European Journal of Operational Research 313 (2024) 92–111

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Hybrid differential evolution algorithm for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources

T. van der Beek[a,*], D. Souravlias[c], J. T. van Essen[b], J. Pruyn[a], K. Aardal[b]

[a] Maritime and Transport Technology, Delft University of Technology, Mekelweg 2, 2628 CD, Delft, the Netherlands
[b] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, the Netherlands
[c] Digital & Information Technology Dept., Port of Rotterdam, World Port Center, Wilhelminakade 901, Rotterdam, 3072 AR, the Netherlands

ARTICLE INFO

Article history:
Received 19 May 2022
Accepted 31 July 2023
Available online 4 August 2023

Keywords:
Project scheduling
Metaheuristics
Resource constrained project scheduling problem
Flexible project structure

ABSTRACT

The resource constrained project scheduling problem with a flexible project structure and consumption and production of resources, involves making a selection of activities and scheduling these activities in order to minimize the makespan, subject to precedence and resource constraints. Since finding a feasible selection of activities is NP-hard, we introduce the concept of group graphs and restrict ourselves to instances with an acyclic group graph. For these instances, which represent many practical cases, we show how to make a feasible selection of activities in polynomial time and use this concept to schedule the selected activities using a hybrid differential evolution algorithm. We compare this algorithm with an algorithm from the literature on special cases of instances without consumption and production of resources, and show that our algorithm creates solutions of higher quality. Furthermore, to compare general instances, we develop an ant colony optimization algorithm that performs slightly better on special cases than the algorithm from literature and show that the hybrid differential evolution algorithm outperforms the ant colony optimization algorithm on general instances.

1. Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is an extensively studied optimization problem with the goal of minimizing the total execution time of a project, subject to precedence and resource constraints. It is widely applicable and has been used in many industries, such as shipbuilding (Hu, Zhang, Wang, Kao, & Ito, 2019), housing construction (Bezerra & Scheer, 2021), employee scheduling (Bellenguez & Néron, 2005), etc. In the standard RCPSP, the list of activities is fixed and all activities have to be scheduled while being constrained by renewable resources, i.e., resources that become fully available again after an activity is done. Examples of renewable resources are workers, machines and cranes.

However, in reality, these assumptions are not always valid due to multiple reasons. First of all, in many construction projects, there are multiple ways of completing projects. Secondly, not all resources are renewed automatically. An example is given from shipbuilding, where assemblies can be produced beforehand in workshops, or directly on the ship (Fafandjel, Rubeša, & Mrakovčic, 2008). After production in a workshop, an assembly has to be installed as a whole on the ship, requiring specialized resources such as cranes. Conversely, direct production on the ship does not require this, although the less ideal working conditions might require better trained workers. Furthermore, when choosing to use sub-assemblies, the resource 'floor space' is required for storage, which does not renew automatically. Instead, an activity 'moving product' or 'installing sub-assembly on ship', with non-zero duration, has to be executed to free up floor space. Another application of non-renewable resources is a limit on the amount of capital tied up or used in a project (Neumann & Schwindt, 2003). Since the capital use might vary based on the selected activities, certain project structures might be infeasible due to different resource requirements. Therefore, it is important to consider these types of resources when studying flexible resource structures (Kellenbrink & Helber, 2015).

* Corresponding author.
E-mail addresses: T.vanderBeek@tudelft.nl (T. van der Beek), D.Souravlias@portofrotterdam.com (D. Souravlias), J.T.vanEssen@tudelft.nl (J.T. van Essen), J.F.J.Pruyn@tudelft.nl (J. Pruyn), K.I.Aardal@tudelft.nl (K. Aardal).

https://doi.org/10.1016/j.ejor.2023.07.043

These additional aspects give rise to the *Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources* (RCPSP-PS/CPR), a generalization of the RCPSP with two extensions.

Firstly, we introduce the flexible project structure. The standard RCPSP answers the question: *At what time should each activity be executed?* The flexible project structure adds another problem: *Which activities should be executed?* This latter problem is called the *selection problem* and was proven to be NP-hard by Barták, Čepek, & Surynek (2007). Secondly, nonrenewable resources with consumption and production, called cumulative resources, are introduced. As opposed to renewable resources, which are fully available after use, cumulative resources are either consumed or produced by an activity. This complicates the problem, since an instance with cumulative resources does not always have a feasible solution. With renewable resources, activities can always be delayed until enough resources are available. With cumulative resources, this is not the case and determining if an instance has a feasible solution is NP-hard (Neumann & Schwindt, 2003). Although both the flexible project structure and the cumulative resources have been studied separately, to the best of our knowledge, the RCPSP-PS/CPR has not been studied.

This paper has three contributions. Firstly, we adapt the *Mixed Integer Linear Programming* (MILP) formulation of Van der Beek, Van Essen, Pruyn, & Aardal (2022) by adding cumulative resources. Secondly, group orderings are introduced; a fast method to find feasible solutions to the selection problem, to be used within heuristics. This method is only applicable to instances with certain characteristics, which are discussed in this paper. However, these characteristics are present in many practical instances of the RCPSP-PS/CPR. Finally, we present a *Hybrid Differential Evolution* (HDE) algorithm and evaluate its performance against special cases from the literature, optimal solutions and an *Ant Colony Optimization* (ACO) benchmark algorithm.

In Section 2, we present an overview of the literature related to the RCPSP-PS/CPR. Subsequently, we formulate the problem in Section 3. Then, we present a method to create feasible selections and use this to create the HDE algorithm and the ACO algorithm in Section 4. This is followed by a comparison of computational results in Section 5 and conclusions in Section 6.

2. Literature review

The RCPSP is introduced by Pritsker, Watters, & Wolfe (1969) and proven to be NP-hard by Blazewicz, Lenstra, & Rinnooy Kan (1983). In this section, we focus on the extensions considered in this paper: consumption and production of resources and a flexible project structure. For these extensions, the main points of interest are modeling decisions and metaheuristics used. For the general RCPSP, an overview of hybrid metaheuristic approaches is given by Pellerin, Perrier, & Berthaut (2019). They conclude that most algorithms are population-based. Furthermore, they conclude that local-search-based permutations and *forward-backward improvements* (Wang & Lui, 2017) are most used in the best performing approaches.

We now provide a brief review on research that considers *scheduling with nonrenewable resources* or *scheduling with cumulative resources*. We use the term nonrenewable resources to denote the case of a set of resources that are consumed and not *automatically* renewed. However, in this paper, we consider *scheduling with cumulative resources*. Cumulative resources denote the case of a set of nonrenewable resources that cannot only be consumed but also produced.

When considering scheduling with cumulative resources, two types of closely related problems have to be studied: *activity-based* scheduling and *event-based* scheduling. The general RCPSP consists of activities to be planned that have a certain duration and precedence relationships. Conversely, the *Event Scheduling Problem* (ESP) consists of events; tasks with zero duration and maximal and minimal time lags to other events, where time lag is the time difference between two events. Notice that every instance of the RCPSP can be modeled as an instance of the ESP by replacing each task with a duration zero and prescribed maximal and minimal time lags to other events, while the converse is not true; the basic version of the RCPSP does not include maximum time lags between activities. Therefore, the RCPSP is a special case of the ESP.

Event-based scheduling with inventory constraints is introduced by Neumann & Schwindt (2003). Here, resources have both an upper and a lower bound; the inventory level cannot exceed its capacity and it is not allowed to drop below a certain level. They formulate the problem and answer some structural questions, e.g., the NP-completeness of the feasibility problem. Furthermore, they present a branch-and-bound procedure and evaluate its performance. The problem introduced by Neumann & Schwindt (2003) has also been investigated in a paper by Laborie (2003), who applied a constraint programming approach with consistency tests. Furthermore, Carlier, Moukrim, & Xu (2009) introduce the *project scheduling problem with consumption and production of resources* (RCPSP/CPR), which uses an event-based approach. For this problem, they provide a scheduling algorithm that computes the optimal schedule by enumerating over all linear orders of events. Koné, Artigues, Lopez, & Mongeau (2013) also consider the problem with cumulative resources. They provide a time-indexed MILP model for the discrete time RCPSP/CPR and a flow-based formulation for the continuous-time RCPSP/CPR. Furthermore, they present a formulation combining activities and events with continuous-time variables determining the occurrence times of the events and binary variables determining whether an activity is executed at the same time of an event. Similarly, Sahli, Carlier, & Moukrim (2016) present different models for the ESP with consumption and production of resources. Like Koné et al. (2013), they give a time-indexed and a flow-based formulation. Besides this, they also give an event-partitioning-based formulation.

Most papers listed above investigate lower bounds and exact formulations. However, the research on heuristic methods for the RCPSP/CPR is limited. Carlier et al. (2009) shortly discuss how the exact enumeration algorithm can be modified to obtain a heuristic method. Furthermore, Shirzadeh Chaleshtarti, Shadrokh, Khakifirooz, Fathi, & Pardalos (2020) present a genetic algorithm for the RCPSP with nonrenewable resources. This consist of the standard version of the RCPSP, with a fixed amount of initially available resources that can only be consumed and not produced. Since there is no flexible project structure and resources can only be consumed, the amount of initial resources fully defines feasibility regarding nonrenewable resources. Therefore, resource infeasibility is not taken into account for this problem.

Besides cumulative resources, the second extension is the flexible project structure. Although there are different variants under different names in the literature, the main concept of this extension is that only a subset of all activities have to be executed.

The flexible project structure is firstly introduced by Barták et al. (2007). However, they do not consider the RCPSP, but a different scheduling problem: temporal networks where resources are not considered and the goal is to satisfy maximum and minimum time lags between activities. For this problem, they introduce the problem of a flexible structure and, amongst other things, show that the selection problem is NP-hard. One of the earliest RCPSP variants with a flexible project structure is the *Extended RCPSP*, introduced by Kuster, Jannach, & Friedrich (2009). They study a disruption management problem, which they model as an RCPSP with an initial activation state and substitution criteria. These substitution criteria define what changes are allowed to the initial state.

Table 1
Overview of models with a flexible project structure.

Paper	Separate scheduling and selection	Independent succession	Exclusivity criterion	Nonrenewable resources	Cumulative resources
Kuster et al. (2009)		✓	✓		
Čapek et al. (2012)	✓		✓		
Kellenbrink & Helber (2015)	✓		✓	✓	
Tao & Dong (2017)		✓		✓	
Servranckx & Vanhoucke (2019)		✓	✓		
Van der Beek et al. (2022)	✓	✓	✓		
This paper	✓	✓	✓	✓	✓

They give a custom evolutionary algorithm to heuristically solve this problem. Furthermore, Čapek, Šcha, & Hanzálek (2012) study a variant of the RCPSP with a flexible project structure, unary resources, time-lags and sequence dependent setup times. They represent the branching structure by Petri nets and give both an MILP and a constructive heuristic algorithm. The RCPSP with a flexible project structure is studied in Kellenbrink & Helber (2015). They model this by distinguishing between mandatory and optional activities, and introduce a set of choices to decide which optional activities have to be executed. They include nonrenewable resources, but only with consumption of these resources and without production. To heuristically solve this problem, they use a genetic algorithm. Another formulation is given by Tao & Dong (2017), who represent the problem by an AND-OR project network. They call this problem the RSPCP with alternative activity chains and give an extended simulated annealing algorithm to heuristically solve it. Furthermore, in Tao & Dong (2018), they extend the problem by adding multiple modes of executing an activity and by considering multi-objective optimization. This new problem is heuristically solved by a hybrid algorithm consisting of tabu search and a genetic algorithm. Servranckx & Vanhoucke (2019) define the RCPSP with alternative subgraphs. This problem consists of branches, where each branch represents a subset of activities that can be executed. This is heuristically solved using tabu search. Furthermore, Van der Beek et al. (2022) introduce an MILP model where the choices are based on *selection-groups*; a group consisting of an activator activity and a set of successor activities. If an activator activity is executed, exactly one successor activity has to be executed. A solution method is given that uses cutting planes and constraint propagation for preprocessing, after which the problem is solved to optimality by a commercial MILP solver.

We now compare the RCPSP-PS/CPR to the problems introduced in this literature review. For this, we use three concepts from Van der Beek et al. (2022). The first concept is separate scheduling and selection. This means that the graph representing all selection constraints can be separate from the graph representing all precedence constraints. The second concept is *Independent Succession* (IS). A model with IS allows for multiple activities to simultaneously cause the execution of the same other activity. If a model does not feature IS, each activity can only be executed based on at most one other activity. Finally, the *Exclusivity Criterion* (EC) is introduced. If a model includes EC, choices are *exclusive*; only one alternative can be selected from a set of candidate activities. As shown in Van der Beek et al. (2022), is is possible to model 'at-least-one' constraints in models with EC, making models with EC the general case. With the introduced terms, the different models can be compared. This is done in Table 1 and shows that the models of Kuster et al. (2009), Kellenbrink & Helber (2015), Servranckx & Vanhoucke (2019), Van der Beek et al. (2022), Tao & Dong (2017) are special cases of the RCPSP-PS/CPR. However, the problem considered in Čapek et al. (2012) contains additional extensions, and therefore, is not a special case of the RCPSP-PS/CPR.

In conclusion, although there is various research on the different elements of the RCPSP-PS/CPR, this has not been combined yet. Furthermore, the research on heuristic methods considering cumulative resources is limited. To the best of our knowledge, no implementation of this exists.

3. RCPSP-PS/CPR

In this section, a formal description of the RCPSP-PS/CPR is given. First, Section 3.1 describes the problem and introduces all required notation. Subsequently, Section 3.2 gives an MILP formulation for the RCPSP-PS/CPR. This model is based on the model for the RCPSP-PS from Van der Beek et al. (2022). Like nearly all formulations for the RCPSP with a flexible project structure, it uses a time-indexed formulation. Therefore, we use the constraints from the time-indexed formulation of Sahli et al. (2016) to model the cumulative resources.

