European Journal of Operational Research 335 (2026) 105–117

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

Progressive hedging algorithm for the resource constrained project scheduling problem with modular production

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

ARTICLE INFO

Keywords:
Resource constrained project scheduling problem
Flexible project structure
Stochastic optimization
Modular production
Progressive hedging

ABSTRACT

In modular shipbuilding, modules are used to lower construction costs and decrease lead times. Achieving these decreases in both costs and time requires making the right choices in material use and activity planning. Therefore, we introduce the *Resource Constrained Project Scheduling Problem with Modular Production* in which decisions are made for the inventory level of resources, activity selection and activity scheduling, in order to maximize profit minus inventory costs. Since these decisions have to be made before uncertain project arrival information is revealed, a *scenario-tree* based approach is used that optimizes over multiple scenarios simultaneously. An *Integer Linear Programming* formulation is introduced for this problem and a *Progressive Hedging* algorithm to find good solutions to this problem, along with two extensions to this algorithm. A computational study is performed, where the activity selection decisions are used to model choices in modular production and outsourcing. The basic PH algorithm outperforms using a commercial solver to find feasible solutions to the ILP model, in terms of both solution quality and computing time. However, the basic PH algorithm still has a hard time converging to an implementable solution, which makes the algorithm rely heavily on a repair step. The introduced extensions improve the convergence properties significantly, and can also be used to prioritize solution quality and/or computing time.

1. Introduction

The shipbuilding industry is an industry producing long-term products, where building times usually vary from a few months to a few years. In order for shipbuilding companies to remain competitive, improvements are made on both the technical aspect and the process aspect. Improving the shipbuilding process is done for multiple reasons, of which one is lead time reduction. The lead time is the time between ordering a product and receiving it. Since ships are essential elements for many trade operations and on-sea construction projects, delayed access to ships can postpone other operations (Athanasia et al., 2012).

One method of reducing lead time is to build ships to stock. Although this is done in some specific cases (Ship&Offshore, 2024), shipbuilding usually is a one-off industry where each product is unique. This prohibits keeping pre-built ships in inventory. A possible method to obtain the benefits of pre-building, while maintaining product flexibility, is **assemble-to-order** (ATO) production (Storch & Sukapanpotharam, 2003). In ATO production, a product is divided into pre-built modules that are assembled on customer order to decrease the production time. However, shipbuilding is a complex process and implementing an ATO strategy is not straightforward. Having items or modules in inventory can significantly reduce lead times, but will also incur costs. These costs are, among others, insurance costs, storage costs and depreciation costs. This raises an important question: what should be the inventory of items and modules with long lead times?

Once an inventory level has been decided upon for each module, used modules will have to be replenished. This can be done by the shipyard itself, or the production can be outsourced to third parties. Outsourcing generally induces additional costs, such as costs for transportation. However, in-house production influences the resource capacity at the shipyard. This can affect the lead times of incoming shipbuilding projects.

Finally, using modules modifies the structure of the project schedule for shipbuilding. Using a module usually requires cranes for transportation and assembly, as opposed to building without a module, which usually requires more on-board crew. Furthermore, it also has effects on other activities. For example, if the installation space is already enclosed by other parts of the ship, the pre-assembled module cannot be moved

* 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.2026.02.026
Received 28 April 2024; Accepted 16 February 2026
Available online 19 February 2026

to the desired installation location. However, without using modules, it might still be possible to transport separate parts and assemble them at the required location.

Modular design and production has also been introduced in other industries then shipbuilding. One of the earliest and most successful industries is the computer industry. By using modularity, Dell designed a line of technologically competitive, lower-priced PCs. This was done by creating an ATO production process, which was able to eliminate most of the inventory and reduce costs (Zhu et al., 2014). Another industry that has adopted modular production is the automotive industry (Frigant & Lung, 2002). In this industry, different parts of the world use different modularization strategies such as outsourcing (Sako & Said, 1999) or in-house modular production (Pandremenos et al., 2009).

In general, modular construction results in a scheduling problem where decisions have to be made on activity starting times, resource inventory levels, outsourcing decisions and project structure decisions. Furthermore, due to the size and duration of the production process in, for example, shipbuilding, it can be beneficial to make different decisions per project.

Determining activity starting times for a single project with a fixed project structure where precedence constraints and resource availability need to be taken into account, can be modelled as the **Resource Constrained Project Scheduling Problem** (RCPSP) (Pritsker et al., 1969). To allow for project structure decisions or modularization choices, we have introduced the RCPSP with a flexible project structure in Van der Beek et al. (2025) for which an exact solution approach was developed. To also allow for consumption and production of nonrenewable resources, the **Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources** (RCPSP-PS/CPR) was introduced in Van der Beek et al. (2023). This allows the modelling of, for example, floor capacity. This combination of flexible project structure and the consumption and production of resources had not been studied before. Also the implementation of heuristic methods considering nonrenewable resources with consumption and production had not been studied before. Therefore, we have developed a **Hybrid Differential Evolution** (HDE) algorithm to solve the RCPSP-PS/CPR which performs well when compared to an exact solution method and an alternative heuristic.

In this paper, we extend the RCPSP-PS/CPR by considering a multi-project setting in which projects arrive over time with a stochastic arrival time. We also include setting the inventory level for nonrenewable resources and maximizing profit leading to the **Resource Constrained Project Scheduling Problem with Modular construction and new Project arrivals** (RCPSP-MP). As decisions have to be made at arrival of a project, this influences scheduling capabilities in future projects. Therefore, the problem can be considered a *multi-stage stochastic optimization problem*. To find solutions to this problem, we present a modified version of the **Progressive Hedging** (PH) algorithm that uses the hybrid differential evolution algorithm introduced in Van der Beek et al. (2023) to solve the scenario subproblems. This algorithm consists of a basic version and two extensions that improve the performance.

In Section 2, an overview is presented of related research for RCPSP-PS/CPR, dynamic multi-project scheduling, and multi-stage optimization. Subsequently, in Section 3, the RCPSP-MP is described and formulated. After this, in Section 4, the solution method is given. Finally, this solution method is evaluated in Section 5 and the paper is concluded in Section 6.

2. Literature review

In this section, we review the literature related to the RCPSP-MP, which can be categorized as a **Dynamic Resource Constrained Multi-Project Scheduling Problem (DRCMPSP)** where each individual project is an instance of the RCPSP-PS/CPR. Therefore, we first present literature on the RCPSP-PS/CPR followed by relevant literature on the DRCMPSP. Since relevant literature on the DRCMPSP is sparse, we also present related literature on general multi-stage optimization.

2.1. Resource constrained project scheduling problem with a flexible project structure and consumption and production of resources

The RCPSP is introduced by Pritsker et al. (1969) and is proven to be NP-hard by Blazewicz et al. (1983). In this section, we focus on the extensions that we consider for each individual project in our multi-project problem: a flexible project structure and consumption and production of resources.

Carlier et al. (2009) introduce the **Resource Constraint Project Scheduling Problem with Consumption and Production of Resources** (RCPSP/CPR), which uses an event-based approach. For this problem, they provide a scheduling algorithm that computes the optimal schedule by enumerating over all linear orders of events. They also shortly discuss how the exact enumeration algorithm can be modified to obtain a heuristic method. Koné et al. (2013) consider the problem with cumulative resources. They provide a time-indexed ILP model for the discrete time RCPSP/CPR and a flow-based formulation for the continuous-time RCPSP/CPR. Shirzadeh Chaleshtarti et al. (2020) present a genetic algorithm for the RCPSP with nonrenewable resources. This consists of the standard version of the RCPSP, with a fixed amount of initially available resources that can only be consumed and not produced. Since there is no flexible project structure and resources can only be consumed, the amount of initial resources fully defines feasibility regarding nonrenewable resources. Therefore, resource infeasibility is not taken into account for this problem.

Besides the consumption and production of resources, the second extension is the flexible project structure. Although there are different variants under different names in the literature, the main concept of this extension is that only a subset of all activities have to be executed. One of the earliest RCPSP variants with a flexible project structure is the *Extended RCPSP*, introduced by Kuster et al. (2009). They study an RCPSP with an initial activation state and substitution criteria, which define what changes are allowed to the initial state. They give a custom evolutionary algorithm to heuristically solve this problem. Furthermore, Čapek et al. (2012) study a variant of the RCPSP with a flexible project structure, unary resources, time-lags and sequence dependent setup times. They represent the branching structure by Petri nets and give both an MILP and a constructive heuristic algorithm. The RCPSP with a flexible project structure is studied in Kellenbrink and Helber (2015). They model this by distinguishing between mandatory and optional activities, and introduce a set of choices to decide which optional activities have to be executed. They include nonrenewable resources, but only with consumption of these resources and without production. To heuristically solve this problem, they use a genetic algorithm. Another formulation is given by Tao and Dong (2017), who represent the problem by an AND-OR project network. They call this problem the RSPCP with alternative activity chains and give an extended simulated annealing algorithm to heuristically solve it. Furthermore, in Tao and Dong (2018), they extend the problem by adding multiple modes of executing an activity and by considering multi-objective optimization. This new problem is heuristically solved by a hybrid algorithm consisting of tabu search and a genetic algorithm. Servranckx and Vanhoucke (2019) define the RCPSP with alternative subgraphs. This problem consists of branches, where each branch represents a subset of activities that can be executed. This is heuristically solved using tabu search. Furthermore, Van der Beek et al. (2025) introduce an MILP model where the choices are based on *selection-groups*. A solution method is given that uses cutting planes and constraint propagation for preprocessing, after which the problem is solved to optimality by a commercial MILP solver.

