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, Netherland*
[b] *Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands*

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). The objective of the RCPSP-PS is to find a minimal makespan schedule subject to precedence and resource constraints, while only having to execute a subset of all activities. We present a general model, which is based on a precedence graph and a task selection graph. Furthermore, we introduce an exact solution method including procedures for generating cutting planes and variable reduction. It is shown that both the lower bound obtained from the linear relaxation, and the computation time needed to obtain integer solutions are improved using these procedures.

Keywords: Project scheduling, Resource constrained project scheduling problem, Integer programming, Flexible project structure

1. Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is an optimization problem aimed at scheduling activities. It comprises a list of activities that have to be scheduled, while satisfying a list of precedence constraints and resource availability constraints. The problem aims at minimizing the makespan of the project and is used in various applications, such

*Corresponding author
Email 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)

as assembly scheduling or employee scheduling (Artigues et al., 2008). The problem was proven to be NP-hard by Blazewicz et al. (1983) and much research has been done in finding heuristic methods (Pellerin et al., 2020). Additionally, generalizations of the RCPSP have been studied in great detail (Hartmann and Briskorn, 2010).

While the RCPSP assumes that all activities have to be executed, this is not always required. In many applications, like housing construction (Servranckx and Vanhoucke, 2019), highway project construction (Wu et al., 2010), modular shipbuilding (Rubeša et al., 2011) and aircraft turnaround scheduling (Kellenbrink and Helber, 2015), there are multiple ways of completing a project. For example, in modular shipbuilding, all components can be installed directly in the ship, or first assembled in a module and subsequently installed as a whole. This results in an RCPSP that can be completed by executing only a subset of all activities. This is called the *Resource Constrained Project Scheduling Problem with a flexible Project Structure* (RCPSP-PS).

The RCPSP-PS consists of two sub-problems. First, the decision has to be made which subset of activities to execute. This is called the *activity selection problem*. Secondly, a schedule has to be made with these selected activities, which gives rise to the *activity scheduling problem*.

In large assembly projects, the choices of which activity to execute next is often associated with flow of components and/or material. If these components, e.g. a ship hull, cannot be split up, the choices can be exclusive; only one alternative can be selected from a subset of activities. This is what we call the *exclusivity criterion*, which complicates the activity selection problem as is shown in this paper.

Furthermore, to better represent project scheduling in reality, two properties are often required. The first is the separation of scheduling and selection logic. Furthermore, we introduce the concept of *independent succession*: a choice to execute a certain activity, can be forced by multiple other activities.

An example can be given from modular shipbuilding, where a ship is produced by combining multiple construction modules. For each module, we have two options: Construct it locally with available materials, or ship it from another yard. Shipping of modules can be combined, such that we only need to execute one shipping activity for all modules. Therefore, if at least one module is shipped, the activity 'shipping modules' has to be executed and finished before installing the modules. To model this, we require both the separation of scheduling and selection logic, and independent succession.

Although multiple papers introduce various models for the RCPSP-PS, there has not been a combination of the exclusivity criterion, independent succession and separation of scheduling and selection logic. Furthermore, there is little research on cutting planes and related exact methods for the RCPSP-PS.

To fill this gap, this paper presents a new model for the selection logic, based on modifying the model of Kellenbrink and Helber (2015) to add independent succession. This allows for a simplification of the model, such that no distinction has to be made anymore between optional activities and activities that are always executed. Furthermore, an exact solution algorithm is given. This algorithm uses a variable reduction method based on the *critical path method* (Zhou et al., 2013) and two types of cutting planes. The proposed solutions methods are then evaluated on restricted instances against the model of Kellenbrink and Helber (2015), and against each other on non-restricted instances.

In Section 2, we present an overview of the literature on exact methods for the RCPSP and on different models for the RCPSP-PS. Subsequently, the problem description and MILP formulation is given in Section 3. The solution algorithm is given in Section 4, of which the results are presented and discussed in Section 5. Finally, Section 6 concludes the paper.

2. Literature overview

The 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 been focused on developing heuristic and exact solution methods. An overview of this research can be found in, for example, Pellerin et al. (2020) and Lombardi and Milano (2012). These two review papers discuss heuristic and exact methods, respectively. In this section, we focus on exact solution methods for and generalizations of the RCPSP.

The standard RCPSP, without generalizations, is often solved by MILP solvers or constraint programming solvers (Herroelen and Leus, 2005). One way of reducing the solution space in both methods is variable reduction. For the RCPSP, this is often based on the Critical Path Method, which defines the earliest time an activity can start, based on precedence constraints, while ignoring resource constraints.

Another way of reducing the solution space is by finding better lower bounds. Stronger lower bounds for MILP problems can reduce the number of branch-and-bound nodes needed to be explored. For constraint programming, optimization can be done by iteratively proving (in)feasibility

for varying makespans. Stronger lower bounds can decrease the number of iterations in this process.

Various researchers developed cutting planes for the RCPSP. Two types of bounds can be derived for the RCPSP: *constructive* and *destructive* bounds. Constructive (or direct) methods directly calculate a bound by relaxing the problem, while destructive bounds are determined by setting a maximal value on the objective function value and proving infeasibility; if a schedule with maximal makespan T is impossible, then $T+1$ is a valid lower bound.

Brucker and Knust (2000) provide destructive bounds based on constraint propagation and linear programming. The constraint propagation method keeps track of the minimal durations between activities and iteratively updates this based on the precedence constraints. The linear programming method relaxes the precedence and non-preemption constraints. Baptiste and Demassey (2004) use the same relaxation and provide additional bounds based on so-called energetic reasoning and weak versions of preemption and precedence. Energetic reasoning for the RCPSP is defined by comparing the demand of resources within a certain time interval with the supply of resources.

Hardin et al. (2008) provide a class of valid inequalities for the RCPSP with a single resource. These are based on covers; subsets of activities that cannot be executed all simultaneously due to limited resource availability. Furthermore, they give sufficient conditions under which these inequalities are facet-defining and provide lifting procedures. Other lower bounds for the RCPSP are given by Haouari et al. (2012), who provide three classes of lower bounds based on energetic reasoning and an efficient way of generating these.

A generalization of the RCPSP that enables different ways of executing activities is the *Multi-Mode Resource Constrained Project Scheduling Problem* (MRCPSP) (Talbot, 1982). In this problem, each activity has multiple execution modes with varying durations and/or resource usage. An overview of variations and solution methods can be found in Wglarz et al. (2011).

For the MRCPSP, various exact methods are developed. Zhu et al. (2006) provide a branch-and-cut algorithm that uses both reduction of variables based on constraint-propagation, and cutting planes based on resource conflicts and precedence relations. Araujo et al. (2020) propose two new models for the MRCPSP and introduce preprocessing cuts based on feasible subsets: sets of job-mode combinations that can be executed simultaneously. Furthermore, they give a branch-and-cut algorithm that uses 5 types of cutting planes in parallel.

