European Journal of Operational Research 262 (2017) 262–273

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Innovative Applications of O.R.

A hybrid metaheuristic for resource-constrained project scheduling with flexible resource profiles

Martin Tritschler, Anulark Naber, Rainer Kolisch*

TUM School of Management, Technical University of Munich, Arcisstr. 21, München 80333, Germany

ARTICLE INFO

Article history:
Received 22 April 2015
Accepted 1 March 2017
Available online 7 March 2017

Keywords:
Project scheduling
Flexible resource profiles
Schedule generation scheme
Metaheuristics

ABSTRACT

We consider a generalization of the resource-constrained project scheduling problem (RCPSP), namely the RCPSP with flexible resource profiles (FRCPSP) in discrete time periods. In the FRCPSP, for each activity the given resource requirement is allocated in a variable number of contiguous periods in which the activity is processed. As the resource allocation can be adjusted between time periods, the resulting resource profile of the activity becomes flexible. The FRCPSP consists of scheduling activities and determining for each activity a resource profile and, thus, a duration in order to minimize the makespan. We propose a Hybrid Metaheuristic for the FRCPSP. It contains the Flexible Resource Profile Parallel Schedule Generation Scheme which employs the concepts of delayed scheduling and non-greedy resource allocation, embedded in a genetic algorithm. The best-found schedules are further improved in a variable neighborhood search by transferring resource quantities between selected activities. The results of a computational study demonstrate that the proposed method yields significantly better solutions than three benchmark methods on problem instances with up to 200 activities.

1. Introduction

The resource-constrained project scheduling problem (RCPSP) consists of scheduling a set of activities in order to minimize the project completion time (makespan) under the constraints of limited resource availability and finish-to-start precedence relations with zero time-lags. In the RCPSP, activity durations are given and the resource allocation to each activity is assumed constant for its entire duration. However, in real-world projects it is often the case that only the total resource requirement of each activity is known beforehand, whereas the activity durations and the resource allocation must be planned accordingly. An activity's duration indeed results from the resource quantities allocated in the processing periods of the activity. As the quantities of allocated resources may vary between time periods, the activity's "resource profile" (Naber & Kolisch, 2014) is flexible and not limited to rectangular shapes as in the RCPSP.

As an example, consider the common case of human labor as a project resource. An activity may have a resource requirement of 6 person-days to complete. Hence, the activity may be scheduled with a constant resource allocation of 2 persons for 3 days. The activity may also be scheduled with a non-constant resource allocation and a resulting flexible resource profile of 2 persons for the first 2 days and 0.5 persons, i.e., 1 person working half-time, for the next 4 days.

A model for a project scheduling problem that integrates such decisions on the resource allocation was first proposed by Kolisch, Meyer, Mohr, Schwindt, and Urmann (2003). Naber and Kolisch (2014) denote this problem as the RCPSP with flexible resource profiles (FRCPSP). The FRCPSP consists of scheduling activities and determining for each activity a resource profile and, thus, a duration in order to minimize the makespan.

As a generalization of the RCPSP, the FRCPSP belongs to the class of NP-hard problems. Hence, it becomes intractable for growing problem sizes. A study by Naber and Kolisch (2014) shows that already for problem instances with 20 activities, a commercial solver is unable to always find optimal solutions within 2 hours of CPU time. Due to the high practical relevance and applicability of the FRCPSP, metaheuristic solution methods are deemed more appropriate to solve larger problem instances within reasonable time.

In this paper, we propose a Hybrid Metaheuristic (HM) for the FRCPSP in discrete time periods. The HM consists of a genetic algorithm (GA) combined with a variable neighborhood search (VNS). Both make use of the new Flexible Resource Profile Parallel Schedule Generation Scheme (FSGS). The FSGS uses the activity list representation of Hartmann (1998) in combination with two new lists governing non-greedy resource allocation and delayed scheduling. We extend the GA of Hartmann (1998) because it has proven to be

* Corresponding author.
E-mail addresses: martin.tritschler@tum.de (M. Tritschler), anulark.naber@tum.de (A. Naber), rainer.kolisch@tum.de (R. Kolisch).

http://dx.doi.org/10.1016/j.ejor.2017.03.006
0377-2217/

one of the best metaheuristics for the RCPSP. Its activity list encoding together with the application of a schedule generation scheme always ensures feasible solutions, while the GA framework allows for an extension to solve the specifics of the FRCPSP. With the GA we explore the solution space both systematically and randomly in search of promising solutions. The VNS is then applied for a local search in quest of further improvements (see Raidl, Puchinger, & Blum, 2010) by transferring resource quantities from non-critical to critical activities based on an analysis of resource flows (see Artigues, Michelon, & Reusser, 2003).

The remainder of this paper is organized as follows. In Section 2, we give a brief problem description before we provide a review of relevant literature in Section 3. In Section 4, we outline solution characteristics relevant to the design of the HM. The HM and its components are described in Section 5 together with an illustrative example. We report computational results in Section 6 and close the paper with concluding remarks in Section 7.

2. Problem description

We follow the FRCPSP definition of Naber and Kolisch (2014) and, therefore, just summarize it briefly. Table 1 provides a summary of the notation used in this paper. A set of n nonpreemptive activities $V = \{1, \ldots, n\}$ as well as the dummy source activity 0 and the dummy sink activity $n+1$ are given. To schedule the activities, we consider a planning time horizon of discrete time periods $t \in T$. Each activity i has to start at the beginning of a period $s_i \in T$ and complete at the end of a period $c_i \in T$. In the remainder of this paper we only refer to the period in which the activity starts or completes. Assuming $s_0 = c_0 = 0$, the makespan is defined as $c_{max} = \max_{i \in V}(c_i)$. All activities are subject to the finish-to-start precedence relations with zero time-lag given in set E. A precedence relation $(i \rightarrow j) \in E$ requires that $s_j > c_i$, that is an activity can only start after all its predecessors have been completed.

Each activity $i \in V$ requires a subset R_i of the given set of resources R. For each resource $r \in R_i$, a resource profile has to be determined. This profile specifies the allocated resource quantity $q_{irt} \in \mathbb{R}_{>0}$, in each contiguous period t over the duration of activity i. This duration $d_i = c_i - s_i + 1$ results from a nonincreasing function of the quantity of allocated resources, under the assumption that resource quantities are continuously divisible and additive.

Central to the problem is the concept of blocks. We define a "block" for resource r of activity i as a number of consecutive periods with a constant allocated resource quantity.

A schedule f for the FRCPSP specifies for each activity $i \in V$ a start period s_i, a duration d_i, and a resource profile for each required resource $r \in R_i$. The FRCPSP is to determine a schedule such that the makespan is minimized. The resource profiles have to adhere to the following three constraints:

1. The quantity of resource r allocated to activity i in each processing period t has to be within the range of the lower and upper resource usage bounds: $\underline{q}_{ir} \leq q_{irt} \leq \overline{q}_{ir}$.
2. The quantity of resource r allocated to activity i has to remain constant for at least a minimum block length (Fündeling, 2006) of $\underline{l}_{ir}$ consecutive time periods.
3. The total quantity of resource r allocated to activity i has to meet or cover the resource demand w_{ir}: $\sum_{t=s_i}^{c_i} q_{irt} \geq w_{ir}$. Exceeding the resource demand might be necessary in order to guarantee problem feasibility when taking into account minimum block lengths.

All resources are renewable with b_r denoting the availability of resource r in each period. Resources are categorized into three types (Naber & Kolisch, 2014):

Table 1
Summary of notation.

Indices	
i, j	Activity
$r, \hat{r}$	Resource, specifically the principal resource
t	Time period
Sets	
A, E	Active and eligible activities
C	Critical activities
E	Immediate precedence relations
P	Selected activity pairs (i, j)
R, R_i, R_i^{pi}	Resources, required resources of activity i, required principal and independent resources of activity i
T	Discrete time horizon
T_{ij}	Time periods for resource transfer of activity pair (i, j)
V, V^{rec}	Activities, activities with rectangular resource profiles
Parameters	
b_r	Availability of resource r
$\underline{c}_i, \overline{c}_i$	Earliest and latest completion period of activity i
$\underline{d}_i, \overline{d}_i$	Lower and upper bounds of duration of activity i
$\underline{l}_{ir}$	Minimum block length of resource r and activity i
n	Number of activities
$p_\lambda, p_\rho, p_\sigma$	Mutation rates for λ, ρ, and σ
$\underline{q}_{ir}, \overline{q}_{ir}$	Lower and upper usage bounds of activity i for resource r
$\underline{s}_i, \overline{s}_i$	Earliest and latest start period of activity i
T_{min}	Lower bound of the makespan
w_{ir}	Requirement of activity i for resource r
α_{ir}, β_{ir}	Coefficient and constant of linear resource function for dependent resource r of activity i
λ	Activity list
$\rho, \rho_i, \overline{\rho}_i$	Resource limit list, resource limit of activity i, upper bound of resource limit of activity i
$\sigma, \sigma_i, \overline{\sigma}_i$	Start delay list, start delay of activity i, upper bound of start delay of activity i
Ω	Maximum number of generated schedules per problem instance
Variables	
c_i	Completion period of activity i
c_{max}	Project makespan
d_i	Duration of activity i
f, f^{rec}	Schedule, schedule featuring only rectangular resource profiles
$\vec{f}_{gen}$	Unique best schedule and corresponding solution representation for each GA generation
k	Neighborhood and number of resource transfers
l_{irt}	Periods in the block of resource r for activity i up to the period t
q_{irt}	Quantity of resource r allocated to activity i in period t
s_i	Start period of activity i
φ_r	Current leftover quantity of resource r
ϑ_{ijrt}	Transfer quantity of resource r for activity pair (i, j) in period t
ξ_{ir}	Remaining requirement of activity i for resource r