Van der Beek et al. (2023) is the first to introduce the extension of the RCPSP with both the flexible project structure and consumption and production of resources. They introduce the concept of group graphs and show how to make a feasible selection of activities in polynomial time.

They use this concept to schedule the selected activities using a hybrid differential evolution algorithm.

2.2. Dynamic resource constrained multi-project scheduling problem

Relevant research on the DRCMPSP is sparse and can be categorized as either reactive baseline scheduling or finding scheduling policies for online decision making. Pamay et al. (2014) and Capa and Ulusoy (2015) focus on baseline scheduling where an initial baseline schedule is created, for which the number of later modifications done has to be minimized. Melchiors et al. (2018) and Satic et al. (2022) modelled the problem as a Markov decision process and used dynamic programming to determine optimal policies for small instances. Satic et al. (2024) used approximate dynamic programming to determine good policies for realistic instances. The obtained policies indicate for each time period which of the unknown available tasks need to be processed resulting in online decision making. Our approach can be categorized as proactive scheduling where all tasks of an arriving project need to be scheduled while taking into account the potential future arrival of new projects. Next to this, our setting is different in the sense that we consider each project to be an RCPSP-PS/CPR whereas in related literature each project is considered to be an instance of the RCPSP.

2.3. Multi-stage optimization

Since literature on the DRCMPSP is sparse, we broaden our view to general **Stochastic Programming** (SP). This is a framework for modeling decisions under uncertainty and was originally introduced by Dantzig (1955). Multi-stage SP involves making decisions over multiple **stages** (i.e. decision making moments). Between each stage, some uncertain data is revealed. Since multi-stage SP models are usually difficult to solve, the probability distributions are often discretized by creating realizations of the uncertain parameters. Such a realization is called a **scenario**. To solve a discretized problem, decomposition methods can be used. These methods fall into two categories: stage based decomposition, such as the L-shaped method (Van Slyke & Wets, 1969), and scenario based methods, such as **dual decomposition** (DD) (Carøe & Schultz, 1999) and progressive hedging (Rockafellar & Wets, 1991). An overview of recent developments of these methods is given in Torres et al. (2019). They state that both the L-shaped method and dual decomposition often lead to long computing times, particularly in the case when the MILP problem has a poor LP relaxation. Additionally, an overview of multi-stage methods is given in Bakker et al. (2020), where it is stated that scenario decomposition methods are generally attractive for multi-stage decomposition methods. Finally, there are several PH methods that find solutions to the decomposed subproblems by a meta-heuristic (Hasannia Kolaee & Mirzapour Al-e-Hashem, 2022; Haugen et al., 2001; Løkketangen & Woodruff, 1996). Since the subproblems of the RCPSP-MP are computationally demanding, the possibility of quickly solving them is an important requirement. Thus, for the reasons stated above, PH is deemed as a promising algorithm for the RCPSP-MP. Therefore, we now present research on the PH algorithm.

Progressive Hedging (Rockafellar & Wets, 1991) iteratively solves each scenario subproblem separately while trying to converge to a feasible solution. Originally, it was proposed for convex stochastic problems, where convergence to an optimum is guaranteed. Although this guarantee is not available when integer variables are present, it has been applied successfully for these types of problems as a heuristic. However, this creates two problems: first, the subproblems might not be solvable in a reasonable amount of time. Second, there are no theoretical guarantees for convergence to a feasible solution.

To address the issue of finding good solutions to subproblems within a reasonable time, researchers have been using heuristic algorithms. One of the first implementations of this is presented in Løkketangen and Woodruff (1996). They give a general method for mixed integer multi-stage stochastic programming problems with binary variables. They use a general version of PH, although they find good solutions to the scenario subproblems with a tabu search algorithm. They apply this method to a general production planning problem. A more specific application of the PH algorithm is given by Haugen et al. (2001). They apply PH to a multi-stage lot sizing problem, with costs for producing and backlogging. Similar to Løkketangen and Woodruff (1996), they heuristically solve the subproblems. Furthermore, Hasannia Kolaee and Mirzapour Al-e-Hashem (2022) study a problem in medical tourism, that involves the allocation and transportation of patients to hospitals. To find good solutions, they present a PH algorithm that finds solutions to the subproblems with a genetic algorithm.

Furthermore, various research has focused on improving convergence behaviour of the PH algorithm for non-convex problems. Watson and Woodruff (2011) study a resource allocation problem and provide several algorithmic improvements, which result in better convergence behavior. First, they provide a method for defining the penalty-term multipliers separately per variable. Second, they provide various ways of variable fixing to improve convergence. Third, they propose a new termination criterium, tailored to their specific optimization problem. Finally, they give a method for detecting cyclic behaviour in penalty terms and a mitigation for this. Furthermore, Crainic et al. (2011) present PH metaheuristics for a problem in selecting arcs for stochastic network design, with various improvement strategies. They give an adjustment strategy for the penalty terms, which increased/decreased costs if an arc is chosen in the minority/majority of scenarios. Furthermore, they adjust costs based on deviations from the average. Guo et al. (2015) combine PH with DD, where they use PH to create a starting point for the DD algorithm and present a method to transform PH penalty weights to Lagrange multipliers for DD. They apply this algorithm to a server location problem and a unit commitment problem for electricity generation. Furthermore, Lamghari and Dimitrakopoulos (2016) combine PH, mixed integer linear programming, and heuristics to find solutions to a pit mining problem. They first run PH while finding solutions to the scenario subproblems with a heuristic algorithm, after which they use a mixed integer linear programming solver to solve a restricted problem to reach convergence. Additional acceleration techniques are given by Peng et al. (2019). They investigate a stochastic resource allocation problem, where the cost of a production process is minimized, including inventory, backorder and production cost. For this, a PH algorithm is used that solves the subproblems with a MILP solver. They introduce three methods to define the penalty term in the PH algorithm. Furthermore, they introduce acceleration techniques based on penalty-linearization, variable fixing, early terminations and warm starts. Finally, we discuss the work of Jiang et al. (2021). They study a stochastic service network design problem and introduce the idea of soft clustering: creating scenario bundles with probabilistic membership. This means that a scenario can be part of multiple bundles simultaneously.

2.4. Contribution

In conclusion, it can be stated that the current research on the DRCMPSP does not give useful research directions for the RCPSP-MP as no flexible structure is considered and either a reactive baseline scheduling approach or online decision making approach is used. Therefore, we considered different SP methods, of which the PH algorithm seemed most promising for the considered setting. Although PH only has proven convergence properties for convex problems, it is used as a heuristic for non-convex problem fairly successful. However, as research indicates, this often requires various acceleration/improvement techniques.

Therefore, this research adds to existing literature in the following ways: (1) we consider a proactive scheduling approach opposed to either reactive baseline scheduling or online decision making to make sure that the completion time of a project is known at the moment of arrival, (2) we allow for a flexible project structure and consumption and production of resources for all arriving projects, and (3) we consider the inventory level of nonrenewable resources to be a decision

variable. Compared to our previous work in Van der Beek et al. (2023), we extend the RCPSP-PS/CPR by determining resource inventory levels and considering profit maximization and stochastic project arrivals resulting in the Resource Constrained Project Scheduling Problem with Modular Production (RCPSP-MP). For the RCPSP-MP, we introduce an ILP formulation and present a PH algorithm to solve this problem. The results are compared against the results obtained by solving the ILP to optimality using a commercial solver for small instances. Finally, we compare different extensions of the PH algorithm.

Note that we do not focus on developing solutions methods for the RCPSP-PS/CPR as this was already studied in Van der Beek et al. (2023).