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*. 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. They call this model 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 MILP 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*, 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 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 modelled with this approach. They solve this model 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*. This is a model 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) uses a network with different types of ac-

Paper	Separate scheduling and selection	Independent succession	Exclusivity criterion	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	✓	✓	✓	✓	

Table 1: Overview of models with a flexible project structure

tivities (OR, AND and OUT) to model both the selection and scheduling problem, while adding the extension of supporting multiple projects. Besides the standard objective of makespan minimization, they also consider time balance and resource balance as objective functions. They provide 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 3 papers include an exact method, for 2 of these it comprises of only an MILP formulation.

3. RCPSP-PS

In this section, the problem is formalized. First, Section 3.1 gives a description of the problem, which includes a description of the selection graph with selection groups. Subsequently, Section 3.2 gives a MILP formulation for the RCPSP-PS. Finally, the activity selection problem is discussed in Section 3.3.

3.1. Problem description

The RCPSP-PS consists of a set of activities N of which a subset has to be executed while minimizing the makespan of executing the selected

activities. The time horizon during which these activities are scheduled is represented by a set of discrete time periods 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. 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. These relationships can be represented in a graph by creating a node for each activity and an edge for each precedence relationship. This graph is called the *precedence graph*. Furthermore, the set $\mathcal{P}_j$ contains all predecessors of activity j and the set $\mathcal{S}_i$ contains all successors of activity i in the precedence graph.

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 $g \in G$ consists of 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. With these, we also define the set of *selection groups with full precedence* $H \subseteq G$. This set contains all selection groups with a time 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 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. An 'and'-relationship can be modeled by multiple unit selection groups.

The selection groups can be represented by a *selection graph*. This is a graph in which each activity is represented by a node and the selection groups are represented by directed edges. A single edge denotes that the source and target activity belong to a unit selection group. Multiple edges with an arc between them indicate a non-unit selection group, where the source activity is the activator and the target activities the successor activities. This is illustrated in Figure 1.

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

An example of a precedence graph and a selection graph is given in

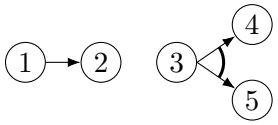

Figure 1: A unit selection group $(a_g, S_g) = (1, \{2\})$ (left) and a non-unit selection group $(a_g, S_g) = (3, \{4, 5\})$ (right).

Figure 2. The selection graph in this example contains 5 selection groups, with $(0, \{1, 2\})$ being the only non-unit selection group. This imposes that if activity 0 is executed, either activity 1 or activity 2 has to be executed. Furthermore, if activity 0 is executed, activity 3 has to be executed as well. Finally, for activities 1, 2 and 3, it follows that if one of them is executed, activity 4 has to be executed as well.

The precedence graph is very similar, and thus, activities 1, 2 and 3 can only start after the end of activity 0, and activity 4 can only start after the end of activities 1, 2 and 3, only considering executed activities. Furthermore, since there is a directed edge between activity 2 and 3, activity 3 can only start after activity 2 is finished, if both are executed. If activity 2 is not executed, activity 3 can start directly after finishing activity 0.

In Kellenbrink and Helber (2015), the selection logic was mainly modeled by *choices*, *activity causing a choice*, and *optional activities per choice*. Respectively, these are analogous to selection groups, activators and successor activities. The difference between these concepts is that selection groups support independent succession. Furthermore, to simplify the model, the concept of *activities caused by another activity* in Kellenbrink and Helber (2015) are not included the presented model, but can be modeled by unit-selection groups instead. This was done to simplify both modeling and mathematical analysis for cutting planes, which are presented in Section 4. Finally, this paper does not consider nonrenewable resources. This was done since the focus of the proposed solution method is the flexible project structure. Since nonrenewable resources are not a standard component of the RCPSP, we exclude this extension from the model.

3.2. Model

To model the problem, we introduce binary decision variables X_{it}, which are equal to one if activity $i \in N$ starts at time $t \in T$, and zero otherwise. 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 of 0 and the final activity can only be executed after

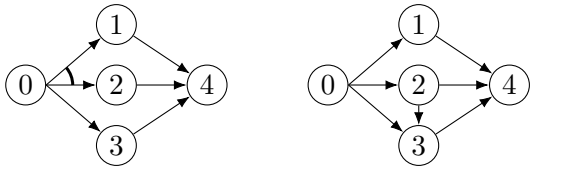

Figure 2: Example of graphs.

all other executed activities. Objective function (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) imposes 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) imposes 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 the resource capacity λ_r. Since the linear relaxation of Objective function (1a), Constraints (1b), (1c), (1e) to (1h) and (1j) is quite weak, we add Constraints (1i). For each selection group with full precedence $g \in H$, these constraints ensure that the sum of the starting times of all successor activities S_g is larger than or equal to the finishing time of the activator activity a_g. Finally, Constraints (1j) specify that the decision variables X_{it} are binary.

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

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

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

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

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

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

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

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

$$\sum_{t\in T}(t+d_{a_g})X_{a_g t} \leq \sum_{j\in S_g}\sum_{t\in T} tX_{jt}, \quad \forall g \in H, \tag{1i}$$

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

Note that this model can be easily generalized by letting the availability of resources vary with time. Furthermore, varying resource usage can be modeled by splitting up activities.

3.3. Activity selection problem

After introducing the model, we now give a formal definition for the activity selection problem. Furthermore, a note is made for modeling the case in which the exclusivity criterion does not hold. Recall that the exclusivity criterion is the criterion that per selection group, if is the activator activity is executed, exactly one successor has to be executed. With this, the *selection problem* is defined as follows: Given a selection graph, find a selection of activities including activity 0, such that for each selection group 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).

Although the exclusivity criterion of the selection group is important for certain real-world applications, there are also cases in which the exclusivity criterion does not hold and where more than one successor can be executed per selection group. In these cases, there is an 'at least one'-requirement instead of an 'exactly one'-requirement. This can be modeled by adding dummy activities (activities with no duration and no resource requirements) before each successor activity, as is demonstrated in Figure 3.

On the left side in Figure 3, selection group $(0, \{3, 4\})$ is shown with the exclusivity criterion; either 3 or 4 can be executed. However, activity 3 and

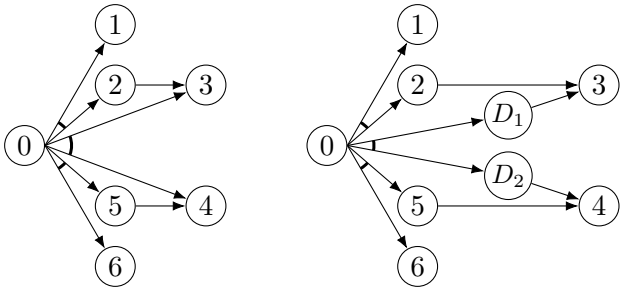

Figure 3: Instances where from activities 3 and 4, exactly one (left) and at least one (right) activity has to be executed.