1. A principal resource $\hat{r}$ is the main resource of an activity and its allocated quantity may define the required quantities of other resources. An activity requires at most one principal resource, but a project may contain multiple activity-specific principal resources.
2. A dependent resource r of activity i is a resource whose allocated quantity q_{irt} depends on the positive allocated quantity $q_{i\hat{r}t}$ of the activity's principal resource $\hat{r}$ through a nondecreasing linear resource function $q_{irt} \geq \alpha_{ir} \cdot q_{i\hat{r}t} + \beta_{ir}$ with coefficient α_{ir} and constant β_{ir}. An activity may require multiple dependent resources. The rationale for the nondecreasing linear resource function is that for processing an activity each principal resource often requires a dependent resource in a certain relative amount. For example, principal resource programmer $\hat{r}$ might need dependent resource computer r. In this case the function is $q_{irt} = q_{i\hat{r}t}$. A similar example is the principal resource bio-lab technician requiring the dependent resource fluorescence microscope (Naber & Kolisch, 2014).

3. An independent resource of an activity is a resource whose allocated quantity is independent from any other resources. An activity may also require multiple independent resources.

Additional parameters can be obtained for each activity, such as the lower bound of duration $\underline{d}_i = \max_{r\in R_i}\left(\max(\lceil w_{ir}/\overline{q}_{ir}\rceil, \underline{l}_{ir})\right)$. The upper bound of duration $\overline{d}_i$ is calculated by using $\underline{q}_{ir}$ instead of $\overline{q}_{ir}$. By using $\underline{d}_i$ and $\overline{d}_i$ in the preprocessing techniques of Naber and Kolisch (2014), the activity's earliest start period $\underline{s}_i$ and its earliest completion period $\underline{c}_i = \underline{s}_i + \underline{d}_i - 1$ are obtained. $\overline{s}_i$ and $\overline{c}_i$ are the latest start and completion periods derived from an upper bound of the makespan. Finally, $T_{min} = \max_{i\in V}(\underline{c}_i)$ defines a lower bound of the makespan.

The FRCPSP is also related to other project scheduling problems. The RCPSP is a special case of the FRCPSP characterized by equal lower and upper resource usage bounds, no minimum block length, and only independent resources. The Multi-mode RCPSP (MMRCPSP) is similar to the FRCPSP in that each activity can be processed in multiple ways (Węglarz, Józefowska, Mika, & Waligóra, 2011). However, the MMRCPSP has a set of predetermined modes with constant resource allocation per activity from which one mode has to be selected. The FRCPSP also varies from the discrete time-resource tradeoff problem (DTRTP), where the activity duration is a function of the activity's resource usage, which is assumed constant (Węglarz et al., 2011).

3. Literature review

The FRCPSP was initially studied by Kolisch et al. (2003) in the context of real-world pharmaceutical research projects. They propose a mixed integer program (MIP) formulation as well as a priority rule heuristic with a serial or parallel schedule generation scheme. Their schedule generation scheme (SGS) follows the greedy principle of scheduling activities as early as possible and allocating the largest possible resource quantities. We denote an SGS that applies these greedy principles as a "standard" SGS. In the following, we distinguish the FRCPSP with continuously divisible resources (continuous resources) from the FRCPSP with discrete resources.

For the FRCPSP with continuous resources, four MIP models are proposed and compared in Naber and Kolisch (2014). They also apply a priority rule heuristic with a standard serial SGS to compute an upper bound of the makespan. In parallel to this work, Schramme (2014) has developed an MIP model and a GA with a non-greedy serial SGS for the FRCPSP without minimum block lengths and without dependent resources. Our proposed FSGS differs from the SGS of Schramme (2014), as we specifically address the minimum block length and determine activity durations as a result of the resource allocation, whereas Schramme (2014) considers activity durations as input parameters. Tritschler, Naber, and Kolisch (2014) propose the Self-Adapting Genetic Algorithm, which follows the idea of Hartmann (2002) to use a self-adaptive parameter to select from a standard serial or parallel SGS.

The FRCPSP with continuous resources is a relaxation of the FRCPSP with discrete resources and, hence, it holds that the optimal objective function of the FRCPSP with continuous resources is less than or equal to the optimal objective function of the FRCPSP with discrete resources. For discrete resources, the sets of resource profiles for the activities are finite, which renders the problem purely combinatorial. There are also two different paradigms in satisfying the resource requirements. The one addressed by Fündeling and Trautmann (2010), Schramme (2014) and Baumann, Fündeling, and Trautmann (2015) specifically allocates resources in the exact amount required by each activity. The second addressed by Naber and Kolisch (2014) and Tritschler et al. (2014) allows resources to be allocated at least by the amount required by each activity.

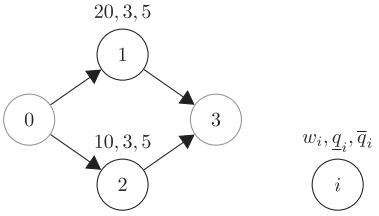

Fig. 1. Project featuring a single resource with availability of $b = 7$ and a minimum block length of $l = 2$ for all activities.

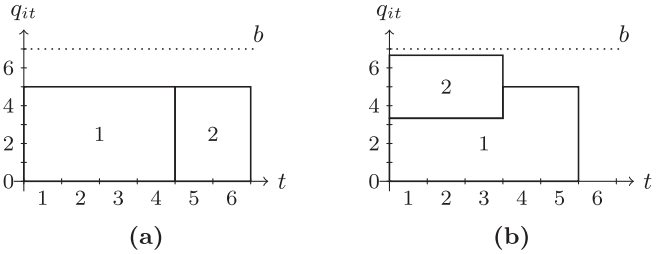

Fig. 2. Examples of schedules for the project given in Fig. 1.

Naber and Kolisch (2015) investigate empirically whether the two paradigms yield equivalent makespans, however, under continuous resources. The computational results show that the latter can yield reductions both in makespans and computational time.

For the FRCPSP with discrete resources, Zimmermann (2016) proposes an MIP-based heuristic. Baumann et al. (2015) provide an MIP model and also consider relaxations resulting in continuously divisible resource quantities and equality resource requirements. Fündeling and Trautmann (2010) apply a priority rule heuristic with a standard serial SGS and Fündeling (2006) provides in addition a parallel SGS which allocates resources to the maximum number of activities by first fulfilling their lower resource usage bounds. Both approaches assume that all activities in a project require the same principal resource. For projects entirely limited to a single resource, Ranjbar and Kianfar (2010) employ a GA with a serial SGS based on an a priori generated set of feasible resource profiles which are, however, limited to specific shape types. For a related problem with several additional constraints inspired from practice, Kuhlmann (2003) proposes multiple GA variants.

4. Solution characteristics

In this section, we argue that approaches beyond the greedy principles mentioned in Section 3 are required to generate optimal solutions for the FRCPSP. The examples in this section apply to both the standard serial and the standard parallel SGS.

4.1. Non-greedy resource allocation

The greedy principle of always allocating the maximum resource quantity does not necessarily lead to the minimum makespan because it may prevent the start of other activities, as pointed out by Fündeling and Trautmann (2010), or increase their durations. An optimal schedule may feature periods in which the resource allocation to an activity is reduced, whereas in other periods it is maximized in order to utilize the available resources.

For the project in Fig. 1, Fig. 2a shows a schedule resulting from a greedy resource allocation. Activity 1 starts with its maximum resource allocation of 5 units. Hence, activity 2 cannot start until period 5 and the resulting makespan is 6 periods. Fig. 2b shows an optimal schedule resulting from a non-greedy resource allocation of only $3\frac{1}{3}$ resource units to both activities in periods 1 to 3.

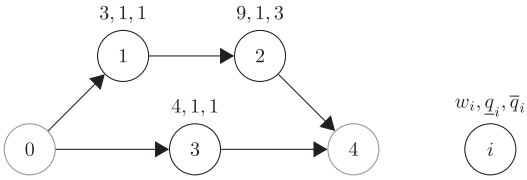

Fig. 3. Project featuring a single resource with availability of 3 and a minimum block length of 3 for all activities.

However, in periods 4 and 5, the resource quantity allocated to activity 1 is increased to the maximum of 5 units in order to utilize the available resources. The resulting makespan of 5 periods is optimal.

4.2. Delayed scheduling

Scheduling activities as early as possible does not necessarily lead to the minimum makespan (see Baumann et al., 2015). By delaying the start of an activity to a later period, the activity may be able to exploit a higher resource availability which otherwise could not be utilized. This effect is caused by the minimum block length.

For the project given in Fig. 3, Fig. 4a illustrates a schedule in which all activities start as early as possible. Activity 2 starts in period 4, where due to the processing of activity 3 only 2 resource units are available. Respecting the minimum block length of 3 periods, 1.8 resource units are allocated for 3 periods to activity 2, resulting in a makespan of 8. By delaying the start of activity 2 to period 5, where 3 resource units are available, an optimal schedule with the minimum makespan of 7 can be obtained, as shown in Fig. 2b.

