TU Delft

Delft University of Technology

Exact solution methods for the Resource Constrained Project Scheduling Problem with a flexible Project Structure

van der Beek, T.; van Essen, J. T.; Pruyn, J.; Aardal, K.

DOI
10.1016/j.ejor.2024.10.029

Publication date
2025

Document Version
Final published version

Published in
European Journal of Operational Research

Citation (APA)
van der Beek, T., van Essen, J. T., Pruyn, J., & Aardal, K. (2025). Exact solution methods for the Resource Constrained Project Scheduling Problem with a flexible Project Structure. *European Journal of Operational Research*, *322*(1), 56-69. https://doi.org/10.1016/j.ejor.2024.10.029

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

European Journal of Operational Research 322 (2025) 56–69

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete optimization

Exact solution methods for the Resource Constrained Project Scheduling Problem with a flexible Project Structure

T. van der Beek [a,*], 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, Netherlands
[b] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands

ARTICLE INFO

Keywords:
Project scheduling
Resource Constrained Project Scheduling Problem
Integer programming
Flexible project structure

ABSTRACT

The Resource Constrained Project Scheduling Problem with a flexible Project Structure (RCPSP-PS) is a generalization of the Resource Constrained Project Scheduling Problem (RCPSP). In the RCPSP, the goal is to determine a minimal makespan schedule subject to precedence and resource constraints. The generalization introduced in the RCPSP-PS is that, instead of executing all activities, only a subset of all activities has to be executed. We present a model that is based on two graphs: one representing precedence relations and one representing the activity selection structure. The latter defines which subset of activities has to be executed. Additionally, we present theoretical properties of this model and give an exact solution method that makes use of these properties by generating cutting planes and setting bounds on variables. Furthermore, three problem properties are introduced to classify problems in the literature. We compare our model to a model from literature on instances that possess a subset of these three problem properties and find a reduction in computing time. Furthermore, by comparing results on instances that possess all problem properties, it is shown that the computing times are decreased and better lower bounds are found by the cutting planes and variable bounds presented in this paper.

1. Introduction

In many real-world projects, a schedule is needed that determines the order in which activities are executed to complete a project such that the project makespan is minimized. Furthermore, many of these projects can be completed in multiple ways, since various (sub-)tasks can be achieved by different (groups of) activities. Projects like this are found in, for example, housing construction (Servranckx & Vanhoucke, 2019), highway project construction (Wu et al., 2010), modular shipbuilding (Rubeša et al., 2011) and aircraft turnaround scheduling (Kellenbrink & Helber, 2015). For these type of projects, two decisions have to be made for activities: are they executed, and if so, at what time? The *Resource Constrained Project Scheduling Problem with a flexible Project Structure* (RCPSP-PS) can be used to model projects of this structure.

In the RCPSP-PS, a list of activities is given as input, which can be used to complete a project. Since there are alternative ways of executing certain tasks, it has to be decided which activities have to be executed in order to complete the project. This is called the *activity selection problem*. Subsequently, the selected activities have to be scheduled, while satisfying precedence constraints. Furthermore, each activity has resource requirements, and the total amount of resources required at any time cannot exceed a given capacity. Scheduling the selected activities according to these constraints is called the *activity scheduling problem*.

In this paper, we define three problem properties to classify research on various models for the RCPSP-PS (see Section 2). Although several models in the literature feature some of these properties, there is not yet a model that combines all three. Therefore, we introduce a new model for the selection logic, based on the model of Kellenbrink and Helber (2015). This model supports problems with all three of these properties. Secondly, in addition to introducing this model, we present theoretical properties based on the number of activities that can be executed within a group of activities. Specifically, we identify groups for which *at least one* activity has to be executed, and groups for which *at most one* activity has to be executed. Finally, we give an example of how these theoretical properties can be used in an exact solution method, based on variable reduction and cutting planes. The proposed solutions methods are then evaluated against the model of Kellenbrink and Helber (2015) on instances that possess a subset of problem properties, and against each other on instances that possess all problem properties.

* Corresponding author.
E-mail addresses: T.vanderbeek@tudelft.nl (T. van der Beek), 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.2024.10.029
Received 3 October 2022; Accepted 22 October 2024
Available online 6 November 2024
0377-2217/© 2024 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

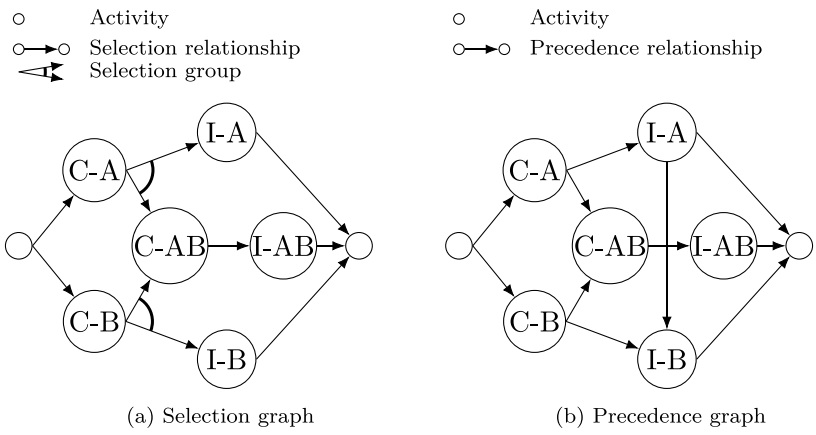


Fig. 1. Example of building a module AB on a ship.

In Section 2, an example problem is provided along with a description of the introduced problem properties, which are subsequently used to classify literature on different models for the RCPSP-PS in Section 3. The problem description and Integer Linear Program (ILP) formulation are given in Section 4. After this, the theoretical properties of the selection logic are presented in Section 5. To give an example of how these properties can be used in exact solution methods, Section 6 presents a solution method based on cutting planes and variable reduction. Finally, the computational results are presented and discussed in Sections 7 and 8 concludes the paper.

2. Problem properties and example RCPSP-PS

In this section, an example of the RCPSP-PS is given, which we use to explain three properties to classify different variants of the flexible project structure in Section 3. Consider a small project building a module AB on a ship. For this, we have two actions: constructing a part, meaning to construct it outside the ship, and installing it, meaning to work directly on the ship. In Fig. 1(a), we present the activity selection graph. This graph defines the logic of which activities to select to be executed. Each node in this graph represents an activity, and each edge defines a selection relationship: if the source node is executed, the target node has to be executed as well. If there is an arc between two or more edges, it means that if the source node is executed, *exactly one* of the target nodes has to be executed.

The construction activities are denoted by 'C-'. Thus, we can see that we always have to construct module part A (C-A) and B (C-B). Subsequently, we can proceed either by installing A and B on the ship (activities I-A and I-B), or by first constructing module AB (C-AB) and then installing module AB (I-AB). In Fig. 1(b), we see the precedence graph of the same problem. Here, an edge (i, j) defines that if both activity i and j are executed, activity j can only start after activity i is finished. We can see that the construction activities have to be performed before the installation activities. Furthermore, if we choose to install module A and B separately, module A has to be installed before module B (edge between (I-A) and (I-B)).

We now present three properties of this problem, which are used in the literature review in Section 3. First, it can be seen that the selection graph and precedence graph are not the same, due to the precedence edge between I-A and I-B. We call this *separate scheduling and selection logic*. Second, we introduce the term *Independent Succession* (IS), to denote whether the problem allows an activity to have multiple other activities that independently can cause this activity to be selected. In the example, the selection of activity C-AB can be caused by either C-A, C-B, or both. Finally, as stated before, from a group of selection-based successors, *exactly one* can be executed. For example, it is not possible to execute both activity I-A and C-AB. If a problem has this feature, we define this as the *Exclusivity Criterion* (EC).

3. Literature overview

The Resource Constrained Project Scheduling Problem (RCPSP) is a classical optimization problem, introduced by Pritsker et al. (1969) and proven to be NP-hard by Blazewicz et al. (1983). Since then, numerous studies have focused on developing heuristic and exact solution methods. An overview of this research can be found in, for example, Pellerin et al. (2019) and Lombardi and Milano (2012). These two review papers discuss heuristic and exact solution methods, respectively. In this section, we focus on the flexible project structures extension for the RCPSP. One problem that enables different ways of completing a project is the *Multi-Mode Resource Constrained Project Scheduling Problem* (MRCPSP) (Talbot, 1982). In this problem, each task has multiple execution modes with varying durations and/or resource usage. An overview of variations and of solution methods for the MRCPSP can be found in Węglarz et al. (2011).

The RCPSP-PS is a generalization of the MRCPSP where only a subset of all activities has to be selected for execution. Both the name of the problem and the way the selection decisions are modeled vary across the literature. One of the earliest models was given by Kuster et al. (2009), who introduce the *Extended RCPSP*, which includes independent succession and the exclusivity criterion. They model the execution decisions by introducing a set of active activities: activities that are initially set to be executed. Substitution activities are introduced for some of these active activities and these substitution activities can be executed instead of the corresponding active activities. Finally, the model is completed by adding a set of dependencies; execution requirements for an activity if another activity is activated or inactivated. To find good feasible solutions for this model, an evolutionary algorithm is used.

Kellenbrink and Helber (2015) separate the scheduling requirements from the precedence requirements, and give a model based on a set of choices and a distinction between optional and mandatory activities. Furthermore, they include EC and nonrenewable resources: resources that do not renew after the activity has ended. They call this problem the RCPSP-PS and solve it heuristically using a genetic algorithm. Each activity has a set of activities it can cause to be selected. Furthermore, the model includes a time-indexed ILP formulation that imposes some restrictions on the selection logic: it is not possible for multiple activities to have the same activity in the set of activities it can cause to be selected. This restricts the modeling process.

Tao and Dong (2017) introduce the *RCPSP with alternative activity chains* that includes nonrenewable resources and IS, where they give a single network that defines both the precedence constraints and the selection constraints based on AND-activities and OR-activities. An AND-activity is an activity for which all successors have to be executed and an OR-activity is an activity for which at least one successor has to be executed. By using a single network for both the precedence

Table 1
Overview of models with a flexible project structure.