4 can also be activated by either activity 2 or 5. In some applications, it would be prefered to allow either activity 3 or 4, or both. This is the case in the right-hand side of Figure 3. Here, the exclusivity criterion is moved to the dummy activities d_1 and d_2. This means that for activity 3 and 4, at least one has to be executed instead of exactly one.

Note that dummy activities only have to be added for successor activities that are also successor activities of other groups. To demonstrate this, consider a selection group $g \in G$, which should be transformed such that from the successor activities S_g, at least one has to be executed instead of exactly one. Then, a dummy activity only has to be added before each successor activity $j \in S_g$, if j is also a successor of another selection group. In Figure 3, this means that if there is no arc from activity 5 to activity 4, dummy activity d_2 is not required. The proof for this is given in Lemma 2 in Appendix B.

4. Solution method

In this section, we present an exact solution method for the RCPSP-PS based on preprocessing and subsequently using a MILP solver. The preprocessing procedure uses a variable reduction method and cutting planes. The variable reduction method is based on the Critical Path Method and is described in Section 4.1. This method sets earliest start times and latest finish times for the individual activities. Subsequently, in Section 4.2, we identify properties of the selection problem. With these properties, we give two types of cutting planes. Finally, in Section 4.3, the solution algorithm is given. The solution algorithm uses a combination of variable reduction and cutting planes to increase the earliest start times. Subsequently, it solves this preprocessed problem by using a MILP solver.

4.1. Variable reduction

One of the most common methods of variable reduction for the RCPSP is using the Critical Path Method to define the earliest and latest start time. However, since the selection graph and precedence graph do not necessarily coincide, this method cannot be used. For example, consider activity i and activity j with $(i, j) \in \mathcal{P}$. The Critical Path Method would then set the earliest starting time of activity j equal to the earliest start time of activity i plus the duration of activity i. However, in the RCPSP-PS, it can happen that activity i is not executed, and therefore, activity j is not restricted by activity i. Therefore, in this subsection, another way of setting the earliest start time s_i and latest start time f_i for each activity $i \in N$ is given.

The variable reduction method is based on what we call a *Non-Empty Execution Set* (NEES): A set of activities of which at least one has to be executed. Variable reduction can then be done based on the following principle: If each activity in a 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 $\max_{j \in A}\{f_j - d_j\}$

In the remainder of this section, a variable reduction method for the RCPSP-PS that uses NEESs is given. First, some notation is introduced. After this, it is shown how to identify a NEES in a subset of activities. Based on this, an algorithm is given that determines the earliest start times and latest finish times.

For the variable reduction method, we need to know whether a candidate set $N' \subseteq N$ contains a NEES. For intuition, consider all activities as the candidate set ($N' = N$) in the selection graph as shown in Figure 4. 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 Figure 4, no choice can be made at selection group g_1 such that no activity in A will be executed.

To identify a NEES in a subset of activities $N' \subseteq N$, we solve the MILP given by Constraint set (2) to select groups and activities. A selected group means that this group is on the path between activity 0 and the NEES, while all selected activities form the NEES. First, we introduce the set G_i, which 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

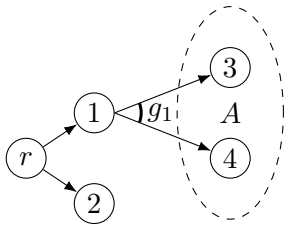

Figure 4: Selection graph example of a NEES. Since either activity 3 or activity 4 always has to be executed, $\{3, 4\}$ is a NEES.

to one if a group or activity is selected, respectively, and zero otherwise. From the starting activity 0, at least one group has to be selected due to Constraints (2a). Then, if a group g is selected, for each successor $i \in S_g$ either a group $h \in G_i$ or activity i itself has to be selected if $i \in N'$. If successor $i \in N'$, 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).

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

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

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

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

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

We now prove that V from a feasible solution to Constraint set (2) constitutes a NEES.

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

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 Constraints (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. □

With this, we now give an algorithm to compute the earliest starting time s_i for each activity $i \in N$.

The first loop in Algorithm 1 loops over all activities in topological order. This ensures that s_i is defined for every predecessor $i \in \mathcal{P}_j$. Furthermore, it defines the sequence $B = \{b_1, \cdots, b_{|B|}\}$. This sequence contains all activities in $i \in \mathcal{P}_j$, ordered by earliest finishing time $s_i + d_i$ in descending order. Loop 2 then takes incremental subsets B' of the ordered set of predecessors P and tries to solve Constraint set (2) with $N' = B'$.