3.1. Problem description

The RCPSP-PS/CPR consists of a set of activities N of which a subset has to be executed. The set N consists of the starting activity 1 and the final activity $|N|$ and $|N|-2$ non-dummy activities. The starting activity 1 starts before all activities and the final activity $|N|$ starts after all activities have been finished. The objective is to minimize the total time of the executed activities, also called the *makespan* of the project. The activities are scheduled in discrete time periods T. Each activity $i \in N$ has a duration of d_i time periods. These activities have to be scheduled while satisfying resource, precedence and selection constraints.

There are two types of resources; renewable resources R^r and cumulative resources R^n. The set of all resources is denoted by $R = R^r \cup R^c$ ($R^r \cap R^c = \emptyset$). For each resource $r \in R$, activity $i \in N$ consumes k_{ri}^- amount of resource r at the start of the activity and produces k_{ri}^+ at the end of the activity. For each renewable resource $r \in R^r$, we have $k_{ri}^- = k_{ri}^+$. Furthermore, each resource has a total capacity of λ_r.

The precedence relationships are defined by the set of tuples $\mathcal{P}$. For each precedence relationship $(i, j) \in \mathcal{P}$, it is required that activity j does not start before the finishing time of activity i.

The flexible project structure is modeled by *selection groups* G. Each selection group $g \in G$ consists of an activator activity a_g and a set of successor activities S_g, with $|S_g| > 0$. For each activator group $g \in G$, it holds that if activator a_g is executed, exactly one of the successor activities S_g has to be executed.

Finding a set of executed activities such that this holds for each group is called the *selection problem*, as given in Definition 1. A representation of the selection problem is given by the *selection graph*. This graph consists of a node for each activity and an arc for each activator-successor relationship. When the selection graph is illustrated, a circular arc is drawn to denote that multiple activator-successor relationships belong to the same group. Figure 1 displays a selection graph with one selection group with multiple successors and four selection groups with one successor. The selection group with multiple successors has activator 1 and successors $\{2, 3\}$.

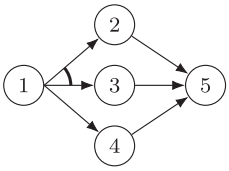


Fig. 1. Example of a selection graph.

Table 2
Notation used in Constraint set (1).

a_g	Activator activity of selection group $g \in G$.
d_i	Duration of activity $i \in N$.
k_{ri}^+	Production of resource $r \in R$ for activity $i \in N$.
k_{ri}^-	Consumption of resource $r \in R$ for activity $i \in N$.
λ_r	Capacity of resource $r \in R$.
G	Set of selection groups.
M	Sufficiently large number.
N	Set of activities.
$\mathcal{P}$	Set of precedence pairs.
R	Set of resources.
S_g	Set of successor activities of selection group $g \in G$.
T	Set of time periods.
X_{it}	Binary variable which is 1 if activity $i \in N$ is executed at time $t \in T$, zero otherwise.

Definition 1 (Selection problem)**.** The selection problem consists of finding a set of executed activities $N' \subseteq N$ with $1, |N| \in N'$, such that for each selection group $g \in G$ it holds that if activator a_g is executed ($a_g \in N'$), there is exactly one executed successor: $|S_g \cap N'| = 1$.

In order to consider all activities in the selection problem, it is imposed that a valid project structure has a path from the starting activity to each other activity. Furthermore, in order to always have the final activity executed, each maximal path in the selection graph ends in the final activity.

3.2. Problem formulation

The problem is modeled with binary decision variables X_{it} for each activity $i \in N$ and for each time $t \in T$. This variable is equal to one if activity i is started at time t and zero otherwise. Objective function (1a) minimizes the starting time of final activity $|N|$, and therefore, the time of the total project. The first activity is always executed due to Constraint (1b). Furthermore, each activity can only be started once, as imposed by Constraints (1c).

The selection problem is captured by Constraints (1d) and (1e). The former impose that if an activator a_g of selection group $g \in G$ is executed, at least one successor activity $i \in S_g$ is executed. The latter ensure that per selection group $g \in G$ with executed activator a_g, at most one successor activity $i \in S_g$ can be executed. Furthermore, the precedence relations are imposed by Constraints (1f) for each precedence relationship $(i, j) \in \mathcal{P}$. Due to these constraints, activity i has to be finished before or at the start of activity j, if both are executed. Here, M is a sufficiently large number ($M \geq |T| + \max_{i \in N} d_i$).

For each resource $r \in R$, Constraints (1g) ensure for each time $t \in T$ that the total consumption minus the total production up to time t is smaller than the total resource capacity λ_r. Finally, Constraints (1h) impose that decision variables X_{it} are binary. In Table 2, all notation is given for the model. Furthermore, in Appendix A, all notation used throughout this paper is listed.

$$\min \sum_{t \in T} t X_{|N|t}, \tag{1a}$$

$$\sum_{t \in T} X_{1t} = 1, \tag{1b}$$

$$\sum_{t \in T} X_{it} \leq 1 \qquad \forall i \in N, \tag{1c}$$

$$\sum_{t \in T} X_{a_g t} \leq \sum_{i \in S_g} \sum_{t \in T} X_{it} \qquad \forall g \in G, \tag{1d}$$

$$\sum_{j \in S_g} \sum_{t \in T} X_{jt} \leq |S_g| - \big(|S_g| - 1\big) \sum_{t \in T} X_{a_g t} \qquad \forall g \in G, \tag{1e}$$

$$\sum_{t \in T} (t + d_i) X_{it} \leq \sum_{t \in T} t X_{jt} + M\Big(1 - \sum_{t \in T} X_{jt}\Big) \quad \forall (i, j) \in \mathcal{P}, \tag{1f}$$

$$\sum_{i \in N} \left(\sum_{\tau=1}^{t} k_{ri}^- X_{i\tau} - \sum_{\tau=1}^{t-d_i} k_{ri}^+ X_{i\tau} \right) \leq \lambda_r \qquad \forall r \in R, t \in T, \tag{1g}$$

$$X_{it} \in \{0, 1\} \qquad \forall i \in N, t \in T. \tag{1h}$$

4. Solution method

In this section, a *Hybrid Differential Evolution* (HDE) algorithm is presented to find feasible solutions for the RCPSP-PS/CPR. Naturally, a search-based algorithm requires a fast way of exploring different solutions to the selection problem. This poses a challenge, since the selection problem is NP-hard (Barták et al., 2007). Therefore, Section 4.1 gives a special case of the selection problem and a polynomial-time algorithm to find feasible solutions to this special case. This algorithm is used within the HDE algorithm that is given in Section 4.2. Furthermore, since the RCPSP-PS/CPR is introduced in this paper, there is no heuristic algorithm for comparison for general instances. Therefore, in Section 4.3, we briefly introduce an *Ant Colony Optimization* (ACO) algorithm for comparison purposes.

4.1. Group orderings

As mentioned before, finding a feasible solution to the selection problem is NP-hard. However, it is often done manually by the planner in real-life cases. In this subsection, we determine the conditions for when it is possible to find feasible solutions to the selection problem in polynomial time. This is done by the introduction of *(feasible) group orderings*: a sorted list containing a subset $G' \subseteq G$. A group ordering is used to find feasible solutions for the selection problem by making sequential decisions for each group in the group ordering, while keeping track of the selected and forbidden activities. In this subsection, we first show how a group ordering is used and give the definition of a *feasible* group ordering. With this, we introduce properties to identify if a group ordering is feasible. Finally, we introduce a special case of the RCPSP-PS/CPR and show how to obtain feasible group orderings for these instances in polynomial time.

By applying Algorithm 1 on a group ordering J, a selection of

Algorithm 1 Using a group ordering J.

```
N^e ← {1}                                   ▷ Executed activities
N^f ← ∅                                     ▷ Forbidden activities
for g ∈ J do
    if a_g ∈ N^e then
        N_g^c ← S_g \ N^f                   ▷ Candidate activities
        if N_g^c ≠ ∅ then
            n ← Select from N_g^c
            N^e ← N^e ∪ {n}
            N^f ← N^f ∪ (S_g \ {n})
        end if
    end if
end for
return N^e
```

activities is obtained. As it is explained below, this selection may or

may not be feasible, depending on certain instance properties. The algorithm keeps track of a list of executed activities N^e and a list of forbidden activities N^f. Then, following the order of group ordering J, each selection group $g \in J$ is evaluated. First, it is checked if the activator a_g is executed. If this is the case, a successor has to be selected. The set of candidate activities N_g^c consists of the set of successor activities S_g, minus the set of forbidden activities. From these candidate activities, one activity n is selected. This selection is based on the algorithm in which Algorithm 1 is used, as is explained in Sections 4.2 and 4.3. Finally, activity n is added to the set of executed activities and all other successors $S_g \backslash \{n\}$ are added to the set of forbidden activities.

As stated in Definition 2, a group ordering is *feasible* if Algorithm 1 always results in a feasible selection of activities, regardless of the selection made in line 7. From the selection problem, as defined by Constraints (1d) and (1e), there are two ways in which a selection can be infeasible for group g: the activator a_g is selected and either none of the successors in S_g are selected, or more than one successor is selected. Since Algorithm 1 keeps track of forbidden activities, the latter case will not happen. The first case can happen in two ways: The set of candidate activities is empty, or an activator a_g is selected for execution after group g has been processed. Based on this, Observation 1 gives a condition for feasibility of a group ordering.

Definition 2 (Feasible group ordering)**.** A feasible group ordering J is an ordered list of selection groups such that applying Algorithm 1 always returns a feasible solution to the selection problem, regardless of the selection choice in line 7. If J does not contain all selection groups G, it is called a *partial* group ordering.

Observation 1. A (partial) group ordering $J = [j_1, \cdots, j_{|J|}]$ is feasible (i.e., Algorithm 1 always returns a feasible activity selection) if it satisfies the following properties:

> **Property 1.** There exist no groups $j_a, j_b \in J$ with $a < b$, such that
>
> $$a_{j_a} \in S_{j_b}. \tag{2}$$
>
> **Property 2.** There does not exist a group $j_a \in J$ such that
>
> $$S_{j_a} \subseteq \bigcup_{i \in \{1,\cdots,a-1\} | S_{j_i} \neq S_{j_a}} S_{j_i}. \tag{3}$$

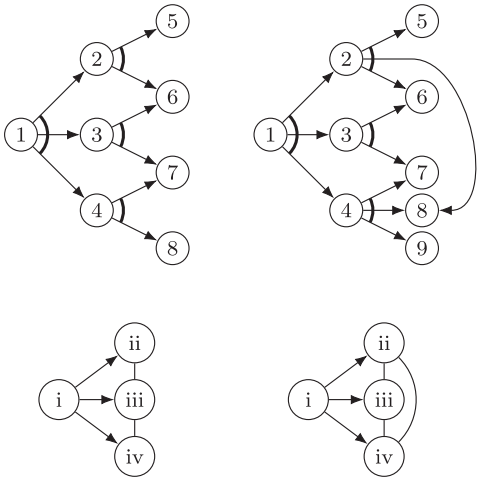
Proof. By Property 1, there will be no group $g \in G$ where a decision has to be made before all groups containing a_g as a successor are considered.

Furthermore, due to Property 2, for each group j_a, there is a successor activity $j \in S_{j_a} \setminus \cup_{i=1}^{a-1} S_{j_i} \neq \emptyset$ that is not a successor activity of an earlier group and therefore can be chosen without causing a conflict. If any earlier group has exactly the same successors, making a choice here automatically results in no conflicts arising when making a choice for j_a. Therefore, these can be excluded from the right-hand side of Eq. (3). □

In order to know if a feasible group ordering exists, we introduce the *group graph*; a graph with a node for each selection group, and edges based on activator-successor relationships. For this, we first introduce the property called *strict successor containment*.

Definition 3 (Strict successor containment)**.** A selection group $g \in G$ has strict successor containment on selection group $h \in G$, if the successors of selection group h are a strict subset of the successors of selection group g; i.e., $S_h \subsetneq S_g$.

Definition 4 states how to create a group graph.



(a) Acyclic group graph. (b) Cyclic group graph.

Fig. 2. Instances with corresponding group graphs.

Definition 4 (Group graph)**.** A group graph $\mathcal{H}$ is a mixed (containing directed and undirected edges) graph based on the selection groups. It is created in the following way:

1. Create a node for each selection group $g \in G$.
2. Create a directed edge from group $g \in G$ to group $h \in G$ if the activator of group h is a successor of group g; i.e., $a_h \in S_g$.
3. If two selection groups $g \in G$ and $h \in G$ have successor overlap ($S_g \cap S_h \neq \emptyset$) and the size of at least one successor set is larger than one ($|S_h| > 1$ or $|S_g| > 1$), create an undirected edge between node g and h.
4. If group $g \in G$ has strict successor containment on group $h \in G$,(i.e., $S_h \subsetneq S_g$), create a directed edge from node h to node g.

If the group graph is acyclic, the corresponding RCPSP-PS/CPR instance has a feasible group ordering, as stated in Theorem 1. The intermediate theory, including the method to create a feasible group ordering from a group graph, is given in Appendix B.

Theorem 1. *If an instance of the RCPSP-PS/CPR has an acyclic group graph, a feasible group ordering can be found in polynomial time.*

Proof. To obtain a feasible group ordering, we execute the following steps:

1. Construct a group graph $\mathcal{H}$, as described in Definition 4.
2. Find a breadth-first ordering for each connected component in $\mathcal{H}$, as described in Lemma 2.
3. Topologically sort the connected components, as described in Lemma 3.