Paper	Separate scheduling and selection	Independent succession	Exclusivity criterion	Nonrenewable resources	Exact method	Heuristic method
Kuster et al. (2009)		✓	✓			✓
Kellenbrink and Helber (2015)	✓		✓	✓	✓	✓
Tao and Dong (2017)		✓		✓	✓	✓
Servranckx and Vanhoucke (2019)		✓	✓			✓
Hauder et al. (2020)		✓	✓		✓	
This paper	✓	✓	✓		✓	

and selection constraints, separation of precedence and selection logic is not possible; every precedence relationship is equal to a selection relationship and vice versa. This means that problems where the precedence and selection constraints do not coincide cannot be modeled with this approach. They solve this problem heuristically using a simulated annealing based algorithm. In Tao and Dong (2018), they extend this to multiple objectives.

Furthermore, Servranckx and Vanhoucke (2019) present the *RCPSP with alternative subgraphs* which includes IS and EC. Their model is based on alternative subgraphs and branches; a branch consists of a set of activities, and an alternative subgraph is a collection of branches of which exactly one has to be executed. This model also has a single network for both precedence and selection logic and is solved heuristically by a tabu search procedure.

Finally, Hauder et al. (2020) use a network with different types of activities (OR, AND, and OUT) to model both the selection and scheduling problem, while adding the extension of supporting multiple projects. The model supports IS and EC. Besides the standard objective of makespan minimization, they also consider time balance and resource balance as objective functions. They provide time indexed Mixed Integer Linear Program (MILP) formulations for these models and a constraint programming method.

An overview of all models is given in Table 1. It can be seen that there is not yet a model combining separation of scheduling and selection, independent succession and the exclusivity criterion. Furthermore, most of the focus is on heuristic methods. Although three papers include an exact method, for two of these, it comprises of only solving an (M)ILP model. Although there are differences in these (M)ILP formulations for the flexible project structure, they all share a similar time-indexed basic RCPSP model.

4. Problem formulation of the RCPSP-PS

In this section, the problem formulation of the RCPSP-PS is given. In Section 4.1, a description of the problem is given, which includes a description of the selection graph with selection groups. Section 4.2 presents an ILP formulation for the RCPSP-PS.

4.1. Problem description

In the RCPSP-PS, a set of activities N is given of which a subset has to be executed to complete the project such that the makespan of the project is minimized. Let $n = |N| - 2$ be the number of non-dummy activities. Then, the starting activity is activity 0 and the final activity is activity $n + 1$. Both of these activities have a duration and resource requirement of zero and the final activity can only be executed after all other executed activities. The time horizon during which these activities are scheduled is represented by a set of discrete time periods $T = \{0, \dots \mathcal{T}\}$. Each activity $i \in N$ has a duration of d_i time periods. Activities have to be scheduled while satisfying resource, precedence and selection constraints. The set of resources is denoted by R. Each resource $r \in R$ has a capacity of λ_r, and each activity $i \in N$ uses k_{ri} units of resource r across the whole duration of the activity. The precedence relationships are defined by a set of tuples $\mathcal{P}$. For each $(i, j) \in \mathcal{P}$, it is required that activity i is finished before the start of activity j. Furthermore, the set $\mathcal{PD}_j$ contains all predecessors of activity j and the set S_i contains all successors of activity i in the precedence graph.

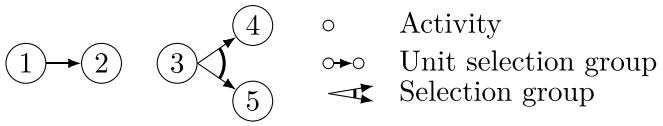


Fig. 2. Selection graphs for a unit selection group $(a_g, S_g) = (1, \{2\})$ (left) and a non-unit selection group $(a_g, S_g) = (3, \{4, 5\})$ (right).

In the RCPSP-PS, only a subset of activities has to be executed. To define the choices on the selection of activities, the concept of *selection groups* (denoted by set G) is introduced. A selection group $g \in G$ is defined by an *activator activity* a_g and a set of one or more *successor activities* S_g. If an activator activity is executed, exactly one of the successor activities has to be executed, which means that a selection group defines an 'exclusive or'-relationship. Since we have separate selection and scheduling logic, there is not necessarily a time-based precedence relationship between each activator activity a_g and (selection-based) successor activity $i \in S_g$. Thus, we introduce the set of *selection groups with full precedence* $H \subseteq G$, to define selection groups where this *is* the case. This set contains all *selection groups* with a *time-based* precedence relationship between the activator and all successors, i.e., $H = \{g \in G : (a_g, j) \in \mathcal{P}, \forall j \in S_g\}$.

We define a *unit selection group* as a selection group $g \in G$ with only one successor (i.e., $|S_g| = 1$). This defines a direct consequential relationship; if activator activity a_g is executed, the single successor activity in S_g will have to be executed as well. In Fig. 2, a unit selection group and a non-unit selection group are shown.

The precedence and selection relationships split up the RCPSP-PS into two problems. The first one is selecting which activities to execute. This is called the *selection problem*. The next question is when to schedule the executed activities. This is defined as the *scheduling problem*.

4.2. Model formulation

We now introduce an ILP formulation for the RCPSP-PS, which is used in the solution methods in two ways: (1) we solve the linear relaxation to find cutting planes, and (2) we solve the model to (near) optimality using Gurobi. For the model, we introduce binary decision variables X_{it} that are equal to one if activity $i \in N$ starts at time $t \in T$, and zero otherwise. Constraints (1a) minimizes the completion time of the final activity, and thus, the total project makespan. The first and final activities are always executed due to Constraints (1b) and (1c), respectively. Note that Constraints (1c) can also follow from the selection groups. Constraints (1d) impose that each activity can only be executed once. Furthermore, Constraints (1e) make sure that if activator activity a_g of selection group $g \in G$ is executed, at least one successor activity $i \in S_g$ has to be executed. Constraints (1f) impose that if activator a_g of selection group $g \in G$ is executed, at most one successor activity is executed. The precedence constraints are set by Constraints (1g). These constraints define that for each $(i, j) \in \mathcal{P}$, if both are executed, the starting time of activity i plus its duration

d_i cannot be larger than the starting time of activity j. Furthermore, Constraints (1h) define that for each resource $r \in R$ and time $t \in T$, the total resource usage is smaller than or equal to the resource capacity λ_r. Finally, Constraints (1i) specify that the decision variables X_{it} are binary.

$$\min \sum_{t \in T} t X_{(n+1)t}, \tag{1a}$$

$$\sum_{t \in T} X_{0t} = 1, \tag{1b}$$

$$\sum_{t \in T} X_{(n+1)t} = 1, \tag{1c}$$

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

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

$$\sum_{j \in S_g} \sum_{t \in T} X_{jt} \leq |S_g| - \left(|S_g| - 1\right) \sum_{t \in T} X_{a_g t}, \quad \forall g \in G, \tag{1f}$$

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

$$\sum_{i \in N} \sum_{u=1}^{d_i} k_{ri} X_{i(t-u+1)} \leq \lambda_r, \quad \forall r \in R, t \in T, \tag{1h}$$

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

In Kellenbrink and Helber (2015), the selection logic was mainly modeled by *choices*, *activities causing a choice*, and *optional activities per choice*. These are analogous to selection groups, activators and successor activities, respectively. However, in Kellenbrink and Helber (2015), it is imposed that if a *caused* activity is selected, exactly one *causing* is selected. This prevents IS. In terms of selection decisions, the model presented in this paper can be seen as a generalized version of the model in Kellenbrink and Helber (2015). However, the current paper does not consider nonrenewable resources, since the focus is on the flexible project structure. However, the constraints for nonrenewable resources, given in Kellenbrink and Helber (2015), could be added directly to the model presented in the current paper.

After introducing the model, we now give a formal definition for the activity selection problem. Recall that the exclusivity criterion is the criterion that per selection group with an executed activator activity, exactly one successor has to be executed. With this, the *selection problem* (imposed by Constraints (1d)–(1f)) is defined as follows: Given a selection graph, find a selection of activities including activity 0, such that for each selection group $g \in G$, exactly one activity j from the set of successor activities S_g is selected if activator activity a_g is selected. This problem is proven to be NP-hard by Barták et al. (2007).

5. Activity execution properties

We now introduce theoretical properties of the RCPSP-PS, which are used to develop the solution method in Section 6. We define two types of activity sets, for which certain execution properties are known, namely *Non-Empty Execution Sets* (NEESs) and *Max-One Execution Sets* (MOESs). The former defines a set of activities of which *at least one* activity *has* to be executed, while the latter is a set where *at most one* activity *can* be executed. These sets can be used in building blocks for solution methods, as is shown in Section 6.

5.1. Non-empty execution sets

The first type of activity set for the RCPSP-PS we introduce is the NEES. As stated above, a NEES is a set of activities of which *at least one* activity has to be executed. In this subsection, we show how to identify a NEES from a set of candidate activities $N' \subseteq N$, based on creating an ILP. For intuition, consider all activities as the candidate set ($N' = N$) in the selection graph as shown in Fig. 3. We traverse the selection graph, starting in the root activity r. Then, a set of activities A is a NEES if, regardless of the choices we make in the selection groups, we always end up in an activity in A. Therefore, if we reach a certain group g that is on the path to A, there should still be a path to A no matter what successor we pick. In Fig. 3, no choice can be made at selection group g_1 such that no activity in A will be executed.

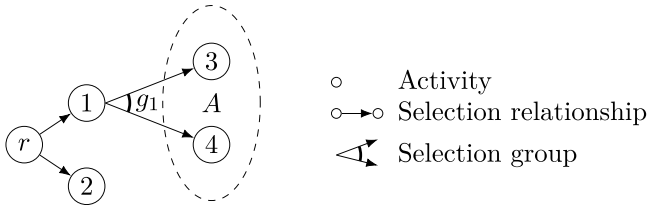


Fig. 3. Selection graph example of a NEES. Since either activity 3 or activity 4 always has to be executed, $A = \{3,4\}$ is a NEES.