5. Hybrid metaheuristic

First, the FSGS is introduced in Section 5.1. Then, the GA is described in Section 5.2 and the VNS in Section 5.3. Generally, we assume that a feasible schedule exists, that is the availability of each resource is sufficient to fulfill the lower bound of resource usage of each activity.

5.1. Flexible resource profile parallel schedule generation scheme

The FSGS implements non-greedy resource allocation, in which it allows periods of limited and periods of maximized resource allocation for the same activity, as well as delayed scheduling. The FSGS always generates feasible solutions. However, due to its heuristic resource allocation it may not always find the optimal resource profiles and makespan. The next sections describe the input parameters, explain the algorithm and provide an example.

5.1.1. Input parameters

The FSGS uses three compact input parameters:

- **Activity list λ**: The sequence of activities for resource allocation is defined by activity list λ (Hartmann, 1998). λ is any precedence feasible permutation of the set of activities V.
- **Resource allocation limit list ρ**: To facilitate non-greedy resource allocation, list $\rho = (\rho_1, \ldots, \rho_n)$ contains for each activity i an integer $\rho_i \in \{0, \ldots, \overline{\rho}_i\}$ that defines the limit for the allocated resource quantity through function $q_{irt} = w_{ir}/(\underline{d}_i + \rho_i)$. Instead of directly encoding the continuous resource quantities for each resource and period, just one integer value ρ_i is required per activity i. The upper bound $\overline{\rho}_i = \min(\overline{d}_i - \underline{d}_i, \overline{c}_i - \underline{s}_i)$ for ρ_i prevents too low limits leading to excessive prolongations of activities.
- **Start delay list σ**: To delay the start of activities, list $\sigma = (\sigma_1, \ldots, \sigma_n)$ contains for each activity i an integer $\sigma_i \in \{0, 1, \ldots, \overline{\sigma}_i\}$ that specifies the number of periods in which activity i is not started, despite a sufficient quantity of remaining resources and fulfilled precedence relations. The upper bound $\overline{\sigma}_i = \overline{s}_i - \underline{s}_i$ for σ_i is calculated from the earliest and latest start periods.

5.1.2. Algorithm

The FSGS extends the period-based approach of the parallel SGS to flexible resource profiles and minimum block lengths. The FSGS increments time periods and considers in each iteration one period t. Resources are allocated to an activity in the order of principal, dependent, and independent resources. An activity is completed as soon as the resource requirements and the minimum block lengths of all of its required resources are met. The duration d_i of activity i results from the allocated resource quantities per period. For simplicity, we assume for now that resource r of activity i is either principal or independent, i.e., $r \in R_i^{pi}$.

For each period t, the FSGS performs the steps given in Algorithm 1. λ provides the sequence of activities to which resources are allocated in all steps. We use ρ to limit the resource allocation in steps 2a and 2b and σ to delay the start of activities in step 2a. The FSGS considers set A of active activities, i.e., activities already started but not yet completed in t, and set E of eligible activities, i.e., activities that have not yet started and whose predecessors have been completed up to $t-1$. φ_r denotes the current leftover capacity of resource r, ξ_{ir} the remaining requirement of activity i for resource r, and the counter $delay_i$ denotes the number of periods that the start of activity i has been delayed. Furthermore, l_{irt} is the length of the block of activity i with respect to resource r until period t. All sets and variables are constantly updated after each operation. These updates are not stated in the algorithm to maintain brevity. We denote the block up to period $t-1$ as the "current block" and the block starting in period t as the "new block". In the following, we explain steps 1 to 3 of Algorithm 1:

Step 1 **Continue active activities to ensure nonpreemption:** For each resource r of activity $i \in A$ for which the minimum block length $\underline{l}_{ir}$ has been met, a quantity equal to the minimum resource usage bound $q_{irt} = \underline{q}_{ir}$ is allocated. If $\underline{l}_{ir}$ has not been met, the current block is continued by allocating the quantity of the previous period $q_{irt} = q_{ir(t-1)}$.

Step 2 **Allocate remaining resources:** Resources are distributed among eligible and active activities in a non-greedy manner according to sequence λ. Two disjunct cases apply:

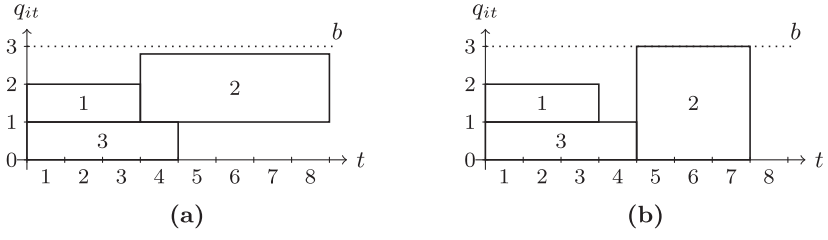

Fig. 4. Examples of schedules for the project given in Fig. 3.

Algorithm 1 Steps of the FSGS in period t.

1. **for** $i \in A$ in sequence of λ, $r \in R_i^{pi}$ **do**
   if $l_{ir(t-1)} \geq \underline{l}_{ir}$ **then**
   $q_{irt} = \underline{q}_{ir}$
   else
   $q_{irt} = q_{ir(t-1)}$
2. **for** $i \in A \cup E$ in sequence of λ, $r \in R_i^{pi} : \varphi_r > 0$ **do**
   (a) **if** $i \in E$ **and** $\forall r' \in R_i : \varphi_{r'} \geq \underline{q}_{ir'}$ **then**
   if $delay_i \geq \sigma_i$ **then**
   $q_{irt} = \min(\varphi_r, \frac{w_{ir}}{\underline{d}_i + \rho_i})$
   else
   $delay_i = delay_i + 1$
   (b) **if** $i \in A$ **and** $l_{ir(t-1)} \geq \underline{l}_{ir}$ **then**
   if $q_{irt} + \varphi_r \geq q_{ir(t-1)}$ **then**
   $q'_{irt} = \max(\underline{q}_{ir}, \min(q_{irt} + \varphi_r, \frac{\xi_{ir}}{\underline{l}_{ir}}, \frac{w_{ir}}{\underline{d}_i + \rho_i}))$
   if $\lceil \frac{\xi_{ir}}{q'_{irt}} \rceil \geq \underline{l}_{ir}$ **and** $(\lceil \frac{\xi_{ir}}{q'_{irt}} \rceil < \lceil \frac{\xi_{ir}}{q_{ir(t-1)}} \rceil$ **or** $q_{ir(t-1)} > \frac{w_{ir}}{\underline{d}_i + \rho_i})$
   $q_{irt} = q'_{irt}$
   else
   $q_{irt} = q_{ir(t-1)}$
   else
   $q_{irt} = q_{irt} + \varphi_r$
   if $\lceil \frac{\xi_{ir}}{q_{irt}} \rceil < 2 \cdot \underline{l}_{ir}$ **then**
   $q_{irt} = \max(\underline{q}_{ir}, \xi_{ir} / \lceil \frac{\xi_{ir}}{q_{irt}} \rceil)$
3. **for** $i \in A$ in sequence of λ, $r \in R_i^{pi} : \varphi_r > 0$ **do**
   Repeat step 2b with $q'_{irt} = \max(\underline{q}_{ir}, \min(q_{irt} + \varphi_r, \frac{\xi_{ir}}{\underline{l}_{ir}}, \overline{q}_{ir}))$

Step 2a **Start eligible activities:** Eligible activities are started based on the delays from σ. Activity $i \in E$ starts if the lower usage bound of each required resource is met and the start of the activity has already been delayed for at least σ_i periods. A delay is counted if activity i is not started, despite a sufficient quantity of remaining resources and fulfilled precedence relations. ρ_i limits the allocated resource quantity to $q_{irt} = \min(\varphi_r, w_{ir}/(\underline{d}_i + \rho_i))$.

Step 2b **Modify resource allocation of active activities:** This step only applies to resources required by active activities $i \in A$ for which the minimum block length has been met. For such resources, a quantity equal to the minimum usage bound has already been allocated in step 1. Now, additional resource quantities are allocated. We distinguish three cases: (1) A new block has to be started due to insufficient resources. (2) The current block is continued. (3) A new block is started in order to change the resource allocation. The operations of each case are as follows:

(1) If the current leftover quantity φ_r of resource r does not suffice to continue the current block, i.e., $\varphi_r < q_{ir(t-1)}$, we add the leftover capacity φ_r to the so far allocated resource quantity q_{irt} by setting $q_{irt} = q_{irt} + \varphi_r$ and, consequently, start a new block.

(2) If φ_r suffices to continue the current block, the algorithm checks whether to allocate the same quantity as in the previous period $t-1$, i.e., $q_{irt} = q_{ir(t-1)}$, or to start a new block based on case (3).