Step 1 consists of 4 operations, described in Definition 4. The last two of these 4 operations have the highest time complexity, namely $O(|G|^2|N|)$. Step 2 and 3 can both be solved by a breadth-first ordering algorithm, which has a time complexity of $O(|G|^2)$. Therefore, the combined time complexity is $O(|G|^2|N|)$. Furthermore, Lemma 3 shows that this group ordering is feasible. □

The heuristics introduced in this paper require a group ordering to create feasible solutions to the selection problem. As Theorem 1 states, this group ordering can be found if the group graph is acyclic. An example is given in Fig. 2. Here, Fig. 2a shows an instance with an acyclic group graph. Therefore, this instance can be solved by the methods of this paper. Conversely, Fig. 2b shows an instance with a cyclic group graph: groups $ii \rightarrow iii \rightarrow iv \rightarrow ii$ form a cycle. Therefore, we cannot use the

breadth-first ordering to find a feasible group ordering. This means that Algorithm 1 is not guaranteed to find a feasible selection, and thus, this instance cannot be solved by the methods proposed in this paper. However, there is no guarantee that there is no feasible group ordering. In the example in Fig. 2b, [*i*, ii, iii, iv] is a feasible group ordering that cannot be found by the proposed method. The reason that this group ordering cannot be found with group graphs, is because there are instances with the same group graph, where this group ordering is not feasible. Indeed, if activity 9 is removed, the group graph remains unchanged. However, group ordering [*i*, ii, iii, iv] does not fulfill Property 2 from Theorem 1, and thus, is infeasible.

4.2. Hybrid differential evolution algorithm

The algorithm introduced is a *Hybrid Differential Evolution* (HDE) algorithm with so-called *Forward Backward Improvement* (FBI). The Differential Evolution algorithm, originally introduced by Storn & Price (1997), was chosen based on multiple successful implementations in the literature (Quoc, The, Doan, & Thanh, 2020; Sallam, Chakrabortty, & Ryan, 2020; Zaman, Elsayed, Sarker, Essam, & Coello Coello, 2021) for RCPSP variants and on good preliminary results.

First, we introduce the solution representation. Subsequently, the main *Differential Evolution* (DE) algorithm is given, which forms the core of the HDE algorithm. Finally, the complete HDE algorithm including FBI is presented.

4.2.1. Solution representation

Since DE is an algorithm for continuous variables, a method to convert a DE solution into a schedule is required. Each agent in the DE algorithm is represented by a $2 \times |N|$ *priority matrix* A that contains two priorities (scalars) per activity. The *selection priority* for activity $i \in N$ is denoted by A_{1i}. This is used to make a selection of executed activities. Furthermore, A_{2i} is the *scheduling priority*, which is used to schedule the selected activities.

To convert a priority matrix A into a feasible schedule, we first determine the executed activities. This is done by using a feasible group ordering J and applying Algorithm 1. In this algorithm, on line 7, the successor with the highest *selection* priority is selected. This results in a set of executed activities N^e.

Next, these executed activities are scheduled according to Algorithm 2. Here, the activities are scheduled iteratively. In each

Algorithm 2 Scheduling activities N^e.

1: $N^a = \{1\}$ ▷ Available activities
2: $N^s = \emptyset$ ▷ Scheduled activities
3: $\mathbf{t} = [0 \text{ for } i \in N^e]$ ▷ Time vector
4: **while** $|N^a| > 0$ **do**
5: $n \leftarrow$ Select from N^a
6: $N^s = N^s \cup \{n\}$
7: $N^a = N^a \cup \{j : j \in N^e \backslash (N^a \cup N^s), P_j \cap N^e \subseteq N^s\} \backslash \{n\}$
8: $\mathbf{t}_n \leftarrow$ earliest time possible for activity n
9: **end while**
10: **return t**

iteration, at line 5, the available activity with the highest *scheduling* priority A_{2i} from all available activities is chosen. An activity $j \in N$ is available if all executed predecessors $P_j \cap N^e$ are scheduled, where P_j is the set of time-based predecessors of activity j.

If an activity $i \in N$ has to be scheduled, it will be at the earliest time possible. This is as soon as all predecessors are finished and the resource availability is sufficient. However, due to cumulative resources not automatically regenerating, it might happen that it is not possible to schedule activity i in a resource feasible way. If this is the case, then activity i is scheduled at the first possible time when all required *renewable* resources are available. This causes resource infeasibility. To guide the search towards the feasible solution region, we penalize the objective function. Let p_{rt} be the resource availability of cumulative resource $r \in R^c$ at time t of a solution. Then, the objective value *used within the heuristic algorithm* is the makespan (which we want to minimize) plus a penalty of $M \cdot \sum_{r \in R^n} \sum_{t \in T} -\min(0, p_{rt})$. Here, M is a sufficiently large number such that a larger resource deficit always results in a larger objective value.

4.2.2. Differential evolution

The core of the HDE algorithm is the differential evolution algorithm. Like many population-based algorithms, this method explores the search space by iteratively creating *offspring* solutions from the current population. Let γ be the size of the population $\mathcal{A}$. Then, based on various implementations in literature (Ali, Elsayed, Ray, & Sarker, 2016; Wu, Wang, & Zhou, 2014), we initialize $\mathcal{A}$ as the set of γ $2 \times |N|$ matrices containing random values between 0 and 1.

After the initialization phase, the iterative improvement phase starts, which is part of a standard DE algorithm. It consists of multiple *generations*. In each generation, the algorithm iterates over the whole population. For each matrix $A \in \mathcal{A}$, define A as the *base matrix*. Secondly, we create a *mutation matrix* B. For this, we randomly select 3 matrices B^1, B^2, B^3 from $\mathcal{A}$ such that A, B^1, B^2 and B^3 are all different. Then, we create the mutation matrix as

$$B = B^1 + w\left(B^2 - B^3\right), \tag{4}$$

where w is the algorithm parameter called *weight factor*. Subsequently, the mutation matrix B is combined with the base matrix A to create a trial matrix T. For this, we define the crossover probability vector $\mathbf{c} = [c_1, c_2]$ with values between 0 and 1 (both w and $\mathbf{c}$ are set by parameter tuning, as is described later). Then, each entry T_{ij} in the trial matrix gets the mutation value B_{ij} with probability c_i, and the base value A_{ij} otherwise. Furthermore, for both rows in T, one entry is picked randomly to be replaced by the corresponding entry in the mutation matrix. This ensures that always at least one entry is replaced.

This process creates a new priority matrix, the trial matrix T. If the objective value of this solution is lower than or equal to the objective value of the base matrix A, the base matrix is replaced by the trial matrix. This process repeats itself, until no improvement of the best solution has been found for ω generations. Then, the best-found solution is returned. Note that although the initial solution matrices contain only values between 0 and 1, there is no restriction on the range of B and thus on solutions further in the iterative process. The complete algorithm is shown in Algorithm 3.

4.2.3. Combine differential evolution with forward backward improvement

Finally, the DE algorithm is combined with a forward backward improvement (Valls, Ballestín, & Quintanilla, 2008). The complete process in shown in Fig. 3. After the DE algorithm terminates, each solution is subjected to the forward-backward improvement. If this improves the best-found solution, the DE algorithm restarts. If it does not find a new best solution, the algorithm terminates.

4.2.4. Example

We now show a small example to illustrate the solution presentation. For this, we consider an instance with a single renewable resource r with $\lambda_r = 1$ and where each non-dummy activity $i \in N \backslash \{1, 7\}$ has a duration of $d_i = 1$ and a resource requirement $k_{ri} = 1$. Furthermore, the selection and scheduling graphs are shown in Fig. 4. Here, the groups are displayed as the numbers *outside of* the activity nodes. Based on the selection graph, the

Algorithm 3 Differential evolution.

$\mathcal{A} \leftarrow$ set of γ random matrices of size $2 \times |N|$
$A^* \leftarrow \arg\min_{A \in \mathcal{A}}$(objective A)
$\theta \leftarrow 0$
while $\theta < \omega$ **do**
  for $A \in \mathcal{A}$ **do**
    $B^1, B^2, B^3 \leftarrow$ Pick randomly without replacement from $\mathcal{A} \backslash \{A\}$
    $B \leftarrow B^1 + w(B^2 - B^3)$
    $f =$ [random from $\{1, \cdots, N\}$, random from $\{1, \cdots, N\}$]
    $T_{ij} \leftarrow \begin{cases} B_{ij} & \text{with probability } c_i \text{ or if } j = f_i \\ A_{ij} & \text{otherwise} \end{cases}$
    if objective $T \leq$ objective A **then**
      Replace $A \in \mathcal{A}$ by T
      if objective $T <$ objective A^* **then**
        $A^* \leftarrow T$
        $\theta \leftarrow -1$
      end if
    end if
    $\theta \leftarrow \theta + 1$
  end for
end while
return A^*

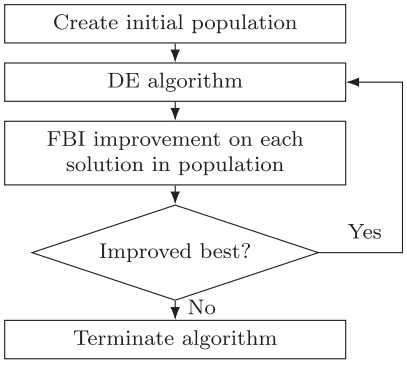


Fig. 3. Hybrid Differential Evolution algorithm.

group graph can be created, which is also shown in Fig. 4. It can be seen that the group graph is acyclic, and by following the procedure in Lemma 3 we obtain group ordering [*i*, ii, iii, iv, v, vi, vii]. Note that this group ordering is constant for an instance and is not modified during the execution of the heuristic algorithm. We let priority matrix A be

$$A = \begin{bmatrix} 0.8 & 0.4 & 0.9 & 0.5 & 0.4 & 0.3 & 0.4 \\ 0.4 & 0.6 & 0.2 & 0.8 & 1.1 & 1.6 & 0.4 \end{bmatrix} \tag{5}$$

and show how to obtain a schedule.

We first consider the selection problem. This problem uses the priorities given in the top row of A. Let N^e be the set of executed activities and start by selecting the root node: $N^e = \{1\}$. Now, we evaluate the first group in the group ordering: $g = i$. Since activity 2 is the only successor, it will be selected: $N^e = \{1, 2\}$. Similarly, group $g =$ ii selects activity 3.

Now, group $g =$ iii is evaluated. The two candidate activities are activity 4 and 5. Since $A_{14} > A_{15}$, we select activity 4 and add activity 5 to the list of forbidden activities. This gives $N^e = \{1, 2, 3, 4\}$ and $N^f = \{5\}$. The next group, $g =$ iv, also has 2 successors. However, since activity 5 is forbidden, only activity 6 remains and has to be selected. Subsequently, the remaining selection groups only have one successor. Thus, this results in $N^e = \{1, 2, 3, 4, 6, 7\}$.

Next, we schedule the set of executed activities N^e, starting at the starting activity by putting $t_1 = 0$. Let $\mathbf{t}$ be the vector of starting times. Based on the precedence constraints, we can now schedule either activity 2 or 3. Since $A_{22} > A_{23}$, we schedule activity 2 at $t_2 = 0$. Proceeding, we obtain $\mathbf{t} = [0, 0, 2, 1, -1, 3, 5]$, where we use -1 to indicate a non-executed activity.

4.3. Ant colony optimization algorithm

Since optimal solutions for general instances of the RCPSP-PS/CPR are unknown, we also present an Ant Colony Optimization (ACO) algorithm for comparison purposes. The choice for this algorithm is made based on the following two points: Firstly, the ACO algorithm was successfully implemented for the classic RCPSP (Merkle, Middendorf, & Schmeck, 2002). Secondly, due to consumption and production of resources, not every schedule is feasible to the RCPSP-PS/CPR. The ACO algorithm allows for problem specific heuristic rules to find feasible solutions.

We now give a general description and pseudo code of the ACO algorithm for the RCPSP-PS/CPR. Since this algorithm is only for comparison purposes on general instances that cannot be solved to optimality, the details are given in Appendix C.

The ACO algorithm has a population of ants. At each iteration, every ant creates a new solution and modifies a common set of pheromone trails based on the quality of the found solution. The pheromone trail then influences other ants while they create new solutions in future iterations. We keep track of three different kind of pheromones: *selection*, *scheduling* and *cumulative selection*. The selection and scheduling problems are solved by their respective pheromone trails and the cumulative selection pheromone trail keeps track of how often a certain activity was selected.

We give a brief overview of the algorithm in Algorithm 4. Here, we use the population parameter γ and the iteration threshold parameter ω. Furthermore, we define $\mathbf{p}$ as the vector containing all pheromone trails and store the activity starting times of the best schedule found under $\mathbf{t}^*$.

At each iteration, each ant creates a new set of executed activities N^e and schedules these activities to create a schedule, represented by the starting times $\mathbf{t}$. If this schedule $\mathbf{t}$ is better than the best schedule represented by $\mathbf{t}^*$, it is stored as new best solution and the iteration threshold counter θ is reset.

After all ants create their solution, the new pheromone contribution $\mathbf{p}'$ is calculated based on the solutions found. The new pheromone trails are then determined by a convex combination of

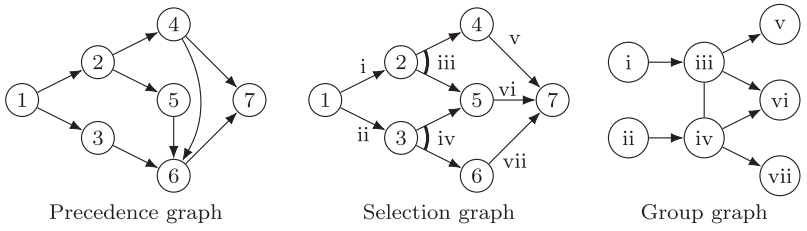


Fig. 4. Precedence, selection and group graph of an example instance.