To identify a NEES in a subset of activities $N' \subseteq N$, we solve the ILP given by Constraint set (2). This ILP selects selection groups and activities, where all selected selection groups form a set of paths, starting at activity 0. As long as each path ends in a selected activity, the set of selected activities form a NEES. To formulate this, we introduce the set G_i that contains all groups $g \in G$ with activator $a_g = i$. Furthermore, we introduce binary variables U_g for each $g \in G$ and V_i for each $i \in N'$, which are equal to one if a selection group or activity is selected, respectively, and zero otherwise.

With this, Constraint (2a) imposes that at least one selection group has to be selected at starting activity 0, which is the start of all paths. Then, the paths are continued due to Constraints (2b) and (2c). These impose that if a selection group g is selected, for each successor $i \in S_g$ either a succeeding group $h \in G_i$ is selected (path continues), or activity i itself has to be selected (path ends in activity i). If activity $i \in N'$, this is imposed by Constraints (2b). Otherwise, binary variable V_i does not exists and one group $h \in G_i$ has to be selected. This is imposed by Constraints (2c). Note that, since Constraint set (2) is used to *identify* a NEES, V_i is never bounded from above by 0, since adding any activity to a NEES still constitutes a NEES.

$$\sum_{g \in G_0} U_g \geq 1, \tag{2a}$$

$$U_g \leq V_i + \sum_{h \in G_i} U_h, \quad \forall g \in G, i \in S_g \cap N', \tag{2b}$$

$$U_g \leq \sum_{h \in G_i} U_h, \quad \forall g \in G, i \in S_g \setminus N', \tag{2c}$$

$$U_g \in \{0,1\}, \quad \forall g \in G, \tag{2d}$$

$$V_i \in \{0,1\}, \quad \forall i \in N'. \tag{2e}$$

We can now show that if $\mathbf{V}$ is a feasible solution to Constraint set (2), it constitutes a NEES.

Lemma 1. *Let $N' \subseteq N$ be a subset of activities and $\mathbf{U}, \mathbf{V}$ be the solution of Constraint set (2). Then, A given by $\{i \in N' : V_i = 1\}$ is a NEES.*

Proof. See Appendix A.1. □

5.2. Max-one execution sets

Secondly, we introduce MOESs. A MOES is a set of activities, for which *at most one* activity is executed in the optimal solution. To

identify these sets, we introduce *Common Rooted Paths* (CRP), as defined in Definition 5.1. For this, we introduce the notation of the *vertex sequence* of a path P; the sequence of all vertices on a path P, denoted by $\mathcal{V}(P)$.

Definition 5.1 (*Common Rooted Path (CRP)*)**.** For two activities i and j, it is said that they have a common rooted path (r, P, Q) if there is another activity r with a path P from r to i and a path Q from r to j with the following properties: The first activities p_1 and q_1 on P and Q after r, respectively, do not belong to the same selection group of activator r. Furthermore, after splitting at r, paths P and Q are disjoint; $\mathcal{V}(P) \cap \mathcal{V}(Q) = \{r\}$.

Based on this definition, we give Proposition 1 to identify whether it is allowed for any two activities to both be executed in the optimal solution.

Proposition 1. *If two distinct activities $i \in N$ and $j \in N$ in a selection graph do not have a common rooted path, at most one of them will be executed.*

Proof. See Appendix A.2. □

Thus, if there is no CRP between any pair of activities in a set, at most one of the activities in this set can be executed. To determine such a set of activities, we introduce a *rooted path graph* (RPG), as defined in Definition 5.2.

Definition 5.2 (*Rooted Path Graph (RPG)*)**.** A rooted path graph $A_G = (N, E)$ is a graph with the same activities N as the selection graph and with an edge $(i, j) \in E$ if and only if there is a common rooted path between i and j.

It then follows that an independent set for the RPG represents a set of activities without any CRP's between them, and thus, a MOES. In Appendix B, we show how to generate an RPG for any acyclic selection graph.

6. Solution method

In this section, we give an example of how NEESs and MOESs can be used in an exact solution method for the RCPSP-PS. This method consists of a preprocessing step based on NEESs and MOESs, and subsequently, using an ILP solver. The preprocessing procedure uses three building blocks: a variable reduction method that is introduced in Section 6.1 and valid inequalities and cutting planes that are introduced in Section 6.2. These building blocks are then combined to create the solution algorithm, which is presented in Section 6.3. The algorithm first reduces the search space by bounding variables, and then solves the resulting ILP using Gurobi.

6.1. Variable reduction

The first building block is the variable reduction method. The idea behind this method is as follows: if each activity in a Non-Empty Execution Set (NEES) A has the same successor activity j in the precedence graph, then the earliest start time s_j of activity j is equal to $\min_{i \in A}\{s_i + d_i\}$. Similarly, if each activity in a NEES A has the same predecessor activity i, the latest finish time f_i of activity i is equal to $\max_{j \in A}\{f_j - d_j\}$

We now give an algorithm, based on the principle described above, to compute the earliest starting time s_i for each activity $i \in N$.

The first loop in Algorithm 1 loops over all activities $j \in N$ in topological order. For each activity $j \in N$, all activities $i \in \mathcal{PD}_j$ are ordered by earliest finishing times $s_i + d_i$ in non-increasing order in sequence $B = \{b_1, \ldots, b_{|B|}\}$. The second loop takes incremental subsets B' of B and tries to solve Constraint set (2) with $N' = B'$. For each iteration of Loop 2, an element k is added to $N' = B'$ until Constraint set (2) becomes feasible. If adding element k to N' results in Constraint set (2) becoming feasible, it follows that $V_k = 1$. Otherwise, Constraint set (2) would also be feasible in the previous iteration. Since B' is ordered in non-increasing order of $s_i + d_i$, B' contained all activities $i \in B$ with $s_i + d_i \geq s_k + d_k$ in the previous iteration. Since solving Constraint set (2) in the previous iteration did not give a feasible solution, B' maximizes $\min_{i \in B'}\{s_i + d_i\}$.

Algorithm 1 Preprocessing algorithm

```
N^(s) ← topological sorting of N on precedence graph
s_i ← 0  ∀  i ∈ N
for all j ∈ N^(s) do                                   ▷ Loop 1
    B ← PD_j, sorted by non-increasing s_i + d_i.
    stop ← False
    n ← 1
    while stop = False AND n ≤ |B| do                  ▷ Loop 2
        B' ← first n elements of B
        Solve Constraint set (2) for N' = B'
        if Constraint set (2) is feasible then
            s_j ← min_{i∈B'} {s_i + d_i}
            stop ← True
        end if
        n ← n + 1
    end while
end for
```

Something similar can be done for the latest finish time f_i for all activities $i \in N$. The difference is that f_i is initially given a value of $|T|$ for every activity and the algorithm runs backwards. This means that Loop 1 is reversed, B contains all successors of j and is ordered in non-decreasing order of $f_i - d_i$. Furthermore, instead of updating s_j, f_j is updated to $\max_{i \in B'}\{f_i - d_i\}$. Finally, after both preprocessing steps, all variables X_{it} with $t < s_i$ or $t > f_i - d_i$ can be set to zero.

6.2. Valid inequalities and cutting planes

The second building block that is used to reduce the search space, are the addition of valid inequalities. These are added for each selection group with full precedence $g \in H \subseteq G$ and are shown in Constraints (3). The inequality states that if a selection group has full precedence, each successor activity has to be executed later than the activating activity, if both are executed.

$$\sum_{t \in T}(t + d_{a_g})X_{a_g t} \leq \sum_{j \in S_g}\sum_{t \in T} tX_{jt}, \quad \forall g \in H, \tag{3}$$

Furthermore, the last building block are two types of cutting planes. The first type is based on groups of activities of which *at least* one has to be selected and is introduced in Section 6.2.1. For the second type, introduced in Section 6.2.2, groups of activities of which *at most* one can be selected are used.

6.2.1. Non-empty execution cutting planes

The first set of cutting planes are based on *Non-Empty Execution Sets*. Recall that these are sets of which at least one activity has to be executed. The separation problem is given by Objective function (4) subject to the constraints of Constraint set (2), in order to find a NEES with less than one executed activity in the relaxed solution. Here, X^*_{it} is the solution obtained by solving the LP relaxation and V_i is the decision variable from Constraint set (2) indicating whether an activity $i \in N$ is in a NEES.

$$\min \sum_{i \in N} V_i \sum_{t \in T} X^*_{it}. \tag{4}$$

By Lemma 1, the index set of V forms a NEES. Thus, if the value of Objective function (4) is smaller than 1, the fractional solution X^*

contains a NEES that has less than one activity executed. Therefore, we add Constraint (5) as a cutting plane, where A is the index set of $\mathbf{V}$; $A = \{i \in N : V_i = 1\}$,

$$\sum_{i \in A} \sum_{t \in T} X_{it} \geq 1. \tag{5}$$

6.2.2. Max-one cutting planes

The second type of cutting planes are named '*max-one cutting planes*'. These cutting planes are based on a Max-One Execution Set (MOES) and a group of activities for which the MOES has full precedence. As stated earlier, a selection group with full precedence has a precedence relationship between the activator and each successor. The idea of the cutting planes is that if one activator activity i from the MOES is executed, then there is always a successor activity j executed that has to be executed after finishing activity i. We give the separation problem for these cutting planes in Constraints (6). For this, we define E as the set of all edges in the RPG. The ILP formulated by Constraints (6) uses the optimal relaxed solution of Constraints (1), denoted by X^*_{it}. The activities of the MOES are captured by binary variables W_i, which are equal to one if activity $i \in N$ is selected for the MOES and zero otherwise. The set of selected successor activities of the MOES is defined by binary variables Z_j for all $j \in N$, where $Z_j = 1$ if activity j is selected and zero otherwise. These two sets of activities are linked by selected selection groups. If a selection group $g \in G$ is selected, then binary variable Y_g equals 1 and 0 otherwise.

