcs of a resource $k' \in \mathcal{R}$ between activities $i' \in V_0$ and $j' \in V_*$ as well as between $u' \in V_0$ and $v' \in V_*$ is tabu if the triple (i', u', k') or (u', j', k') is contained in the tabu list TL_{add} or if the triple (i', j', k') or (u', v', k') is contained in the tabu list TL_{drop}. Similarly, a reverse move between two selected activities $i', j' \in V$ is tabu if for all resources $k' \in \mathcal{R}$ for which at least one arc exists in the resource flow at least one triple (j', i', k') is contained in the tabu list TL_{add} or if at least one triple (i', j', k') is contained in the tabu list TL_{drop}.

Both tabu lists TL_{add} and TL_{drop} are able to avoid cycling (i.e., the revisiting of a solution that has recently been visited) for the neighborhoods considered in this paper. In particular, the tabu list TL_{drop} prevents the tabu search from removing specific arcs

from a resource flow that have recently been inserted, while the tabu list TL_{add} prevents the tabu search from inserting specific arcs to a resource flow that have recently been removed. However, cycles may still occur because tabu restrictions only apply for a limited number of iterations (i.e., the tabu tenure) after which the corresponding solutions could be revisited. In the tabu search algorithm, it is possible to use either only one of the two tabu lists or both depending on the selected tabu tenures. If both tabu lists are used, this generally results in a stronger tabu restriction because the tabu condition is satisfied more frequently for similar tabu tenures.

The tabu tenures t_{add} and t_{drop} are chosen independently for both tabu lists TL_{add} and TL_{drop} based on the size of the selected neighborhood $\mathcal{N}_2$. We set $t_{add} := \text{rand}(a) + \alpha \cdot |\mathcal{N}_2|$ and $t_{drop} := \text{rand}(b) + \beta \cdot |\mathcal{N}_2|$, where a, b are two positive integer values and α, β are selected from the interval $[0, 1[$. The function $\text{rand}(r)$ randomly selects a number from the interval $[0, r[$ using a random number generator with a fixed seed. These values are calculated in each iteration of the tabu search and assigned to the corresponding triples in the tabu lists such that these modifications are tabu for the next t_{add} or t_{drop} iterations, respectively.

To intensify the search process, a total of l_{max} elite solutions can be stored during the search as described in Nowicki and Smutnicki (1996). An elite solution is a solution that improves the currently best solution during an iteration. If such a solution is generated, both the solution as well as the current tabu lists are stored in the set of elite solutions. It should be noted that the tabu list stored here is also updated to contain the move made in the next iteration to avoid making the same move when the tabu search is restarted from this elite solution. Now, if no new solution could be generated during an iteration or if a given number of iterations has passed without an improvement of the best solution found so far, the tabu search restarts from the oldest elite solution.

Otherwise, if the set of elite solutions is empty, the tabu search restarts from a new solution that is generated by the parallel schedule generation scheme. Here, to diversify the search, we calculate the total number of iterations that activities $i \in V$ have been critical and use this information to select these activities with a higher priority (i.e., the priority values based on the selected activity rule are modified accordingly). This ensures that activities that are often critical are scheduled with a higher priority than activities that are less often critical. It should be noted that the tabu search also restarts from a new solution if the best solution could not be improved during a given number of iterations after the tabu search has been restarted from an elite solution.

Finally, the tabu search terminates when a certain stopping condition (e.g., maximal number of iterations reached) is satisfied.

6 Computational results

To evaluate the tabu search algorithm presented above, we implemented it in Java and tested it on a computer with an Intel Core i5-3470 (3.20 GHz) processor with 8 GB RAM (not using multi-threading abilities). We applied this algorithm to the problem instances for the classical RCPSP with 30, 60, 90, and 120 activities as they have been generated by Kolisch and Sprecher (1997). These instances are available online at the website of the project scheduling problem library PSPLIB (http://www.om-db.wi.tum.de/psplib/main.html).