Algorithm 4 Ant Colony Optimization.

$\mathbf{p} \leftarrow \mathbf{1}$
$\mathbf{t}^* \leftarrow \infty$
$\theta \leftarrow 0$
while $\theta < \omega$ **do**
  for $i \in [1, \cdots, \gamma]$ **do**
    $N^e \leftarrow$ Executed activities
    $\mathbf{t} \leftarrow$ Schedule executed activities N^e
    if $\mathbf{t}_{|N|} < \mathbf{t}^*_{|N|}$ **then**
      $\mathbf{t}^* \leftarrow \mathbf{t}$
      $\theta \leftarrow -1$
    end if
  end for
  $\mathbf{p}' \leftarrow$ new pheromone contribution
  $\mathbf{p} \leftarrow \mathbf{p}(1-\rho) + \mathbf{p}' \cdot \rho$
  $\theta \leftarrow \theta + 1$
end while
return $\mathbf{t}^*$

the old pheromone trails and the new contribution, where evaporation parameter ρ determines the influence of the new contribution. Finally, the iteration treshold counter θ is incremented by one. If there has been an improvement in the last ω iterations, a new iteration starts. Otherwise, the algorithm terminates and returns the best schedule found represented by $\mathbf{t}^*$.

5. Computational results

In this section, we present the computational results to evaluate the HDE algorithm. This is done by running a series of tests on instances, both from the literature and newly generated instances. All tests were performed on a single core of a 3.0 GHz Intel XEON CPU with 4 GB RAM. The parameters used for the HDE algorithm are $\gamma = 300$, $\omega = 275$, $\mathbf{c} = [0.15, 0.25]$ and $w = 0.1$. For the ACO algorithm, the parameters are $\gamma = 300$, $\omega = 75$, $q = 1$, $\rho = 0.35$, $\alpha = 0.18$, $\beta = 0.6$, $\mu = 5$ and $\eta = 0.2$. These parameters are the result of a parameter tuning process, which is presented in Appendix D.

Servranckx & Vanhoucke (2019) present both an algorithm to create instances for the RCPSP-AS (without cumulative resources) and a tabu search algorithm for solving these. They model the flexible project structure as networks with multiple *stages*. Each stage is represented by an activity, and possibly one or more *alternatives*. Then, the activities and alternatives across stages are linked to each other to create *branches* of activities that represent an option to execute a part of the project. The algorithm to create instances depends on several parameters, as listed below:

- *Linked parameter*: This parameter ([0,1]) indicates the degree to which different branches are connected to each other.
- *Nested parameter*: This parameter ([0,1]) indicates the degree to which different branches split up further into sub-branches.
- *Flexibility parameter*: This parameter ([0,1]) indicates the degree of alternatives.

For an exact description, see Servranckx & Vanhoucke (2019). From the authors of this paper, a set of instances and objective values was obtained. This was used to build three sets of instances. The first one, named Servranckx & Vanhoucke (2019), simply contains all instances received from the authors, except for 25 instances that are removed for use in the parameter tuning process. These therefore do not contain cumulative resources.

For further evaluation, two smaller, more uniformly distributed subsets are taken from the Servranckx & Vanhoucke (2019) instance set. These sets are named *Optimal* and *Non-optimal* and contain instances for which the optimal solutions are known and unknown, respectively. The optimal solutions were obtained according to the methods presented in Van der Beek et al. (2022).

These smaller instance sets are created by taking a subset of the set Servranckx & Vanhoucke (2019), such that they have an equal number of instances for all possible combinations of values for the properties *linked*, *nested* and *flexibility*. Furthermore, a new instance was created for each selected instance by adding cumulative resources, as described in Van der Beek (2021). Since there are more instances without the optimal solution known, the *Non-optimal* instance set contains more instances (2000) than the *Optimal* instance set (900). Furthermore, the number of activities of all instances is between 102 and 702, the number of renewable resources is 5 and the number of cumulative resources is 2. The instance sets are summarized in Table 3 and the complete instance sets with all results are published in Van der Beek (2021).

Thus, we have three instance sets: Servranckx & Vanhoucke (2019), *Optimal*, and *Non-optimal*. The instance set Servranckx & Vanhoucke (2019) is used to compare the HDE algorithm with the algorithm from that paper. Furthermore, the instance set *Optimal* is used to evaluate the performance of the HDE and the ACO against the optimal solutions. Finally, the instance set *Non-optimal* is used to evaluate the performance of the HDE algorithm on larger instances, for which no optimal solutions are known. Since solutions in Servranckx & Vanhoucke (2019) are only available for half of this set (without cumulative resources), we compare the HDE algorithm against the ACO algorithm to evaluate the HDE algorithm on the complete instance set.

To compare the performance on multiple instances, we introduce the *Bound Optimality Gap* (BOG), which is a lower bound on the optimality gap of the respective solution:

$$BOG(obj) = \frac{obj - obj^*}{obj^*} \cdot 100\%, \tag{6}$$

where obj is the objective value obtained by the considered algorithm and obj^* is the best known objective value for the instance within the considered instance set. Since an instance can be part of multiple instance sets, we only consider the solutions within the respective instance set.

5.1. Instance set: Servranckx & Vanhoucke (2019)

First, we evaluate the instance set that was directly obtained by the authors of Servranckx & Vanhoucke (2019). This instance set contains 3207 instances without cumulative resources along with the objective values from their *Tabu Search* (TS) algorithm. In this section, we compare the results from the TS algorithm to the algorithms presented in that paper.

For the results in Servranckx & Vanhoucke (2019), only objective values of a single run are available. Therefore, for a fair comparison, each algorithm is run once per instance. In Table 4 is shown how each algorithm performs against the TS algorithm. It can be seen here that in more than half of the cases, the ACO and HDE algorithms tie with the TS algorithm.

We obtain Fig. 5 by omitting all ties and plotting the *relative objective* $obj^r = obj/obj^{TS}$, where obj is the objective of the ACO/HDE algorithm and obj^{TS} is the objective of the TS algorithm. Here, it can be seen that the boxes of the boxplot are completely below 1, indicating that both ACO and HDE perform better on most non-tied instances. Furthermore, the average relative objective of ACO and HDE is shown to be at 0.982 and 0.976, respectively. This shows that the HDE algorithm performs better than the algorithm from Servranckx & Vanhoucke (2019), and it suggests that the ACO is a reasonable algorithm for comparison purposes.

Secondly, we compare the BOG for the HDE and the algorithm from Servranckx & Vanhoucke (2019) over the *nested* param-

Table 3
Instance sets.

Name	# of instances	Cum. resources	Optim. solutions
Servranckx & Vanhoucke (2019)	3207	No	Mixed
Optimal	900	in 450 instances	Known
Non-optimal	2000	in 1000 instances	Unknown

Table 4
Comparison of ACO and HDE algorithms against the TS algorithm from Servranckx & Vanhoucke (2019).

Method	ACO	HDE
Better	1197	1335
Tied	1813	1858
Worse	197	14

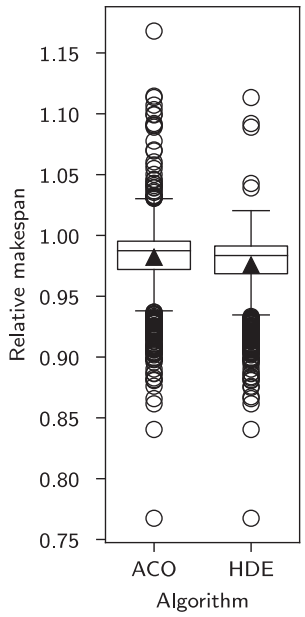


Fig. 5. Comparison of relative makespan $obj^r = \frac{obj}{obj^{TS}}$ of ACO and HDE algorithms, omitting all ties.

eter. This parameter was chosen since it was found to have the most significant correlation with the performance. This is shown in Fig. 6. Here, for the HDE algorithm the outliers of the BOG show a performance-wise decreasing trend for the HDE algoritm of the BOG with the *nested* parameter value. However, both the boxes and the outliers show that the performance of the HDE algorithm is better than the performance of the algorithm from Servranckx & Vanhoucke (2019), even for the maximum *nested* parameter value. Since less clear trends were found for the other instance parameters, *linked* and *flexibility*, these results are placed in Appendix E.

5.2. Instance set: Optimal

The next set of instances is called *Optimal* and contains instances for which the optimal solution is known. To create this set, a new instance was created for each instance in the set Servranckx & Vanhoucke (2019) by adding cumulative resources according to Van der Beek (2021). Subsequently, up to 5 instances (depending on how many optimal solutions are known) were randomly selected for each combination of parameters (*linked*, *flexibility*, *nested*). This resulted in a set of 900 instances of which half contains both types of resources and the other half contains only renewable resources.

Table 5
Number of runs (out of 4500 per category) resulting in the optimal value and number of instances (out of 450 per category) resulting in at least one optimal solution.

Method	ACO		HDE	
Cumulative resources	No	Yes	No	Yes
Optimal runs	3883	3996	4376	4428
Optimal instances	421	432	444	446

For this instance set, both algorithms are run 10 times. Thus, this resulted in 4500 runs per category (with and without cumulative resources). In Table 5, it is shown how many runs resulted in optimality and how many instances have at least one optimal solution in ten runs. It can be seen that in most runs, the optimal value was reached for both methods. Furthermore, it can be seen that the HDE obtains the optimal value at least once for nearly every instance. When comparing between instances with and without cumulative resources, it is shown that both methods perform better for instances with cumulative resources. A possible explanation is that the parameter tuning process contained challenging instances on the aspect of resource feasibility.

Next, we evaluate the optimality gaps for the HDE algorithm. The optimality gap is defined as $\frac{obj-obj^*}{obj^*} \cdot 100\%$, where obj is the found objective value and obj^* the optimal objective value. The optimality gap indicates the difference between the found objective and the optimal objective. Since the HDE algorithm is the main algorithm of this paper, we further evaluate these optimality gaps for this algorithm only. This is shown in Fig. I, where the optimality gaps are shown for different *linked* parameter values. Here, it can be seen that in general the optimality gap is quite low (mostly below 2%). Additionally, an increasing trend can be seen of the optimality gap against the *linked* parameter. This indicates, especially for the highest value, that the HDE algorithm has more difficulty in finding the optimal solution as the *linked* parameter increases. However, since all boxes are reduced to lines of value zero, this trend is only present in the outliers. Since less clear trends were found for the other instance parameters, *nested* and *flexibility*, these results are placed in Appendix E.

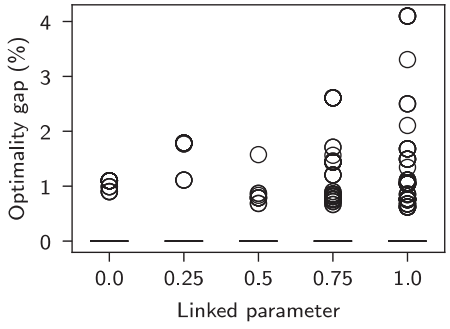


Fig. I. Boxplots of optimality gaps for the HDE algorithm against the *linked* parameter.

5.3. Instance set: Non-optimal

Finally, we consider the instance set *Non-optimal*. This set is created similarly to the instance set *Optimal*, however, it contains

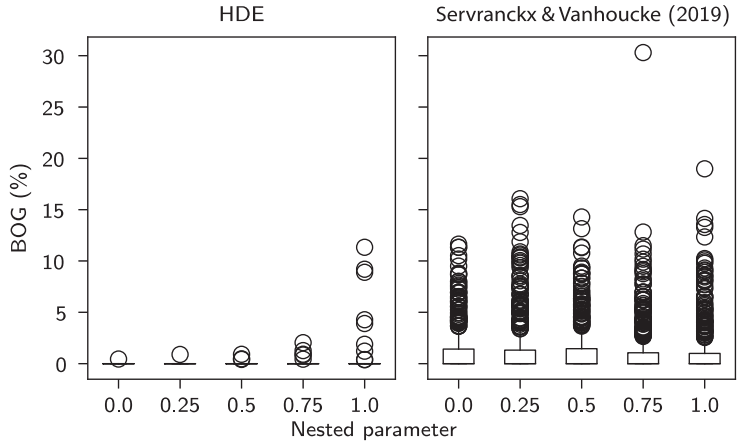


Fig. 6. BOG of HDE and algorithm from Servranckx & Vanhoucke (2019) against the *nested* parameter.

Table 6
For how many instances each algorithm performs better than the other, evaluated on best out of 10 runs and on average of 10 runs.

	Best	Average
ACO	1	6
HDE	147	758
Tie	1852	1236

Table 7
Average values of BOGs, categorized on instances with and without cumulative resources.

Algorithm:	Cumulative resources:	
	No	Yes
ACO	0.275%	0.641%
HDE	0.031%	0.015%

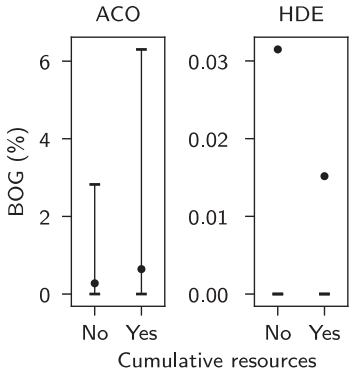


Fig. 7. Mean, 2.5 and 97.5 percentile for BOGs against the resource type for both algorithms.