Constraints (6b) only select full-precedence groups for which the activators have no CRP. Constraints (6c) define that activities can only be selected within the MOES, if they are the activator of a selected full-precedence group. Furthermore, Constraints (6d) select only successors of selected full-precedence groups. The first term in Objective function (6a) represents the finishing times of the MOES and the second term represents the starting times of the successor activities of the activities in the MOES. By maximizing the difference between these two terms, a violation of the precedence relations can be found.

$$\max \sum_{i \in N} W_i \sum_{t \in T} (t + d_i) X^*_{it} - \sum_{j \in N} Z_j \sum_{t \in T} t X^*_{jt}, \tag{6a}$$

$$Y_g + Y_h \leq 1, \quad \forall g \in H, h \in H, (a_g, a_h) \in E, a_g \neq a_h, \tag{6b}$$

$$Y_g \geq W_{a_g}, \quad \forall g \in H, \tag{6c}$$

$$Z_j \geq Y_g, \quad \forall g \in H, j \in S_g, \tag{6d}$$

$$W_i \in \{0,1\}, \quad \forall i \in N \text{ for which } |\{g : i = a_g, g \in H\}| \geq 1, \tag{6e}$$

$$Y_g \in \{0,1\}, \quad \forall g \in H, \tag{6f}$$

$$Z_j \in \{0,1\}, \quad \forall j \in N. \tag{6g}$$

Proposition 2. *Let $\mathbf{X}^*$ be the linear relaxed solution of Constraint set (1). Furthermore, let $\mathbf{W}^*$, $\mathbf{Y}^*$ and $\mathbf{Z}^*$ be the solution of Constraint set (6) and let the value of Objective function (6a) be larger than 0. Then, Constraint (7) is a cutting plane for the RCPSP-PS that cuts of the current solution $\mathbf{X}^*$:*

$$\sum_{i \in N} W^*_i \sum_{t \in T} (t + d_i) X_{it} \leq \sum_{i \in N} Z^*_i \sum_{t \in T} t X_{it}. \tag{7}$$

Proof. See Appendix A.3. □

Both cutting plane types are used as an initial step in solving Constraint set (1). First, the LP-relaxation of Constraint set (1) is solved. Secondly, the separation problems are solved and the cuts are added to the LP-relaxation. This is repeated until no more cuts are found, or when the objective increase is lower than a certain threshold for a fixed number of iterations.

6.3. Constraint propagation algorithm

In the preceding part of this section, a method for variable reduction, valid inequalities and two types of cutting planes and their separation algorithms are given. These form the building blocks of the exact solution method given in the remainder of this section. This method uses these building blocks to set lower bounds on activity starting times and then uses constraint propagation such that increments in these bounds are propagated to other activities. We start by presenting the initialization of the algorithm. After that, we give the individual functions used and, subsequently, the solution algorithm is given.

The algorithm initializes by creating an empty set of cutting planes C_i and generating a set of *forced activities* $\mathcal{F}_i$ for each activity $i \in N$. The set C_i is later populated with cutting planes that are generated specifically for activity $i \in N$. Furthermore, the set of forced activities $\mathcal{F}_i$ for activity $i \in N$ is defined as the set of activities that always have to be executed if activity i is executed. To determine whether activity $j \in N$ is in the set of forced activities $\mathcal{F}_i$, we use a modified graph as input for Constraint set (2). First, we remove activity i and all activities $k \in N$ without a CRP between k and i to obtain N'. Since i is executed, any activity without a CRP to i is not executed. Then, if there is a feasible solution to Constraint set (2) for activity set N' with $V_j = 1$ and $V_p = 0$ for every $p \in N' \setminus \{j\}$, $\{j\}$ is a NEES and is thus always executed. In this case, activity $j \in \mathcal{F}_i$.

Furthermore, we introduce $\mathbf{s} = [s_0, \ldots, s_{n+1}]$ as the vector of earliest starting times for all activities in N, which is initialized to $\mathbf{0}$. With this, we introduce four functions: *find_cutting_planes*$(i, C_i, \mathbf{s})$, *linrelax*$(i, C_i, \mathbf{s})$, *variable_reduction*$(\mathbf{s})$ and *solve*$(\mathbf{s})$.

Function *find_cutting_planes*$(i, C_i, \mathbf{s})$ generates a set of cutting planes as follows. First, we replace activity $n+1$ in Objective function (1a) by activity i and add the constraint $\sum_{t \in T} X_{it} = 1$, which we refer to as *setting the objective function to i*. This gives an ILP where activity i is always executed, while minimizing the starting time of activity i, thus, providing a lower bound on the starting time s_i of activity i. The solution obtained by solving the relaxation of this ILP, with earliest possible starting times $\mathbf{s}$ and cutting planes C_i, is then used to generate additional cutting planes as given by Constraints (5) and (7), which together with the already given cutting planes form the new set C_i. Setting the earliest possible starting times is done by setting $X_{jt} = 0$ for $t < s_j$ for each activity $j \in N$. The function call is aborted as soon as more than 10 consecutive cutting planes did not improve the linear relaxation value, as it was discovered experimentally that increasing this number did not increase performance significantly, while increasing the computational time.

Function *linrelax*$(i, C_i, \mathbf{s})$ solves the linear relaxation of Constraint set (1) combined with Constraints (3) while setting the objective function to i, adding cutting planes C_i and setting the lower bound on activity starting times to $\mathbf{s}$ for all activities in N. The function returns the objective function value, rounded up to the nearest integer, which is a lower bound on the starting time of activity i.

Function *variable_reduction*$(\mathbf{s})$ calls Algorithm 1, with the modification of using $\mathbf{s}$ as initial lower bound instead of setting it to zero in line 2. It then returns lower bounds for all activities.

Finally, the function *solve*$(\mathbf{s})$ first calculates the latest finishing time f_i for each node $i \in N$ as described in Section 6.1 and then solves the ILP while adding valid inequalities from Constraints (3) and setting $X_{it} = 0$ for all $i \in N$ with $t < s_i$ or $t + d_i > f_i$.

Algorithm 2 Solution algorithm

$\mathcal{C} \leftarrow \{\emptyset : i \in N\}$
$\mathbf{s} \leftarrow$ variable_reduction($\mathbf{0}$)
$\mathcal{F} \leftarrow$ forced activities
$N^{(s)} \leftarrow$ topological sorting of N on precedence graph
improved $\leftarrow$ True
while improved = True **do**
  improved $\leftarrow$ False
  for all $i \in N^{(s)}$ **do**
    $C_i \leftarrow C_i \cup \{C_j : j \in \mathcal{F}_i\}$
    $C_i \leftarrow C_i \cup$ find_cutting_planes$(i, C_i, \mathbf{s})$
    $v \leftarrow$ linrelax$(i, C_i, \mathbf{s})$
    if $v > s_i$ **then**
      improved $\leftarrow$ True
      $s_i \leftarrow v$
      $\mathbf{s} \leftarrow$ variable_reduction($\mathbf{s}$)
    end if
  end for
end while
solve($\mathbf{s}$)

With these functions, the solution algorithm is given in Algorithm 2. Initially, the lower bounds $\mathbf{s}$ are set by calling the *variable_reduction*($\mathbf{0}$) function. Subsequently, the sets of forced activities $\mathcal{F}_i$ for every activity $i \in N$ are calculated. The algorithm will now loop over all activities in topological order $N^{(s)}$. For each activity $i \in N^{(s)}$, cutting planes are calculated by setting the objective to i, and by adding cutting planes from all forced activities. The latter is done because a cutting plane for an activity j is valid when this activity is executed. Next, for activity i, a lower bound on the starting time is calculated by using *linrelax*$(i, C_i, \mathbf{s})$. If this improves the current lower bound for i, the *variable_reduction*($\mathbf{s}$) is called to possibly propagate this improvement to other activities. If there is any improvement for at least one of the activities, the loop over all activities is repeated. If not, the while loop terminates. Subsequently, to get an optimal solution, the *solve*($\mathbf{s}$) function is called to solve the ILP.

7. Computational results

In this section, we present the computational results. We first give a brief description of how the instances are generated and how the instance sets are created. Subsequently, the results are presented. These results are used to compare our methods with a method from literature, to evaluate the sensitivity against the time horizon $\mathcal{T}$, and to evaluate the performance of different parts of the algorithm.

7.1. Instances

In this section, we give a brief overview of the instance generation algorithm and give the instance sets we evaluate. The full sets of instances can be found in Van der Beek (2022). The instances are generated by creating a simple network with placeholder activities and replacing these placeholder activities by *subnetworks*. For each instance, subnetworks are generated by the generation procedure of Vanhoucke et al. (2008). This procedure generates instances of the RCPSP based on the number of activities, *Serial/Parallel indicator* (SP), *Resource Factor* (RF) and *Resource Constrainedness* (RC). Each instance contains two types of subnetworks, called *Phase-1 subnetworks* and *Phase-2 subnetworks*. The parameters $N1$ and $N2$ define the number of activities of these respective networks.

The network of placeholder activities is created from a shape array, for example $[3, 3]$, indicating two sequential selection-groups with each three successors. Each successor is replaced by a phase-1 subnetwork, and in the resulting network, the *Replace Number* (RN) determines the number of activities replaced by multiple parallel phase-2 subnetworks. This results in instances where all non-dummy activities are optional.

Table 2
Properties of instance sets.

	LC	MC
RC	[0.5, 0.7, 0.9]	[0.5, 0.7, 0.9]
N1	[3, 4, 5]	[3, 4, 5]
N2	[3, 4, 5]	[3, 4, 5]
Shape	[[2, 2], [2, 3], [3, 2], [2, 2, 3], [2, 3, 2], [3, 2, 2],[3, 3]]	[[2, 2], [2, 3], [3, 3], [2, 3, 2]]
RN	[2, 3]	[2, 3]
AL	[0, 2, 4, 6]	[0, 3, 6]
IS	No	Yes
# instances	1008	576

Finally, additional precedence links are placed according to the *Additional Links* (AL) parameter. If *Independent Succession* (IS) is allowed, then the same number of selection links as precedence links is placed as well.