3. Problem description

The RCPSP-MP is a multi-stage stochastic optimization problem, where at the first stage the resource inventory levels have to be set and, at subsequent stages when a project arrives, the scheduling decisions have to be made for this project. In this section, a formal description of this problem is given. To represent the uncertainty, the concept of a **scenario tree** is used. A scenario tree is a rooted tree where each **scenario tree node** defines a decision moment. The root node defines the first stage and thus the first decision moment. Then, each child node represents a possible next stage realization. From each child node, its respective child node again represents a possible next stage, and so on. Then, a path from the root node to a leaf node represents a scenario: a possible realization of all stages of the stochastic process.

Since a node can be part of multiple root-leaf paths, it can also be part of multiple scenarios. However, at the moment of making the decisions for this node, it is not known which of these scenarios will occur. Therefore, if the problem is decomposed by scenarios, all decisions for a certain node across all of its scenarios should be the same. This is referred to as the **Non Anticipativity Constraints** (NACs). If a solution satisfies all NACs, the solution is called **implementable**.

In the RCPSP-MP, activities have to be planned according to two types of resources: renewable resources R^r and nonrenewable resources R^n. A renewable resource, such as workers or machines, regenerates automatically after use. For each renewable resource $r \in R^r$, the initial availability is input and given by λ_r. A nonrenewable resource is a resource that does not always automatically renew after the consuming activity ends. Instead, the consumption and/or production can be set separately per activity. An example is an inventory item or module, for which a reorder activity has to be performed to replenish it. The availability of both renewable and nonrenewable resources is required to be integer, in order to model resources such as machines, modules, etc. To simplify the replenishment decisions, we assume a fixed base inventory level Y_r that has to be decided upon for each nonrenewable resource $r \in R^n$ at the root node of the scenario tree. For each activity that consumes an amount of resource $r \in R^n$, a reorder activity can be scheduled that replenishes this resource. After setting the inventory levels at the root node, a project that has to be scheduled arrives at each descendant node of the scenario tree. The goal is to maximize the total profit, which consists of profit for executing activities at certain times minus costs for inventory. Thus, to summarize, the projects are scheduled according to the following assumptions:

- In the first decision moment, the starting inventory level is set. In each subsequent decision moment, an incoming project is scheduled.
- Rescheduling is not allowed.
- Each activity can produce, or consume, nonrenewable resources.
- Each activity can use renewable resources, which are renewed when the activity finishes.
- Resource shortages are not allowed.
- Resource levels cannot exceed the initial inventory level.

An example of a scenario tree with 3 types of candidate projects can be seen in Fig. 1. Consider, for example, the scenario represented by nodes 0-1-4. At the root node, the resource inventory levels have to be decided for the nonrenewable resources. Then, at node 1, a project of type A arrives with arrival time $\tau = 10$ and has to be scheduled. At this moment, we only know that we are either in the scenario represented by nodes 0-1-3 or 0-1-4. Finally, due to node 4, a project of type C arrives with arrival time $\tau = 120$.

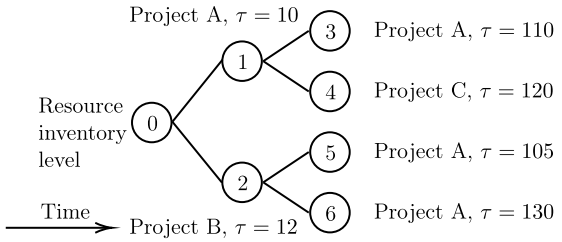


Fig. 1. Example of a scenario tree for 3 stages (2 arriving projects).

We now give a formal description of the scenario tree and the projects contained in it. Let Ω be the set of all scenarios. In practice, the set of scenarios will be constructed by sampling projects from the product portfolio. The number of scenarios will be set as large as possible, constrained by computational time and performance of the solution method. Ψ the set of all scenario tree nodes, Ω_ψ the set of all scenarios that contain scenario tree node $\psi \in \Psi$ and let Ψ_ω be the set of scenario tree nodes in scenario $\omega \in \Omega$. Furthermore, we let scenario tree node $\psi \in \Psi$ with $\psi = 0$ be the root node. Then, we let $\mathcal{P}_\psi$ be the project arriving at non-root node $\psi > 0$ and τ_ψ the arrival time of project $\mathcal{P}_\psi$.

Project $\mathcal{P}_\psi$ consists of a set of possible activities N_ψ including a starting and final activity. The starting activity is forced to be executed and, by the design of the selection graph, also the final activity is always executed. To model the choices of using certain modules, a **flexible project structure** is used. This means that from all activities N_ψ, only a subset has to be selected for execution. The structure of these choices is defined by a set of selection groups G_ψ. Each selection group $g \in G_\psi$ has an activator activity a_g and a set of successor activities S_g. For each selection group $g \in G_\psi$, it holds that if the activator activity a_g is chosen to be executed, exactly one successor in S_g has to be executed as well. When the activator activity a_g is not chosen to be executed, there is no restriction on the number of activities in S_g that can be executed. The set of selected activities has to be scheduled in the set of discrete time periods $T = \{1, \dots, |T|\}$. Each activity $i \in N_\psi$ has a duration of d_i time periods and a profit of p_{it} when scheduled at time $t \in T$. Furthermore, the set P_ψ defines the precedence relationships. For each precedence relationship $(i, j) \in P_\psi$, activity j can start only after activity i has been finished, if both activity i and j are executed. Furthermore, each activity $i \in N_\psi$ requires k_{ri} units of renewable resource $r \in R^r$. For each nonrenewable resource $r \in R^n$, k_{ri}^- units are consumed at the start of activity i and/or k_{ri}^+ units are generated at the end.

Thus, at each non-root scenario tree node $\psi \in \Psi \setminus \{0\}$, a project $\mathcal{P}_\psi$ arrives. Therefore, a scenario $\omega \in \Omega$ then has a set of sequentially arriving projects $\{\mathcal{P}_\psi | \psi \in \Psi_\omega \setminus \{0\}\}$. These projects can be combined by creating finish to start precedence constraints between the final and starting activity of subsequent projects. This creates a multi-project $\mathcal{P}_\omega^m$ per scenario $\omega \in \Omega$, that represents all sequentially arriving projects. The model for the RCPSP-MP represents all these multi-projects simultaneously.

For multi-project $\mathcal{P}_\omega^m$ for scenario $\omega \in \Omega$, let $N_\omega = \cup_{\psi \in \Psi_\omega} N_\psi$ be the set of activities and $G_\omega = \cup_{\psi \in \Psi_\omega} G_\psi$ the set of selection groups. Furthermore, consider activity $i \in N_\omega$, created by project $\mathcal{P}_\psi$ from selection tree node $\psi \in \Psi_\omega$. Then, we define $N_{\omega i}$ as the set of identical activities from selection tree node ψ in *other* scenarios $\omega' \in \Omega, \omega \neq \omega'$. Similarly, $G_{\omega g}$ is the set of identical selection groups of the same selection tree node in different scenarios. Additionally, we let N and G be the union of all activities and selection groups over all scenarios, respectively ($N = \bigcup_{\omega \in \Omega} N_\omega$ and $G = \bigcup_{\omega \in \Omega} G_\omega$). Finally, we define s_ω as the first activity of multi-project $\mathcal{P}_\omega^m$ and let P be the total set of all precedence relationships. With this, we can give the **Integer Linear Program** (ILP) of the RCPSP-MP.

The RCPSP-MP has two types of decision variables that are used to make decisions on resource inventory level, activity selection and activity scheduling. As stated before, Y_r defines the initial resource availability for each nonrenewable resource $r \in R^n$. Furthermore, X_{it} is a binary decision variable, used to both select and schedule activities. This binary decision variable is equal to 1 if activity $i \in N$ starts at time $t \in T$ and zero otherwise. Note that when X_{it} equals zero for all $t \in T$, this means that activity $i \in N$ is not selected.