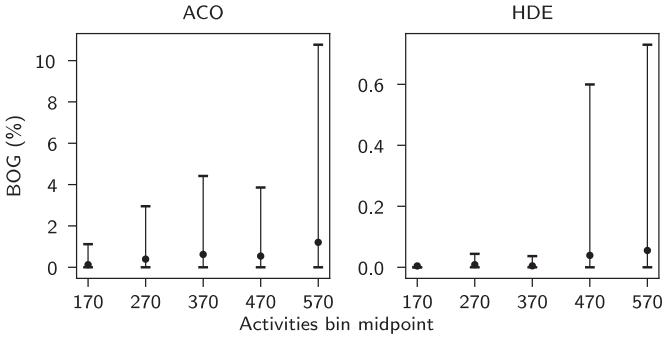


Fig. 8. Mean, 2.5 and 97.5 percentile for BOGs against the activities per instance for both algorithms. Instances are grouped into bins. Each bin has a range of 100 activities.

10 instances per combination of parameter values and only includes instances for which the optimal value is not known. This results in a set of 2000 instances, of which half contain cumulative resources. Since we only have literature results for a part of the instance set and no optimal solutions, the ACO is used as a baseline algorithm for comparison. For each instance, each algorithm is run 10 times.

For cumulative resources, not every activity list can be mapped to a feasible solution. Therefore, it is possible to obtain infeasible solutions from the algorithms. To mitigate this, both algorithms restart if they converge to an infeasible solution, with a maximum of 9 restarts per run. The HDE algorithm did not require any restarts, since each run directly resulted in feasibility. The ACO algorithm required 21 restarts, spread out over 17 runs and 8 instances. The maximum number of restarts required for any run was equal to 2.

Furthermore, we compare the found objective values per algorithm. In Table 6, it is shown per algorithm how many times the best value of all 10 runs is lower than the other algorithm and how often they are tied. Similarly, it compares the average values of all 10 runs and shows how often this value is lower for HDE, ACO and how often it is tied. It can be seen that HDE performs clearly better than ACO, although the majority of instances result in a tie.

We also evaluate the BOGs. The average values are shown in Table 7 and the average values with spread are shown in Fig. 7, categorized by the presence of cumulative resources. It can be seen that ACO performs significantly worse for instances with cumulative resources. Note that comparison on the BOGs show *relative* performance against other runs for the same instance. Therefore, they might contradict the *absolute* performance, indicated in Table 5. Furthermore, we evaluate the BOG compared to the number of activities, as shown in Fig. 8. It shows, not surprisingly, that the difficulty of solving instances increases with the number of activities. Additionally, it can be seen that for each bin of activity numbers, HDE outperforms ACO on both spread and average value.

To evaluate the effects of cumulative resource availability, we use the concept of *resource strength* from Neumann & Schwindt (2003). There, a value of zero indicates that resource profiles of resource-feasible schedules are constant over time and a value of one indicates that the *earliest start schedule* (see Neumann & Schwindt, 2003) is feasible. However, this definition does not hold for the RCPSP-PS/CPR, since not all activities are executed. Nevertheless, the resource strength can still be used as an indication of the level of cumulative resource availability. In Fig. 9, the BOGs are shown against the values of the resource strength. In the ACO algorithm, a decreasing trend can be seen, indicating better perfor-

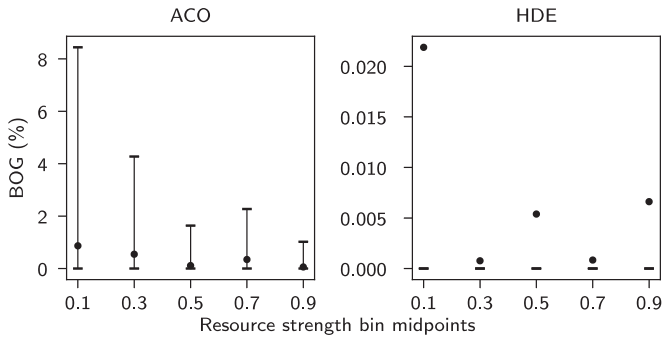


Fig. 9. Mean, 2.5 and 97.5 percentile for BOGs against the resource strength. Resource strength values are grouped into bins. Each bin has a range of 0.2.

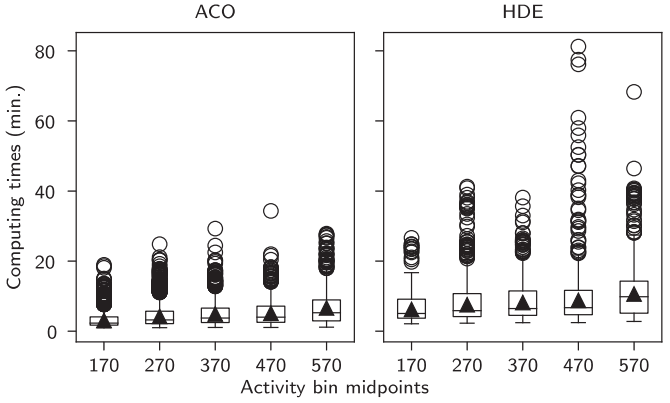


Fig. 10. Durations against the activities per instance for both algorithms. Instances are grouped into bins. Each bin has a range of 100 activities.

mance for instances with a larger resource strength. For the HDE, however, no clear trend can be seen.

Finally, the computing times are shown in Fig. 10. As expected, the computing times increase with the number of activities. It also can be seen that the computing times of the HDE algorithm are significantly longer, even though the parameter tuning processes of both algorithms had the same computing time penalty. A possible explanation for this is the high value of $\omega = 275$ for the HDE algorithm, which defines that after each improvement in objective function value, at least 275 iterations are executed.

6. Conclusion

In this paper, a model is given for the RCPSP-PS/CPR by taking the model from Van der Beek et al. (2022) and adding cumulative resources. Finding a selection of activities that satisfies the selection constraints of the RCPSP-PS/CPR is NP-hard, so in order to find feasible selections quickly in metaheuristics, we restricted the problem. This is done by introducing the concept of group graphs and feasible group orderings.

We showed that if the group graph is acyclic, there exists a feasible group ordering that can be found according to the method presented in this paper. This feasible group ordering can then be used to create feasible selections in polynomial time. This method is used in a HDE algorithm for the RCPSP-PS/CPR. While evaluating this algorithm against solutions from Servranckx & Vanhoucke (2019), it is shown that the HDE algorithm generally creates better solutions. Furthermore, by comparing against optimal solutions, it is shown that in most cases the HDE finds an optimal solution. For instances where no optimal solution is known, the HDE algorithm was compared against a baseline ACO algorithm. Although we showed that this algorithm also performs better than the algorithm from Servranckx & Vanhoucke (2019) on instances without cumulative resources, the HDE algorithm outperforms the ACO algorithm on nearly all metrics.

For future research, we recommend testing the developed methods on real-world instances. This will quantify the value of these algorithms in terms of reduced makespan and costs. Furthermore, from a computational point of view, a decrease in performance can be seen with increasing value of the nested parameter. Therefore, additional research can be performed on methods for instances with large values for this parameter. A possible direction for performance improvement can be experimenting with various initialization methods for the DE population. Finally, the presented methods only work on instances with an acyclic group graph. Although this seems usable for many practical cases and we have not encountered any cases without this property, there might be some real-world exceptions where this is not the case. Therefore, it would be interesting to see if feasible group orderings can be produced for a more general case, or to find alternative methods of solving the selection problem.

Acknowledgments

The authors would like to thank all partners of the NAVAIS project for assistance during this research. The project has received funding from the European Union's Horizon 2020 research and innovation programme (Contract No.: 769419). Furthermore, we would like to thank the anonymous referees for their useful comments which helped to improve the paper.

Appendix A. Notation

Variables	
X_{it}	1 if activity $i \in N$ is executed at time $t \in T$, zero otherwise.
Sets	
G	Selection groups.
N	Activities.
P_j	Predecessors of activity $j \in N$ in the precedence graph.
R	Resources.
R^r	Renewable resources.
R^c	Cumulative resources.
S_g	Successor activities of selection group $g \in G$.
T	Time periods.
$\mathcal{P}$	Time-based predecessor-successor pairs.
Parameters	
a_g	Activating activity of selection group $g \in G$.
d_i	Duration of activity $i \in N$.
k_{ri}	Net resource production of resource $r \in R$ for activity $i \in N$.
k^+_{ri}	Production of resource $r \in R$ for activity $i \in N$.
k^-_{ri}	Consumption of resource $r \in R$ for activity $i \in N$.
M	Sufficiently large number.
q	Parameter for ACO algorithm setting the number of considered solutions.
w	Weight factor HDE algorithm.
α	ACO parameter indicating influence of selection pheromone values.
β	ACO parameter indicating influence of scheduling pheromone values.
γ	Population size in ACO and HDE algorithms.
η	ACO parameter indication probability of adding heuristic contribution.
λ_r	Capacity of resource $r \in R$.
μ	Resource tightness.
ρ	Evaporation coefficient for ACO algorithm.
ω	Threshold parameter for ACO and HDE algorithms.
Vectors	
$\mathbf{c}$	Crossover probability for HDE algorithm.
$\mathbf{p}$	Pheromone trails.
$\mathbf{p}^{cs}$	Cumulative selection pheromone trail.
$\mathbf{p}^{sc}$	Scheduling pheromone trail.

(*continued on next page*)

$\mathbf{p}^{se}$	Selection pheromone trail.
Functions	
$erf(x)$	Gauss error function.
$HC(N^s, N^c, i)$	Heuristic contribution when selecting activity i from candidates $N^c \subseteq N$, given the selection of activities $N^s \subseteq N$.
$NM(obj)$	Normalized makespan.
$PR^{sc}(N^s, N^c, i)$	Scheduling probability in ACO for activity i from candidates N^c, given that activities N^s are already scheduled.
$PR^{se}(N^c_g, i)$	Selection probability in ACO for activity i from candidates N^c_g from group $g \in G$.
$RS_r(N^c, i)$	Resource supply of resource $r \in R$ of activity i from candidates $N^c \subseteq N$.
$RT_r(N^s)$	Resource tightness of scheduled activities $N^s \subseteq N$ for cumulative resource $r \in R^c$.
$SC(N^c, i)$	Scheduling score when scheduling activity i from candidates $N^c \subseteq N$.

Appendix B. Group orderings

This sections provides the intermediate proofs of Section 4.1 to create Theorem 1. We first introduce the following Lemma:

Lemma 1. *If an instance of the selection problem has an acyclic group graph, every group $g \in G$ with $|S_g| = 1$ satisfies Property 2 in Theorem 1 for any (partial) group ordering.*

Proof. Let $S_g = \{i\}$. Then, since the group graph is acyclic, there is no group $h \in G$ with $i \in S_h$ and $|S_h| > 1$. Otherwise, the group graph would contain an undirected edge between node g and node h due to successor overlap and a directed edge from g to h due to strict successor containment, which would result in a cycle.

Thus, if there is a group $h \in G$ with $i \in S_h$, it follows that $S_h = \{i\} = S_g$. In this case, group h is not included in the right-hand side of Eq. (3). Since there does not exist a group $h \in G$ with $i \in S_h$ and $S_h \neq S_g$, the right-hand side will never include activity i and Eq. (3) will never hold for $j_a = g$. □

Subsequently, we show how to construct partial feasible group orderings in Lemma 2.

Lemma 2. *Consider a group graph $\mathcal{H}$ that is acyclic. Then, for each connected component of undirected edges, a breadth-first search ordering starting in any node satisfies both properties in Theorem 1 and thus is a partial feasible group ordering.*

Proof. By assumption, there are no cycles in $\mathcal{H}$. Therefore, all undirected edges form a forest. Now, consider a connected component G' (which is a tree) of size n with breadth-first ordering $J = [j_1, \cdots, j_n]$. As given in the lemma, G' only consists of undirected edges. We will show that J fulfills both properties of Theorem 1.

To prove Property 1, we show that there are no two selection groups $g \in G'$ and $h \in G'$, with $g < h$, such that $a_g \in S_h$. Assume, as a contradiction, that there are two selection groups $g \in G'$ and $h \in G'$, with $g < h$, such that $a_g \in S_h$. Then, there is a directed edge in $\mathcal{H}$ from h to g. Since h and g are both in the connected component G', there is also a path by undirected edges from g to h. This results in a cycle and contradicts the assumption of an acyclic group graph. Therefore, Property 1 from Theorem 1 is always satisfied.

We now prove Property 2. For this, we introduce

$$\mathcal{F}_J(a) = \bigcup_{i \in \{1,\cdots,a-1\} | S_{j_i} \neq S_{j_a}} S_{j_i}, \tag{B.1}$$

which is the right-hand side of Eq. (3) and where j_i is the ith entry of (partial) group ordering J. Furthermore, we consider group $j_a \in J$ where a is the index of group j_a in J. Now, Property 2 is satisfied if for each $j_a \in J$ it holds that $S_{j_a} \not\subseteq \mathcal{F}_J(a)$.

Due to Lemma 1, we know that Property 2 is satisfied if $|S_{j_a}| = 1$, thus we only need to consider the case that $|S_{j_a}| > 1$. Since undirected edges are created between groups with successor overlap if at least one group has more than one successor, we know that if two groups in G' have successor overlap, there is an undirected edge between them.

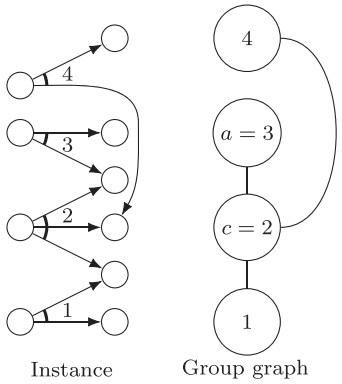


Fig. B.1. A breadth-first ordering J on a connected component in the group graph. This illustrates that for each group $j_a \in J$, there can be only one group $j_c \in J$ with $c < a$.

Since J is ordered breadth-first, there is at most one group $j_c \in J$ with successor overlap and $c < a$. An example of this is shown in Fig. B.1. We now consider two cases: The case that $S_{j_a} \neq S_{j_c}$ and the case that $S_{j_a} = S_{j_c}$.