(3) A new block with an allocation of $q'_{irt} = \max(\underline{q}_{ir}, \min(q_{irt} + \varphi_r, \xi_{ir}/\underline{l}_{ir}, w_{ir}/(\underline{d}_i + \rho_i)))$ is only started by setting $q_{irt} = q'_{irt}$ if the following condition (I) and at least one out of conditions (II) or (III) apply: (I) The remaining resource requirement ξ_{ir} is sufficient to accommodate at least one minimum block length, i.e., $\lceil \xi_{ir}/q'_{irt} \rceil \geq \underline{l}_{ir}$. (II) The new block resulting from an increased resource quantity q'_{irt} is shorter than in the case of continuing the current block, i.e., $\lceil \xi_{ir}/q'_{irt} \rceil < \lceil \xi_{ir}/q_{ir(t-1)} \rceil$. (III) The allocated quantity in the current block is larger than the limit defined by ρ_i, i.e., $q_{ir(t-1)} > w_{ir}/(\underline{d}_i + \rho_i)$. Note that condition (III) can only apply if step 3 has been performed at the beginning of the current block. Based on condition (III), the new block resulting from a decreased resource quantity q'_{irt} again adheres to the limit defined by ρ_i.

Next, if less than two minimum block lengths remain to complete the activity, i.e., $\lceil \xi_{ir}/q_{irt} \rceil < 2 \cdot \underline{l}_{ri}$, the resource allocation has to be kept constant until the activity completes. Starting a new block in the following periods would otherwise increase the duration of the activity. To prevent an overallocation of resources beyond w_{ir} in this case, we set $q_{irt} = \max(\underline{q}_{ir}, \xi_{ir}/\lceil \xi_{ir}/q_{irt} \rceil)$.

Step 3 **Utilize leftover resources:** After completing steps 1 and 2, a feasible partial schedule up to period t has been generated in which resources are allocated in a non-greedy manner to active and eligible activities. However, due to the resource allocation limits from ρ, it may be the case that the available resources are not fully utilized in period t. Hence, in step 3 we exploit these leftover resource quantities by exceeding the allocation limit (see the example from Section 4.1). We allocate the leftover resource quantities to active activities strictly in the sequence of λ if the minimum block length permits it. For this purpose, step 2b is repeated with the modification $q'_{irt} = \max(\underline{q}_{ir}, \min(\varphi_r + q_{irt}, \xi_{ir}/\underline{l}_{ir}, \overline{q}_{ir}))$, which allows allocating resources up to $\overline{q}_{ir}$.

The allocation of dependent resources to activity i is performed as follows. The allocated quantity of dependent resource r is calculated based on the allocated quantity $q_{i\hat{r}t}$ of principal resource $\hat{r}$ by $q_{irt} = \alpha \cdot q_{i\hat{r}t} + \beta$. If φ_r is lower than q_{irt}, then we set $q_{irt} = \max(\underline{q}_{ir}, \varphi_r)$ and, as a consequence, also update the allocated quantity of the principal resource to $q_{i\hat{r}t} = (q_{irt} - \beta)/\alpha$. This requires to recalculate the amounts of other dependent resources too. The process is repeated until $q_{i\hat{r}t}$ adheres to the availability and the resource usage bounds of all dependent resources.

5.1.3. Example

To illustrate the FSGS, consider the project in Fig. 5. Given $\lambda = (3, 1, 4, 2)$, $\rho = (1, 0, 0, 0)$, and $\sigma = (0, 0, 0, 0)$, the FSGS generates the schedule f shown in Fig. 6. In the following, we only relate to

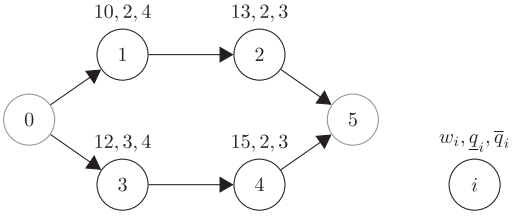

Fig. 5. Project featuring a single resource with availability of $b = 5$ and a minimum block length of $\underline{l}_i = 2$ for all activities.

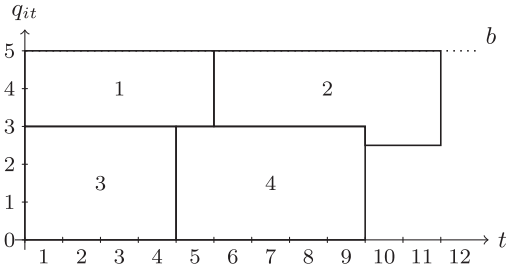

Fig. 6. Schedule f generated by the FSGS for the project from Fig. 5.

periods in which the allocated resource quantity is changed. In all other periods, only step 1 of Algorithm 1 applies. The index for the single resource is omitted.

In period 1, resources are first allocated to activity 3 due to its position in λ. As $\underline{d}_3 = \max(\lceil 12/4 \rceil, 2) = 3$ and $\rho_3 = 1$, a resource quantity of $\min(5, 12/(3+1)) = 3$ is allocated in step 2a. The remaining 2 resource units are allocated to activity 1 in step 2a. Activity 3 completes in period 4. In period 5, first 2 resource units are allocated to activity 1 based on step 1. This is sufficient for activity 1 to complete. Then, the remaining 3 resource units are allocated to start activity 4 in step 2a. In period 6, activity 4 continues with the same allocated quantity as in period 5 due to step 1. Hence, activity 2 can only start with an allocated quantity of 2 units from step 2a. Activity 4 completes in period 9. In period 10, activity 2 requires additional $\xi_2 = 5$ resource units to complete. First, a resource quantity equal to the lower usage bound $q_{2,10} = 2$ is allocated in step 1. As the length of the current block $l_{2,9} = 4$ is greater than $\underline{l}_2 = 2$, step 2b increases the quantity to $q'_{2,10} = \max(\underline{q}_2, \min(q_{2,10} + \varphi, \xi_2/\underline{l}_2, w_2/(\underline{d}_2 + \rho_2))) = \max(2, \min(2+3, 5/2, 15/(5+0))) = 2.5$. The new resulting block has a length of 2 which equals the minimum block length and is 1 period shorter than the continued current block. As the new block does not overallocate resources, the resource quantity is not changed by operation $\max(2, 2.5/\lceil 2.5/2.5 \rceil) = 2.5$. The activity completes in the following period, resulting in a makespan of 11.

5.2. Genetic algorithm

A genetic algorithm (GA) is a population-based metaheuristic inspired by the principles of natural evolution (Holland, 1975). To apply a GA to the FRCPSP, we use the FSGS input parameter lists λ, ρ, and σ as representation for a candidate solution. We employ the FSGS to generate a feasible schedule from a candidate solution and consider the resulting makespan as the fitness value.

Our proposed GA first creates an initial population of candidate solutions as described in Section 5.2.1. In each generation, the elite solution with the lowest makespan is inserted into the next generation. Based on the elite solution's makespan, we update the bounds $\bar{s}_i$, $\bar{c}_i$, $\overline{\rho}_i$, and $\overline{\sigma}_i$, which are thus tightened with every makespan improvement. Next, the GA selects a set of candidate solutions for the next generation by stochastic universal sampling (Baker, 1987). These candidate solutions are modified by the operators described in Section 5.2.2 and constitute the next generation. The whole process is repeated until a termination criterion is fulfilled. The GA returns as result the list $\vec{f}_{gen}$. It contains as unique elements the elite solution and the corresponding schedule for each GA generation, sorted in ascending order of generation count. As elite solutions are maintained, $\vec{f}_{gen}$ is also sorted in non-increasing order of makespan. $\vec{f}_{gen}$ is used as input for the VNS as described in Section 5.3.

5.2.1. Initial population

We construct activity list λ by iteratively adding one activity whose predecessors are already contained in λ, assuming that the dummy source activity is always present in λ. Diversity is introduced by selecting the next activity randomly as well as based on the priority rules employed in Fündeling (2006), namely longest path following (LPF), most work remaining (MWR), and most total successors (MTS). For 75% of the population, we set with a probability of 10% uniform random integer values for ρ_i within its bounds, whereas for the rest we set ρ_i to 0. All σ_i are set to 0 in order to prevent delayed activity starts in the initial population.

5.2.2. Operators

To recombine solution candidates, we apply the precedence-order maintaining two-point crossover of Hartmann (1998). It randomly selects two crossover points in lists λ, ρ, and σ of two parent solutions and generates two new feasible child solutions. We then apply three mutation operators. For λ we use the mutation operator of Hartmann (1998). It exchanges an activity in λ with the one at the next position with a probability of p_λ if the exchange is feasible regarding the precedences of the activities. The mutation of ρ_i has a probability of p_ρ. Here, one of the following two operations is done with equal probability. ρ_i is either replaced by a uniform random integer within its bounds or increased (decreased) by one. Finally, σ_i is replaced with a probability of p_σ by a uniform random integer within its bounds.

5.3. Variable neighborhood search

Variable neighborhood search (VNS) is a metaheuristic which combines local search with systematic change of neighborhoods (Hansen & Mladenović, 2005). We propose a VNS to further improve the best schedules found by the GA in list $\vec{f}_{gen}$ by transferring resource quantities between selected pairs of activities (i, j). VNS is an appropriate metaheuristic for this purpose, as its concept of nested neighborhoods represents combinations of multiple pairwise resource transfers. The VNS intensifies the search along the GA's search trajectory on solutions of already high quality.

We first describe how activity pairs are selected in Section 5.3.1, before we explain the resource transfer in Section 5.3.2. In Section 5.3.3, we define the neighborhoods and embed the resource transfer into the VNS framework. Finally, we provide an illustrated example in Section 5.3.4.