Table 3 Parameter settings for calculating the tabu tenures

n	a	α	b	β
(a) Parameter settings 1				
30	5	0.3	0	0.0
60	10	0.2	0	0.0
90	15	0.2	0	0.0
120	20	0.2	0	0.0
(b) Parameter settings 2				
30	4	0.3	3	0.01
60	7	0.2	3	0.01
90	12	0.2	3	0.01
120	15	0.2	5	0.02

Table 4 Results for the classical RCPSP

n	#	Tabu search							PSPLIB	
		$\Delta_{\min}$	$\Delta_{\max}$	$\Delta^0_{\min}$	$\Delta^0_{\max}$	t_{avg}	$t_{\max}$	opt	Δ	opt
30	480	0.50	0.75	2.49	2.72	2.4	13.4	402	0.00	480
60	480	12.67	13.25	2.54	2.94	16.6	65.0	345	10.37	431
90	480	12.31	12.85	1.87	2.25	46.4	177.6	339	9.48	402
120	600	37.60	38.71	2.98	3.70	221.3	366.5	164	29.18	291

The tabu search algorithm has been set up to terminate after at most 10,000 iterations or, alternatively, if the best known lower bound (or the optimal makespan) has been reached. In each iteration, at most 50 reroute modifications as well as at most 5 reverse modifications may be evaluated. Furthermore, after some preliminary experiments, it turned out best to store $l_{\max} = 1$ elite solution. The algorithm restarts from this elite solution if no improving solution (i.e., a solution that improves the currently best solution) could be found in the last 750 iterations. If no improving solution could be found in the last 1500 iterations, the algorithm restarts from a new solution as described above. The parameters a, b, α, and β used to calculate t_{add} and t_{drop} were selected independently for each set of instances based on preliminary experiments as displayed in Table 3. While (a) displays parameter settings if only the tabu list TL_{add} is used, (b) displays the parameter settings if both tabu lists are used.

Computational results for the problem instances of the classical RCPSP that were obtained based on the two parameter settings from Table 3 can be found in Table 4. The algorithm was run with both settings, one after each other, we report minimum and maximum deviations $\Delta_{\min}$, $\Delta_{\max}$ (both in percent). Similar to the extensive computational studies by Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006), we report deviations from the optimal solution for the instances with 30 activities and deviations from the critical path lower bound LB_0 for the instances with 60, 90, and 120 activities. Furthermore, we list the minimal and maximal improvements $\Delta^0_{\min}$, $\Delta^0_{\max}$ (both in percent) from the initial solution, as well as the average and maximum

Table 5 Results for the RCPSP with transfer times (instances from Krüger 2009)

n	#	Tabu search								
		$\Delta^{\text{opt}}_{\text{min}}$	$\Delta^{\text{opt}}_{\text{max}}$	$\Delta^{\text{LB}_0}_{\text{min}}$	$\Delta^{\text{LB}_0}_{\text{max}}$	Δ^{0}_{min}	Δ^{0}_{max}	t_{avg}	t_{max}	opt
30	480	0.81	1.21	13.92	14.46	3.82	4.16	2.6	13.2	393
60	400	0.84	1.18	4.12	4.52	3.08	3.39	9.9	57.3	337
n	#	Multi-pass		GA						
		Δ^{opt}	opt	Δ^{opt}	opt					
30	480	1.63	322	0.16	459					
60	400	1.61	284	0.38	355					

computational time t_{avg} and t_{max} (both in seconds). Furthermore, we report the number *opt* of instances for which an optimal solution could be found. These results are then compared to the results of the best heuristic solutions as they have been reported on the website of the PSPLIB.

In can be seen that our results are worse than the best PSPLIB results for all sets of instances (up to almost 9 % for the instances with 120 activities).

In terms of computational time, a comparison of the different algorithms is difficult since different hardware was used. However, it is likely, that our algorithm generally requires more time than other algorithms that represent solutions as activity lists due to the large size of the neighborhoods (in particular, the neighborhood $\mathcal{N}^{\text{max,ca},\Delta}_{\text{reroute}}$ can be very large) as well as the time required to evaluate a single solution (which is $\mathcal{O}(n^3)$ due to the calculation of the longest path lengths after a modification). Thus, while our algorithm is still able to obtain reliable results for these instances, it cannot compete with algorithms that are specifically designed to solve the classical RCPSP. Additionally, for the classical RCPSP, the effect of Example 2 does not apply. Most algorithms for the RCPSP are based on the smaller solution spaces of semi-active (or even only active) schedules, for which it is guaranteed that they always contain an optimal solution. Here, the larger solution space of resource flows seems to be disadvantageous.