Consider the first case: $S_{j_a} \neq S_{j_c}$. We know that S_{j_c} is contained in $\mathcal{F}_J(a)$ ($S_{j_c} \subseteq \mathcal{F}_J(a)$). Furthermore, besides group j_c, no other group ordered before j_a in J has successor overlap with group j_a. Therefore, we get

$$S_{j_a} \cap \mathcal{F}_J(a) = S_{j_a} \cap S_{j_c}. \tag{B.2}$$

For contradiction to Property 2, assume that $S_{j_a} \subseteq \mathcal{F}_J(a)$. Then, it follows that $S_{j_a} \cap \mathcal{F}_J(a) = S_{j_a}$. Combining with Eq. (B.2) gives $S_{j_a} = S_{j_a} \cap S_{j_c}$, from which it follows that $S_{j_a} \subseteq S_{j_c}$. Since, by assumption, $S_{j_a} \neq S_{j_c}$, we get $S_{j_a} \subsetneq S_{j_c}$. However, due to strict successor containment, this means that there is a directed edge from j_a to j_c. This edge forms a cycle with the undirected edge, and therefore contradicts the assumption of an acyclic group graph. Thus, in the case of $S_{j_a} \neq S_{j_c}$, the assumption $S_{j_a} \subseteq \mathcal{F}_J(a)$ cannot be true, and therefore, $S_{j_a} \not\subseteq \mathcal{F}_J(a)$. Thus, Property 2 holds.

In the other case, when $S_{j_a} = S_{j_c}$, group j_c is not considered in the union operator of $\mathcal{F}_J(a)$ and no other group in $\mathcal{F}_J(a)$ has successor overlap with S_{j_a}. Therefore, we get $S_{j_a} \cap \mathcal{F}_J(a) = \emptyset$, and thus, $S_{j_a} \not\subseteq \mathcal{F}_J(a)$. This means that Property 2 is satisfied for all cases. □

Subsequently, we can combine all partial feasible group orderings. This is stated in Lemma 3.

Lemma 3. *If a group graph $\mathcal{H}$ is acyclic, there is a feasible group ordering.*

Proof. We can construct a feasible group ordering J by combining all partial group orderings from the connected components of undirected edges in Lemma 2 by *topological sort*; if there is a directed path from connected component of undirected edges G' to connected component of undirected edges H', the partial group ordering of H' has to appear after the partial group ordering of G'. Note that we define nodes without undirected edges as connected components of size 1. By assumption of an acyclic group graph, it is always possible to sort the partial group orderings topologically.

We now prove that J satisfies both properties of Theorem 1 by induction. Let $\mathcal{C} = [G'_1, \cdots, G'_{|\mathcal{C}|}]$ be the topologically sorted collection of connected components and let J'_i be the partial group ordering of G'_i. Then, we denote J as the concatenation of all partial group orderings: $J = (J'_1, \cdots, J'_{|\mathcal{C}|})$.

As the base step for induction, Lemma 2 states that J'_1 satisfies both properties of Theorem 1. For the induction step, we take the

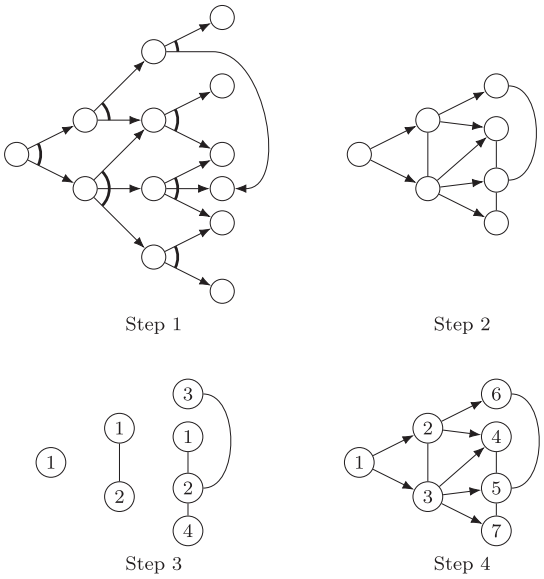


Fig. B.2. Creating a feasible group ordering.

feasible partial group ordering $J(n) = (J'_1, \cdots, J'_n)$ with $n < |\mathcal{C}|$ and append the partial group ordering J'_{n+1} to obtain $J(n+1)$. We now show that $J(n+1)$ satisfies the properties from Theorem 1.

First, we show that for each combination of groups $g \in J(n)$ and $h \in J'_{n+1}$, Property 1 holds. Assume, for contradiction, that it does not hold, and thus, g is a successor of h ($a_g \in S_h$). Then, by Step 2 in Definition 4, there would also be a directed edge from component J'_{n+1} to $J(n)$. This contradicts the topological sorting in combination with an acyclic group graph, and therefore, Property 1 is satisfied.

To prove Property 2, we consider group $h \in J'_{n+1}$ and show that Eq. (3) is never satisfied for $h = j_a$. For this, we define K as the preceding part of the group ordering $\{j_1, \cdots, j_{a-1}\}$ and split this up into $K_1 = \{j_1, \cdots, j_b\} = J(n)$ and $\{j_{b+1}, \cdots, j_{a-1}\} = K_2 \subset J'_{n+1}$.

First, we consider $K_1 = J(n)$. Since Property 2 is satisfied if $|S_h| = 1$ due to Lemma 1, we only need to consider the case where $|S_h| > 1$. For this case, we know that there is no group $g \in J(n)$ with successor overlap between g and h. Otherwise, there would be an undirected edge between g and h and they would be in the same connected component. This means that there is no overlap between all successors of $J(n)$ and S_h : $(\bigcup_{g \in J(n) | S_g \neq S_h} S_g) \cap S_h = \emptyset$.

Now, consider $K_2 \subset J'_{n+1}$. Property 2 holds by Lemma 2, which results in $S_h \notin \bigcup_{g \in K_2 | S_g \neq S_h} S_g$. Combining this with the equation for K_1 gives $S_h \notin \bigcup_{g \in K | S_g \neq S_h} S_g$. This means that Property 2 is satisfied for each $h \in J_{n+1}$.

In conclusion, if $J(n)$ is a feasible group ordering, so is $J(n+1)$. Since $J(1)$ is feasible, so is $J = J(|\mathcal{C}|)$. □

An example of the process of obtaining a feasible group ordering can be seen in Fig. B.2. First, the selection graph is given. Subsequently, in Step 2, the group graph is presented. The connected components based on undirected edges are evaluated in Step 3. Each connected component is ordered by breadth-first. In Step 4, a final ordering is obtained, respecting both the topological sorting and the partial orderings of Step 3. It can be seen that this group ordering satisfies both properties of Theorem 1, and therefore, using Algorithm 1 always results in a feasible selection. With this, we now can state Theorem 1.

Appendix C. Ant colony optimization algorithm

In this section, we give the full details of the Ant Colony Optimization (ACO) algorithm. This algorithm has multiple ants creating new solutions at each iteration, and updating a common set of pheromone trails to influence how solutions are created in future iterations.

We first discuss the pheromone trails: what is the structure and how are they updated. Secondly, we explain the contribution of the heuristic rule. Subsequently, we explain how individual ants use this information to create new solutions. Finally, we combine these parts to give the full algorithm.

C1. Pheromone trails

We keep track of three kind of pheromones: *selection*, *scheduling* and *cumulative selection*. The selection pheromone $\mathbf{p}^{se}$ is a vector containing an entry p_i^{se} for every activity $i \in N$ and indicates how often activities occur in high quality solutions. Furthermore, the entries of the scheduling pheromone vector $\mathbf{p}^{sc}$ are defined for every combination of activities, resulting in an entry p_{ij}^{sc} for every pair of activities $i, j \in N$. These pheromones are used to schedule a selection. Finally, the vector $\mathbf{p}^{cs}$ contains the cumulative selection pheromone values, indexed similarly to the selection pheromone vector: p_i^{cs} for every activity $i \in N$. The cumulative selection pheromones are used to negate the effect of selection choices on the scheduling pheromone. Combing all pheromone values gives the combined pheromone vector $\mathbf{p}$.

At each iteration, after all ants created their solution, the pheromone trails are updated. This is done by the following update equation:

$$\mathbf{p} := \mathbf{p}(1-\rho) + \mathbf{p}' \cdot \rho. \tag{C.1}$$

Here, ρ is the evaporation coefficient (set between 0 and 1) and $\mathbf{p}'$ the new pheromone contribution. We can split up the entries of $\mathbf{p}'$ into $\mathbf{p}'^{se}$, $\mathbf{p}'^{sc}$ and $\mathbf{p}'^{cs}$, which are the pheromone contributions of selection, scheduling and cumulative selection, respectively.

Let $\mathcal{A}$ be the set of solutions found by all ants in the current iteration and let A^* be the best solution of all previous iterations. Then, we take $\mathcal{B}$ as the best q solutions from $\mathcal{A} \cup \{A^*\}$, where q is an algorithm parameter. Thus, we have $\mathcal{B} = \{B_1, \cdots, B_q\}$. Each solution $B_m \in \mathcal{B}$ is then a priority matrix. Here, the top row contains $(B_m)_{1i} = 1$ if activity $i \in N$ is executed and $(B_m)_{1i} = 0$ otherwise. The bottom row contains a priority vector for all selected activities, as to be used in Algorithm 2. Now, we calculate the selection pheromone contribution $\mathbf{p}'^{se}$. This is done by the following equation:

$$p_i'^{se} = \sum_{m=1}^{q} \frac{(B_m)_{1i}}{m} \forall i \in N. \tag{C.2}$$

Thus, in Eq. (C.2), the mth best solution gives a contribution of $1/m$ to every activity that is executed. For the scheduling pheromone update, the following equation is used:

$$p_{ij}'^{sc} = \sum_{m=1}^{q} \left(\begin{cases} 1/m \text{ if } (B_m)_{2i} > (B_m)_{2j} \text{ and } i, j \in N^e \\ 0 \text{ otherwise} \end{cases} \right) \forall i, j \in N. \tag{C.3}$$

In this equation, the mth best solution gives a contribution of $1/m$ to every trail (i, j), if i and j are both selected ($i, j \in N^e$) and i appears before j in the activity list. Finally, we update the cumulative selection pheromone as follows:

$$p_i'^{cs} = \sum_{m=1}^{q} (B_m)_{1i} \forall i \in N. \tag{C.4}$$

For each node, a contribution of 1 is added if this node is executed in the selected solution. Combining $\mathbf{p}'^{se}$, $\mathbf{p}'^{sc}$ and $\mathbf{p}'^{cs}$ gives $\mathbf{p}'$ that can be used in Eq. (C.1) to calculate the new pheromone values.

C2. Heuristic rule

Pheromone trails are the most important part in order to create new solutions. However, these are not always sufficient in order to enter the feasible region. Therefore, we implement a heuristic rule in order to direct the algorithm to the feasible region.

As for HDE, the scheduling of a set of executed activities N^e is done by Algorithm 2. In each iteration of this algorithm, there is a set of *scheduled activities* N^s and a set of *candidate activities* N^c, where one candidate activity $i \in N^c$ has to be selected to be added to the scheduled activities. Furthermore, we introduce the net resource production of resource $r \in R$ for activity $i \in N$ as $k_{ri} = k^+_{ri} - k^-_{ri}$.

A solution is infeasible when, at any time, there is a deficit of cumulative resources. Therefore, we use a heuristic rule consisting of two parts: the *Resource Tightness* $RT_r(N^s)$ and the *Resource Supply* $RS_r(N^c, i)$. The resource rightness $RT_r(N^s)$ estimates how close a cumulative resource $r \in R^c$ is to depletion, based on the scheduled activities N^s, and acts as an activation function; if a resource is low, its corresponding resource tightness is high which results in a higher priority of replenishing this resource. The resource supply $RS_r(N^c, i)$ represents the amount of cumulative resource $r \in R^c$ generated by candidate node $i \in N^c$, and thus, the preference of selecting this activity.

The RT is then defined as follows:

$$RT_r(N^s) = \frac{1}{2} erf\left(2 - \frac{\lambda_r + \sum_{j \in N^s} k_{rj}}{\mu \lambda_r}\right) \forall r \in R^c. \tag{C.5}$$

The top side of the fraction in Eq. (C.5) represents the net amount of resource r generated by all scheduled activities N^s plus the initial available amount of resource r. Thus, this is a measure for the total available amount of resource r divided by the initial amount of resource r times the resource tightness parameter μ. Then, the Gauss error function erf is used together with the constants $\frac{1}{2}$ and 2 to create Eq. (C.5). This equation is close to zero when the available amount of resource r is equal to or higher than $\mu\lambda_r$, where λ_r is the original resource availability. If the resource availability decreases, the RT increases, and thus, more priority is given to the generation of this resource.

Secondly, the resource supply $RS_r(N^c, i)$ represents the net resource production, normalized to a [0,1] range. In order to prevent division by zero, we define the Resource Supply to be 0 if all candidate activities N^c have the same net resource generation. This gives the following equation:

$$RS_r(N^c, i) = \begin{cases} \frac{k_{ri} - \min_{j \in N^c} k_{rj}}{\max_{j \in N^c} k_{rj} - \min_{j \in N^c} k_{rj}}, & \text{if } \min_{j \in N^c} k_{rj} < \max_{j \in N^c} k_{rj} \\ 0, & \text{otherwise.} \end{cases} \tag{C.6}$$

Finally, we get the heuristic contribution by averaging over the product of the resource tightness and the resource supply for each resource:

$$HC(N^s, N^c, i) = \frac{1}{|R^c|} \sum_{r \in R^c} RT_r(N^s) \cdot RS_r(N^c, i). \tag{C.7}$$

Each term in the sum of Eq. (C.7) consists of the resource tightness times the resource supply. This means that the heuristic contribution for activity $i \in N^c$ is increased the most, if it generates a cumulative resource (high resource supply) that is near to depletion (high resource tightness).