5.3.1. Activity selection

For most activity pairs a resource transfer either is infeasible or does not reduce the makespan. Thus, we construct subset P of the set of all activity pairs. The idea is to select activity pairs $(i, j) \in P$ such that resource quantities are only transferred from "non-critical" activity i to "critical" activity j in order to reduce the duration of j. Critical activities, as defined below, correspond to a longest path in a network representation of the schedule which utilizes the resource flow concept of Artigues et al. (2003). We construct set P for schedule f in five steps:

1. **Prevent cycles:** Due to the flexible resource profiles, the resource quantity allocated to an activity over time may first decrease and then increase again. In this case, there is a resource flow from the activity via other activities to itself. Hence, the resulting resource flow network contains cycles and the calculation of a longest path becomes NP-hard (Garey & Johnson, 1979). To derive an acyclic resource flow network, we transform schedule f into schedule f^{rec} in which all activities $i^{rec} \in V^{rec}$ feature rectangular resource profiles for all required resources. Resource profiles that contain more than one block are split into multiple new activities that each represent one single block. In each period in which an activity $i \in V$ from schedule f starts a new block, a new corresponding activity $i^{rec} \in V^{rec}$ starts in schedule f^{rec}.
2. **Generate resource flow network:** An acyclic resource flow network is derived from f^{rec} by adapting the algorithm of Artigues, Demassey, and Neron (2008, p. 34). In the resulting resource flow network, an arc from activity i^{rec} to j^{rec} exists if there is a positive resource flow from i^{rec} to j^{rec}. The weight of the arc is set to $\min(s_{j^{rec}} - s_{i^{rec}}, d_{i^{rec}})$.
3. **Identify critical activities:** A given maximum number of longest paths in the resource flow network is calculated. For a longest path, we denote activity $j \in V$ as critical if it corresponds to an activity from set V^{rec} which is on this longest path.

Set C contains all critical activities $j \in V$ corresponding to this longest path.

4. **Select activity pairs:** Set P contains activity pairs (i, j) in which activity i is non-critical, while activity j is critical and its duration is greater than its minimum duration, and the set T_{ij} of periods for resource transfer is nonempty. T_{ij} consists of the periods in which activity i currently has an allocated quantity above its lower usage bound, while activity j has a quantity below its upper usage bound for a shared principal or independent resource r:

$$P = \{(i,j) | i \notin C, j \in C, d_j > \underline{d}_j, T_{ij} \neq \emptyset\}$$

$$T_{ij} = \{t \in T | \exists r \in R_i^{pi} \cap R_j^{pi} : (q_{irt} > \underline{q}_{ir}, q_{jrt} < \overline{q}_{jr})\}$$

5. **Break ties:** In case of multiple longest paths, multiple sets P occur. In this case, we select the set P that has the highest makespan reduction potential $\sum_{(i,j)\in P}(d_j - \underline{d}_j)$.

5.3.2. Resource transfer

For $(i, j) \in P, r \in R_i^{pi} \cap R_j^{pi}$ and $t \in T_{ij}$, we define the maximum transfer quantity ϑ_{ijrt} as the minimum between the maximum possible sendable and receivable resource quantity, i.e., $\vartheta_{ijrt} = \min(q_{irt} - \underline{q}_{ir}, \overline{q}_{jr} - q_{jrt})$. The quantity of dependent resources is determined based on the quantity of the principal resource. Starting from feasible schedule f we perform k resource transfers for activity pairs $(i, j) \in P$ to generate a new feasible schedule f' as follows. We apply the FSGS based on λ, ρ, and σ. However, in each period $t \in T_{ij}$ the allocated resource quantity in steps 2a and 2b of Algorithm 1 is modified in an additional operation to $q_{irt} = \max(\underline{q}_{ir}, \min(\overline{q}_{ir}, q_{irt} - \vartheta_{ijrt}))$ for sending activity i and to $q_{jrt} = \max(\underline{q}_{jr}, \min(\overline{q}_{jr}, q_{jrt} + \vartheta_{ijrt}))$ for receiving activity j. In step 2b of Algorithm 1, a new block is started if the remaining resource requirement ξ_{ir} suffices to generate at least one minimum block length.

The resource transfer is only performed if both activities meet the minimum block length. This requirement is not considered in the activity selection, as in case of multiple resource transfers we have to actually generate a schedule in order to determine the resulting block lengths.

5.3.3. Schedule improvement

We define neighborhood k of feasible schedule f as the set of all feasible schedules which can be generated from f by k resource transfers for activity pairs $(i, j) \in P$. The VNS integrates two components: (1) the first-improvement variable neighborhood descent of Hansen and Mladenović (2005) which is a local search method employing a purely deterministic neighborhood change mechanism and (2) a perturbation method to escape local minima based on $\vec{f}_{gen}$ from the GA:

1. The VNS starts in neighborhood $k = 1$ of the last schedule f from $\vec{f}_{gen}$. This schedule has the shortest makespan found by the GA. In each local search step in neighborhood k, we generate a new schedule f' from the current incumbent f by k resource transfers based on activity pairs from P. If no makespan improvement is found within the given maximum number of generated schedules per neighborhood, the VNS proceeds to the next neighborhood $k = k + 1$. If the makespan of f' is reduced, the VNS moves from f to f' and continues the search in neighborhood $k = 1$ of schedule f'. The information which transfers have been performed is saved.
2. If f is not improved within the given maximum number of non-improving schedules or if the maximum allowed neighborhood has been reached, the VNS escapes the local optimum f by selecting the next unique schedule f' from $\vec{f}_{gen}$ and moving from f to f'. Thus, the VNS exploits the results of the GA and operates on schedules of already high quality instead of moving to a randomly generated schedule.

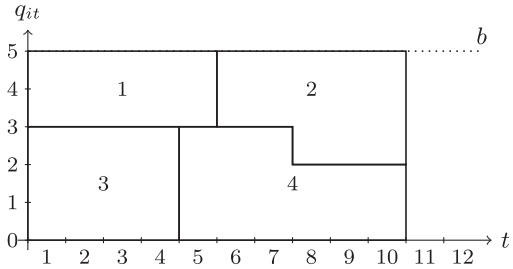

Fig. 7. Improved schedule f' resulting from schedule f of Fig. 6.

The VNS terminates and returns the overall best-found schedule if $\vec{f}_{gen}$ has been fully processed or the given maximum number of schedules has been generated or the lower bound of the makespan T_{min} has been reached.

5.3.4. Example

Assume that schedule f from Fig. 6 for the project from Fig. 5 is processed by the VNS. In the activity selection, schedule f is transformed into schedule f^{rec}. The non-constant resource profile of activity 2 in f contains 2 blocks. Hence, it corresponds to 2 activities with rectangular shape resource profiles in f^{rec}, one from periods 6 to 9 and the other from periods 10 to 11. The resulting resource flow network contains 2 longest paths of length 11. For the first longest path with critical activities $C = \{1, 2\}$, activities 1 and 2 are parallel to activity 4. Hence, we obtain periods $T_{4,1} = \{5\}$ and $T_{4,2} = \{6, 7, 8, 9\}$. We get $P = \{(4, 1), (4, 2)\}$ with a reduction potential of $(d_1 - \underline{d}_1) + (d_2 - \underline{d}_2) = (5 - 3) + (6 - 5) = 3$ periods. However, the other longest path with critical activities $C' = \{3, 4, 2\}$ has no reduction potential. For all activities in C', the only parallel activity which is not in C' itself is activity 1. As the allocated resource quantity of activity 1 is equal to its lower usage bound in all periods, no resource transfer is possible.

Neighborhood $k = 1$ of f contains all schedules resulting from $k = 1$ resource transfer. The first resource transfer for activity pair (4, 1) with $T_{4,1} = \{5\}$ in period 5 is not performed, as activity 1's remaining resource requirement of 2 units does not suffice to accommodate the minimum block length. For the resource transfer of activity pair (4, 2) with $T_{4,2} = \{6, 7, 8, 9\}$, the minimum block lengths of activities 4 and 2 prevent a transfer in periods 6 and 7. In period 8, a transfer of $\vartheta_{4,2,8} = \min(3 - 2, 3 - 2) = 1$ units is feasible, resulting in 3 resource units allocated to activity 2 and 2 units allocated to activity 4, as shown in schedule f' of Fig. 7. Due to the minimum block length, both activities have the same allocation in period 9. Both activities complete in period 10. As a result, the makespan is reduced by 1 period to 10, which is optimal.

6. Computational study

In this section, we present the computational study. First, we describe the design of the study in Section 6.1, before we present the test problem instances in Section 6.2 and give details on the parameter settings in Section 6.3. In Section 6.4, we report and analyze the computational results. Lastly, Section 6.5 demonstrates an application of the HM to the FRCPSP with discrete resources.

6.1. Test design

We compare the HM to the following benchmark methods:

- **SGA**: The Self-Adaptive Genetic Algorithm of Tritschler et al. (2014) uses an activity list representation and an additional self-adaptive parameter (Hartmann, 2002), the SGS flag, which determines whether a standard serial or a standard parallel SGS is used.
- **PRS**: The Parallel Random Sampling heuristic resembles the method of Kolisch et al. (2003). It constructs activity lists by random sampling and generates schedules with a standard parallel SGS.
- **SRS**: The Serial Random Sampling heuristic is the same as the PRS but uses a standard serial SGS.
- **MIP**: As a reference we employ a commercial solver using the best-performing MIP model FP-DT3 of Naber and Kolisch (2014). The best solution obtained after a time limit of two hours per problem instance is considered as a reference value.