$$\max \sum_{\omega \in \Omega} \sum_{i \in N_\omega} \sum_{t \in T} \frac{p_{it} X_{it}}{|\Omega|} - \sum_{r \in R^n} c_r Y_r \tag{1}$$

$$\sum_{t \in T} X_{s_\omega t} = 1, \forall \omega \in \Omega, \tag{2}$$

$$\sum_{t \in T, t < \tau_\psi} X_{it} = 0, \forall \psi \in \Psi, i \in N_\psi \tag{3}$$

$$\sum_{t \in T} X_{it} \leq 1, \forall i \in N, \tag{4}$$

$$\sum_{t \in T} X_{a_g t} \leq \sum_{i \in S_g} \sum_{t \in T} X_{it}, \forall g \in G, \tag{5}$$

$$\sum_{j \in S_g} \sum_{t \in T} X_{jt} \leq |S_g| - \left(|S_g| - 1\right) \sum_{t \in T} X_{a_g t}, \forall g \in G, \tag{6}$$

$$\sum_{t \in T} (t + d_i) X_{it} \leq \sum_{t \in T} t X_{jt} + M\left(1 - \sum_{t \in T} X_{jt}\right), \forall (i,j) \in P, \tag{7}$$

$$\sum_{i \in N_\omega} \sum_{t'=t-d_i+1}^{t} k_{ri} X_{it'} \leq \lambda_r, \forall \omega \in \Omega, r \in R^r, t \in T, \tag{8}$$

$$\sum_{i \in N_\omega} \left(\sum_{t'=1}^{t} k_{ri}^- X_{it'} - \sum_{t'=1}^{t-d_i} k_{ri}^+ X_{it'} \right) \leq Y_r, \forall \omega \in \Omega, r \in R^n, t \in T, \tag{9}$$

$$\sum_{i \in N_\omega} \left(\sum_{t'=1}^{t} k_{ri}^- X_{it'} - \sum_{t'=1}^{t-d_i} k_{ri}^+ X_{it'} \right) \geq 0, \forall \omega \in \Omega, r \in R^n, t \in T, \tag{10}$$

$$\sum_{t \in T} t X_{it} = \sum_{t \in T} t X_{jt}, \forall \omega \in \Omega, i \in N_\omega, j \in N_{\omega i}, \tag{11}$$

$$X_{it} \in \{0,1\}, \forall i \in N, t \in T, \tag{12}$$

$$Y_r \in \mathbb{Z}_{\geq 0}, \forall r \in R^n. \tag{13}$$

The objective is to schedule the projects such that the profit is maximized. Profit p_{it} for activity $i \in N$ is a non-increasing function in t as an earlier completion of a project is usually preferred. Even though this seems equivalent to minimizing the makespan, we focus on profit such that we can balance profit with inventory costs. Generally, the profit is zero for most activities, except for the final one or for some milestone activities. Furthermore, deducting a constant value (relative to t) from p_{it} allows for a fixed cost of executing activity i, which can be used to model outsourcing costs. From this profit, we subtract the inventory costs, which are determined by multiplying the resource availability Y_r of renewable resource $r \in R^n$ by the fixed cost c_r. With this, we get Objective function (1).

Furthermore, Constraints (2) specify that for each scenario, the starting activity has to be selected for execution. Constraints (3) define that activities can only be scheduled after the project has arrived. Subsequently, Constraints (4) impose that each activity is scheduled to start in at most one time period. The selection groups are handled by Constraints (5) and (6). For each selection group $g \in G$, the former define that if the activator activity a_g is selected for execution, at least one successor activity is selected as well. The latter define that if the activator activity a_g is selected for execution, at most one successor can be selected. Furthermore, Constraints (7) impose that, for each precedence relationship $(i,j) \in P$, when both activity i and j are executed, activity j starts after activity i is finished. The resource constraints are set by Constraints (8) to (10). Constraint set (8) imposes the constraints on renewable resources, while the latter do this for the nonrenewable resources. Constraints (9) impose that the total resource consumption cannot be larger than the initial availability Y_r, thus assuring that the resource inventory levels cannot be negative. Part of the resource inventory level costs are due to keeping the items in the inventory, such as costs for storage space, insurance costs or financing costs. Therefore, the available resource inventory level cannot be higher than the initial resource inventory level Y_r, for each nonrenewable resource $r \in R^n$. Thus, Constraints (10) define that the total resources consumption cannot be negative. Subsequently, Constraints (11) represent the NACs. They impose that for different scenarios, the identical activities within the same selection tree nodes must have the same selection and scheduling time decisions. Finally, Constraints (12) and (13) define the variable domains. All notation introduced in this section is listed in Table 1.

Table 1
Notation for Section 3.

a_g	Activating activity of selection group $g \in G$.
c_r	Unit cost of initial availability of nonrenewable resource $r \in R^n$.
d_i	Duration of activity $i \in N$.
G_ω	Selection groups in scenario $\omega \in \Omega$.
$G_{\omega g}$	Selection groups identical to selection groups $g \in G_\omega$ for scenario $\omega \in \Omega$
k_{ri}	Net resource production of resource $r \in R$ for activity $i \in N$.
k_{ri}^+	Production of resource $r \in R$ for activity $i \in N$.
k_{ri}^-	Consumption of resource $r \in R$ for activity $i \in N$.
M	Sufficiently large number.
N	Activities.
N_ω	Activities in multi-project $\mathcal{P}_\omega^m$ of scenario $\omega \in \Omega$.
$N_{\omega i}$	Activities identical to activity $i \in N_\omega$ for scenario $\omega \in \Omega$ in all other scenarios.
$\mathcal{P}_\omega^m$	Multi-project representing all projects of scenario $\omega \in \Omega$.
$\mathcal{P}_\psi$	Project arriving at scenario $\psi \in \Psi$.
P	Predecessor-successor pairs.
p_{it}	Profit obtained by scheduling activity $i \in N$ at time $t \in T$.
R	Resources.
R^n	Nonrenewable resources.
R^r	Renewable resources.
s_ω	Starting activity of multi project $\mathcal{P}_\omega^m$ of scenario $\omega \in \Omega$.
S_g	Successor activities of selection group $g \in G$.
T	Time periods.
X_{it}	1 if activity $i \in N$ is executed at time $t \in T$, zero otherwise.
Y_r	Initial resource availability of nonrenewable resource $r \in R^n$.
λ_r	Capacity of renewable resource $r \in R^r$.
Ψ	Scenario tree nodes.
Ω	Scenarios.

In the given ILP formulation, the pre-built modules in ATO production are included in the set of nonrenewable resources R^n for which the inventory level needs to be determined. Alternatively, instead of using a module, the separate parts can be assembled immediately on the ship, represented by other nonrenewable resources for which the inventory needs to be determined. As both types of nonrenewable resources have their own fixed costs, lead time to replenish, and required resources for assembly, solving the ILP will determine which pre-built modules to use to optimize profit in ATO production.

Note that this ILP formulation is similar to the formulation introduced in Van der Beek et al. (2023). We have added multiple projects and scenarios in Constraint sets (2), (3), and (11) and we have included renewable resources in Constraint set (8).

4. Solution method

To find solutions to the RCPSP-MP, a **Progressive Hedging** (PH) algorithm is created. This algorithm uses PH to make decisions for the resource inventory level and activity selection. Subsequently, the **Hybrid Differential Evolution** (HDE) algorithm introduced in Van der Beek et al. (2023) is used to create the full schedules for each scenario in Ω separately. An overview of the complete algorithm is given in Fig. 2. In this section, we first present the solution representation. Subsequently, we give a short description of the HDE algorithm, after which the PH algorithm is given. Finally, we present two extensions that improve the performance of the PH algorithm. All notation introduced in this section is listed in Table 2.

Table 2
Notation for Section 4.

$alt\_obj(\cdots)$	Altered objective function.
LR	Lag parameter for resource inventory level variables.
LS	Lag parameter for selection variables.
N^s	Selectable activities.
$obj(\vec{x}_\omega)$	Unaltered objective function value of solution $\vec{x}_\omega$.
PM	Penaly multiplicator.
r	Penalty ratio.
SC	Differential evolution scaling parameter.
$\vec{w}_\omega$	Lagrangian multiplier for scenario $\omega \in \Omega$.
$\vec{x}_\omega$	Compressed solution vector of scenario $\omega \in \Omega$.
$\bar{\vec{x}}_\omega$	Average compressed solution vector of scenario $\omega \in \Omega$.
γ	HDE population size.
Ψ_ω	Scenario tree nodes in scenario $\omega \in \Omega$.
Ω_ψ	Scenarios that contain scenario tree node $\psi \in \Psi$.

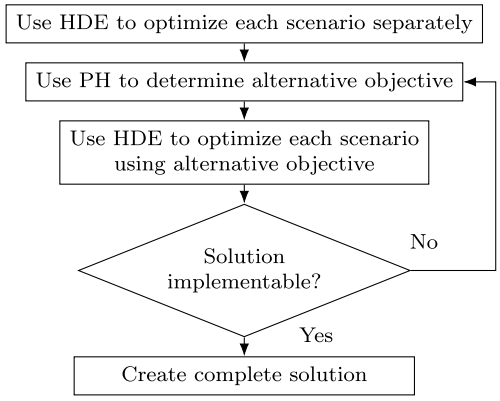


Fig. 2. Progressive Hedging algorithm.

4.1. Solution representation

The PH algorithm decomposes the optimization problem by scenarios. These scenarios are then, iteratively, optimized heuristically while the objective function is altered to steer the solution towards implementability, i.e., identical activities within the same selection tree nodes must have the same selection and scheduling time decisions across all of its scenarios. This means that we consider two optimization processes: The **complete optimization process** that finds one implementable solution that is feasible for all scenarios in the scenario trees using PH, and the **subproblem optimization process**, where each scenario is optimized separately using HDE.