For each iteration of Loop 2, an element j is added to $N' = B'$ until Constraint set (2) becomes feasible. If adding element j to N' results in Constraint set (2) becoming feasible, it follows that $V_j = 1$. Otherwise, Constraint set (2) would also be feasible in the previous iteration. Since B' is ordered in descending order of $s_i + d_i$, in the previous iteration, B' contained all activities $i \in B$ with $s_i + d_i \geq s_j + d_j$. 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

```
1: N^(s) ← topological sorting of N on precedence graph
2: s_i ← 0 ∀i ∈ N
3: for all j ∈ N^(s) do                                   ▷ Loop 1
4:     B ← P_j, sorted by non-increasing s_i + d_i.
5:     stop ← False
6:     n ← 1
7:     while stop = False AND n ≤ |B| do                  ▷ Loop 2
8:         B' ← first n elements of B
9:         Solve Constraint set (2) for N' = B'
10:        if Constraint set (2) is feasible then
11:            s_j ← min_{i∈B'}{s_i + d_i}
12:            stop ← True
13:        end if
14:        n ← n + 1
15:    end while
16: 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 ascending 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.

In our implementation, T is determined by using an initial solution created by a simple heuristic. This heuristic uses a MILP solver to obtain a feasible selection of activities. Then, these activities are ordered by starting at activity 0 and iteratively picking a precedence-feasible activity at random.

4.2. Cutting Planes

Due to Constraints (1g), the linear relaxation of the proposed MILP is very weak. To strengthen the formulation, two types of cutting planes are presented. The first type is based on groups of activities of which *at least* one has to be selected. For the second type, groups of activities of which *at most* one can be selected are used.

The procedure is the same for both cutting plane types: First, the linear relaxation is solved to obtain the relaxed solution. Subsequently, the sepa-

ration problem is solved to obtain a cutting plane that cuts off the relaxed solution. This cutting plane is added to the original problem and a new solution is obtained. This is repeated until no more cutting planes can be found or when the objective has not increased for a fixed number of iterations.

4.2.1. Non-Empty Execution cutting planes

In this subsection, a method of adding cutting planes is presented that are generated based on *Non-Empty Execution Sets*. Recall that these are sets in which at least one activity has to be executed. The separation problem is given by Objective function (3) 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.

$$\min \sum_{i \in N} V_i \sum_{t \in T} X^*_{it}. \tag{3}$$

By Lemma 1, the index set of V forms a NEES. Thus, if the value of Objective function (3) is smaller than 1, the fractional solution X^* contains a NEES that has less than 1 activity executed. Therefore, we add Constraint (4) as a cutting plane, where A is the index set of V; $A = \{i \in N : V_i = 1\}$,

$$\sum_{i \in A} \sum_{t \in T} X_{it} \geq 1. \tag{4}$$

4.2.2. Max-one cutting planes

The final type of cutting planes is based on *max-one execution sets* (MOES). We call cutting planes generated by this method 'max-one cutting planes'. Like the model strengthening constraints given by Constraints (1i), this method requires selection groups with full precedence. A MOES $N' \subseteq N$ is a set of activities of which at most one will be executed in the optimal solution. To identify these sets, *common rooted paths* (CRP) are used as defined in Definition 4.1. 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 $\mathcal{V}(P)$.

In the remainder of this section, we will give a definition of a CRP and prove that if there is a CRP between two activities, at most one of these activities will be executed. Subsequently, we present an algorithm to find all CRPs in the selection graph. Finally, these CRPs are used to generate a new type of cutting planes.

Definition 4.1 (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. Consider two activities i and j that are both executed in the solution. This is illustrated in Figure 5. 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. □

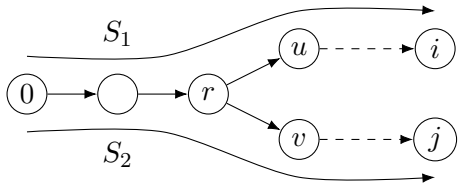

Figure 5: Example for CRP. There is a CRP $(r, (r, u, \cdots, i), (r, v, \cdots, j))$.

All MOES can be found by using a *rooted path graph* (RPG), as defined in Definition 4.2.

Definition 4.2 (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 in the RPG represents a set of activities without any CRP's between them, and thus, a MOES. 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). 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.

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 $j \in S_g$.

Given these definitions, we now present Algorithm 2, 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; $E^{(f)} \leftarrow E^{(f)} \cup \{(r, i) : r \in N, i \in \Theta_r\}$

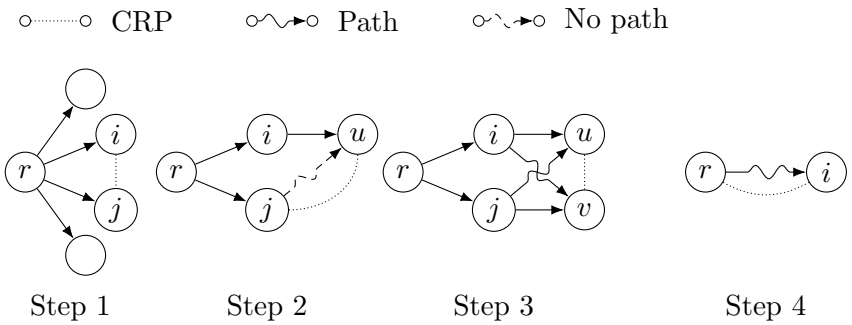

Figure 6: Steps in RPG algorithm.

These steps are illustrated in Figure 6.

Algorithm 2 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

We now show that Algorithm 2 creates an RPG.

Theorem 1
Algorithm 2 creates a rooted path graph if the selection graph is acyclic.

Proof. **If there is an edge created by Algorithm 2, there is a CRP:** For Step 1, 2, and 3, we will prove this by induction. The base case is given by Step 1. In 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:

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.

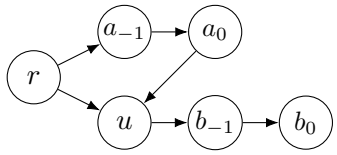

Figure 7: (i, j) created by Step 2, with $u \in \mathcal{V}(Q)$.

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 Figure 7, 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 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 Figure 8. 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 2: Consider activities i and j with a CRP $\{r, P, Q\}$, $P = (r, p_1, \cdots, p_n)$ and $Q = (r, q_1, \cdots, q_m)$. 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). □

We now give the separation problem for maximum execution cutting planes in Constraint set (5). These cutting planes are based on a 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 activity i from the MOES is executed, then there is always an activity j executed that has to be executed after finishing activity i.

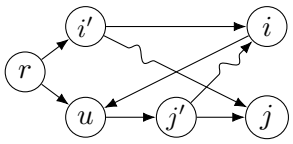

Figure 8: (i, j) created by Step 3, with $u \in \mathcal{V}(Q)$.

The MILP formulated by Constraint set (5) uses the optimal relaxed solution of Constraint set (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 $Y_g = 1$ and $Y_g = 0$ otherwise.