Furthermore, in order to analyze the influence of the HM components on the solution quality, three HM variants are compared:

- **GA-FSGS**: The FSGS is embedded into the GA without the VNS. This combination is used to evaluate the impact of the VNS.
- **MP-FSGS**: In order to assess the GA's impact, the FSGS is employed in a multi-pass method (see Kolisch & Hartmann, 2006). By using the FSGS, the method generates a large number of schedules and selects the one with the best objective function value. Following Fündeling and Trautmann (2010), we generate three λ with the priority rules LPF, MTS, and MWR and ρ and σ set to zero. We generate the remaining λ by random activity selection and ρ and σ chosen randomly within the given bounds.
- **GA-SGS**: The GA operates only on λ and is combined with a standard parallel SGS. This combination is used to evaluate the FSGS.

The methods are compared based on a maximum number of generated schedules per problem instance (Ω). This widely accepted methodology has previously been applied in studies on the RCPSP (Kolisch & Hartmann, 2006) and the MRCPSP (van Peteghem & Vanhoucke, 2014). According to Kolisch and Hartmann (2006), the termination criterion has the advantages that it is platform independent and allows direct comparison with future studies. Each started schedule generation process is counted as one schedule. If a makespan equals T_{min}, an optimal schedule is obtained and the method terminates. We report the required computation time of each method as single-threaded CPU time.

All methods, except the MIP, are implemented in Java 7 on a desktop PC with a 3.3 gigahertz Intel Core i3-2120 CPU and 4 gigabytes of RAM. Naber and Kolisch (2014) solve their MIP model with CLPEX on a computer with a 3.4 gigahertz Intel Core i7-3770 CPU and 16 gigabytes RAM.

6.2. Test data

The study is conducted on problem instances from test sets A and B of Fündeling and Trautmann (2010). Test set A contains instances with up to 4 resources, derived from the RCPSP instances of the PSPLIB (Kolisch & Sprecher, 1997). The study only includes instances of test set A with at most 55 activities because for larger instances no MIP results are available. These 509 instances of test set A are labeled as instance set $A_{\leq 55}$, where the subscript indicates the number of activities.

Test set B consists of instance sets B_{10}, B_{20}, B_{40}, B_{100}, and B_{200} with 10, 20, 40, 100, and 200 activities, respectively, and up to 4 resources. Fündeling and Trautmann (2010) generate 480 problem instances in each set by using a factorial design of the problem parameters order strength, resource factor, and resource strength. The higher the order strength (OS) is, the more precedence relations are in the project network. OS values of 0.25, 0.5, and 0.75 are used. The resource factor (RF) indicates the number of required resources per activity (Kolisch & Sprecher, 1997). Its values are set to 0.25, 0.5, 0.75, and 1. For example, a value of 0.5 indicates that each activity requires 2 out of the 4 resources. The resource strength (RS) measures the scarcity of resources by comparing the resource requirements to the resource availability (Kolisch & Sprecher, 1997). RS values of 0, 0.25, 0.5, and 0.75 are used. A lower value indicates a higher scarcity. For RS = 0, there is for each resource at least one activity that may exclusively occupy the resource due to the upper bound of resource usage $\overline{q}_{ir}$ (see Fündeling & Trautmann, 2010). For RS = 1, the resource availability does not constrain the scheduling. The minimum block length is randomly assigned to values between 2 and 4 periods.

In each problem instance, all activities require the same principal resource $\hat{r}$. To ensure that a feasible solution exists we follow Naber and Kolisch (2014) and set for dependent resource r the resource function coefficients $\alpha_{ir} = (\overline{q}_{ir} - \underline{q}_{ir})/(\overline{q}_{i\hat{r}} - \underline{q}_{i\hat{r}})$ and $\beta_{ir} = \underline{q}_{ir} - \underline{q}_{i\hat{r}}\alpha_{ir}$. Before solving a problem instance, the preprocessing of Naber and Kolisch (2014) is applied.

6.3. Parameter settings

The parameters of the HM have been determined in a prestudy. To adapt the GA to different problem sizes, we set its population size according to the function $\min(10{\cdot}n, 400)$ of the number of activities n. We use a mutation rate for the activity list of $p_\lambda = 5\%$, which is in line with the values used by Hartmann (1998), as well as $p_\sigma = 0.5\%$ and $p_\rho = 5\%$. We facilitate the interplay between GA and VNS by adapting the number of schedules as termination criterion based on the problem size. The VNS schedule limit is set to $\Omega_{VNS} = \lfloor \Omega \cdot \max(n/200, 0.25) \rfloor$ schedules, the GA schedule limit is set to $\Omega_{GA} = \Omega - \Omega_{VNS}$. In the VNS, we use a limit of 1000 non-improving schedules per incumbent solution, 200 generated schedules per neighborhood, a maximum neighborhood of 5, and a limit of 5 longest paths in the activity selection.

For the SGA a population size of $\min(5{\cdot}n, 200)$ is used. The activity lists in the initial population are constructed in the same manner as in the HM. The SGS flag is set randomly with the equal probability to serial or parallel. The mutation rate for the activity list and for the SGS flag are both set to 5% according to Hartmann (2002).

6.4. Computational results

In the following, whenever stating statistical significance we refer to the $\alpha = 0.05$ significance level verified in a Kruskal–Wallis one-way analysis of variance or a Mann–Whitney U test.

6.4.1. Solution quality

First, we report on the solution quality. Table 2 lists the average relative deviation Δ_{mip} from the best MIP solution and the average relative deviation Δ_{lb} from the lower bound of the makespan T_{min}. The optimal MIP solutions are obtained for all instances of set B_{10}, whereas for sets B_{20}, B_{40}, and $A_{\leq 55}$ not all MIP solutions are optimal. For the larger sets B_{100} and B_{200} no MIP solutions are available. For each instance set, the average results for schedule limits Ω of 1000, 5000, 15,000, and 25,000 schedules are given and the average result across all schedule limits is set in bold. The last row of the table provides the overall results across all instance sets and schedule limits. The differences between the methods are statistically significant in all rows.

Considering the overall results, the HM yields better results than all other methods with statistical significance. The advantage of the HM is largest on data set B_{10}, where all MIPs are solved to optimality and Δ_{mip} equals the optimality gap. For 25,000 schedules the HM's optimality gap of 0.14% is about 9 times lower than

Table 2
Average deviation in percent from best MIP solution (Δ_{mip}) and from T_{min} (Δ_{lb}).

	HM		SGA		PRS		SRS	
Set / Ω	Δ_{mip}	Δ_{lb}	Δ_{mip}	Δ_{lb}	Δ_{mip}	Δ_{lb}	Δ_{mip}	Δ_{lb}
$A_{\leq 55}$	**0.03**	**5.60**	**2.12**	**7.82**	**2.64**	**8.41**	**3.89**	**9.83**
1000	0.64	6.27	2.52	8.30	3.26	9.09	4.66	10.68
5000	0.01	5.58	2.13	7.83	2.67	8.45	3.95	9.89
15,000	−0.24	5.30	1.95	7.63	2.37	8.10	3.57	9.47
25,000	−0.27	5.25	1.88	7.53	2.26	7.98	3.40	9.28
B_{10}	**0.24**	**5.40**	**1.31**	**6.57**	**1.55**	**6.81**	**1.59**	**6.88**
1000	0.44	5.62	1.34	6.61	1.56	6.83	1.62	6.91
5000	0.22	5.38	1.30	6.56	1.55	6.82	1.58	6.87
15,000	0.15	5.31	1.30	6.56	1.55	6.81	1.58	6.87
25,000	0.14	5.28	1.30	6.56	1.55	6.81	1.58	6.87
B_{20}	**0.28**	**4.24**	**0.75**	**4.77**	**1.28**	**5.34**	**1.31**	**5.40**
1000	0.60	4.60	0.86	4.90	1.63	5.73	1.70	5.84
5000	0.27	4.23	0.75	4.78	1.31	5.37	1.30	5.39
15,000	0.15	4.10	0.70	4.71	1.12	5.16	1.15	5.23
25,000	0.10	4.04	0.69	4.70	1.07	5.10	1.09	5.16
B_{40}	**−1.88**	**4.08**	**−1.73**	**4.29**	**−0.05**	**6.21**	**0.32**	**6.74**
1000	−1.66	4.35	−1.53	4.55	0.49	6.84	0.93	7.44
5000	−1.88	4.09	−1.72	4.31	−0.01	6.26	0.42	6.85
15,000	−1.98	3.96	−1.82	4.18	−0.29	5.94	0.04	6.41
25,000	−2.01	3.93	−1.85	4.13	−0.40	5.81	−0.09	6.26
B_{100}		**3.94**		**4.05**		**7.15**		**8.09**
1000		4.07		4.21		7.68		8.68
5000		3.94		4.08		7.21		8.14
15,000		3.89		3.97		6.92		7.84
25,000		3.87		3.93		6.80		7.71
B_{200}		**3.41**		**3.55**		**7.12**		**8.20**
1000		3.46		3.65		7.53		8.67
5000		3.40		3.57		7.16		8.27
15,000		3.39		3.51		6.95		7.98
25,000		3.39		3.48		6.83		7.88
Overall	**−0.33**	**4.46**	**0.63**	**5.20**	**1.37**	**6.86**	**1.81**	**7.55**

Table 3
Average time to generate 1000 schedules in seconds.