These two optimization processes both have a unique solution representation. The vector representing a solution has continous entries, since this is required by both the PH and the HDE algorithm. The subproblem optimization algorithm HDE uses the **subproblem solution vectors**. Each subproblem solution vector is a concatenation of a **resource vector**, a **selection priority vector** and a **scheduling priority vector**. The resource vector has $|R^n|$ entries. Each entry represents, after being rounded, the initial resource availability Y_r for nonrenewable resource $r \in R^n$. The selection and scheduling priority vectors indicate the priority for each activity $i \in N$ to be selected or scheduled, respectively. To convert these priority vectors into a schedule, we first use the selection priority vector to select activities. This is done by iteratively selecting the activity with the highest priority from all available activities given the precendence constraints and already selected activities. After this, in a similar way, the activity with the highest scheduling priority is scheduled iteratively from all currently available activities at the earliest time possible. For more details on this, we refer to Van der Beek et al. (2023).

Additionally, the complete optimization algorithm PH uses **compressed solution vectors**. This vector contains only the nonrenewable resource availabilities and information on the selection of activities. The PH algorithm tries to converge the compressed solution vectors to an implementable solution. Since the focus of the RCPSP-MP is on resource inventory level and modularization choices, and because of the increased computational difficulty of converging on starting times, implementability on starting times is handled later in the algorithm. Furthermore, we can differentiate the activities to be selected by activities that represent an actual choice, and activities where the selection follows from other choices. Since the PH algorithm makes the solutions converge to be implementable by penalizing entries in the solution vector, we do not want to distort these penalties by activities that do not represent a choice. Therefore, we define a **selectable activity** as any successor activity in the set of successor activities S_g of selection group $g \in G$, when successor group S_g contains at least two successor activities. Thus, the set of selectable activities can be denoted by $N^s = \{i \in N | g \in G, i \in S_g, |S_g| > 1\}$.

Then, we create the compressed solution vector $\vec{x}_\omega$ by concatenating the vector of nonrenewable resource availabilities Y and a vector consisting of a binary value for each selectable activity that indicates whether activity $i \in N$ is selected or not.

4.2. Hybrid differential evolution

The individual scenario subproblems are solved by the *Hybrid Differential Evolution* (HDE) algorithm, introduced in Van der Beek et al. (2023). This is a differential evolution algorithm with a Forward Backward Improvement (FBI) step (Valls et al., 2008). The HDE algorithm has a population of γ solution vectors represented by set $\mathcal{X}$. In each iteration, a **trial** solution vector $\vec{b}$ is created for each solution $\vec{x} \in \mathcal{X}$ in the population. This is done by first randomly selecting 3 solutions $\vec{a}^1$, $\vec{a}^2$, and $\vec{a}^3$ from set $\mathcal{X}$ such that $\vec{x}$, $\vec{a}^1$, $\vec{a}^2$, and $\vec{a}^3$ are all different. Then, a **mutation** solution vector $\vec{m}$ is created using scaling parameter SC as follows: $\vec{m} = \vec{a}^1 + SC(\vec{a}^2 - \vec{a}^3)$. Then, each entry of the trial solution vector $\vec{b}$ has the corresponding entry value from mutation solution vector $\vec{m}$ with probability RP and the corresponding entry value from solution $\vec{x}$ with probability $1 - RP$. Trial solution $\vec{b}$ replaces the original solution $\vec{x}$ in set $\mathcal{X}$ if it has a better objective.

This process is executed iteratively for the complete population. If no best new solution has been found for TP iterations, an FBI improvement step is performed for all solutions, which first schedules the activities as late as possible in decreasing order of their finish times and then schedules the activities as early as possible in increasing order of their start times. If this does not improve any solutions, the process is terminated. For the complete details of this algorithm, we refer to Van der Beek et al. (2023).

4.3. Progressive hedging

We now present the structure of the PH algorithm. This algorithm uses the, in PH literature called, **Lagrangian multiplier** $\vec{w}_\omega$ for each scenario $\omega \in \Omega$ and **penalty ratio** r, to alter the objective functions of the subproblems to enforce implementability. Initially, r is set to its initialization value $r^{init} > 0$ and each $\vec{w}_\omega$ is set to a zero vector with the same length as $\vec{x}_\omega$. Furthermore, the PH algorithm initializes a compressed solution $\vec{x}_\omega$ for each scenario $\omega \in \Omega$, obtained by optimizing the subproblem solution vector with the HDE algorithm and converting to the compressed version. This solution maximizes the **unaltered objective function**, as given by

$$obj(\vec{x}_\omega) = \sum_{i \in N_\omega} \sum_{t \in T} p_{it} X_{it} - \sum_{r \in R^n} c_r Y_r. \tag{14}$$

After this initialization, the iterative process starts. In each iteration, the penalization of each compressed solution vector $\vec{x}_\omega$ for scenario $\omega \in \Omega$ is done by comparing $\vec{x}_\omega$ to the average solution vector $\bar{\vec{x}}_\omega$ over all scenarios. An implementable set of solutions correlates to $\vec{x}_\omega = \bar{\vec{x}}_\omega$ for every scenario $\omega \in \Omega$: each decision value is equal across all scenarios. To calculate $\bar{\vec{x}}_\omega$, we note that each scenario $\omega \in \Omega$ consists of multiple scenario tree nodes Ψ_ω. Since $\vec{x}_\omega$ consists of the resource inventory

vector (set in the root scenario tree node) and activity selection entries for selectable activities (related to projects in non-root nodes), each of the $|R^n| + |N^s|$ entries of $\vec{x}_\omega$ can be mapped surjectively to a scenario tree node. With this, we can create $\bar{\vec{x}}_\omega$: for each entry in $\vec{x}_\omega$ that maps to scenario tree node $\psi \in \Psi_\omega$, we average the corresponding entries in $\vec{x}_{\omega'}$ for all scenarios $\omega' \in \Omega_\psi$ that also contain scenario tree node ψ. To formalize this, we introduce the **solution indicator vector**

$$U(\omega, \omega', \psi) = \left[U(\omega, \omega', \psi)_1, \dots, U(\omega, \omega', \psi)_{|R^n|+|N^s|}\right] \tag{15}$$

for every combination of $\omega \in \Omega$, $\omega' \in \Omega$ and $\psi \in \Psi_\omega$. If node $\psi \notin \Psi_{\omega'}$, we get $U(\omega, \omega', \psi) = \vec{0}$. Otherwise, if $\psi \in \Psi_{\omega'}$, we let the jth entry of $U(\omega, \omega', \psi)$ be 1 if the jth variable in $\vec{x}_\omega$ corresponds to a selectable activity in scenario tree node ψ, and zero otherwise. With this, we can express $\bar{\vec{x}}_\omega$ as

$$\bar{\vec{x}}_\omega = \sum_{\psi \in \Psi_\omega} \sum_{\omega' \in \Omega_\psi} \frac{1}{|\Omega_\psi|} U(\omega, \omega', \psi)^T \cdot \vec{x}_{\omega'}. \tag{16}$$

To steer the solution set to be implementable, we first modify the Lagrangian vector $\vec{w}_\omega$, by adding the penalty ratio times the difference between current solution $\vec{x}_\omega$ and average solution $\bar{\vec{x}}_\omega$: $\vec{w}_\omega \leftarrow \vec{w}_\omega + r(\vec{x}_\omega - \bar{\vec{x}}_\omega)$. If an entry of $\vec{x}_\omega$ is larger than the average solution vector $\bar{\vec{x}}_\omega$, the corresponding entry in $\vec{w}_\omega$ is increased. Conversely, entries lower than the average result in a decreased entry in the Lagrangian vector $\vec{w}_\omega$. This is then used, together with r and $\bar{\vec{x}}_\omega$, to alter the objective function. In the standard PH implementation, the **altered objective function** is given by:

$$alt\_obj\left(\vec{x}_\omega, \bar{\vec{x}}_\omega, \vec{w}_\omega, r\right) = obj(\vec{x}_\omega) - \vec{w}_\omega^T \vec{x}_\omega - r\|\vec{x}_\omega - \bar{\vec{x}}_\omega\|^2, \tag{17}$$

where the last two terms are penalization terms. The first penalization term is based on the Langrangian vector $\vec{w}_\omega$. Recall that positive entries of $\vec{w}_\omega$ are the result of larger than average values in $\vec{x}_\omega$, since we start with $\vec{w}_\omega$ equal to zero and increase $\vec{w}_\omega$ when $\vec{x}_\omega$ is larger than average. Therefore, this term penalizes high values if the solution from the previous iteration (or initialization) was above the average solution vector. Similarly, negative values of $\vec{w}_\omega$ correspond to entries in $\vec{x}_\omega$ that are smaller than average. Therefore, this term guides the solution towards the average value. The second penalty term penalizes the squared distance between $\vec{x}_\omega$ and $\bar{\vec{x}}_\omega$, times the penalty ratio r. This has a similar effect, penalizing any deviation from the average. At the end of each iteration, the penalty ratio is multiplied by the **Penalty Multiplicator** parameter $PM > 1$. This technique is taken from Crainic et al. (2011). The iterative process continues until the solutions are implementable, or until a maximum number of iterations is reached.