Furthermore, we solved the instances for the RCPSP with transfer times consisting of 30 and 60 activities as they were generated by Krüger and Scholl (2009) based on the corresponding problem instances from the PSPLIB. It should be noted that they generated instances for which the optimal solutions are known. There are 480 instances with $n = 30$ and 400 instances with $n = 60$. The results for these instances can be found in Table 5. We again report our results for the best solutions that have been obtained by one of the two parameter settings listed in Table 3. These are then compared to the results obtained by a multi-pass heuristic using the serial and parallel schedule generation scheme with forward–backward improvement and different priority rules (cf. Krüger 2009), as well as the results obtained by a genetic algorithm with a stopping condition of 5000 schedules (cf. Krüger 2009). For all of these approaches, we report the deviations from the optimal solution as well as from the lower bound LB_0, and

the improvements of the initial solutions. It should be noted that we used the adapted lower bound LB_0 introduced in Krüger (2009) that also incorporates the transfer times. In particular, if two activities i and j with $(i, j) \in A$ require $r_{ik} + r_{jk} > R_k$ units of resource k, additionally the transfer time Δ_{ijk} is regarded when calculating the longest path lengths.

For these instances, the solutions obtained by our algorithm are worse than those obtained by the genetic algorithm introduced in Krüger (2009). A closer examination of the results reveals that in particular instances with a large resource factor RF as well as a small resource strength RS are hard to solve due to the large number of resource transfers associated with these instances (i.e., the resulting solution space is very large). For example, we have an average deviation of 5.20 % from the optimal solutions for the 30 hard instances with 30 activities where RF $= 1.0$ and RS $= 0.2$. In comparison, Krüger (2009) reports an average deviation of 1.03 % for these instances for the genetic algorithm.

Again, it seems to be difficult to compare computational times. For the genetic algorithm by Krüger (2009), on an Intel Pentium 4 processor with 1 GB RAM an average computational time of $t_{\text{avg}} = 5.3$ s was reported for the instances with 30 activities and $t_{\text{avg}} = 45.8$ s for $n = 60$.

It should be noted that the extended instances from Krüger (2009) seem to be biased due to the process by which they have been generated. At first, optimal solutions for the PSPLIB instances were calculated using a branch-and-bound algorithm. Based on these solutions (i.e., vectors containing the starting times of the activities), Krüger then calculated a resource flow representing the optimal solution. Finally, she extended the instances by transfer times based on an integer linear program such that the transfer times between the activities are maximized while at the same time ensuring that optimality of the calculated resource flow as well as the optimal makespan is retained.

To ensure this condition, Krüger has chosen maximal transfer times $\Delta_{ijk}^{\max}$ for resources $k \in \mathcal{R}$ between each pair of activities $i, j \in V_{\text{all}}$ based on the calculated resource flows. In particular, if a resource transfer $f_{ijk} > 0$ of resource k exists between activities i and j, the maximal transfer time is given by $\Delta_{ijk}^{\max} = \phi \cdot (S_j - (S_i + p_i))$ with $\phi \in [0, 1]$ (Krüger uses the value $\phi = 0.5$), i.e., the maximal transfer time is bounded by the gap between the end of activity i and the start of activity j. Otherwise, if no resource transfer of resource k exists between activities i and j (i.e., for $f_{ijk} = 0$), a random value $\Delta_{ijk}^{\max} \in [0, \max_k]$ is selected where the upper bound $\max_k = 3 \cdot \lceil \frac{\Theta_k}{\vartheta_k} \rceil$ depends on time span Θ_k between the first and the last usage of resource k in the optimal schedule as well as on the number ϑ_k of jobs that require resource k during this time span.

As a result of this, the calculation of the transfer times for the extended instances is strongly influenced by the corresponding optimal resource flows. In particular, the transfer times for resources k between activities i and j with $f_{ijk} > 0$ are generally small, while the transfer times between activities i and j with $f_{ijk} = 0$ are larger. Moreover, the transfer times for resources $k \in \mathcal{R}$ between two consecutive critical activities $i \in V_0$ and $j \in V_*$ with $f_{ijk} > 0$ are always equal to zero (otherwise, the optimal makespan could not be retained). These observations can also be verified by taking a closer look at the extended instances. For example, for hard instances with

Table 6 Results for the RCPSP with transfer times (new instances)

n	#	$\Delta^{LB_0}_{min}$	$\Delta^{LB_0}_{max}$	Δ^{0}_{min}	Δ^{0}_{max}	t_{avg}	t_{max}
30	96	33.11	34.48	7.05	7.89	7.3	12.9
60	96	35.79	37.16	6.22	7.24	36.5	60.6
90	96	36.21	37.79	5.61	6.66	94.4	165.2
120	120	89.27	91.43	4.87	5.95	265.8	359.2

RF = 1.0 and RS = 0.2, it can be seen that almost all transfer times are equal to zero (i.e., the optimal schedule is very tight and a lot of resource transfers occur between the activities). For this reason, the extended instances as they have been introduced in Krüger (2009) are not entirely representative of the RCPSP with transfer times (in particular, for these instances, the phenomenon from Example 2 may not hold).