Set	HM	GA	VNS	SGA	PRS	SRS
$A_{\leq 55}$	0.14	0.11	0.24	0.05	0.06	0.02
B_{10}	0.08	0.08	0.17	0.04	0.04	0.02
B_{20}	0.20	0.18	0.28	0.09	0.09	0.05
B_{40}	0.50	0.38	0.65	0.21	0.21	0.10
B_{100}	1.55	1.19	2.72	0.84	0.69	0.31
B_{200}	4.44	3.58	9.08	2.61	1.85	0.70
Overall	**1.15**	**0.92**	**2.19**	**0.64**	**0.49**	**0.20**

those of the other methods. The HM further improves when increasing Ω from 15,000 to 25,000 schedules, whereas the other methods are stagnant or likely trapped to suboptimality. For the medium data sets $A_{\leq 55}$ and B_{40}, the HM finds many new best-known solutions. On the large data sets B_{100} and B_{200}, the HM and the SGA clearly outperform the random sampling heuristics.

6.4.2. Computation time

The average computation time in seconds required to generate 1000 schedules per problem instance (based on the average for 25,000 generated schedules) is listed in Table 3. Although not perfectly linear, the HM scales well. By doubling the number of activities, the HM's computation time grows by a constant factor of 2.5, when considering the range from 10 to 100 activities. For 200 activities, the factor slightly increases to 2.86. The HM's VNS requires more time than the HM's GA due to the repeated generation and analysis of resource flows in the activity selection. As the VNS is not executed on all solutions, its impact on the computation time of the HM is small. The key driver for computation time is the used SGS, as the FSGS performs more complex operations to determine the resource quantity, compared to the other SGSs (see Section 5.1.2).

6.4.3. Instance parameters

Let us now assess the influence of the instance parameters on the solution quality. Fig. 8 visualizes the gap to T_{min} for different values of order strength (OS), resource factor (RF), resource strength (RS), and minimum block length. The averages across all four schedule limits Ω for instance sets B_{10} to B_{200} are plotted in Fig. 8.

The order strength has a negligible impact on the solution quality, in line with the results of Fündeling (2006). The impact of the minimum block length is similarly small. The resource strength and the resource factor have a significant impact on the solution quality. These observations are similar to those for the RCPSP (see Kolisch, Sprecher, & Drexl, 1995). For instances which are highly resource-constrained, i.e. for which RS = 0 holds, the advantage of the HM and the SGA over the random sampling heuristics is most distinct.

6.4.4. Components of the hybrid metaheuristic

For the different HM variants, Table 4 lists the gap to T_{min} averaged across all four schedule limits Ω. The difference in the results is statistically significant in all rows of the table. Considering pairwise comparisons for the overall results in the last row, the GA-FSGS generates better results than the MP-FSGS and the GA-SGS with statistical significance. This clearly demonstrates the positive impacts of the GA (between GA-FSGS and MP-FSGS) and the FSGS (between GA-FSGS and GA-SGS). The improvements of the complete HM compared to the GA-FSGS are statistically insignificant when considering all instances. However, this analysis also includes instances on which the VNS is not even executed. Since

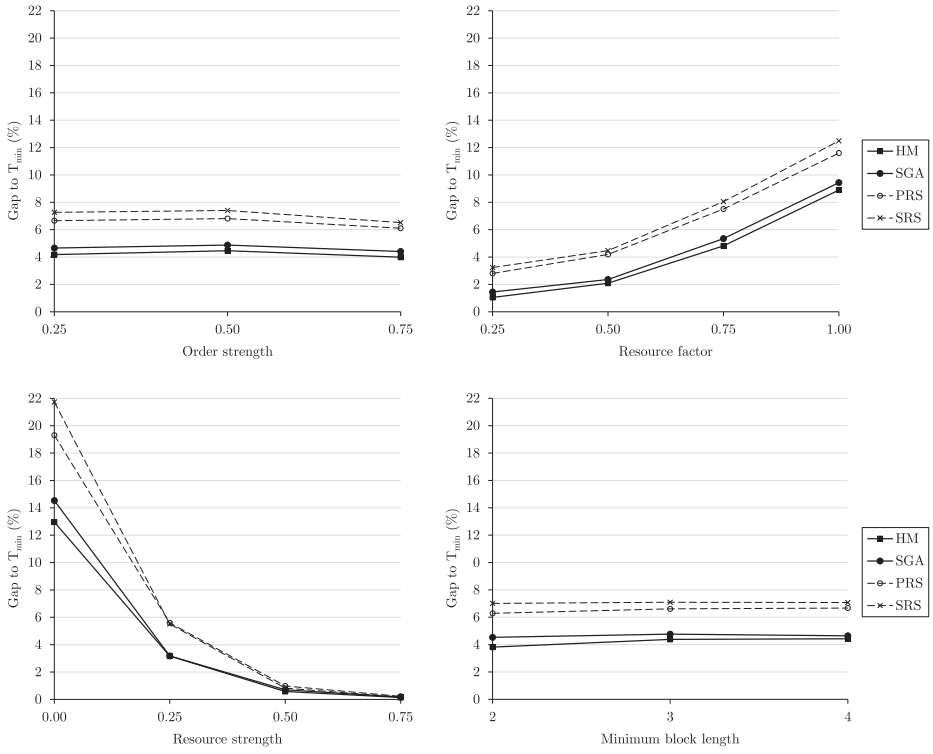

Fig. 8. Influence of instance parameters on solution quality.

Table 4
Average gap to T_{min} in percent.

Set	HM	GA-FSGS	MP-FSGS	GA-SGS
$A_{\leq 55}$	5.60	5.69	6.92	7.88
B_{10}	5.40	5.41	5.60	6.85
B_{20}	4.24	4.28	5.02	5.09
B_{40}	4.08	4.17	5.84	4.54
B_{100}	3.94	4.09	5.23	4.16
B_{200}	3.41	3.58	4.24	3.69
Overall	**4.46**	**4.55**	**5.49**	**5.39**

Table 5
VNS: Improved instances as % of the instances on which VNS is executed (Inst %) and makespan reduction per improved instance in periods (ΔC_{max}) and in percent (ΔC_{max}%).

Set	Inst %	ΔC_{max}	ΔC_{max}%
$A_{\leq 55}$	5.59	1.06	2.27
B_{10}	1.56	1.06	2.44
B_{20}	7.57	1.23	1.39
B_{40}	20.63	1.34	0.85
B_{100}	54.24	2.12	0.57
B_{200}	61.84	3.84	0.55
Overall	**22.91**	**2.58**	**1.34**

the VNS operates within the HM after completion of the GA, the VNS is not started in case the GA has already solved an instance to optimality.

The potential of the VNS becomes visible when considering only the instances on which the VNS is actually executed. Only for this subset of instances, Table 5 lists the percentage of instances for which the HM including VNS yields a lower makespan than the GA-FSGS (Inst %). The larger the problem size, the more instances are improved by the VNS, peaking at 62% on instance set B_{200}. Note that the VNS schedule limit Ω_{VNS} is also highest on this instance

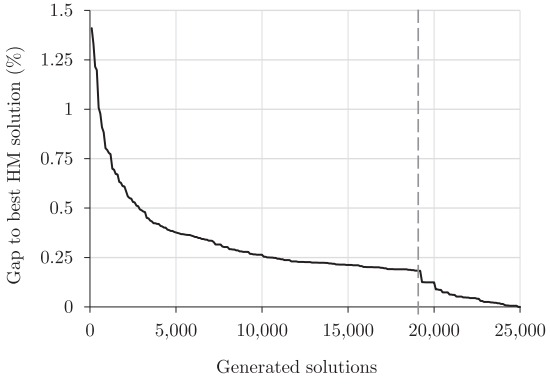

Fig. 9. Convergence of the HM.

set. On average, the VNS improves 23% of the considered instances, which corresponds to 12% of all instances. Table 5 also provides the absolute makespan reduction per improved instance in periods (ΔC_{max}) and in percent of the GA-FSGS's makespan (ΔC_{max}%).

Next, we analyze the characteristics of the best solutions obtained by the HM regarding the use of the resource allocation limit and the start delay. On average, 59% of the best solutions generated by the HM employ a resource allocation limit $\rho_i > 0$ and 25% a start delay $\sigma_i > 0$.

Finally, Fig. 9 provides the average gap to the best solution found as a function of the generated schedules for instance sets B_{10} to B_{200} and $\Omega = 25{,}000$. Since the VNS's start is determined by its variable schedule limit Ω_{VNS}, the vertical dashed line indicates the earliest start at 18,000 generated schedules. Until 18,000 generated schedules the GA shows a typical convergence behavior. From 18,000 generated schedules on, the VNS leads to a considerable further improvement of the solutions.

Table 6
Solution quality for the FRCPSP with discrete resources.