Constraints (5b) only select full-precedence groups for which the activators have no CRP. Constraints (5c) define that activities can only be selected within the MOES, if they are the activator of a selected full-precedence group. Furthermore, Constraints (5d) select only successors of selected full-precedence groups. The first term in Objective function (5a) represents the finishing times of the MOES, 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.

$$\max \sum_{i \in N} W_i \sum_{t \in T} (t + d_i) X^*_{it} - \sum_{j \in N} Z_j \sum_{t \in T} t X^*_{jt}, \tag{5a}$$

$$Y_g + Y_h \leq 1, \quad \forall g \in H, h \in H, (a_g, a_h) \in E^{(f)}, a_g \neq a_h, \tag{5b}$$
$$Y_g \geq W_{a_g}, \quad \forall g \in H, \tag{5c}$$
$$Z_j \geq Y_g, \quad \forall g \in H, j \in S_g, \tag{5d}$$
$$W_i \in \{0, 1\}, \quad \forall i \in N \text{ for which } |\{g : i = a_g, g \in H\}| \geq 1, \tag{5e}$$
$$Y_g \in \{0, 1\}, \quad \forall g \in H, \tag{5f}$$
$$Z_j \in \{0, 1\}, \quad \forall j \in N. \tag{5g}$$

Proposition 2. *Let X^* be the linear relaxed solution of Constraint set (1). Furtermore, let W^*, Y^* and Z^* be the solution of Constraint set (5) and let*

the value of Objective function (5a) be larger than 0. Then, Constraint (6) is a cutting plane for the RCPSP-PS that cuts of the current solution X^*:

$$\sum_{i \in N} W_i^* \sum_{t \in T} (t + d_i) X_{it} \leq \sum_{i \in N} Z_i^* \sum_{t \in T} t X_{it}. \tag{6}$$

Proof. Constraints (5b) imposes that groups can only be selected if there is no CRP between the activators. Therefore, in combination with Constraints (5c), 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 (6) 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 Equation (7). If no activity i' for which $W_{i'}^* = 1$ is executed, the first term of Constraint (6) is zero and, therefore, it is a cutting plane.

$$\sum_{i \in N} W_i^* \sum_{t \in T} (t + d_i) X_{it} = \sum_{t \in T} (t + d_{i'}) X_{i't} \leq \sum_{t \in T} t X_{j't} \leq \sum_{i \in N} Z_i^* \sum_{t \in T} t X_{it}. \tag{7}$$

Therefore, as long as Objective function (5a) has a value larger than 0, Constraint (6) cuts of the current solution X^*. □

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 treshold for a fixed number of iterations.

4.3. Constraint propagation algorithm

In the preceding part of this section, a method for variable reduction and two types of cutting planes and their separation algorithms are given. In the remainder of this section, we combine these methods with a MILP solver to create a solution algorithm for the RCPSP-PS. This is based on constraint propogation; increments in lower bounds per activity are propagated to set bounds on other activities.

First, the initialization and individual functions used in this algorithm are presented. Subsequently, the solution algorithm is given.

The solution algorithm first generates a set of cutting planes $\mathcal{C}_i$ for each activity $i \in N$ as follows. First, we replace activity $n+1$ in Objective function (1a) by activity i and adding the constraint $\sum_{t \in T} X_{it} = 1$, which we refer to as *setting the objective function to i*. This gives a MILP 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 MILP is then used to generate cutting planes as given by Constraints (4) and (6), which together form set $\mathcal{C}_i$.

Subsequently, the algorithm generates a set of forced activities $\mathcal{F}_i$ for each activity $i \in N$. A 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 all activities $k \in N$ without a CRP between k and i. 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 $p \in N', p \neq j$, then $\{j\}$ is a NEES and is thus always executed. In this case, activity $j \in \mathcal{F}_i$.

Furthermore, we introduce $\mathbf{s} = [s_0, \cdots, s_{n+1}]$ as the vector of earliest starting times for all activities in N, which is initialized to $\mathbf{0}$. With this, we introduce three functions: *linrelax*(i,$\mathcal{C}_i$,$\mathbf{s}$), *variable_reduction*($\mathbf{s}$) and *solve*($\mathbf{s}$,$\mathcal{C}_i$). The first function, *linrelax*($i, \mathcal{C}_i, \mathbf{s}$), solves the linear relaxation of Constraint set (1) while setting the objective activity to i, adding cutting planes $\mathcal{C}_i$ and setting the lower bound on activity starting times to $\mathbf{s}$ for all activities in N. The latter is done by setting $X_{jt} = 0$ for $t < s_j$ for each activity $j \in N$. 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.

The function *variable_reduction*($\mathbf{s}$) calls Algorithm 1, with the modification of using $\mathbf{s}$ as initial lower bounds instead of setting it to zero in line 2. It then returns lower bounds for all activities. Finally, the function *solve*($\mathbf{s}$,$\mathcal{C}_{n+1}$) first calculates the latest finishing time f_i for each node $i \in N$ as described in Section 4.1 and then solves the MILP while adding cutting planes $\mathcal{C}_{n+1}$ (since $n+1$ is the objective activity) and setting $X_{it} = 0$ for all $i \in N$ with $t < s_i$ or $t + d_i > f_i$.

Algorithm 3 Solution algorithm

$\mathcal{C} \leftarrow$ cutting planes
$\mathcal{F} \leftarrow$ forced activities
for all $i \in N$ **do**
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\mathcal{C}_j : j \in \mathcal{F}_i\}$
end for
$\mathbf{s} \leftarrow \mathbf{0}$
$N^{(s)} \leftarrow$ topological sorting of N on precedence graph
improved $\leftarrow$ True
while improved = True **do**
    improved $\leftarrow$ False
    for all $i \in N^{(s)}$ **do**
        $v \leftarrow \text{linrelax}(i, \mathcal{C}_i, \mathbf{s})$
        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($\mathbf{s}, \mathcal{C}_{n+1}$)

With these functions, the solution algorithm is given in Algorithm 3. Initially, the lower bounds $\mathbf{s}$ are set to $\mathbf{0}$. Subsequently, the sets of cutting planes $\mathcal{C}_i$ and sets of forced activities $\mathcal{F}_i$ for every activity $i \in N$ are calculated. Since a cutting plane for an activity j is valid when this activity is executed, all cutting planes from forced activities $\mathcal{F}_i$ are valid when the objective function is set to activity i. Therefore, each set of cutting planes is updated by adding the cutting planes from all forced activities. The algorithm will now loop over all activities in topological order $N^{(s)}$. For each activity $i \in N^{(s)}$, a lower bound will be calculated by using *linrelax*(i,$\mathcal{C}_i$,$\mathbf{s}$). 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 will be repeated. If not, the while loop will terminate. Subsequently, to get an optimal solution, the *solve*($\mathbf{s}$,$\mathcal{C}_{n+1}$) function is called to solve the MILP.

5. Computational results