Due to this, we generated and solved additional instances (see http://www2.informatik.uni-osnabrueck.de/kombopt/data/rcpsp/). In particular, we randomly selected two instances for each configuration from the sets with 30, 60, 90, and 120 activities and extended them by transfer times (i.e., we have 96 instances with 30, 60, and 90 activities, respectively, as well as 120 instances with $n = 120$). These instances were extended by transfer times as follows. First of all, the transfer times Δ_{0jk} and $\Delta_{i,n+1,k}$ involving the dummy activities are set to zero. Furthermore, to generate the remaining transfer times between the real activities, we represented each activity $i \in V$ as a randomly generated point (x_{ik}, y_{ik}) with $0 \leq x_{ik}, y_{ik} \leq 5$ for each resource $k \in \mathcal{R}$ and calculated the transfer times based on the Euclidean distance as

$$\Delta_{ijk} = \min \left\{ 5, \left\lceil \sqrt{(x_{ik} - x_{jk})^2 + (y_{ik} - y_{jk})^2} \right\rceil \right\}.$$

This ensures that the transfer times are randomly selected from the interval [0, 5] and the triangle inequality is satisfied. We selected this interval because the processing times for all instances are from the interval [1, 10] and we wanted to ensure that the transfer times are mostly smaller than the processing times.

In Table 6, we report the results of our algorithm based on the parameter settings from Table 3. As before, we report the minimal/maximal deviations $\Delta^{LB_0}_{min}$, $\Delta^{LB_0}_{max}$ (in percent) from the adapted lower bound LB_0, the minimal/maximal improvements Δ^{0}_{min}, Δ^{0}_{max} (in percent) from the initial solution, and the average/maximal computation time t_{avg} and t_{max} (in seconds).

In this table, it can be seen that our approach is able to achieve considerable improvements from the initial solutions (up to an average of more than 7.9 %). This improvement is larger than in Tables 4 and 5 where the average improvement is always smaller than 4.2 %.

It should be noted that the adapted lower bound LB_0 is a very weak lower bound for this problem. Already for the classical RCPSP, the average deviations from the lower bound LB_0 are between 8 and 40 %. Here, with the addition of transfer times between the activities, this problem grows even more prominent (cf. Krüger 2009). Thus, it can be assumed that the actual deviations from the optimal makespan are much smaller than the deviations from the lower bound LB_0 reported above.

Table 7 Comparison of the tabu search algorithm with the truncated branch-and-bound algorithm described in Neumann et al. (2003) (new instances)

n	#	Tabu search (TS)					Branch-and-bound (BB)			
		$\Delta_{\min}$	$\Delta_{\max}$	$\Delta_{\min}^{\text{best}}$	$\Delta_{\max}^{\text{best}}$	opt	Δ	Δ^{best}	t_{avg}	opt
30	96	27.56	28.93	0.22	1.18	40	32.90	3.41	352.5	42
60	96	35.16	36.53	0.13	1.25	30	48.77	7.98	450.2	25
90	96	36.21	37.79	0.16	1.32	26	54.10	10.09	456.9	23
120	120	89.27	91.43	0.08	1.23	2	133.90	21.47	600.9	0

n	#	Comparison		
		TS < BB	TS = BB	TS > BB
30	96	43	45	8
60	96	64	27	5
90	96	63	27	6
120	120	119	0	1

Additionally, we solved the new instances by a truncated branch-and-bound algorithm described in Neumann et al. (2003) (Chapter 2.14). This algorithm was designed to solve a more general problem, namely the RCPSP with generalized precedence constraints (i.e., minimal and maximal time lags) as well as sequence-dependent setup times. We ran this algorithm on a Windows computer with an Intel Core i5-2410 (2.30 GHz) processor with 4 GB RAM and a time limit of 600 s.

In Table 7, we report the results and compare them to the results obtained by our algorithm. Here, it should be noted that some solutions could be verified to be optimal by the branch-and-bound algorithm or if the makespan of the generated solution equals the adapted lower bound LB_0.