C3. Creating solutions

Each ant creates a new solution at each iteration. This is done by combining the information from the pheromone trails with the heuristic contribution. First, the ant creates a selection of executed activities N^e. This is done by executing Algorithm 1. Here, at line 7, the next activity i from selection group $g \in G$ is selected from all candidates N^c_g with probability $PR^{se}(N^c_g, i)$:

$$PR^{se}(N^c_g, i) = \frac{(p^{se}_i)^\alpha}{\sum_{j \in N^c_g} (p^{se}_j)^\alpha}, \tag{C.8}$$

where α is a parameter indicating the influence of higher pheromone trail values.

Subsequently, the executed activities are scheduled according to Algorithm 2. On line 5, candidate activity $i \in N^c$ is selected from candidates $N^c \subseteq N$, given that activities in set N^s are already scheduled, with probability $PR^{sc}(N^s, N^c, i)$. To calculate this probability, we first calculate the *Scheduling Score* $SC(N^c, i)$:

$$SC(N^c, i) = \left(\frac{\sum_{j \in N^c \setminus \{i\}} p^{sc}_{ij}}{|N^e|}\right)^\beta \cdot \frac{1}{p^{cs}_i}. \tag{C.9}$$

Here, β is a parameter indicating the influence of higher pheromone trail values. We then add the heuristic contribution $HC(N^s, N^c, i)$ with probability η, where η is an algorithm parameter, and normalize the scores to a 0–1 range. This gives the following expression for the scheduling probabilities:

$$PR^{sc}(N^s, N^c, i) = \begin{cases} \frac{SC(N^c,i) + HC(N^s,N^c,i)}{\sum_{j \in N^c} SC(N^c,j) + HC(N^s,N^c,j)} & \text{with probability } \eta \\ \frac{SC(N^c,i)}{\sum_{j \in N^c} SC(N^c,j)} & \text{otherwise.} \end{cases} \tag{C.10}$$

C4. Full algorithm

Finally, we combine the above to present the ACO algorithm, as given in Algorithm C.1. For this algorithm, we use the population parameter γ and the iteration threshold parameter ω from the HDE algorithm. The algorithm initializes by setting all pheromone trail values to 1.

Algorithm C.1 Ant Colony Optimization

```
1: p ← 1
2: t* ← ∞
3: θ ← 0
4: while θ < ω do
5:     for i ∈ [1, ··· , γ] do
6:         N^e ← Executed activities by Algorithm 1 and Equation 16
7:         t ← Schedule by Algorithm 2 and Equation 18
8:         if t_|N| < t*_|N| then
9:             t* ← t
10:            θ ← −1
11:        end if
12:    end for
13:    p'^se ← by Equation 10
14:    p'^sc ← by Equation 11
15:    p'^cs ← by Equation 12
16:    p' = [p'^se, p'^sc, p'^cs]
17:    p := p(1 − ρ) + p' · ρ
18:    θ ← θ + 1
19: end while
20: return t*
```

At each iteration, each ant creates a new schedule by first selecting activities according to Algorithm 1 and subsequently scheduling these according to Algorithm 2. If this new schedule represented by $\mathbf{t}$ is better than the best solution represented by $\mathbf{t}^*$,

t is stored as best solution. After all ants have created a new solution, the pheromone values are updated. This process is repeated until there has not been an improvement for ω iterations.

Appendix D. Parameter tuning

This section describes the parameter tuning process, which consists of a local-search algorithm. This algorithm iteratively varies a single parameter across a range of values. For each value, the scheduling algorithm is executed on a set of 50 instances, half with renewable resources and half without. The tuning algorithm selects a parameter, varies this parameter, and selects the value with the lowest mean objective function value. Then, the same is done for the next parameter. This is repeated until the parameters have not changed in a full iteration of all parameters, thus assuring a local minimum.

In order to equalize the running times, a maximum running time of 15 minutes is used. A penalty of 10 is added for every second above this. Furthermore, a penalty of 10 is added for every unit of cumulative resource unavailability. For the HDE algorithm, this resulted in parameters $\gamma = 300$, $\omega = 275$, $w = 0.1$, $\mathbf{c} = [0.15, 0.25]$. For the ACO algorithm, the parameters are $\gamma = 300$, $\omega = 75$, $q = 1$, $\rho = 0.35$, $\alpha = 0.18$, $\beta = 0.6$, $\mu = 5$ and $\eta = 0.2$. In Figs. D.1 and D.2, the average makespan plus resource penalties and the average computing times are shown for the HDE algorithm. In Figs. D.3 and D.4, this is shown for the ACO algorithm. Note that for some parameters, the influence of the computing time prevents the lowest objective value (makespan plus resource penalty) to be picked. For example, parameter q has its best objective value at $q = 5$, but $q = 1$ is selected due to its shorter computing time.

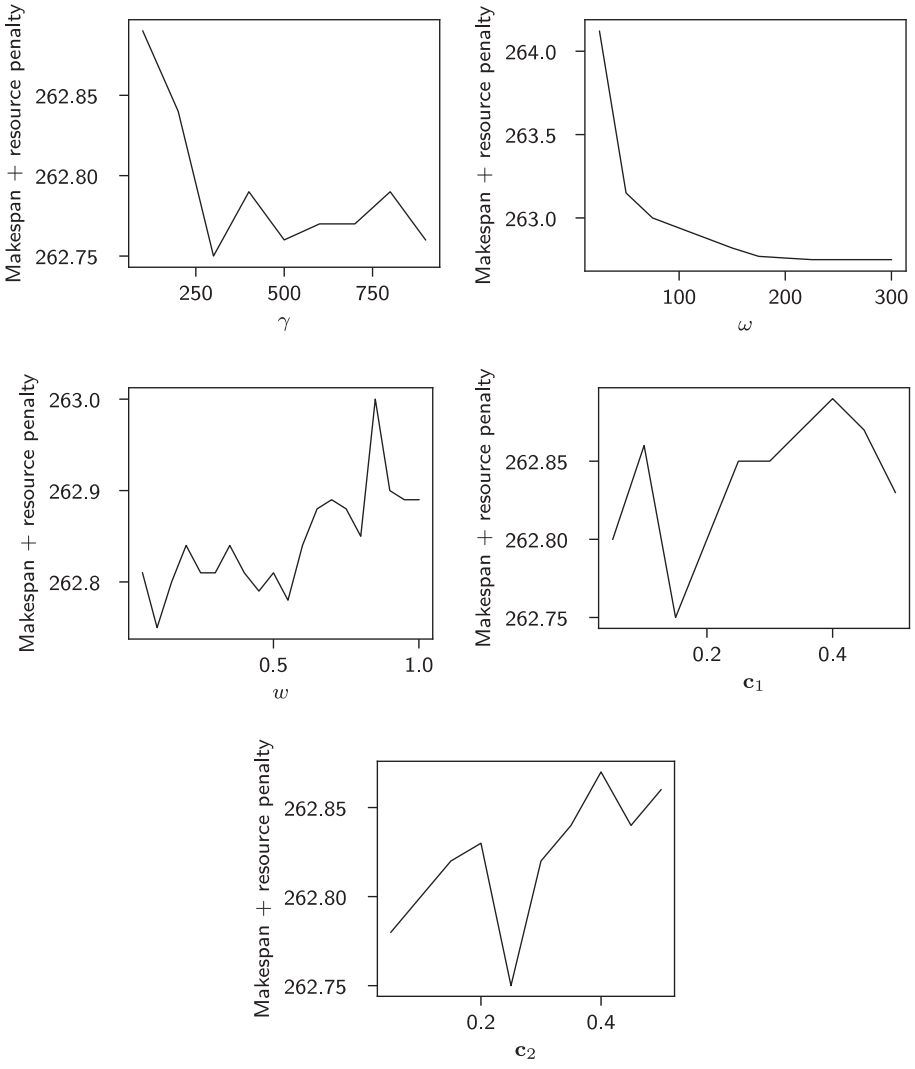


Fig. D.1. Average makespan plus resource penalties for the final iteration of the HDE parameter tuning process.

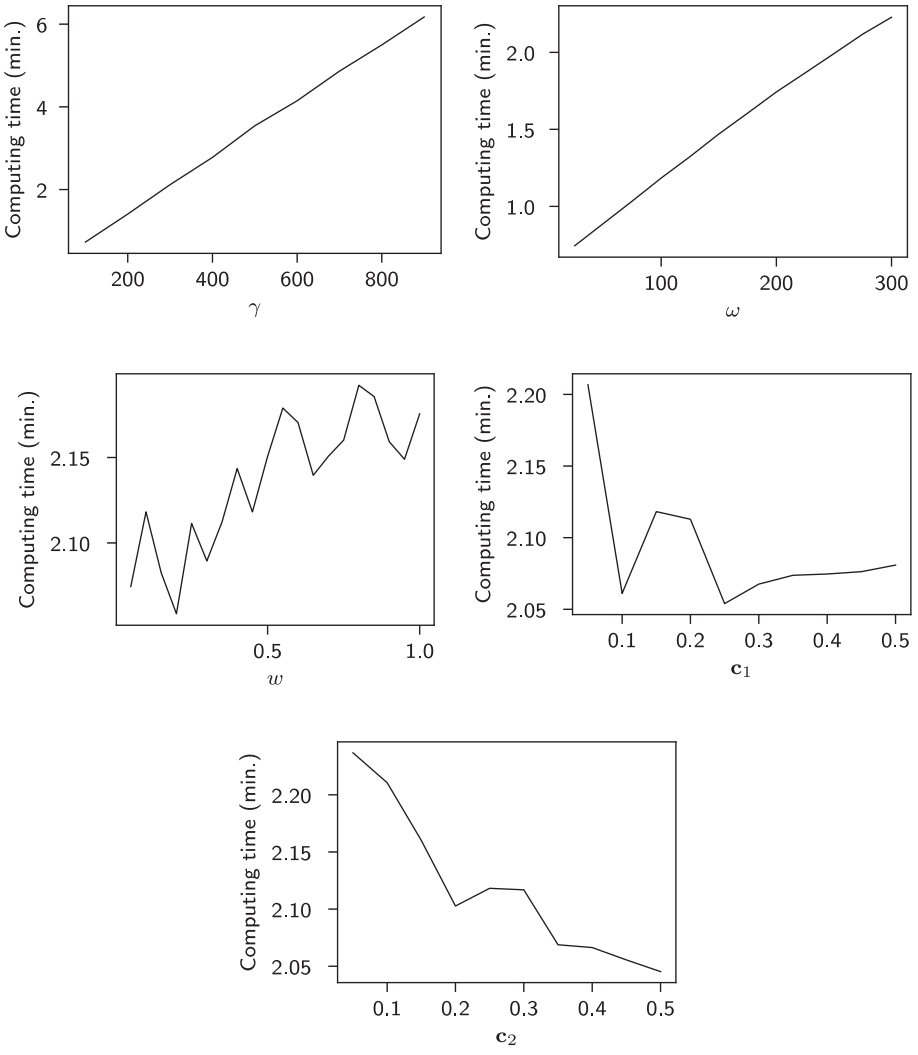


Fig. D.2. Average computing times for the final iteration of the HDE parameter tuning process.

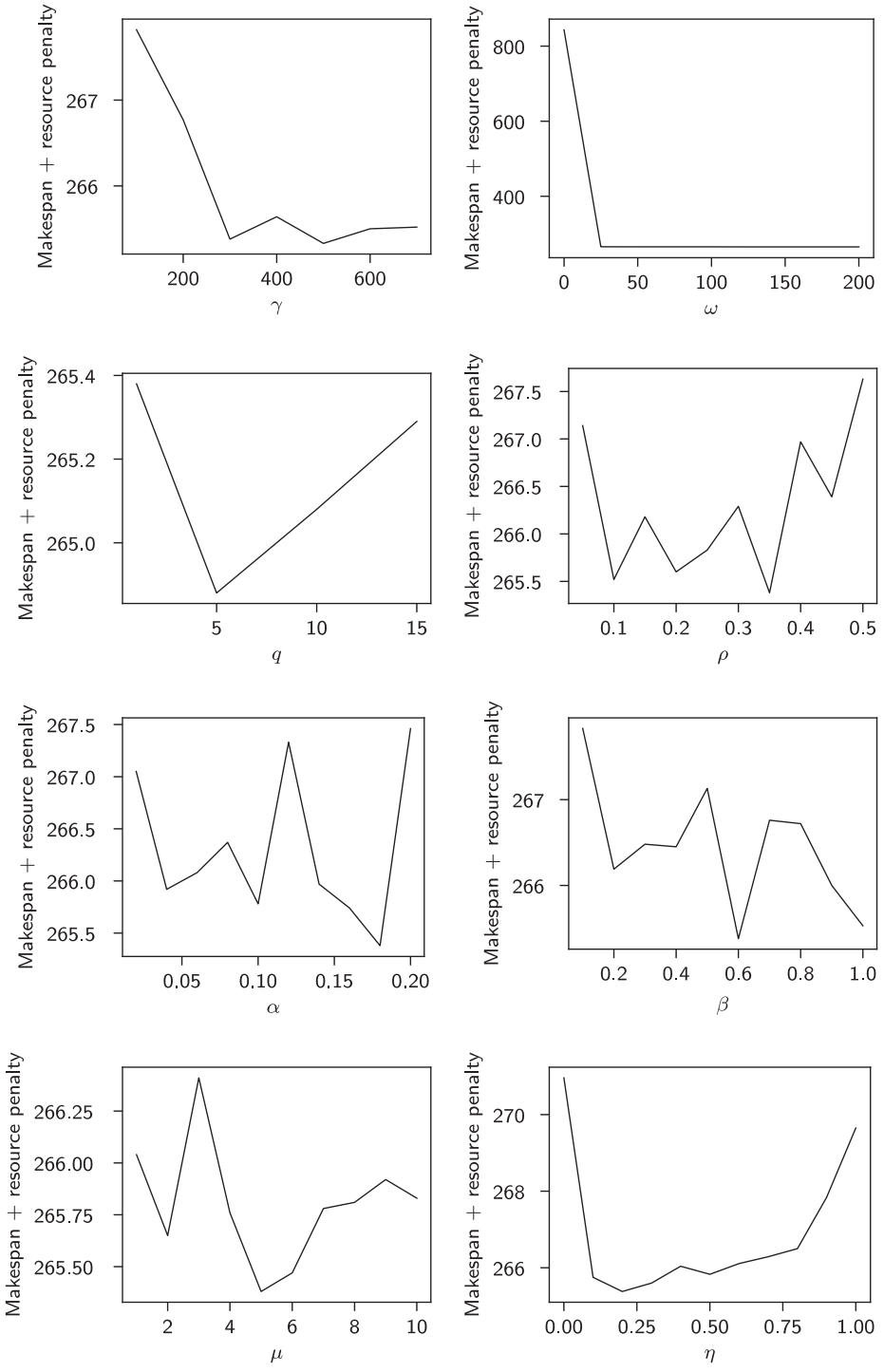


Fig. D.3. Average makespan plus resource penalties for the final iteration of the ACO parameter tuning process.