To evaluate the performance of the cutting planes and algorithms introduced above, computational tests are performed on 30 instances. These instances have 30 to 70 activities, and are grouped per number of activities. There are 8 instances for each number of activities. To generate the instances, the generation procedure from Vanhoucke et al. (2008) is used for the precedence graph, and the selection graph is generated by modifying this precedence graph. Both the tests and the instance generation code is available at van der Beek (2021).

The tests were performed on a computer with an Intel i7 1.9 GHz processor with 8GB Ram. For each instance, two tests were done. The first one is to evaluate the lower bound obtained by relaxing the integrality constraints. Secondly, the integer solutions are computed for the most promising methods. The linear relaxations were all solved in a timespan of minutes. For the integer solution, Gurobi 8.1.1 (Gurobi, 2021) was used with a maximum computation time of 6 hours and all standard settings, except for setting MIPFocus=3.

In total, we compare 5 solution methods, as described below:

1. *Basic*: Solve the MILP given by model Constraint set (1) for final activity $n+1$; i.e., calling function *solve*($\mathbf{0}$,$\emptyset$).

2. *Cutting planes*: *Basic* method with the addition of cutting planes from Constraints (4) and (6) for only final activity $n+1$; i.e., calling function *solve*($\mathbf{0}$,$\mathcal{C}_{n+1}$)

3. *Variable reduction*: *Basic* method combined with the variable reduction method given in Algorithm 1; i.e., calling function *solve*($\mathbf{s}$,$\emptyset$) with $\mathbf{s}$ from *variable_reduction*($\mathbf{0}$).

4. *Combined*: *Basic* method combined with both the *cutting planes* method and the *variable reduction* method; i.e., calling function *solve*($\mathbf{s}$,$\mathcal{C}_{n+1}$) with $\mathbf{s}$ from *variable_reduction*($\mathbf{0}$).

5. *Constraint propagation*: Algorithm 3.

5.1. Linear relaxation

An important indicator of the strength of each method is the lower bound resulting from solving the linear relaxation of the problem. Since the *Constraint propagation* method is a combination of all other methods, the lower

bound reached by this method is always at least as large as the other methods. Therefore, to compare methods, this can be used to normalize the lower bounds for the other methods.

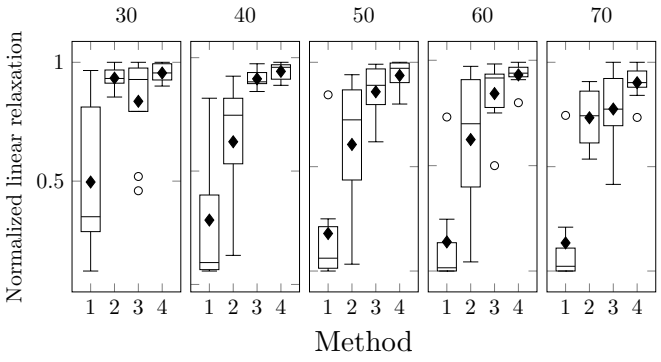

Figure 9: Lower bounds per method, normalized to the *Constraint propagation* method. Methods: 1. Basic 2. Cutting planes 3. Variable reduction 4. Combined.

The values of the lower bounds are shown in Figure 9. Since all values are normalized to the value of the *Constraint propagation* method, the value of this method is always equal to 1 and therefore not shown in the figure.

Compared to the *Basic* method, one can see that the improvement in lower bound increases with instance size for all methods. On average, for the set of largest instances, the lower bound computed by *Constraint propagation* method is 7.42 times as large as the lower bound achieved by the *Basic* method.

Furthermore, the values for the *Combined* method are higher than both the *Cutting planes* method and the *Variable reduction* method for each number of activities. This shows that combining these methods give a higher lower bound than they do separately.

5.2. Integer solutions

To evaluate the strength of preprocessing, the integer solutions are computed. This is done by first preprocessing (adding cutting planes or removing variables), and subsequently using the commercial MILP solver Gurobi for which each run is limited to 6 hours. Since these tests take significantly longer than the tests for the linear relaxation, only the two most promising methods (*Combined* and *Constraint propagation*) are tested, along with the *Basic* method for comparison purposes.

First, the number of instances where an optimal solution was found is evaluated. Figure 10 shows that the *Combined* method solves in total one instance more to optimality than the basic method. It should be noted, though, that this gain is obtained for the smaller instances. For instances with 60 activities, the *Combined* method performs worse than the *Basic* method. The *Constraint propagation* method, however, outperforms the *Basic* method for each instance and solves 3 additional instances to optimality.

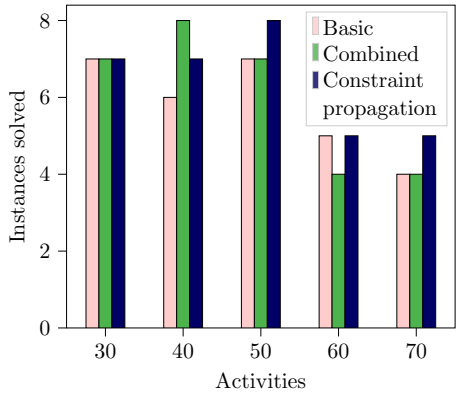

Figure 10: Number of instances solved to optimality per method. Each number of activities has 8 instances.

Secondly, the computation time is evaluated per method. The average computation times are given in Table 2. Furthermore, the spread can be seen in Figure 11. Here it can be seen that, except for the instances of size 40, the *Constraint propagation* method is the fastest method on average. For the instances of size 40, the *Combined* method is nearly 3 times as fast as the other methods.

Since the computation time, which consists of preprocessing time and computation time, for the *Constraint propagation* method is shorter than the *Basic* method for most instance groups, it can be concluded that the time spent on preprocessing is worth it due to the decrease in solver duration.

Finally, the lower and upper bounds will be evaluated. To compare the upper bounds across instances, we will evaluate the gap between the upper bound u and the best found lower bound l^*, normalized by the best found lower bound. This gives $\frac{u-l^*}{l^*}$. Similarly, to normalize the lower bound l, we evaluate $\frac{u^*-l}{u^*}$, where u^* is the best found upper bound per instance.

The normalized upper bounds are shown in Figure 12. It is shown that

Number of activities	Basic	Combined	Constraint propagation
30	3509	3331	3039
40	5201	1796	5362
50	3790	4587	1569
60	9827	11012	9279
70	11435	11235	9651

Table 2: Average computation times in seconds.

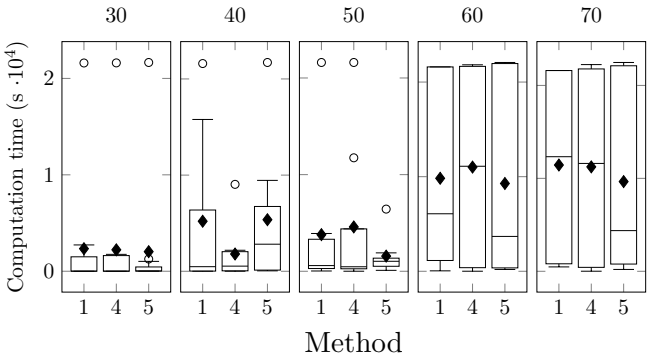

Figure 11: Computation time. Methods: 1. Basic 4. Combined 5. Constraint propagation.

for instances with up to 50 activities, the found upper bounds are equal. Note that for instances of 40 activities, Figure 10 shows that optimality was proven more often by the *Combined* method. Therefore, it can be concluded that for other methods the optimal solution was found, but no optimality guarantee was given.

For the larger instances, no clear benefit is shown for the *Combined* method and the *Constraint propagation* method. On average, the *Constraint propagation* method performs best for the instances with 60 activities, but for the instances with 70 activities the *Basic* method has the best upper bound.