	HM		GA-FSGS	
	Δ_{mip}	Δ_{lb}	Δ_{mip}	Δ_{lb}
B_{10}	**0.29**	**5.77**	**0.31**	**5.79**
1000	0.55	6.06	0.58	6.09
5000	0.26	5.74	0.28	5.76
15,000	0.18	5.64	0.19	5.66
25,000	0.18	5.65	0.19	5.66
B_{20}	**−0.04**	**4.56**	**−0.02**	**4.58**
1000	0.26	4.89	0.27	4.91
5000	−0.03	4.57	−0.02	4.58
15,000	−0.17	4.41	−0.14	4.44
25,000	−0.22	4.36	−0.19	4.39
B_{40}	**−2.01**	**4.47**	**−2.01**	**4.47**
1000	−1.67	4.86	−1.77	4.76
5000	−2.04	4.43	−2.02	4.45
15,000	−2.14	4.31	−2.10	4.36
25,000	−2.18	4.26	−2.14	4.31
B_{100}		**4.21**		**4.34**
1000		4.34		4.42
5000		4.20		4.34
15,000		4.17		4.30
25,000		4.15		4.29
Overall	**−0.58**	**4.75**	**−0.57**	**4.79**

6.5. Application to the FRCPSP with discrete resources

Beside the continuous resources (CR) addressed in this paper, there are practical cases in which resource quantities are not infinitely divisible, such as those addressed in Fündeling and Trautmann (2010) and Baumann et al. (2015). In this section, we demonstrate, as a preliminary attempt, how the proposed HM and its GA-FSGS variant can also be applied to solve the FRCPSP with discrete resources (DR). It is important to note that despite dealing with discrete resources, the DR computational results cannot be compared with those of Fündeling and Trautmann (2010) and Baumann et al. (2015), since their problems specifically allocate resources in the exact amount required by each activity, while our FRCPSP allows resources to be allocated at least by the amount required by each activity. In other words, more resources may be assigned to activities, if the overall makespan can be reduced, while attempting to satisfy the required minimum block length.

The main variable that differentiates DR from CR is the assigned resource quantity q_{irt}, as obtained in Algorithm 1 of the FSGS. We, therefore, transform a continuous value of q_{irt} to an integer value by either rounding up or down to respectively observe the resource requirements of each activity and the limited availability of each resource. Additionally, for the VNS to yield feasible DR solutions, we assume, for each activity i and resource r, integer values of the resource requirement w_{ir} and both lower and upper bounds of resource quantity, $\underline{q}_{ir}$ and $\overline{q}_{ir}$. These conditions result in an integer value of the maximum quantity ϑ_{ijrt} that may be transferred for each activity pair (i, j) by the VNS as explained in Section 5.3.2.

We conduct pilot DR runs whose computational results of solving test sets B_{10}, B_{20}, B_{40}, and B_{100} are summarized in Table 6. Note that the VNS is applicable to these instance sets as they uphold the aforementioned integer conditions of resource requirements and bounds. The rounding operations for DR are expected to result in a negligible increment in runtimes. Similar to Table 2, Table 6 reports the solution quality in terms of the average deviations from the best MIP DR solution and from the lower bound of the makespan T_{min} for both HM and GA-FSGS . While the reference T_{min} values remain unaffected, the best MIP DR solutions are obtained from solving the best model FP-DT3 of Naber and Kolisch (2014), however, with the integrality imposed on all resource quantity variables. Despite the fact that the MIP DR model is generally more difficult to solve than the CR one, all MIP solutions are found optimal for test set B_{10}. For each of those instances whose MIPs are not solved to optimality, the best makespan or, otherwise, the upper bound of the makespan obtained from the DR modified heuristic of Naber and Kolisch (2014) is used as the MIP reference value instead.

Due to the additional integrality of resource variables, the average gap Δ_{lb} of DR is, as expected, slightly higher than that of CR across all test sets. The average gap Δ_{mip} of DR is, on the other hand, slightly better than that of CR in all test sets except B_{10}. This favorable outcome is attributed to the lower quality of MIP solutions found within limited runtime.

The effectiveness of the VNS is also reported by the percentage of instances on which the VNS is executed and for which the HM including VNS yields a lower makespan than the GA-FSGS. On set B_{10} 2.14% of the instances are improved, on set B_{20} 7.57%, on set B_{40} 22.26%, and on set B_{100} 58.42%, leading to an average improvement of 22.60% of all instances on which the VNS is executed. The results also verify that the average VNS improvement is positively correlated to the problem size, as in the continuous case.

Despite the valid applicability of the HM and GA-FSGS to deal with discrete resources, it should be emphasized that the proposed HM and especially the VNS are designed for continuous resources and that, as it is out of scope of this paper, more tailoring of the HM and the VNS is recommended for further study to improve the solution quality for the FRCPSP with discrete resources and beyond, for example, mixed-types of resources.

7. Conclusions

We proposed a Hybrid Metaheuristic (HM) for the resource-constrained project scheduling problem with flexible resource profiles. It uses the Flexible Resource Profile Parallel Schedule Generation Scheme (FSGS) which employs the concepts of delayed scheduling and non-greedy resource allocation to construct feasible schedules. The FSGS generates significantly lower makespans for the FRCPSP than a standard parallel schedule generation scheme. The best schedules are further improved in a novel variable neighborhood search by transferring resource quantities between activities that are selected based on an analysis of resource flows. Overall, the HM yields significantly better results than three benchmark (meta-)heuristics and generates near-optimal solutions in short computation time.

Further research may include an extension of the HM to deal with both discrete and continuous resources, incorporation of a mathematical model to optimally allocate resources, and perhaps an extended application of the HM to the FRCPSP in continuous time.

Acknowledgments

The authors would like to thank Norbert Trautmann for providing the test instances and the anonymous reviewers for giving comments and suggestions. The support of Anulark Naber by the German Research Foundation (Deutsche Forschungsgemeinschaft) under grant NA 1059/1-1 is graciously acknowledged.

References

Artigues, C., Demassey, S., & Neron, E. (2008). Resource-constrained project scheduling: Models, algorithms, extensions and applications. *Control systems, robotics and manufacturing series*. London and Hoboken: ISTE and Wiley.

Artigues, 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.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms and their application* (pp. 14–21). Hillsdale, USA: L. Erlbaum Associates.

Baumann, P., Fündeling, C.-U., & Trautmann, N. (2015). The resource-constrained project scheduling problem with work-content constraints. In C. Schwindt, & J. Zimmermann (Eds.), *Handbook on project management and scheduling*. In *International handbooks on information systems: 1* (pp. 533–544). Springer International Publishing.

Fündeling, C.-U. (2006). Ressourcenbeschränkte Projektplanung bei vorgegebenen Arbeitsvolumina (Resource-constrained project scheduling with given work contents). *Gabler Edition Wissenschaft Produktion und Logistik*. Wiesbaden: Dt. Univ.-Verl.

Fündeling, C.-U., & Trautmann, N. (2010). A priority-rule method for project scheduling with work-content constraints. *European Journal of Operational Research, 203*(3), 568–574.

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. *Series of books in the mathematical sciences*. San Francisco: W.H. Freeman.

Hansen, P., & Mladenović, N. (2005). Variable neighborhood search. In E. Burke, & G. Kendall (Eds.), *Search methodologies* (pp. 211–238). US: Springer.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics, 45*(7), 733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics, 49*(5), 433–448.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

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., Meyer, K., Mohr, R., Schwindt, C., & Urmann, M. (2003). Ablaufplanung für die Leitstrukturoptimierung in der Pharmaforschung (Scheduling of lead structure optimization in pharmaceutical research). *Zeitschrift für Betriebswirtschaft, 73*(8), 825–848.

Kolisch, R., & Sprecher, A. (1997). PSPLIB - a project scheduling problem library. *European Journal of Operational Research, 96*(1), 205–216.

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

Kuhlmann, A. (2003). *Entwicklung eines praxisnahen Project-Scheduling-Ansatzes auf der Basis von genetischen Algorithmen (Development of a practical project scheduling approach based on genetic algorithms)*. Berlin: Logos-Verlag.

Naber, A., & Kolisch, R. (2014). MIP models for resource-constrained project scheduling with flexible resource profiles. *European Journal of Operational Research, 239*(2), 335–348.

Naber, A., & Kolisch, R. (2015). MIP models for resource-constrained project scheduling with flexible resource profiles: Comparisons between Baumann & Trautmann (2013) and Naber & Kolisch (2014). *Technical Report*. TUM School of Management, Technical University of Munich.

Raidl, G. R., Puchinger, J., & Blum, C. (2010). Metaheuristic hybrids. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics*. In *International series in operations research & management science: 146* (pp. 469–496). New York: Springer.

Ranjbar, M., & Kianfar, F. (2010). Resource-constrained project scheduling problem with flexible work profiles: A genetic algorithm approach. *Scientia Iranica, 17*(1), 25–35.

Schramme, T. (2014). *Modelle und Methoden zur Lösung des ressourcenbeschränkten Projektablaufplanungsproblems unter Berücksichtigung praxisrelevanter Aspekte (Models and methods to solve the resource-constrained project scheduling problem under consideration of practical aspects)*. Paderborn: Paderborn University.

Tritschler, M., Naber, A., & Kolisch, R. (2014). A genetic algorithm for the resource-constrained project scheduling problem with flexible resource profiles. In T. Fliedner, R. Kolisch, & A. Naber (Eds.), *Proceedings of the 14th international conference on project management and scheduling* (pp. 230–233). Munich: TUM School of Management.

van Peteghem, V., & Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research, 235*(1), 62–72.

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes – a survey. *European Journal of Operational Research, 208*(3), 177–205.

Zimmermann, A. (2016). An MIP-based heuristic for scheduling projects with work–content constraints. In *2016 IEEE international conference on industrial engineering and engineering management (IEEM)* (pp. 1195–1199).