However, $\vec{x}_\omega$ represents only the resource inventory level and activity selection decisions. Therefore, after terminating the iterative process, the resulting solution still needs to be converted to a schedule. First, if $\vec{x}$ is not implementable, the repair function repair($\vec{x}$) converts it to an implementable solution. This is done by setting each resource inventory entry to its maximum value across all scenarios. Second, the activity selection entries are rounded sequentially. After rounding an activity selection variable, all other activities in the same selection group are fixed to prevent selection infeasibilities. This is done by using *group orderings*, with the definition, generation procedure, and use described in Van der Beek et al. (2023).

After the repair process, an implementable schedule is created for multi-project $\mathcal{P}_\omega$ for each scenario $\omega \in \Omega$. For this, we loop over all scenarios $\omega \in \Omega$ and, subsequently, over its scenario tree nodes $\psi \in \Psi_\omega$. For each scenario tree node, we denote all preceding projects in the same scenario by Ψ_{prec}. If we did not yet compute a schedule for project $\mathcal{P}_\psi$, we run the HDE algorithm for this project, while fixing all variables from the preceding projects. Then, finally, we combine all solutions X_ψ from all scenario tree nodes $\psi \in \Psi_\omega$ to get scenario solution X_ω.

This gives the PH algorithm, as given in Algorithm 1, that is guaranteed to create feasible schedules for the RCPSP-MP, if one exists. In the remainder of this section, we present improvement techniques for this algorithm.

Algorithm 1 Progressive hedging algorithm.

$r \leftarrow r^{init}$ ▷ Initialization
for $\omega \in \Omega$ **do**
  $\vec{w}_\omega \leftarrow \vec{0}$
  $\vec{x}_\omega \leftarrow \arg\max_{\vec{x}_\omega \in \vec{X}_\omega} obj(\vec{x}_\omega)$
end for

while termination criterion not met **do** ▷ Iterative improvements
  for $\omega \in \Omega$ **do**
    $\bar{\vec{x}}_\omega \leftarrow \sum_{\psi \in \Psi_\omega} \sum_{\omega' \in \Omega_\psi} \frac{1}{|\Omega_\psi|} U(\omega, \omega', \psi) \odot \vec{x}_{\omega'}$
    $\vec{w}_\omega \leftarrow \vec{w}_\omega + r(\vec{x}_\omega - \bar{\vec{x}}_\omega)$
    $\vec{x}_\omega \leftarrow \arg\max_{\vec{x}_\omega \in \vec{X}_\omega} alt\_obj(\vec{x}_\omega, \bar{\vec{x}}_\omega, \vec{w}_\omega, r)$
  end for
  $r \leftarrow r \cdot PM$
end while

if $\vec{x}$ is not implementable **then** ▷ Make solution implementable
  $\vec{x} \leftarrow$repair($\vec{x}$)
end if

$X_\psi \leftarrow \emptyset, \quad \forall \psi \in \Psi$ ▷ Create complete solution
for $\omega \in \Omega$ **do**
  for $\psi \in \Psi_\omega$ **do**
    if $X_\psi = \emptyset$ **then**
      $\Psi_{prec} \leftarrow \{\psi' \in \Psi_\omega | \psi' < \psi\}$
      $X_\psi \leftarrow$ Run HDE for $\mathcal{P}_\psi$, with choices for projects $\mathcal{P}_{\psi'}$ for $\psi' \in \Psi_{prec}$ fixed by $X_{\psi'}$
    end if
  end for
  $X_\omega \leftarrow$Combine X_ψ for $\psi \in \Psi_\omega$
end for

4.4. Extensions

In order to improve the performance of the PH algorithm, two improvement techniques are presented here. The first one is a new technique and the second one is based on adaptations from improvement techniques in the literature.

The first improvement technique introduced is **Overshoot limitation**, which is one of the contributions of this paper. As shown in Eq. (17), the Lagrangian penalty is $\vec{w}_\omega^T \vec{x}_\omega$. Consider entry $x_i \in \vec{x}_\omega$ that has been higher than average for some iterations, such that the corresponding entry in $\vec{w}_\omega$ has become positive. This gives an incentive for x_i to decrease. However, especially with the resource inventory level variables that can take on a range of values, this can result in overshoot: there is an incentive to be as low as possible, not just to be close to the average value. Therefore, we replace $\vec{x}_\omega$ by $\vec{x}'_\omega$ in the second term of Eq. (17), with for each entry $x'_i \in \vec{x}'_\omega$:

$$x'_i = \begin{cases} \bar{x}_i & \text{if } (w_i > 0 \text{ and } x_i < \bar{x}_i) \text{ or } (w_i < 0 \text{ and } x_i > \bar{x}_i), \\ x_i & \text{otherwise,} \end{cases} \tag{18}$$

using corresponding entries $x_i \in \vec{x}_\omega$, $\bar{x}_i \in \bar{\vec{x}}_\omega$ and $w_i \in \vec{w}_\omega$. Thus, entries of $\vec{x}_\omega$ for which $w_i > 0$ have no incentive to become as small as possible, since the penalty given by $w_i x'_i$ is equal for all values of x_i below $\bar{x}_i$ (and vice verse for entries with $w_i < 0$). This means that $\vec{x}'_\omega$ is capped at the average value to prevent overshoot. This gives us the altered cost when overshoot limitation is used:

$$alt\_obj\left(\vec{x}_\omega, \bar{\vec{x}}_\omega, \vec{w}_\omega, r\right) = obj(\vec{x}_\omega) - \vec{w}_\omega^T \vec{x}'_\omega - r\|\vec{x}_\omega - \bar{\vec{x}}_\omega\|^2. \tag{19}$$

Second, we introduce **Variable bounding**, based on the **variable fixing** technique from Watson and Woodruff (2011). Here, in each iteration of the PH algorithm, for variable x_i, an upper bound for future iterations is set on the largest value for this variable, across all scenarios. Similarly, a lower bound is set by the lowest value of x_i across all

scenarios. Note that since variable x_i is binary, an upper bound of 0 or a lower bound of 1 results in variable fixing. Furthermore, we introduce the lag parameters LR and LS, defining the lag in variable bounding for resource inventory level and activity selection variables, respectively. A lag means that a variable is only bounded, after it has satisfied a bound for at least that many iterations.

5. Computational study

This section presents the computational study. First, the instance generation methods and instance sets are described. After this, the computational results are given. The full instances and computational results are given in Van der Beek (2022).

5.1. Problem instances

In this subsection, we present the instances used in the computational study. This is done by first describing the structure of the instances. Secondly, the instance sets are described. The structure of the instances represents choices that have to be made in modular shipbuilding.

Every instance consists of a scenario tree. Each node in this tree, except for the root node, corresponds to a project. A project is created by using a **base project** and replacing multiple activities by a **Module Option** (MO). An MO is a set of activities for which a module can be used. Each MO consists of four alternative subnetworks: The **Module Alternative** (MA), the **Direct Construction Alternative** (DCA), the **Outsource Alternative** (OA) and the **Reconstruction Alternative** (RA). The first two represent the process of installing a module: the MA represents the use of pre-assembled module components and the DCA represents building the module directly, without the use of module components. The latter two represent the replenishment processes of the required module components: In the OA, the module components are ordered from a third party and the RA reconstructs the components locally.

For each module option, we randomly select an activity $i \in N \setminus \{0, |N|\}$ in the base project. This activity is then replaced by a selection group, with a dummy activity (zero duration) as activator activity and the first activities of the MA and DCA as successors. Additionally, the final activities of both alternatives are linked to the original successors of activity i. Thus, activity i is replaced by two subnetworks, MA and DCA, of which exactly one has to be executed. Since the MA represents the use of a pre-assembled module, it requires the use of nonrenewable resources that represent the module component(s). Therefore, if the MA is executed, it triggers one of the replenishments networks. These replenishment networks, OA and RA, produce the same number of nonrenewable resources as their corresponding MA. Since ordering or reconstructing can be done as soon as the schedule is decided upon, the OA and RA only have precedence relationships to the starting activity of the base project.

After creating a project for each scenario tree node $\psi \in \Psi$, these projects are combined sequentially to create multi-project $\mathcal{P}^m_\omega$, by creating finish to start precedence relationships between the finish and start nodes of the respective projects. Then, all multi-projects are combined to create the scenario tree. Although there is generally some level of overlap between projects, we model them as sequentially to focus on the modularization decisions. Since the OA and RA do not have a precedence relationship with the final activity, they can interfere with future projects.