In this table, $\Delta_{\min}$, $\Delta_{\max}$, and Δ (in percent) denote the deviations from the optimal makespan (if verified) or from the adapted lower bound LB_0, while $\Delta_{\min}^{\text{best}}$, $\Delta_{\max}^{\text{best}}$, and Δ^{best} (in percent) denote the deviations from the best solution found by the tabu search or the branch-and-bound algorithm. The value *opt* denotes the number of solutions that could be verified to be optimal, while the time t_{avg} (in seconds) denotes the average computation time of the branch-and-bound algorithm (bounded by the given time limit). Here, a solution is regarded as optimal if the makespan is equal to the adapted lower bound LB_0 or to the optimal makespan $C_{\max}^*$ (with $C_{\max}^* \geq \text{LB}_0$) verified by the branch-and-bound algorithm. Finally, we compare how often the best solution found during the first or the second pass of the tabu search algorithm is better (TS < BB), equal (TS = BB) or worse (TS > BB) than the solution found by the branch-and-bound algorithm.

Overall, it can be seen that our algorithm was able to obtain considerably better results than the branch-and-bound algorithm for these problem instances. In particular, it is often able to generate solutions with either a better or at least an equal makespan within a shorter amount of time compared to the solutions obtained by the branch-and-

bound algorithm. Finally, it should be noted that in some cases, solutions generated by the tabu search could be proved to be optimal despite the fact that these solutions have not been proved to be optimal by the branch-and-bound algorithm. This is the case if the makespan of the solution obtained by the tabu search algorithm is equal to the adapted lower bound LB_0.

7 Concluding remarks

In this paper, we introduced a tabu search algorithm for the RCPSP with transfer times that is based on a resource flow representation and two types of modifications. Apart from theoretical results (in particular, studying the connectivity of the neighborhoods), we report computational results for both, the classical RCPSP as well as the RCPSP with transfer times. While our algorithm could not compete with state-of-the-art algorithms for the classical RCPSP, we were able to obtain promising results for the RCPSP with transfer times. Especially, an extension of the approach to the RCPSP with first- and second-tier resource transfers in Poppenborg (2014) makes a further consideration of this approach worthwhile.

For further research, it would be interesting to consider further reductions of the neighborhoods as well as a more efficient evaluation of neighbors. Also extensions to other regular objective functions (like the maximum lateness or sum of completion times) could be studied.

Acknowledgments We would like to thank Doreen Becker for providing her test instances and results from Krüger (2009) as well as Christoph Schwindt for giving us the code of the branch-and-bound algorithm described in Neumann et al. (2003). Additionally, we are very grateful for the constructive comments of two referees which helped us to improve the presentation of the paper.

8 Appendix

Theorem 2 *For the RCPSP with transfer times, the neighborhood $\mathcal{N}_1$ is connected.*

Proof To prove this, it is necessary to calculate topological orderings for AON-flow networks based on given resource flows. A topological ordering defines a sequence in which the activities can be processed such that all precedence constraints as well as all resource transfers between the activities are observed (i.e., if a precedence constraint $(i, j) \in A$ or a resource transfer $f_{ijk} > 0$ for some resource $k \in \mathcal{R}$ exists between activities $i \in V_0$ and $j \in V_*$, activity i comes before activity j in the topological ordering).

However, it should be noted that a topological ordering is not necessarily unique. In particular, if no directed path exists from activity i to activity j and from activity j to activity i, no particular order has to exist between activities i and j (i.e., activity i can come either before or after activity j in a topological ordering). This is the case, for example, if both activities can be processed in parallel by different resource units. In this case, we assume that activities that can be processed in parallel are ordered according to increasing numbers. Then, a unique topological ordering can be calculated for each AON-flow network.

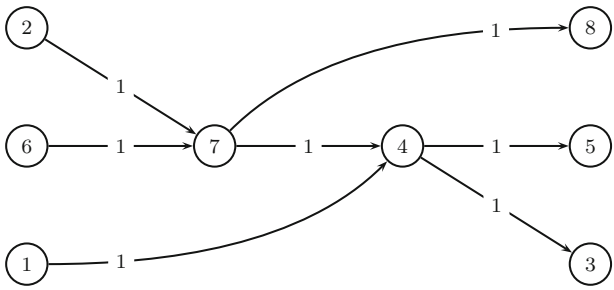
Let $\mathcal{F}$ and $\mathcal{F}'$ be two arbitrary feasible resource flows with $\mathcal{F} \neq \mathcal{F}'$. Furthermore, let $\pi = (\pi_0, \ldots, \pi_{n+1})$ and $\pi' = (\pi'_0, \ldots, \pi'_{n+1})$ be the topological orderings of the corresponding AON-flow networks. We now show that the neighborhood $\mathcal{N}_1$ is connected by transforming resource flow $\mathcal{F}'$ into resource flow $\mathcal{F}$ by a finite number of modifications in $\mathcal{N}_1$.