An important part in any time-based scheduling formulation of the RCPSP is the size of time horizon $\mathcal{T}$. Decreasing this can significantly decrease the number of variables and often speed up the computation. To determine our value of $\mathcal{T}$, we create a feasible solution using a simple heuristic: The selection problem is solved by iteratively adding nodes, starting at the root node, until a feasible selection is obtained. In the case of multiple candidate nodes, the selection is made randomly. Then, a schedule is made in a similar way: iteratively adding random selected nodes in a precedence feasible way, starting at the root node. The now sorted selected nodes form an activity list, and a serial generation scheme is used to create a schedule (Kolisch & Hartmann, 1999). Note that this simple heuristic works for the structure of the instances considered in this paper, but not necessarily for all instances of the RCPSP-PS. In case of other structures, more sophisticated heuristics should be used. If no heuristic is available, the sum of all durations form an (very bad) upper bound that can be used.

With these parameters, we create two sets of instances: *Literature Comparison* (LC) and *Method Comparison* (MC). For all sets, the parameters RF and SP are constant (RF = 0.75 and SP = 0.5), as we capture the variance in resources and shape by the RC parameter and shape array, respectively. Furthermore, each instance has 4 resources. For all other parameters, the values are shown in Table 2. Here, each list shows the considered values per parameter and an instance is generated for each combination, with certain combinations to be removed after computational results indicated that these instances are unsolvable within a solving time of 6 h.

The largest instance set, LC, has no IS, and therefore, can be solved by the model in Kellenbrink and Helber (2015). This set is thus used to compare the proposed method in this paper with a model from literature. The instance set MC includes IS, and therefore, can only be solved with the proposed methods in this paper. This instance set is used to gain insight in the improved performance by various parts of the proposed solution method.

7.2. Results

All tests are performed on Intel Xeon Gold 5128 2.3 GHz server core. To solve the ILPs, Gurobi 8.1.1 (Gurobi, 2021) was used with a time limit of 6 h and the following settings to focus on finding the optimal solution by branch and bound: MIPFocus = 3, Heuristics = 0 and RINS = 0. We consider four different solution methods:

1. *Gurobi Basic Method (GBM)*: Solve the ILP given by model Constraint set (1) for final activity $n + 1$.
2. *Variable reduction (VR)*: GBM combined with Constraints (3) and the variable reduction method given in Algorithm 1; i.e., calling function *solve*($\mathbf{s}$) with $\mathbf{s}$ from *variable_reduction*($\mathbf{0}$).
3. *Constraint propagation (CP)*: Algorithm 2.

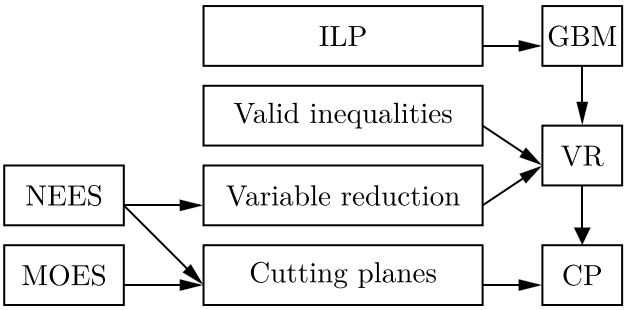


Fig. 4. Methods GBM, VR and CP and their building blocks.

Table 3
Summary of results for instance set *Literature comparison*. The numbers of best lower bounds and best upper bounds have 260 and 867 ties, respectively.

	GBM	CP	GKH
# Optimal solutions	356	364	256
Average computational time (10^3 s)	15.11	14.95	16.85
Average optimality gap	0.130	0.126	0.215
# Best lower bound	452	420	4
Average normalized lower bound	0.872	0.876	0.793
# Best upper bound	104	99	5
Average normalized upper bound	1.0029	1.0032	1.0116
Average number of variables	35 048	15 275	35 048
Average number of constraints	2018	2340	1093

4. *Gurobi Kellenbrink and Helber (2015) (GKH)*: Solve ILP model proposed in Kellenbrink and Helber (2015). Note that their variable reduction method only works on non-optional activities, and thus does not reduce variables for our instances.

The first three methods can be seen as extensions of each other, as is illustrated in Fig. 4. The VR method is the GBM, with additional constraints and certain variables set to zero. Furthermore, the CP method solves the same ILP as the VR method, although additional variables are set to zero.

In the remainder of this section, the methods are evaluated against each other. For clarity, all numerical values of the tables in this section are given in Appendix C.

7.2.1. Literature comparison

First, the instance set LC is evaluated. To compare both basic ILP models, each instance is solved by GBM and GKH. Furthermore, each instance is solved by the CP method to evaluate the improvement. Due to the large size of this instance set and the computational effort required, we evaluate this set only on the full (CP) method and leave the analysis for the intermediate step (VR) for the MC instance set.

A summary of the results is given in Table 3. Here, it can be seen that the number of optimal solutions obtained by the GBM and CP is significantly higher than by method GKH. Furthermore, the CP method is, on average, about 32 min faster than the GKH method. However, this is skewed due to the non-solvable instances. When comparing only the *interesting instances*, a decrease in average computational time of 50% is achieved. We define an interesting instance as an instance that has at least one method solving it to optimality, and at least one method not solving it to optimality within 5 min. This can be seen in Fig. 5(a), where the computational time of all 334 interesting instances are shown. Still, the solving time is capped at 6 h.

Furthermore, we define the optimality gap as $(ub - lb)/ub$ where lb and ub are the lower and upper bound found by the ILP solver, respectively. In Table 3 and Fig. 5(b), this shows a similar trend as for computational time. Furthermore, to gain more insight in this optimality gap, we evaluate the best lower and upper bounds found during the ILP solver process separately. Similar to the optimality gap, we normalize these to the best upper bound found for each instance. The averages are then computed over all instances. Furthermore, we evaluate how often a certain method reached the best bound, compared to other methods, excluding all *ties*. A tie is defined as an instance where all three methods reach the same bound. It can be seen that on both bounds and optimality gap, the GBM and CP method outperform the GKH method. Furthermore, the GBM has a larger number of best lower bounds, but the CP method has a higher average lower bound. From this, it can be deduced that if CP has the best bound, on average it is with a larger difference than if the GBM has the best bound. Furthermore, it can be seen that the GBM performs better than the CP method on all aspects of the upper bound. Therefore, the better average optimality gap of the CP method compared to the GBM can be attributed to the better performance of the lower bounds.

When comparing the size of the problems, it can be seen in Table 3 that the GBM model is considerably larger than the GKH model. Both have the same number of variables, but the GBM model has about twice as many constraints. However, the constraint propagation reduces the size significantly. Although some more constraints (and thus rows) are added due to Constraints (3), the number of variables is more than halved.

Upon evaluating computational times, we found two major trends: with the number of activities and with the RC value. The average computational times for different values of these parameters are plotted in Figs. 6(a) and 6(b). It can be seen that the solving difficulty increases with the number of activities. We see that the difference between the methods decreases with the number of activities, presumably due to the influence of unsolved instances (which are capped at 6 h). Furthermore, while evaluating the dependency on the RC parameter, it can be seen that values around 0.7 are significantly more difficult. A possible explanation could be that since resource constraints are difficult to solve, a lower constrainedness results in an easier instance, while a high value would significantly decrease the feasible region, and thus, the number of schedules that have to be evaluated.

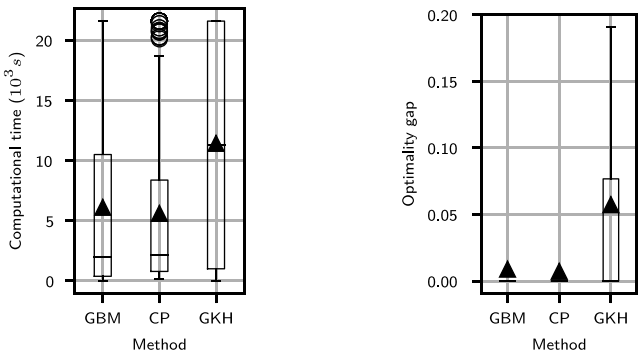
7.2.2. Method comparison

Finally, we evaluate the instance set MC to gain insight in the relative improvements of different parts of the CP method. A summary of the results is given in Table 4 and the total computational time per method for interesting instances (instances that are not too easy or too hard) are shown in Fig. 7(a).

It can be seen that, although the CP method has the lowest average computational time, the largest part of this decrease in computational time can be attributed to the VR method. However, when considering the number of optimal solutions obtained, the improvement due to the CP method is considerably larger than the improvement due to the VR method, indicating that the CP method is especially useful on more difficult instances. A possible explanation for this is that the CP method requires additional computation, which might only be beneficial for more complicated instances.

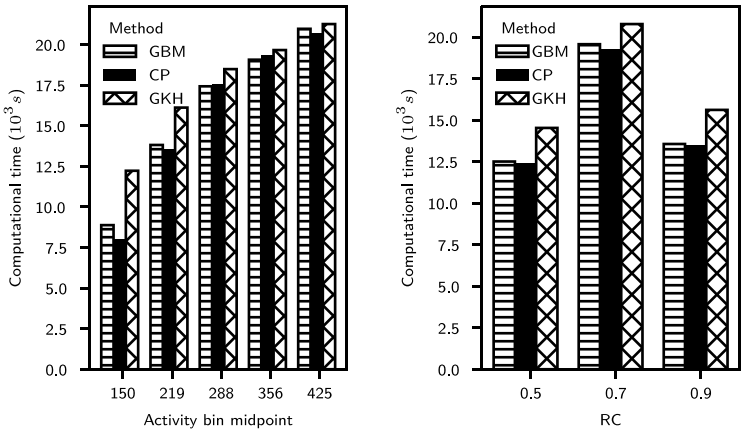
The optimality gaps show counter-intuitive behavior, with the GBM having the smallest gap. When evaluating the average normalized lower and upper bounds, it can be seen that this optimality gap is mainly due to the better upper bounds found by the GBM. Looking at the number of best lower and upper bounds, it seems that the GBM performs better than both the VR and CP method. However, this is not necessarily the case for the lower bounds, since this comparison is slightly skewed due to CP and VR performing well on the same instances. If only GBM and CP would be compared on lower bounds, GBM would have 122 best lower bounds and CP 125 (excluding ties between GBM and CP). However, some of these best lower bounds of CP are further improved by VR. Therefore, CP performs relatively worse when comparing all three methods. For the upper bounds, however, GBM performs better than both the VR and the CP method.