Thus, the procedure explained above creates a single instance, consisting of scenarios of multiple sequentially linked projects. In Fig. 3, an example of a single project is shown. Here, the base network consists of five activities. From this, we replace activity 1 by a module option. It can be seen that this creates 4 alternatives: DCA, MA, RA and OA. The DCA and MA replace activity 1, while the RA and OA are placed after the root activity in the precedence graph and after the MA in the selection graph.

The method described above is used to create two instance sets: *Exact* and *Heuristic-only*. The former has 209 instances that are small enough such that feasible solutions can be found efficiently by solving the ILP given in Constraint set (3). The goal of this instance set is to evaluate the performance of using a commercial solver to solve the ILP formulation. The latter instance set, *Heuristic-only*, contains 206 larger instances for which Constraint set (3) cannot be used to find feasible solutions, since the commercial solver usually will not find a feasible solution within the allowed computing time. The goal of this instance set is to compare the various variants of the PH algorithm.

5.2. Computational results

This subsection presents the computational results. These are divided into two parts: the results for the *Exact* instance set and the results for the *Heuristic-only* instance set. The instance set *Exact* contains smaller instances, for which the ILP model in Constraint set (3) is used to find feasible (but not necessarily optimal) solutions. The goal of these instances is to evaluate the ILP model against the PH algorithm and its extensions. The instance set *Heuristic-only* contains instances with more scenarios. These instances are only solved with the PH algorithms, with the goal of evaluating the extensions. All tests are performed on a single core of a 3.0 GHz Intel XEON CPU with 4 GB RAM.

The algorithm parameters were decided upon by creating a small set of instances and running a local search based algorithm that iteratively varies one parameter. The parameters for the HDE algorithm for the subproblems are determined by running the HDE algorithm on subproblems separately. Then, with this, the parameters for the PH algorithm are decided upon by iteratively varying one parameter using a local search based algorithm and fixing it to the best encountered value. This resulted in the parameters $\gamma = 100$, $SC = 0.1$, $RP = 0.1$, $TP = 100$, $r^{init} = 1.6$, $PM = 1.001$, $LR = 8$ and $LS = 7$.

During these computational tests, the following methods are used:

1. **ILP Method** (IM): Running the Gurobi optimizer on the ILP model in Constraint set (3) for a fixed amount of time and returning the best found feasible solution.
2. **Basic Progressive Hedging** (BPH): The basic PH algorithm, as presented in Section 4.3, without any extensions.
3. **Bounded Method** (BM): The BPH with the *variable bounding* extension from Section 4.4.
4. **Overshoot-limitation Method** (OM): The BPH with the *overshoot limitation* extension from Section 4.4
5. **Combined Method** (CM): The BPH with both the *overshoot limitation* and the *variable bounding* extensions.

Due to the repair step in the PH algorithm, methods 2 to 5 always create implementable solutions. The IM, however, does not have this guarantee. Based on time limitations for the computational study, the maximal computing time for the IM is set to 18 h and the maximum number of iterations for the PH algorithms is set to 250.

Furthermore, to compare methods, instances need to be normalized in some way to put equal emphasis on smaller and larger instances. This is done by introducing the **Bound Optimality Gap** (BOG), which is an upper bound on the optimality gap of the respective solution:

$$BOG(obj) = \frac{obj - obj^*}{obj^*} \cdot 100\%. \tag{20}$$

Here, obj is current objective value, and obj^* the best known objective value for that instance.

5.2.1. Exact

First, the instance set *Exact* is evaluated. This set contains instances with only a few scenarios, such that feasible solutions can be found by the ILP method. Table 3 shows a summary of the tests results. Here, it can be seen that for nearly half of the instances, a feasible solution

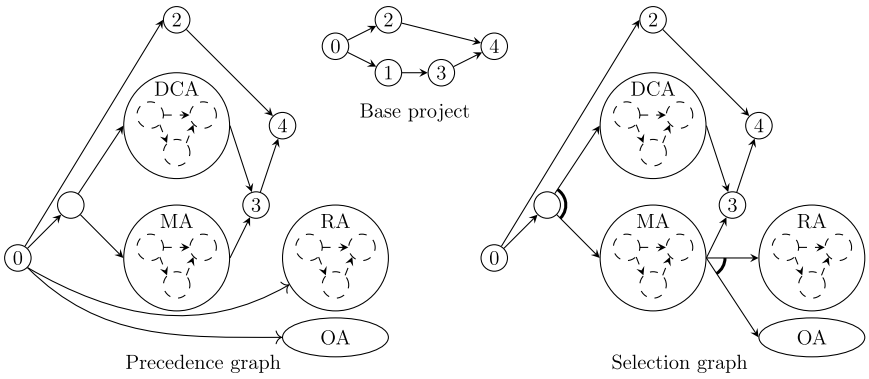
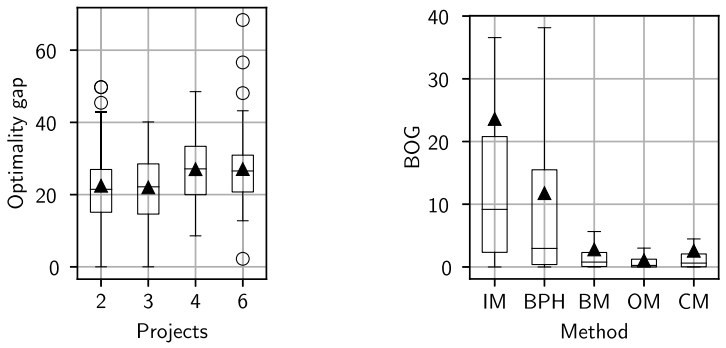


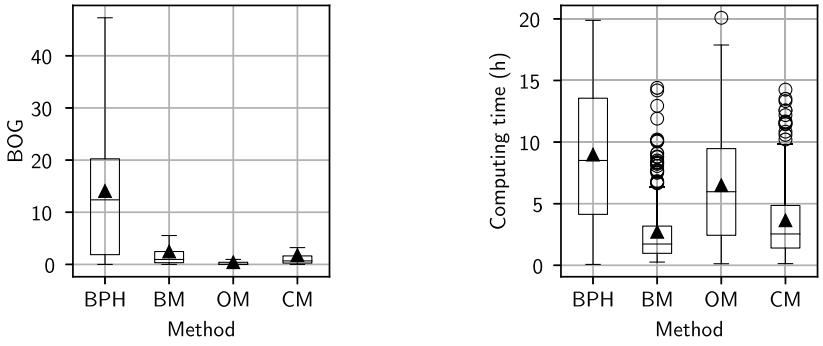
Fig. 3. Insertion of a single module option in a base project.



(a) OG per number of projects in scenario.

(b) BOG per method (without outliers).

Fig. 4. Results for instance set *Exact*.



(a) BOG per method (without outliers).

(b) Computing times.

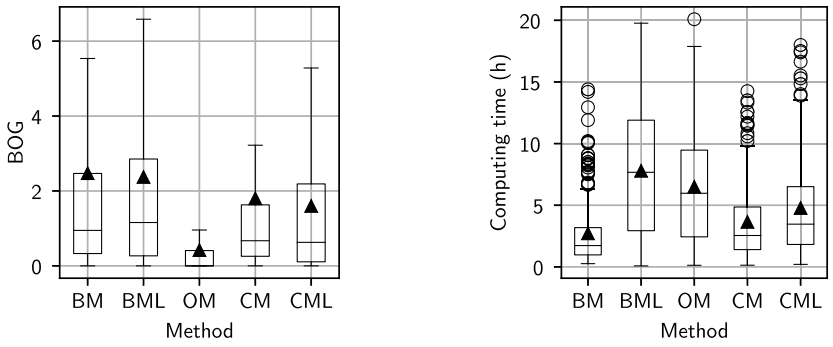
Fig. 5. Results for all PH methods for instance set *Heuristic-only*.

is found. However, nearly none of these solutions were proven to be optimal. Consequently, nearly all tests were terminated due to the time limit, which results in the average computing time being close to this time limit. Furthermore, the optimality gaps (OG) are evaluated. This is defined as

$$OG = \frac{lb - obj^*}{obj^*} \cdot 100\%, \tag{21}$$

where lb is the highest lower bound found by the ILP-solver and obj^* is the best found feasible solution, either from the ILP-solver or from the PH heuristics. Using both the feasible solution from the ILP-solver and the PH methods is done to obtain a meaningful OG, even when no feasible solution is found by the ILP-solver. In Fig. 4a, the optimality gaps are given against the total number of projects in the scenario tree for each instance. Here, a slightly increasing trend can be seen. Furthermore, it can be seen that even for small instances, the optimality gap is relative large. Since the optimality gap includes the best found solution by the IM, it shows that there often remains a large gap between the ILP-based lower bound and the best found ILP solution. This shows the computational difficulty of solving the problem with just an ILP solver (IM), and therefore, the need for heuristic methods (BPH, BM, OM and CM).