First, we consider all activities $i, j \in V$ with $f'_{ijk} > 0$ for some resource $k \in \mathcal{R}$ in resource flow $\mathcal{F}'$ (i.e., activity i comes before activity j in π') where activity j comes before activity i in π for resource flow $\mathcal{F}$. This implies that in resource flow $\mathcal{F}$ there are no direct or indirect resource transfers from activity i to activity j. If such activities i, j exist, it is always possible to select two of these activities such that in resource flow $\mathcal{F}'$ no other path exists from activity i to activity j via other activities. This follows from the following reasoning.

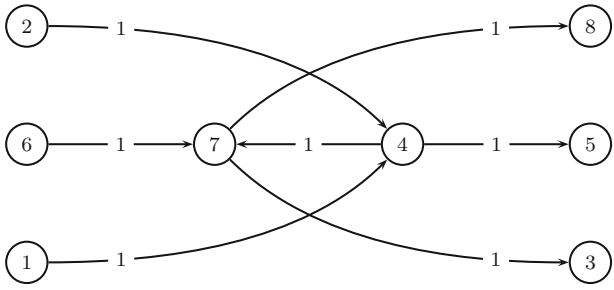
Let $u, w \in V$ be two of these candidates for which at least one path $P = (u, v_1, \ldots, v_\mu, w)$ via μ other activities $v_1, \ldots, v_\mu \in V$ exists from activity u to activity w. In this case, none of the activities v_1 to v_μ can simultaneously be a successor of activity u as well as a predecessor of activity w in the topological ordering π for resource flow $\mathcal{F}$. Otherwise, if an activity $v \in \{v_1, \ldots, v_\mu\}$ came after activity u as well as before activity w in the topological ordering π for resource flow $\mathcal{F}$, this would imply that also activity u comes before activity w which contradicts the condition that activity w comes before activity u in the topological ordering π. Thus, it is always possible to identify two consecutive activities $i \in \{u, v_1, \ldots, v_\mu\}$ and $j \in \{v_1, \ldots, v_\mu, w\}$ on this path such that activity i comes after activity j in π. By this reasoning, we find two activities $i, j \in V$ with the above property.

Now, the arcs $(i, j)_k$ for all resources $k \in \mathcal{R}$ with $f'_{ijk} > 0$ between the selected activities i and j can be reversed based on a modification in the neighborhood $\mathcal{N}_{\text{reverse}}$. In the resulting resource flow, activity j comes before activity i, i.e., the order of these two activities is reversed and matches the order of the activities in the topological ordering π for resource flow $\mathcal{F}$. Additionally, incoming resource transfers to activity i as well as outgoing resource transfers from activity j have to be adapted as described above. Here, it can be inferred from Fig. 7 that the order in which these activities have to be processed in relation to activities i and j does not change. Instead, all activities $u \in V_0$ from which resource units are redirected to activity j are still predecessors of activity i in the resulting topological ordering and all activities $v \in V_*$ to which resource units are redirected from activity i are still successors of activity j.

It should be noted that activities $u \in V_0$ from which resource units are transferred to activity i in resource flow $\mathcal{F}'$ (as well as direct and indirect predecessors of these activities u) that are not affected by a reverse move (i.e., no resource units are redirected from these activities u to activity j) can be processed in parallel to activity j (as well as in parallel to direct and indirect successors of activity j) in the resulting resource flow unless other directed paths exist between these activities. Similarly, activities $v \in V_*$ to which resource units are transferred from activity j in resource flow $\mathcal{F}'$ (as well as direct and indirect successors of these activities v) that are not affected by a reverse move (i.e., no resource units are redirected from activity i to these activities v) can be processed in parallel to activity i (as well as in parallel to direct and indirect successors of activity i) in the resulting resource flow unless other directed paths between these



(a) Before modification (with $\pi = (1, 2, 6, 7, 4, 3, 5, 8)$)



(b) After modification (with $\pi = (1, 2, 4, 5, 6, 7, 3, 8)$)

Fig. 20 Reversing the arc between activities 7 and 4

activities exist. Here, while the topological ordering of these activities may be different in the resulting resource flow, no additional arcs are inserted into the resource flow that might have to be reversed to obtain the order π.