Furthermore, Table 4 lists the average sizes of the ILPs. It can be seen that the largest part of the reduced number of variables is due



(a) Computational time. (b) Optimality gap (without outliers).

Fig. 5. Computational time and optimality gap for instance set LC.



(a) Per activity bin (b) Per RC value

Fig. 6. Average computational times for instance set LC.

Table 4
Summary of results for instance set MC. The number of best lower bounds and best upper bounds have 298 and 512 ties, respectively.

	GBM	VR	CP
# Optimal solutions	323	325	336
Average computational time (10^3 s)	11.41	11.04	10.89
Average optimality gap	0.057	0.059	0.058
# Best lower bound	127	88	113
Average normalized lower bound	0.9449	0.9452	0.9453
# Best upper bound	38	24	34
Average normalized upper bound	1.0027	1.0049	1.0036
Average number of constraints	1535	1767	1767
Average number of variables	21 253	11 871	11 000

Table 5
Average computational time for instance set MC.

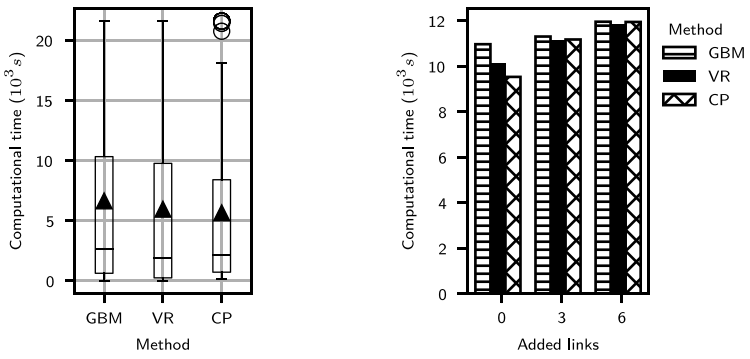
Added links	Computational time (10^3 s)		
	GBM	VR	CP
0	10.97	10.14	9.53
3	11.31	11.14	11.18
6	11.96	11.84	11.95

to the VR method, averaging to a reduction of 44%. The CP method in turn has a further reduction of 4%. Both the VR and CP method use the same valid inequalities, resulting in the same number of constraints: 15% more than the GBM.

Finally, we evaluate the performance of the methods compared to the AL parameter, which gives an indication of the similarity between the selection graph and the precedence graph. The average computational time per AL value is shown in Fig. 7(b) and Table 5.

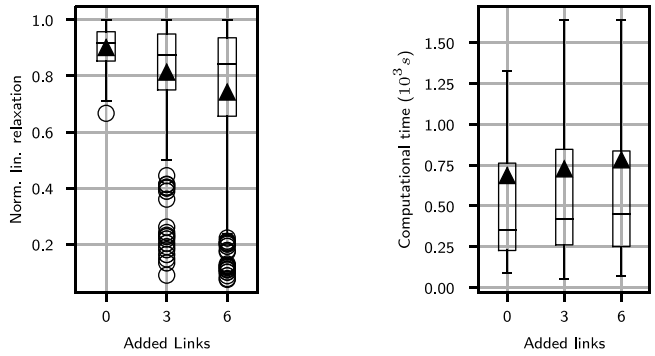
It can be seen that there is an increasing trend of computational time against the AL value. Furthermore, it is interesting to see that the VR method performs slightly better in terms of computational time on all AL values except zero. To analyze the performance against the AL value in more detail, we evaluate the linear relaxation of both the CP and the VR method. We define the *relative linear relaxation* as $\frac{lr_2}{lr_3}$, where lr_2 and lr_3 are the linear relaxation values of the VR and CP methods, respectively. Thus, a value of 1 means no improvement in linear relaxation due to the CP method. Note that CP solves the same ILP model as VR, although with more variables set to zero. Therefore, the relative linear relaxation is never larger than one. The lower the value of the relative linear relaxation, the larger the improvement due to the CP method.

In Fig. 8(a), the relative linear relaxation values are shown. It can be seen that the relative performance of the CP method increases with the AL value. However, by plotting the computational times of the preprocessing part of the CP method in Fig. 8(b), it can be seen that the computational time also increases with AL values. As seen in Table 6, there is a decrease in ILP solver time due to the CP method for each AL value. However, for AL values larger than 0, this decrease is smaller than the increased computational time due to preprocessing. Therefore, the CP does not outperform the VR on these instances, because the additional computational time needed is larger than the



(a) Interesting instances. (b) Average computational time.

Fig. 7. Computational times for instance set MC.



(a) Normalized linear relaxation of VR compared to CP. (b) Computational time of preprocessing step for CP (without outliers).

Fig. 8. Analysis on AL value.

Table 6
Partial computational times (10^3 s) for instance set MC for different values of AL.

AL:	0	3	6
VR ILP Solver	10.14	11.14	11.84
CP preprocess	0.73	1.06	1.22
CP ILP Solver	8.81	10.12	10.73

decrease in computational time obtained by having a better model. A possible explanation for the increase in preprocessing time, is that added links increase the influence of NEESs across the project network, thus possibly introducing new iterations in the preprocessing algorithm. However, when evaluating the number of solved instances for an AL value of 3 (108 for VR and 111 for CP) and 6 (100 for VR and 102 for CP), the use of the CP method can still be valuable for these instances.

8. Conclusions

In this paper, we developed a general model for the RCPSP-PS by introducing the concept of *selection groups*. Based on this model, two types of subsets of activities were identified: 'non-empty execution sets' and 'max-one execution sets', which provide information on the number of executed activities within these sets. With these sets, cutting planes and a constraint propagation technique were introduced, along with an algorithm that combines these methods.

Computational tests show that the basic ILP model performs significantly better than the most similar model from literature, although it must be noted that we only test on instances without nonrenewable resources. Furthermore, an improvement on both computational time and number of optimal solutions found is achieved by both the *Variable reduction* and the *Constraint propagation* method. Additionally, although for the final instance set the average optimality gap did not improve, the best lower bound found did. To further improve the results, different formulations can be explored, as this has shown to be beneficial for the standard RCPSP.

Besides the direct computational improvements, the methods presented in this paper decrease the solution space. Therefore, they can be directly implemented in other exact approaches, such as constraint programming. Furthermore, the mathematical proofs in this paper can be used as building blocks for other theoretical or computational improvements, for example, in decomposition based approaches such as Sprecher (2002). Finally, the theoretical properties can be used in heuristic algorithms.

CRediT authorship contribution statement

T. van der Beek: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation. **J.T. van Essen:** Writing – review & editing, Validation, Supervision. **J. Pruyn:** Writing – review & editing, Supervision, Project administration, Funding acquisition. **K. Aardal:** Supervision.

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. Proofs

A.1. Proof of Lemma 1

Proof. We proof this by induction, by creating and updating a set B iteratively. At each iteration, B satisfies two properties:

1. B is a NEES.
2. B contains only activities $i \in N'$ with $\sum_{g \in G_i} U_g \geq 1$ or $V_i = 1$.

Start with $B = \{0\}$. Since the source activity 0 always has to be executed, it is a NEES. Furthermore, due to Constraint (2a), it also satisfies Property 2.

For the induction step, assume that set B satisfies both properties. For each activity $i \in B$ with $V_i = 0$ (and thus $\sum_{g \in G_i} U_g \geq 1$ by Property 2), let set C_i contain all successor activities of i that satisfy Property 2; $C_i = \{j : \exists g \in G_i | j \in S_g \wedge V_j + \sum_{h \in G_j} U_h \geq 1\}$. Since $\sum_{g \in G_i} U_g \geq 1$ by assumption, there is at least one group $g \in G_i$ with $U_g = 1$. Then, by Constraints (2b) and (2c), it follows that all successors of this group satisfy Property 2 and are therefore included in set C_i. Thus, if i is executed, at least one activity in C_i is executed. We now create $\bar{B}$ by replacing i by C_i; $\bar{B} = B \cup C_i \setminus \{i\}$. Since B is a NEES, so is $\bar{B}$. Thus, $\bar{B}$ satisfies both properties. Finally, take $\bar{B}$ as the new B.

For each iteration, each activity is replaced by its successors unless $V_i = 1$. Since the graph is acyclic, the iterative process will terminate, in which case $V_i = 1$ for every $i \in B$, since at some point G_i will be empty. As shown, both properties are maintained in this process. Therefore, the final set B is a NEES. Because A contains all activities $i \in N'$ with $V_i = 1$, this means that $B \subseteq A$. Since B is a NEES, which means that at least one activity in B has to be executed, then any superset of B is also a NEES. Therefore, A is a NEES as well and since A results from a solution to Constraint set (2), this solution constitutes a NEES. □

A.2. Proof of Proposition 1

Proof. Consider two activities i and j that are both executed in the solution. This is illustrated in Fig. 9. There has to be a path of executed activities from the start activity 0 to both i and j. Call these paths S_1 and S_2, respectively. If these paths split at an activity r to successor activities u and v ($u \neq v$), it follows that u and v cannot be in the same selection group due to Constraints (1f). Since paths can merge after splitting, take activity r as the activity immediately before the last split, and the remaining paths as P and Q, which give a CRP (r, P, Q). Let $\ell(P)$ be the number of activities on path P. If there is no split, assume w.l.o.g, $\ell(P) < \ell(Q)$. Then, j lies in the extension of i, and $(i, \{i\}, \{i\} \cup \{\mathcal{V}(Q) \setminus \mathcal{V}(P)\})$ gives a CRP between i and j.