For the lower bounds, however, a more clear pattern is visible. This is plotted in Figure 13 and the average values are shown in Table 3. It can be seen that for instances with 40 or more activities, the lower bounds obtained by the *Combined* method or by the *Constraint propagation* method outperform, on average, the lower bound obtained by the *Basic* method.

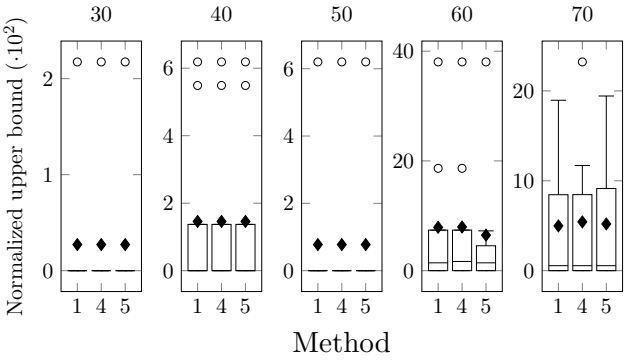

Figure 12: Upper bounds. Methods: 1. Basic 4. Combined 5. Constraint propagation.

Number of activities	Basic	Combined	Constraint propagation
30	0.09	0.27	0.09
40	1.38	0.00	0.14
50	0.73	0.64	0.00
60	4.45	3.79	3.81
70	3.66	2.73	3.09

Table 3: Average normalized lower bounds ($\cdot 10^2$).

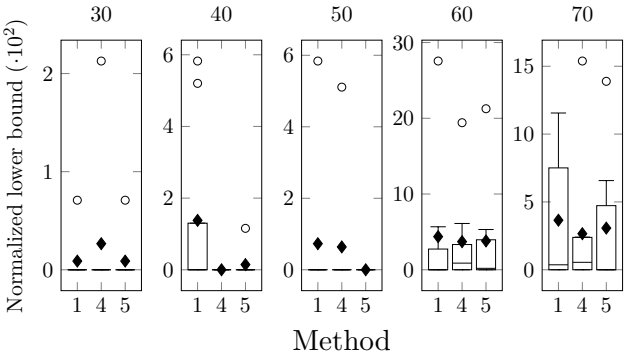

Figure 13: Lower bounds. Methods: 1. Basic 4. Combined 5. Constraint propagation.

6. Conclusions

In this paper, we developed a general model for the RCPSP-PS. This model represents the RCPSP-PS in a general form, by introducing *selection groups* and the selection graph as the only new element compared to the

traditional RCPSP, which enables easier modeling and mathematical analysis. 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 these techniques significantly increase the lower bound on the objective value for nearly all instances. Furthermore, we ran a commercial MILP solver on the preprocessed instances. In these tests, it was shown that the extra duration of further preprocessing is smaller than the time gained due to the decrease in solving duration, therefore validating the use of the preprocessing algorithm.

Thus, the proposed solution methods decrease the computational time for nearly all instances. To further decrease the computational time, multiple cutting planes could be added per iteration instead of only one. As the proposed solution methods decrease the solution space, they can also be incorporated into other exact approaches, such as constraint programming. Furthermore, they can assist in the development of heuristic methods by verification and validation of the found heuristic solutions.

7. 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).

References

Araujo, J.A., Santos, H.G., Gendron, B., Jena, S.D., Brito, S.S., Souza, D.S., 2020. Strong bounds for resource constrained project scheduling: Preprocessing and cutting planes. Computers & Operations Research 113, 104782. doi:10.1016/j.cor.2019.104782.

Artigues, C., Demassey, S., Néon, E., Sourd, F., 2008. Resource-Constrained Project Scheduling. ISTE, London, UK. doi:10.1002/9780470611227.

Baptiste, P., Demassey, S., 2004. Tight LP bounds for resource constrained project scheduling. OR Spectrum 26, 251–262. doi:10.1007/s00291-003-0155-1.

Barták, R., Čepek, O., Surynek, P., 2007. Modelling alternatives in temporal networks. Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007 , 129–136doi:10.1109/SCIS.2007.367680.

van der Beek, T., 2021. Instances, fileformat and generator script for the Resource Constrained Project Scheduling Problem with a flexible Project Structure. doi:10.4121/14124080.v1.

Blazewicz, J., Lenstra, J., Kan, A., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5, 11–24. doi:10.1016/0166-218X(83)90012-4.

Brucker, P., Knust, S., 2000. Linear programming and constraint propagation-based lower bound for the RCPSP. European Journal of Operational Research 127, 355–362. doi:10.1016/S0377-2217(99)00489-0.

Gurobi, 2021. Gurobi Optimizer Reference Manual. `http://www.gurobi.com`.

Haouari, M., Kooli, A., Néron, E., 2012. Enhanced energetic reasoning-based lower bounds for the resource constrained project scheduling problem. Computers and Operations Research 39, 1187–1194. doi:10.1016/j.cor.2011.05.022.

Hardin, J.R., Nemhauser, G.L., Savelsbergh, M.W.P., 2008. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. Discrete Optimization 5, 19–35. doi:10.1016/j.disopt.2007.10.003.

Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207, 1–14. doi:10.1016/j.ejor.2009.11.005.

Hauder, V.A., Beham, A., Raggl, S., Parragh, S.N., Affenzeller, M., 2020. Resource-constrained multi-project scheduling with activity and time flexibility. Computers and Industrial Engineering 150, 106857. doi:10.1016/j.cie.2020.106857, `arXiv:1902.09244`.

Herroelen, W., Leus, R., 2005. Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165, 289–306. doi:10.1016/j.ejor.2004.04.002.

Kellenbrink, C., Helber, S., 2015. Scheduling resource-constrained projects with a flexible project structure. European Journal of Operational Research 246, 379–391. doi:10.1016/j.ejor.2015.05.003.

Kuster, J., Jannach, D., Friedrich, G., 2009. Extending the RCPSP for modeling and solving disruption management problems. Applied Intelligence 31, 234–253. doi:10.1007/s10489-008-0119-x.

Lombardi, M., Milano, M., 2012. Optimal methods for resource allocation and scheduling: A cross-disciplinary survey. Constraints 17, 51–85. doi:10.1007/s10601-011-9115-6.

Pellerin, R., Perrier, N., Berthaut, F., 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 280, 395–416. doi:10.1016/j.ejor.2019.01.063.

Pritsker, A.A.B., Watters, L.J., Wolfe, P.M., 1969. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. Management Science 16, 93–108. doi:10.1287/mnsc.16.1.93.

Rubeša, R., Fafandjel, N., Kolić, D., 2011. Procedure for Estimating the Effectiveness of Ship Modular Outfitting. Engineering Review 31, 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, 841–860. doi:10.1016/j.ejor.2018.09.005.

Talbot, F.B., 1982. Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case. Management Science 28, 1197–1210. doi:10.1287/mnsc.28.10.1197.

Tao, S., Dong, Z.S., 2017. Scheduling resource-constrained project problem with alternative activity chains. Computers and Industrial Engineering 114, 288–296. doi:10.1016/j.cie.2017.10.027.

Tao, S., Dong, Z.S., 2018. Multi-mode resource-constrained project scheduling problem with alternative project structures. Computers and Industrial Engineering 125, 333–347. doi:10.1016/j.cie.2018.08.027.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L.V., 2008. An evaluation of the adequacy of project network generators with sys-

tematically sampled networks. European Journal of Operational Research 187, 511–524. doi:10.1016/j.ejor.2007.03.032.

Wglarz, 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, 177–205. doi:10.1016/j.ejor.2010.03.037.

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, 379–388. doi:10.1016/j.aei.2010.07.002.

Zhou, J., Love, P.E., Wang, X., Teo, K.L., Irani, Z., 2013. A review of methods and algorithms for optimizing construction scheduling. Journal of the Operational Research Society 64, 1091–1105. doi:10.1057/jors.2012.174.

Zhu, G., Bard, J.F., Yu, G., 2006. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing 18, 377–390. doi:10.1287/ijoc.1040.0121.

A. Notation

Variables