An example for this situation is visualized in Fig. 20. Here, an extract of a resource flow is shown before as well as after the only arc between activities 7 and 4 has been reversed. In the resulting resource flow, activity 4 comes before activity 5. Additionally, activity 6 (from which no resource unit is redirected to activity 4) can be processed in parallel to activities 4 and 5 and activity 5 (to which no resource unit is redirected from activity 7) can be processed in parallel to activities 7, 3, and 8 such that the topological ordering of these activities changes based on their numerical ordering.

Thus, each reverse move changes the order of two activities $i, j \in V$ into the same order as in the topological ordering π for resource flow $\mathcal{F}$ and no additional arcs result from these moves that have to be reversed to obtain π. For this reason, it is possible to transform resource flow $\mathcal{F}'$ into a resource flow $\mathcal{F}''$ by a finite number of modifications in the neighborhood $\mathcal{N}_{\text{reverse}}$ such that the topological ordering of all activities $i, j \in V$ on directed paths in resource flow $\mathcal{F}''$ is the same as for resource flow $\mathcal{F}$.

Now, if $f''_{ijk} = f_{ijk}$ holds for all resource transfers of resource $k \in \mathcal{R}$ between activities $i \in V_0$ and $j \in V_*$ in resource flows $\mathcal{F}''$ and $\mathcal{F}$, resource flow $\mathcal{F}'$ has been transformed into resource flow $\mathcal{F}$ and we are finished. Otherwise, due to flow conservation, at least two resource transfers $f''_{ijk} > 0$ and $f''_{uvk} > 0$ between activities $i, u \in V_0$ and $j, v \in V_*$ have to exist with the following properties:

- The amount of resource units of resource k transferred on these arcs is larger than in resource flow $\mathcal{F}$, i.e., $f_{ijk} < f''_{ijk}$ and $f_{uvk} < f''_{uvk}$ hold.
- Without loss of generality, it can be assumed that the amount of resource k transferred from activity i to activity v in resource flow $\mathcal{F}''$ is smaller than in resource flow $\mathcal{F}$, i.e., $f_{ivk} > f''_{ivk}$ holds.
- Activity j does not come before activity u in the topological ordering π for resource flow $\mathcal{F}$. This is assured because all activities share at least one common predecessor (e.g., the dummy activity 0) as well as one common successor (e.g., the dummy activity $n+1$) and all activities on directed paths from 0 to $n+1$ are ordered according to the topological ordering π.

After such activities have been identified in resource flow $\mathcal{F}''$, it is possible to reroute an amount of $q = \min\{f''_{ijk} - f_{ijk}, f''_{uvk} - f_{uvk}, f_{ivk} - f''_{ivk}\}$ units of resource k from activity i to activity v as well as from activity u to activity j by a modification in the neighborhood $\mathcal{N}_{\text{reroute}}$. Now, we can consider the four affected resource transfers before and after the modification in comparison to resource flow $\mathcal{F}$:

- We have $f_{ijk} < f''_{ijk}$ before the modification. After the modification, $q \leq f''_{ijk} - f_{ijk}$ units of resource k are redirected from activity i to activity v, i.e., we have $f_{ijk} \leq f''_{ijk} - q$. Thus, the deviation of the resulting resource transfers is reduced by q units.
- We have $f_{uvk} < f''_{uvk}$ before the modification. After the modification, $q \leq f''_{uvk} - f_{uvk}$ units of resource k are redirected from activity u to activity j, i.e., we have $f_{uvk} \leq f''_{uvk} - q$. Thus, the deviation of the resulting resource transfers is reduced by q units.
- We have $f_{ivk} > f''_{ivk}$ before the modification. After the modification, $q \leq f_{ivk} - f''_{ivk}$ units of resource k are redirected from activity i to activity v, i.e., we have $f_{ivk} \geq f''_{ivk} + q$. Thus, the deviation of the resulting resource transfers is reduced by q units.
- In the worst case, we have $f_{ujk} \leq f''_{ujk}$ before the modification. After the modification, q units of resource k are redirected from activity u to activity j, i.e., we have $f_{ujk} < f''_{ujk} + q$. Thus, the deviation of the resulting resource transfers is increased by q units.