Thus, if activities i and j are both executed, there is always a CRP between them. This means that at most one of them can be executed if there is no CRP. □

A.3. Proof of Proposition 2

Proof. Constraints (6b) impose that groups can only be selected if there is no CRP between the activators. Therefore, in combination with Constraints (6c), there is no CRP between the set of activities for which $W_i^* = 1$. Therefore, for an integer solution, the left hand side of Constraint (7) contains *at most one* non-zero summation term $\sum_{t \in T}(t + d_i)X_{it}$, for which $\sum_{t \in T} X_{it} = 1$. Consider the case that one activity i' for which $W_{i'}^* = 1$ is executed. Then, there exists at least one selection group $g \in H$ with activator i' for which $Y_g^* = 1$ and thus there exists at least one executed activity j' that is a successor of group g and for which $Z_{j'}^* = 1$. Due to the full precedence, we obtain Eq. (8). If no activity i' for which $W_{i'}^* = 1$ is executed, the left-hand side of Constraint (7) is zero, and therefore, it is a cutting plane.

$$\sum_{i \in N} W_i^* \sum_{t \in T}(t + d_i)X_{it} = \sum_{t \in T}(t + d_{i'})X_{i't} \leq \sum_{t \in T} tX_{j't} \leq \sum_{i \in N} Z_i^* \sum_{t \in T} tX_{it}. \quad (8)$$

Therefore, as long as Objective function (6a) has a value larger than 0, Constraint (7) cuts of the current solution X^*. □

Appendix B. Rooted path graphs

In this section, we show how to create a Rooted Path Graph (RPG). Let Ω_i be the set of all successors of i in the *selection graph* (recall that S_i is the set of all successors in the *precedence graph*). Furthermore, let Θ be all pairs of activities (i, j) with a path between $i \in N$ and $j \in N$ in the selection graph and Θ_i all activities reachable from $i \in N$ in the selection graph. Finally, let Γ be the set of activity pairs (i, j) for which $i \in N$ and $j \in N$ are successors in the same selection group, i.e., for all $(i, j) \in \Gamma$ there exists a selection group $g \in G$ such that $i \in S_g$ and $j \in S_g$.

Furthermore, we distinguish between two types of CRP's: **splitted** CRP's and **extended** CRP's. A splitted CRP (r, P, Q) splits up at activity r and has $\ell(P) > 1$ and $\ell(Q) > 1$ (note that paths P and Q both include activity r), where $\ell(P)$ is the number of activities on path P. In an extended CRP there is no split and one path is an extension of the other, i.e., the root is i, $P = \{i\}$ and Q is a path from i to j. In Fig. 9, there is a splitted CRP between i and j. However, there also is an extended CRP between, for example, r and i: $(r, (r), (r, u, \ldots, i))$.

Given these definitions, we now present Algorithm 3, which creates an RPG. Three sets of edges are introduced: Final edges $E^{(f)}$, active edges $E^{(a)}$ and new edges $E^{(n)}$. There are four steps which add edges to these sets:

Step 1 For any activity $r \in N$, let $F_r^{(1)}$ be the set of edges between any two successors (i, j), with $i \neq j$, if i and j are not successors in the same selection group; $F_r^{(1)} = \{(i, j) : i \in \Omega_r, j \in \Omega_r, (i, j) \neq \Gamma, i \neq j\}$. Add these edges to the set of active edges, i.e., $E^{(a)} \leftarrow E^{(a)} \cup F_r^{(1)}$.

Step 2 For any active edge $(i, j) \in E^{(a)}$, create a set of edges $F_{ij}^{(2)}$. For each successor activity u of activity i, add edges (j, u) and (u, j) to this set if u is not reachable from j. We call this **extending** (i, j) on i to u. This gives $F_{ij}^{(2)} = \{(j, u) : u \in \Omega_i, u \notin \Theta_j\} \cup \{(u, j) : u \in \Omega_i, u \notin \Theta_j\}$. Add these edges to the set of new edges: $E^{(n)} \leftarrow E^{(n)} \cup F_{ij}^{(2)}$.

Step 3 For any active edge $(i, j) \in E^{(a)}$, create the set $F_{ij}^{(3)}$. Add edge (u, v) to this set if u is a successor of i and v is a successor of j, $u \neq v$, both u and v are not equal to i or j, u is reachable from j and v is reachable from i. This gives $F_{ij}^{(3)} = \{(u, v) : u \in \Omega_i, v \in \Omega_j, u \neq v, u \notin \{i, j\}, v \notin \{i, j\}, u \in \Theta_j, v \in \Theta_i\}$. Add this set to the set of new edges: $E^{(n)} \leftarrow E^{(n)} \cup F_{ij}^{(3)}$. After this loop, add the set of active edges to the set of final edges ($E^{(f)} \leftarrow E^{(f)} \cup E^{(a)}$) and replace the set of active edges by the new set of edges ($E^{(a)} \leftarrow E^{(n)}$). If the set of new edges is not empty, empty this set ($E^{(n)} = \emptyset$) and go back to Step 2. Otherwise, proceed to Step 4.

Step 4 For every activity $r \in N$, add final edges (r, i) for every i reachable from r; $E^{(f)} \leftarrow E^{(f)} \cup \{(r, i) : r \in N, i \in \Theta_r\}$

These steps are illustrated in Fig. 10.

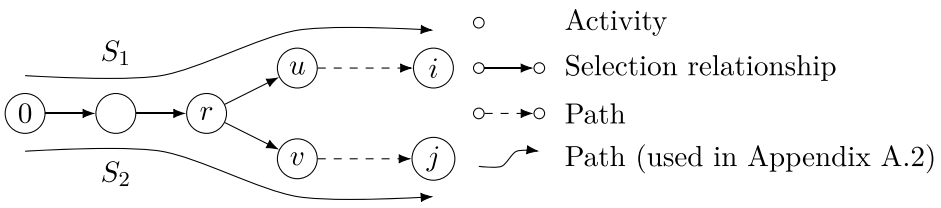


Fig. 9. Example showing a splitted CRP between i and j and an extended CRP between r and i.

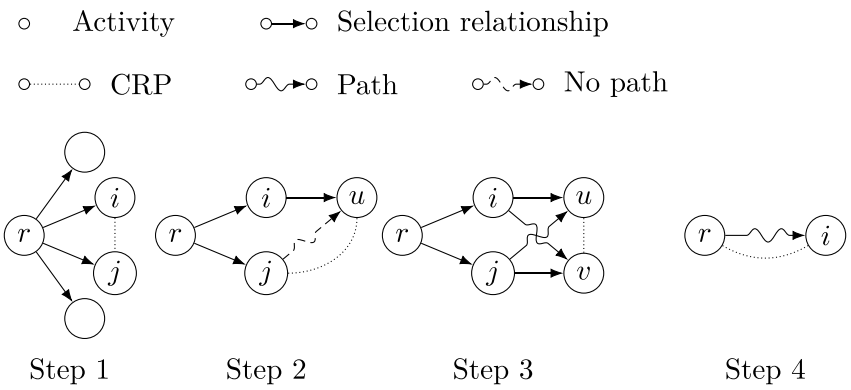


Fig. 10. Steps in RPG algorithm.

Algorithm 3 Rooted Path Graph

$E^{(f)} \leftarrow \emptyset$
$E^{(a)} \leftarrow \emptyset$
$E^{(n)} \leftarrow \emptyset$
for all $r \in N$ **do**
  $E^{(a)} \leftarrow E^{(a)} \cup F_r^{(1)}$ ▷ Step 1
end for

do
  for all $(i,j) \in E^{(a)}$ **do**
    $E^{(n)} \leftarrow E^{(n)} \cup F_{ij}^{(2)}$ ▷ Step 2
    $E^{(n)} \leftarrow E^{(n)} \cup F_{ij}^{(3)}$ ▷ Step 3
  end for
  $E^{(f)} \leftarrow E^{(f)} \cup E^{(a)}$
  $E^{(a)} \leftarrow E^{(n)}$
  $E^{(n)} \leftarrow \emptyset$
while $|E^{(a)}| > 0$

for all $i \in N$ **do**
  $E^{(f)} \leftarrow E^{(f)} \cup \{(i,j) : j \in \Theta_i\}$ ▷ Step 4
end for

Theorem 1 states that Algorithm 3 creates an RPG.

Theorem 1. *Algorithm 3 creates a rooted path graph if the selection graph is acyclic.*

Proof. If there is an edge created by Algorithm 3, there is a CRP:

For Step 1, 2, and 3, we will prove this by induction. The base case is given by Step 1. If an edge (i,j) is created in this step, the CRP is given by the splitted CRP $(r, \{r,i\}, \{r,j\})$. Since Step 2 and 3 take input edges from Step 1, 2, and 3, we assume for the induction step that each input edge (i,j) for Step 2 and 3 has a splitted CRP (r,P,Q).

For Step 2, w.l.o.g, let u be the successor activity of the final activity i in path P. If $u \notin \mathcal{V}(Q)$, then adding u to P gives a splitted CRP for activities u and j, and we are done. If $u \in \mathcal{V}(Q)$, there are three cases in which edge (i,j) could have been created:

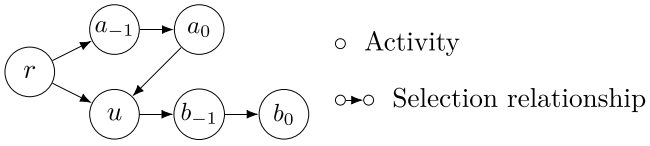


Fig. 11. (i,j) created by Step 2, with $u \in \mathcal{V}(Q)$.