V_i	1 if activity $i \in N$ is selected for the NEES and zero otherwise
W_i	1 if activity $i \in N$ is selected for the MOES and zero otherwise
X_{it}	1 if activity $i \in N$ is executed at time $t \in T$, zero otherwise
Y_g	1 if group $g \in H$ is selected and zero otherwise
Z_i	1 if activity $i \in N$ is selected as successor activity and zero otherwise

Parameters

a_g	Activating activity of selection group $g \in G$
d_i	Duration of activity $i \in N$
f_i	Latest finish time of activity $i \in N$
k_{ri}	Usage of resource $r \in R$ for activity $i \in N$.
l	Lower bound on objective.
M	Very large number
n	Number of non-dummy activities
s_i	Earliest start time of activity $i \in N$
u	Upper bound on objective.
$\ell(P)$	Number of vertices in path P
λ_r	Capacity of resource $r \in R$

Sets

$E^{(a)}$	Active edges representing a CRP
$E^{(f)}$	Final edges representing a CRP
$E^{(n)}$	New edges representing a CRP
G	Selection groups
H	Selection groups with full precedence
N	Activities
R	Resources
S_g	Successor activities of selection group $g \in G$
T	Time periods
$\mathcal{C}_i$	Cutting planes for activity $i \in N$
$\mathcal{F}_i$	Forced activities for activity $i \in N$
$\mathcal{P}$	Precedence relationships (tuples of 2 activities)
$\mathcal{P}_j$	Predecessors of activity $j \in N$ in the precedence graph
$\mathcal{R}_i(A)$	All paths starting in i and ending in an activity in set A, with only the last activity in set A
$\mathcal{S}_i$	Successors of activity $i \in N$ in the precedence graph
$\mathcal{V}(P)$	Vertex set of path P
Γ	All activity pairs (i, j) if $i \in N$ and $j \in N$ are successors in the selection graph of the same selection group
Θ	Pairs (i, j) of activities where i is reachable from j of vice versa
Θ_i	All activities that are reachable from activity $i \in N$
Ω_i	Successors of activity $i \in N$ in the selection graph

B. Dummy variables

Lemma 2

Consider a selection group $g \in G$ for which S_g does not contain the final activity $n+1$. This selection group will be modified such that at least one successor $i \in S_g$ has to be executed if the activator a_g is executed, instead of exactly one. This can be achieved by applying the following algorithm:

Step 1 *Let $S'_g = \{i : i \in S_g, \{k : k \in G, i \in S_k, k \neq g\} \neq \emptyset\}$ be a subset of S_g containing only activities that are also successors of another group. Create a dummy activity D_i for each successor activity $i \in S'_g$.*

Step 2 *Create a selection group h with $a_h = a_g$ and $S_h = \{D_i : i \in S'_g\} \cup \{i \in S_g \setminus S'_g\}$*

Step 3 *Create a selection group h_i for each $i \in S'_g$ with $a_{h_i} = D_i$ and $S_{h_i} = \{i\}$.*

Step 4 *Remove selection group g.*

Proof. The algorithm adds dummy activities for all activities $i \in S'_g$. We first show that if we do this for all activities $i \in S_g$ instead, we impose an 'at least one' constraint instead of an 'exactly one' constraint. After that, we show that if we remove all dummy activities (and corresponding groups) for $i \in S_g \setminus S'_g$, the solution stays feasible and the optimal solution value does not change.

First, apply the algorithm with $S'_g = S_g$. Now, Constraints (1e) and (1f) impose for selection group h that if a_h is executed, *exactly* one dummy variable has to be executed. Consequently, Constraints (1e) impose that *at least* one activity $i \in S_g$ has to be executed, since activity $i \in S_g$ can also be executed when the activating dummy activity D_i is not executed.

Let $\mathcal{A}$ be the problem instance obtained by performing the algorithm for $S'_g = S_g$. Furthermore, let $\mathcal{B}$ be the problem instance we obtain by using S'_g instead of S_g. Converting $\mathcal{A}$ to $\mathcal{B}$ can be done as follows: For each activity $i \in S_g \setminus S'_g$, remove dummy activity D_i, remove selection group h_i, and replace successor activity D_i in selection group h by the original successor activity i.

To show that the 'at least one'-criterium also holds for $\mathcal{B}$, we will show that each solution X to instance $\mathcal{A}$ can be converted to a solution Y to instance $\mathcal{B}$ and vice versa, while keeping the same objective value.

Let X be a solution to problem $\mathcal{A}$. We now show that solution X can be converted to a feasible solution Y for problem $\mathcal{B}$ with the same objective value. Converting is done by projecting all values of X on Y and modifying the values for $i \in S_g \setminus S'_g$ if needed.

Firstly, we consider the case where activity $i \in S_g \setminus S'_g$ is not executed in solution X. By Constraints (1e), it follows that dummy activity D_i is also not executed. Therefore, removing D_i and selection group h_i does not cause any infeasibilities and Y remains a feasible solution for $\mathcal{B}$.

Secondly, consider the case where activity $i \in S_g \setminus S'_g$ is executed. If dummy activity D_i was executed, the number of executed successor activities for selection group h stays the same in solution Y, which therefore remains feasible for problem $\mathcal{B}$. If D_i was not executed, which is possible considering Constraints (1e), there is an infeasibility in problem $\mathcal{B}$ for group h in Constraints (1f). However, since activity i is not the successor of any other group, it can be set to not executed. This does not cause any infeasibilities because Constraints (1e) impose in the direction of activator to successor and not in the reverse direction.

Therefore, any solution X for problem $\mathcal{A}$ can be converted to a solution Y for problem $\mathcal{B}$. Since the value of the objective activity $n+1$ is not changed, the objective value remains equal.

Next, we show that a solution Y for problem $\mathcal{B}$ can be converted to a solution X for problem $\mathcal{A}$ with equal objective value. This is done by projecting Y on X and setting the values for the dummy activities as required.

Again, consider activity $i \in S'_g \setminus S_g$. If this activity is not executed, set the corresponding dummy activity D_i to not executed in X as well. Then, the problem remains feasible.

Now, consider the case where activity $i \in S'_g \setminus S_g$ is executed. In this case, none of the dummy activities are executed and dummy activity D_i can be set to executed in solution X to obtain a feasible solution. Similar as for the reverse case, the objective activity $n+1$ is not changed, and therefore, the objective value remains equal.

Thus, there exists a solution X for problem $\mathcal{A}$ if and only if there exists a solution Y for problem $\mathcal{B}$ with the same objective value. Therefore, the 'at least one'-criterion from problem $\mathcal{A}$ is also imposed on problem $\mathcal{B}$. $\square$