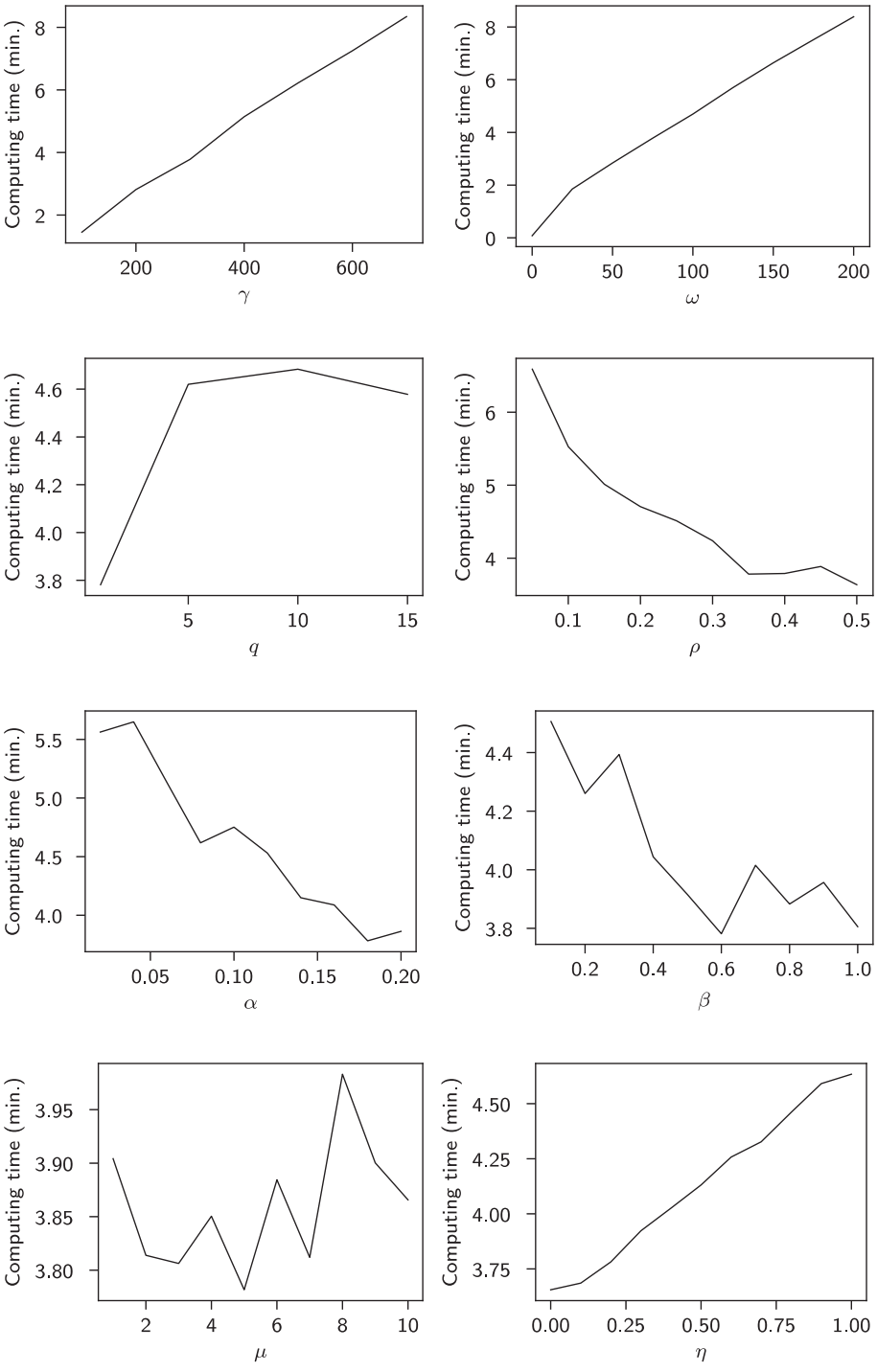


Fig. D.4. Average computing times for the final iteration of the ACO parameter tuning process.

Appendix E. Additional computational results

Additional computational results are given in Figures E.1–E.4.

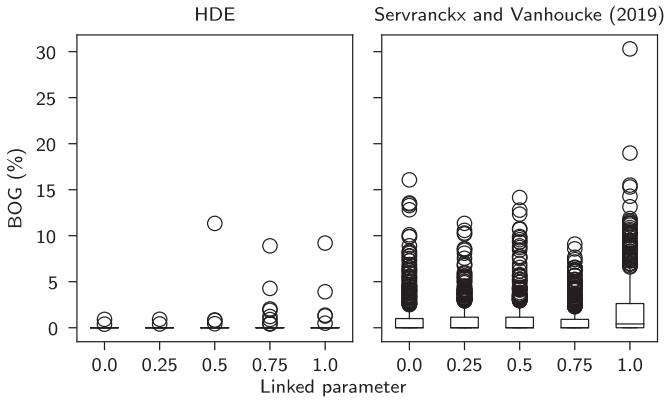


Fig. E.1. BOG of HDE and Servranckx & Vanhoucke (2019) algorithm against the *linked* parameter.

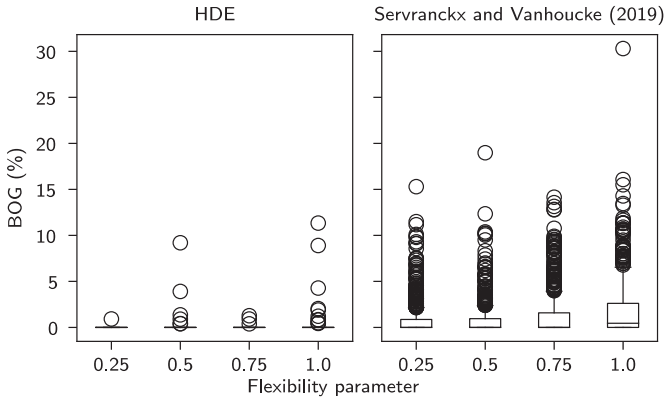


Fig. E.2. BOG of HDE and algorithm from Servranckx & Vanhoucke (2019) against the *flexibility* parameter.

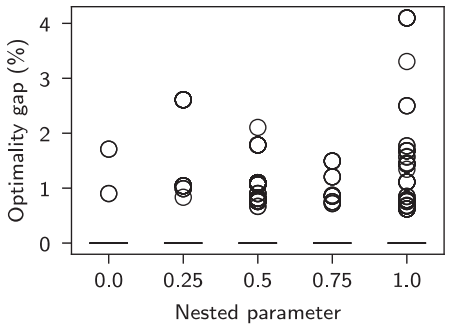


Fig. E.3. Boxplots of optimality gaps for the HDE algorithm against the *nested* parameter.

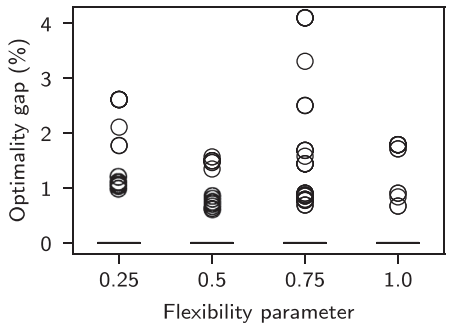


Fig. E.4. Boxplots of optimality gaps for the HDE algorithm against the *flexibility* parameter.

References

Ali, I. M., Elsayed, S. M., Ray, T., & Sarker, R. A. (2016). A differential evolution algorithm for solving resource constrained project scheduling problems. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics): vol. 9592* (pp. 209–220).

Barták, R., Čepek, O., & Surynek, P. (2007). Modelling alternatives in temporal networks. In *Proceedings of the 2007 IEEE symposium on computational intelligence in scheduling, CI-Sched 2007*.

Bellenguez, O., & Néron, E. (2005). Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), LNCS: Vol. 3616* (pp. 229–243).

Bezerra, P., & Scheer, S. (2021). *A metaheuristic procedure combined with 4D simulation as an alternative for the scheduling process of housing complexes*: 98. Springer International Publishing.

Blazewicz, J., Lenstra, J., & Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics, 5*(1), 11–24.

Čapek, R., Šcha, P., & Hanzálek, Z. (2012). Production scheduling with alternative process plans. *European Journal of Operational Research, 217*(2), 300–311.

Carlier, J., Moukrim, A., & Xu, H. (2009). The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. *Discrete Applied Mathematics, 157*(17), 3631–3642.

Fafandjel, N., Rubeša, R., & Mrakovčic, T. (2008). Procedure for measuring shipbuilding process optimisation results after using modular outfitting concept. *Strojarstvo, 50*(3), 141–150.

Hu, S., Zhang, Z., Wang, S., Kao, Y., & Ito, T. (2019). A project scheduling problem with spatial resource constraints and a corresponding guided local search algorithm. *Journal of the Operational Research Society, 70*(8), 1349–1361.

Kellenbrink, C., & Helber, S. (2015). Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research, 246*(2), 379–391.

Koné, O., Artigues, C., Lopez, P., & Mongeau, M. (2013). Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal, 25*(1-2), 25–47.

Kuster, J., Jannach, D., & Friedrich, G. (2009). Extending the RCPSP for modeling and solving disruption management problems. *Applied Intelligence, 31*(3), 234–253.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence, 143*(2), 151–188.

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation, 6*(4), 333–346.

Neumann, K., & Schwindt, C. (2003). Project scheduling with inventory constraints. *Mathematical Methods of Operations Research, 56*(3), 513–533.

Pellerin, R., Perrier, N., & Berthaut, F. (2019). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research, 175*(2), 707–721.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science, 16*(1), 93–108.

Quoc, H. D., The, L. N., Doan, C. N., & Thanh, T. P. (2020). New effective differential evolution algorithm for the project scheduling problem. In *2020 2nd International conference on computer communication and the internet (ICCCI)* (pp. 150–157). IEEE.

Sahli, A., Carlier, J., & Moukrim, A. (2016). Comparison of mixed integer linear programming models for the event scheduling problem with consumption and production of resources. *IFAC-PapersOnLine, 49*(12), 1044–1049.

Sallam, K. M., Chakrabortty, R. K., & Ryan, M. J. (2020). A two-stage multi-operator differential evolution algorithm for solving resource constrained project scheduling problems. *Future Generation Computer Systems, 108*, 432–444.

Servranckx, T., & Vanhoucke, M. (2019). A tabu search procedure for the resource–constrained project scheduling problem with alternative subgraphs. *European Journal of Operational Research, 273*(3), 841–860.

Shirzadeh Chaleshtarti, A., Shadrokh, S., Khakifirooz, M., Fathi, M., & Pardalos, P. M. (2020). A hybrid genetic and lagrangian relaxation algorithm for resource-constrained project scheduling under nonrenewable resources. *Applied Soft Computing Journal, 94*, 106482.

Storn, R., & Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

Tao, S., & Dong, Z. S. (2017). Scheduling resource-constrained project problem with alternative activity chains. *Computers and Industrial Engineering, 114*, 288–296. September

Tao, S., & Dong, Z. S. (2018). Multi-mode resource-constrained project scheduling problem with alternative project structures. *Computers and Industrial Engineering, 125*, 333–347. (November 2017)

Valls, V., Ballestín, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 185*(2), 495–508.

Van der Beek, T., Van Essen, J.T., Pruyn, J., & Aardal, K. (2022). Exact solution methods for the resource constrained project scheduling problem with a flexible project structure. https://optimization-online.org/?p=18944.

Van der Beek, T. (2021). Instances, file format, generator script and results for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources. doi:10.4121/16764205.

Wang, J.-c., & Lui, W.-r. (2017). Forward-backward improvement for genetic algorithm based optimization of resource constrained scheduling problem. In *DEStech transactions on computer science and engineering* (pp. 349–356).

Wu, L., Wang, Y., & Zhou, S. (2014). Improved differential evolution algorithm for resource-constrained project scheduling problem. *Journal of Systems Engineering and Electronics, 21*(5), 798–805.

Zaman, F., Elsayed, S., Sarker, R., Essam, D., & Coello Coello, C. A. (2021). An evolutionary approach for resource constrained project scheduling with uncertain changes. *Computers & Operations Research, 125*.