As a result of this, the deviation for three of the resulting resource transfers is reduced by q units while it is increased by at most q units for one resource transfer. Then, because all other resource transfers remain unchanged, the deviation of the resulting resource flow from $\mathcal{F}$ is always reduced by at least $2q$ compared to the deviation of $\mathcal{F}''$ and $\mathcal{F}$. Furthermore, because the modified resource transfers are selected such that activity u comes before activity j in the topological ordering π for resource flow $\mathcal{F}$, no arc is inserted that would have to be reversed based on a modification in the neighborhood $\mathcal{N}_{\text{reverse}}$.

Thus, because each reroute move reduces the deviation of the corresponding flows (and at least one resource transfer is set to the same value as in resource flow $\mathcal{F}$), it is possible to transform $\mathcal{F}''$ into a resource flow $\mathcal{F}'''$ by a finite number of modifications in the neighborhood $\mathcal{N}_{\text{reroute}}$ such that $f'''_{ijk} = f_{ijk}$ holds for all $i \in V_0$, $j \in V_*$, and $k \in \mathcal{R}$, i.e., resource flow $\mathcal{F}'$ can be transformed into resource flow $\mathcal{F}$ by a finite number of modifications in the neighborhood $\mathcal{N}_1$. □

References

Artigues C (2010) The resource-constrained project scheduling problem. In: Artigues C, Demassey S, Néron E (eds) Resource-constrained project scheduling. ISTE, London, pp 19–35

Artigues C, Michelon P, Reusser S (2003) Insertion techniques for static and dynamic resource-constrained project scheduling. Eur J Oper Res 149(2):249–267

Artigues C, Roubellat F (2000) A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. Eur J Oper Res 127(2):297–316

Brucker P, Knust S (2011) Complex scheduling. Springer, Berlin

Demeulemeester EL, Herroelen W (2002) Project scheduling: a research handbook. International Series in Operations Research & Management Science. Kluwer Academic Publishers, Boston

Floyd RW (1962) Algorithm 97: shortest path. Commun ACM 5(6):345

Fortemps P, Hapke M (1997) On the disjunctive graph for project scheduling. Found Comput Decis Sci 22(3):195–209

Glover F, Laguna M (1998) Tabu search. Kluwer Academic Publishers, Dordrecht

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207(1):1–14

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 127(2):394–407

Hurink J, Knust S (2001) List scheduling in a parallel machine environment with precedence constraints and setup times. Oper Res Lett 29(5):231–239

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90(2):320–333

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174(1):23–37

Kolisch R, Sprecher A (1997) PSPLIB—a project scheduling problem library. Eur J Oper Res 96(1):205–216

Krüger D (2009) Multi-project scheduling with resource transfers. Books on Demand GmbH, Norderstedt

Krüger D, Scholl A (2009) A heuristic solution framework for the resource constrained (multi-) project scheduling problem with sequence-dependent transfer times. Eur J Oper Res 197(2):492–508

Krüger D, Scholl A (2010) Managing and modelling general resource transfers in (multi-) project scheduling. OR Spectr 32(2):369–394

Mika M, Waligóra G, Węglarz J (2006) Modelling setup times in project scheduling. In: Józefowska J, Węglarz J (eds) Perspectives in modern project scheduling, vol 92. Springer, New York, pp 131–163

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions. Springer, Berlin

Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Manag Sci 42(6):797–813

Poppenborg J (2014) Modeling and optimizing the evacuation of hospitals based on the RCPSP with resource transfers. PhD thesis, Clausthal University of Technology, Clausthal-Zellerfeld, Germany

Roy B, Sussmann B (1964) Les problèmes d'ordonnancement avec constraintes disjonctives. Technical report note D.S. no. 9 bis, SEMA, Paris, France

Schwindt C, Trautmann N (2000) Batch scheduling in process industries: an application of resource-constrained project scheduling. OR Spectr 22(4):501–524

Schwindt C, Trautmann N (2003) Scheduling the production of rolling ingots: industrial context, model, and solution method. Int Trans Oper Res 10(6):547–563

Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 80(1):94–102

van Laarhoven PJM, Aarts EHL, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40(1):113–125

Vanhoucke M (2008) Setup times and fast tracking in resource-constrained project scheduling. Comput Ind Eng 5(4):1062–1070