1. Edge (i,j) created by Step 1: Since $Q = (r,j)$ and u is on path Q, it follows that $u = r$ or $u = j$. The former would result in a cycle $r \to i \to u = r$, which is not possible since we have an acyclic graph. The latter contradicts $u \notin \Theta_j$, so (i,j) cannot be created by Step 1.
2. Edge (i,j) created by Step 2. Let $(i,j) = (a_0, b_0)$, where the index 0 stands for the number of iterations, counting backwards. There are now two cases, (a_0, b_0) was created by extending edge (a_{-1}, b_0) to a_0 on a_{-1}, or by extending edge (a_0, b_{-1}) to b_0 on b_{-1}. Consider the last case. By assumption, $u \in \mathcal{V}(Q)$ and u is a successor of a_0. Therefore, as shown in Fig. 11, there is a path $a_0 \to u \to b_0$. This means that b_0 is reachable from a_0, which is a contradiction since Step 2 only extends if b_0 is not reachable from a_0.
   This means that if (a_0, b_0) is created by Step 2, it has to be extended to a_0 from input edge (a_{-1}, b_0). The same logic holds for this input edge (a_{-1}, b_0); it cannot be created by extending (a_{-1}, b_{-1}) to b_0 on b_{-1}. Thus, (a_0, b_0) is created by iteratively extending edge (a_{-n}, b_0) to a_{-n+1} on a_{-n}. Taking n as large as possible, we get edge (a_{-n}, b_0), which is not created by Step 2. Therefore, (a_{-n}, b_0) has to be created by either Step 1 or Step 3. Consider the case that (a_{-n}, b_0) is created by Step 1. This means that both activities a_{-n} and b_0 are successor activities of the root activity $r \in N$. Since u cannot be reachable from b_0 and an activity is always reachable by itself, $b_0 \in \Theta_{b_0}$ and thus $u \neq b_0$. Therefore, since $u \in \mathcal{V}(Q)$ and $Q = (r, b_0 = j)$, we have that $u = r$, and thus, there is a path $a_0 \to u = r \to a_{-n} \to \cdots \to a_0$, which is a cycle.
   If (a_{-n}, b_0) is created by Step 3, call the input edge (a_{-n-1}, b_{-1}). Since $u \in \mathcal{V}(Q)$ and $u \neq b_0$, it follows that b_{-1} is reachable from

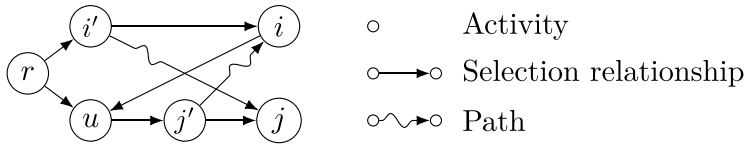


Fig. 12. (i, j) created by Step 3, with $u \in \mathcal{V}(Q)$.

Table 7
Numerical values of Fig. 5(a).

Method	1	3	4
Mean	6.12	5.62	11.44
Median	1.98	2.16	11.34
Minimum (excl. outliers)	0.03	0.15	0.01
Maximum (excl. outliers)	21.60	18.66	21.60
Q1	0.39	0.79	1.01
Q3	10.51	8.38	21.60

Table 8
Numerical values of Fig. 5(b).

Method	1	3	4
Mean	0.009	0.007	0.057
Median	0.000	0.000	0.000
Minimum (excl. outliers)	0.000	0.000	0.000
Maximum (excl. outliers)	0.000	0.000	0.191
Q1	0.000	0.000	0.000
Q3	0.000	0.000	0.077

Table 9
Numerical values of Fig. 6(a).

Activity bin midpoint	Computational time (10^3 s)		
	1	3	4
150	8.87	8.02	12.22
219	13.81	13.57	16.11
288	17.42	17.55	18.48
356	19.06	19.34	19.66
425	20.96	20.70	21.26

Table 10
Numerical values of Fig. 6(b).

RC	Computational time (10^3 s)		
	1	3	4
0.5	12.50	12.40	14.53
0.7	19.57	19.24	20.78
0.9	13.56	13.51	15.61

Table 11
Numerical values of Fig. 7(a).

Method	1	2	3
Mean	6.65	5.96	5.66
Median	2.64	1.90	2.15
Minimum (excl. outliers)	0.01	0.01	0.13
Maximum (excl. outliers)	21.60	21.60	18.09
Q1	0.63	0.25	0.72
Q3	10.33	9.77	8.41

Table 12
Numerical values of Fig. 7(b).

Added links	Computational time (10^3 s)		
	1	2	3
0	10.97	10.14	9.53
3	11.31	11.14	11.18
6	11.96	11.84	11.95

Table 13
Numerical values of Fig. 8(a).

Added links	0	3	6
Mean	0.90	0.81	0.74
Median	0.92	0.87	0.84
Minimum (excl. outliers)	0.71	0.50	0.24
Maximum (excl. outliers)	1.00	1.00	1.00
Q1	0.85	0.75	0.66
Q3	0.96	0.95	0.94

Table 14
Numerical values of Fig. 8(b).

Added links	0	3	6
Mean	0.69	0.73	0.78
Median	0.35	0.42	0.45
Minimum (excl. outliers)	0.09	0.05	0.07
Maximum (excl. outliers)	1.33	1.64	1.64
Q1	0.23	0.26	0.25
Q3	0.76	0.85	0.84

u, i.e., $b_{-1} \in \Theta_u$. By construction in Step 3, a_{-n} is reachable from b_{-1}. Therefore, there is a path $u \to b_{-1} \to a_{-n} \to a_0 \to u$, which is also a cycle. This means that there is a cycle in both cases. This contradicts the fact that we have an acyclic graph. Therefore, edge (i, j) cannot be created by Step 2.

3. Edge (i, j) created by Step 3. Let (i', j') be the input edge which created (i, j), with i' and j' in P and Q, respectively. This is illustrated in Fig. 12. Since $u \in \mathcal{V}(Q)$ and $u \neq j$, it follows that j' is reachable from u. This gives a path $i \to u \to j' \to i$. This creates a cycle, which is a contradiction.

Therefore, if $u \in \mathcal{V}(Q)$, edge (i, j) could not be created. Thus, $u \notin \mathcal{V}(Q)$ and Step 2 creates a splitted CRP.

Step 3, with input edge (i, j), creates an active edge (u, v) representing a splitted CRP if $v \notin \mathcal{V}(P)$ and $u \notin \mathcal{V}(Q)$. Thus, assume $u \in \mathcal{V}(Q)$. Then, there is a cycle from u to j to u, which is a contradiction. Therefore $u \notin \mathcal{V}(Q)$. The same arguments holds for $v \notin \mathcal{V}(P)$.

Thus, given a splitted CRP, Step 2 and 3 produce a splitted CRP. Since Step 1 produces only splitted CRP's, Step 2 and 3 do as well by induction. Finally, each edge in Step 4 is an extended CRP.

If there is a CRP, there is an edge created by Algorithm 3:

Consider activities i and j with a CRP $\{r, P, Q\}$, $P = (r, p_1, \dots, p_n)$ and $Q = (r, q_1, \dots, q_m)$. Let $p_n = i$ and $q_m = j$. If either i is reachable from j or vice versa ($(i, j) \in \Theta$), Step 4 creates an edge. Therefore, assume that $(i, j) \notin \Theta$.

Step 1 creates an edge between p_1 and q_1. If $p_1 = i$ and $q_1 = j$, we are done, so assume $p_1 \neq i$ and/or $q_1 \neq j$. Now, if $p_2 \in \Theta_{q_1}$ and $q_2 \in \Theta_{p_1}$, Step 3 creates edge (p_2, q_2). If $p_2 \notin \Theta_{q_1}$ and/or $q_2 \notin \Theta_{p_1}$, Step 2 creates an edge further along the CRP in at least one path (P or Q). Therefore, in each iteration, an edge is created along the CRP to an activity on either P, Q or both. Continue this until, w.l.o.g, there is an edge created to activity i on path P.

Let q_a be the last activity on Q with an edge (i, q_a). Step 2 iteratively creates a new edge (i, q_{a+1}) as long as $q_{a+1} \notin \Theta_i$. This is either repeated until $q_{a+1} = j$ and edge (i, j) is created, or until $q_{a+1} \in \Theta_i$. In the latter case, there is a path $i \to q_{a+1} \to j$, so Step 4 will create edge (i, j). □

Appendix C. Numerical results

In this section, numerical values for figures in the manuscript are given (see Tables 7–14).

References

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: vol. 2007*.

Van der Beek, T. (2022). Instances and file format for the resource constrained project scheduling problem with a flexible project structure. http://dx.doi.org/10.4121/21106768.v1.

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

Gurobi (2021). Gurobi optimizer reference manual.

Hauder, V. A., Beham, A., Raggl, S., Parragh, S. N., & Affenzeller, M. (2020). Resource-constrained multi-project scheduling with activity and time flexibility. *Computers & Industrial Engineering*, *150*, Article 106857.

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

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In *Project scheduling* (pp. 147–178).

Kuster, J., Jannach, D., & Friedrich, G. (2009). Extending the RCPSP for modeling and solving disruption management problems. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, *31*(3), 234–253.

Lombardi, M., & Milano, M. (2012). Optimal methods for resource allocation and scheduling: A cross-disciplinary survey. *Constraints*, *17*(1), 51–85.

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.

Rubeša, R., Fafandjel, N., & Koli, D. (2011). Procedure for estimating the effectiveness of ship modular outfitting. *Engineering Review*, *1*(1), 55–62.

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.

Sprecher, A. (2002). Network decomposition techniques for resource-constrained project scheduling. *Journal of the Operational Research Society*, *53*(4), 405–414.

Talbot, F. B. (1982). Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, *28*(10), 1197–1210.

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

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

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., & Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, *187*(2), 511–524.

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

Wu, I. C., Borrmann, A., Beißert, U., König, M., & Rank, E. (2010). Bridge construction schedule generation with pattern-based construction methods and constraint-based simulation. *Advanced Engineering Informatics*, *24*(4), 379–388.