Furthermore, a comparison of all instances for which feasible solutions are found for all methods, is given in Table 4. Here, it can be seen that both the mean BOG and the computing time is significantly better for all PH methods. In Fig. 4b, the BOGs are shown for each method. It can be seen that the BPH has a significant improvement compared to the IM. Furthermore, all extensions result in an even larger reduction of BOG. However, when evaluating the number of instances for



(a) BOG per method (without outliers). (b) Computing times.

Fig. 6. Results for PH methods, while including lag on variable bounding, for instance set *Heuristic-only*.

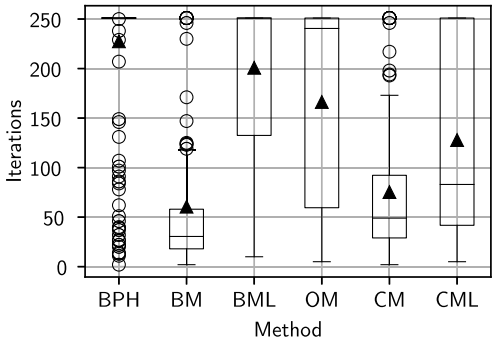


Fig. 7. Number of iterations per method for instance set *Heuristic-only*.

Table 3
Summary of tests for all 209 instances for instance set *Exact* using the ILP-solver.

Optimal solutions found	2
Feasible solutions found	97
Average computing time (h)	17.89
Average optimality gap	24.68
Standard deviation optimality gap	10.62

Table 4
Comparison for all methods for all 97 instances in the instance set *Exact*, where a feasible solution is found by the IM.

Method	IM	BPH	BM	OM	CM
Mean BOG	23.60	11.77	2.80	1.08	2.59
Standard deviation BOG	39.13	17.24	4.71	2.17	4.37
# Best feasible solutions	17	16	22	36	24
Mean computing time (h)	17.76	4.33	0.35	2.04	0.49
Std. deviation computing time (h)	1.86	3.98	0.59	1.91	0.76

which a method reaches the best found solution (including ties), the BPH performs slightly worse than the IM. Here, it can be seen that the improvements are required to outperform the IM. Evaluating only the performance and not the computing time, it can be seen that the OM performs best.

5.2.2. Heuristic-only

Next, the instance set *Heuristic-only* is evaluated, in order to gain insight in all PH methods. The summary of these results is shown in Table 5. Furthermore, Fig. 5 shows the BOGs and computing times per method. It can be seen that all extensions result in a significant improvement of the basic method in all metrics. When evaluating the BOG and number of best solutions found, it can be seen that the OM performs best. However, this method also takes significantly longer than the other improvement methods, with the BM having the lowest computing time. The CM, which combines the BM and OM, has its computing times and BOGs between these two methods. A possible explanation for the higher computing time, compared to the BM, is that the overshoot limitation dampens the penalty terms in the PH algorithm. This results in slower converging bounds, and thus, in an increased computing time.

Table 5
Summary of results for all 206 instances of instance set *Heuristic-only*.

Method	BPH	BM	OM	CM
BOG mean	14.1	2.5	0.4	1.8
BOG standard deviation	13.4	3.9	0.8	2.9
Mean computing time (h)	9.0	2.7	6.5	3.7
Std. deviation computing time (h)	5.6	2.7	4.6	3.1
# Implementable without repair	30	193	109	194
# Best solution	13	27	109	27

Additionally, Table 5 shows the number of solutions that were implementable without the repair step in Algorithm 1. It can be seen that the BPH usually did not reach implementability within the allowed number of iterations. The relatively large number of implementable solutions for the BM and the CM indicate that the use of variable bounding significantly improves the convergence of the PH algorithm. However, even though the OM required more repairs, it has significantly more best solutions found. Upon further inspection of the instances where OM reached the best solution found, 35 of these instances had to be repaired for the OM, while having at least one other method where reparation was not needed. This indicates that even without reaching implementability directly, it is still possible to reach good solutions. Note that the number of implementable solutions without repair and number of best solutions found being equal for OM is thus a coincidence.

Furthermore, we evaluate the use of lag in variable bounding for the BM and the CM. To indicate the variations including lag of these methods, the suffic 'L' is used. Fig. 6 shows the BOGs and computing times for all methods. It can be seen that introducing lag reduces the mean BOG and increases the computing time. For the BM, introducing lag results in computing times larger than the OM, while not having better BOG-based performance. For the CM, introducing lag results in a lower mean BOG. Compared to the OM, the CML has lower computing times, but also higher BOGs.

Finally, Fig. 7 shows the number of iterations per method. It can be seen that the BPH usually does not reach implementability and relies heavily on the repair function. Furthermore, it can be seen that introducing variable bounding significantly decreases the number of iterations, while introducing lag increases this. The relatively large number of iterations needed for the BML compared to the combined lag method, might be explained by the fact that the bounded method relies heavily

on occasional dips and peaks in variables. Introducing lag limits the effect of this. For the combined method, due to the dampening effect of the overshoot limitation, bounds are formed more gradually and are less affected by the lag.

6. Conclusion

In this paper, the RCPSP-MP is introduced to model a stochastic modular production problem. This was done by modifying the RCPSP-PS/CPR by adapting the objective function, introducing resource inventory level variables for nonrenewable resources, and by converting to a scenario-tree based formulation. This model then determines the resource inventory levels, activity selection, and activity scheduling decisions, while optimizing the mean of the objective functions over all scenarios.

An ILP formulation is given for the deterministic equivalent of the RCPSP-MP. Although this formulation can theoretically be solved to optimality by branch-and-bound solvers, the RCPSP-MP consists of multiple instances of the RCPSP-PS/CPR, which are by itself NP-hard. Therefore, as is shown in the computational results, the ILP model is rarely solved to optimality, even for small instances. Thus, a PH algorithm is introduced that converges to an implementable solution for the resource inventory level and activity selection decisions. After these are converged, the activity scheduling decisions are determined. Furthermore, two extensions are introduced to accelerate convergence: variable bounding and overshoot limitation.

In the computational results, it is shown that all variations of the PH algorithm outperform the ILP-based solution method. However, the basic algorithm relies heavily on the repair step at the end of the algorithm and does not usually converge to an implementable solution without this. The improvement techniques significantly improve the convergence rate, while creating better solutions in less time. In terms of solution quality, overshoot limitation performs best. Adding variable bounding to this decreases the computing time, but also the solution quality. Introducing lag on variable bounding somewhat improves the solution quality in cost of computing time, although the solution quality does not improve beyond the solutions found by only using overshoot limitation.

The PH algorithm converges on resource inventory level and activity selection decisions, while deciding on the scheduling decisions afterwards. Although this assumption generally aligns with the practice of modular production, it also means that the computational results have to be viewed in the context of the generated instances, which have the assumption that scheduling decisions have less influence on future projects than resource inventory level and activity selection problems. One direction of future research would be to relax this assumption and evaluate the performance.

Another direction for future research would be to test the algorithm on real-life data. This will also provide managerial insights on which and how many modules should be kept in storage, how many should be outsourced, and how many should be produced.

CRediT authorship contribution statement

T. Van Der Beek: Writing – review & editing, Writing – original draft, Methodology; **J. T. Van Essen:** Writing – review & editing, Validation, Supervision, Methodology; **J. Pruyn:** Writing – review & editing, Supervision, Funding acquisition; **K. Aardal:** Writing – review & editing.

Declaration of competing interest

The project has received funding from the European Union's Horizon 2020 research and innovation programme (Contract No.: 769419)

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).

Appendix A. Instances

This section describes the instances used in this paper. First, in Section A.1, the instance generation method is presented. Secondly, Section A.2 describes the instance sets created by this method.

A.1. Instance generation

This subsection presents the instance generation method. Each instance is generated by creating a scenario tree defined by the $Shape$ vector. This vector defines, for each level in the scenario tree, the number of child nodes. For example, the instance shown in Fig. 1 has a shape factor of $[2, 2]$, since the root node splits into two child nodes, which in turn both split into two child nodes as well. After creating the scenario tree, a project is created for each non-root node. Each of these projects is created by using a *base project* and replacing activities by *Module Options* (MO). The base project is an RCPSP instance, created by the instance generation algorithm from Demeulemeester et al. (2003). The algorithm requires the following input parameters: The number of activities $|N|$, the **Serial/Parallel** (SP) indicator, the **Resource Factor** (RF) and the **Resource Constrainedness** (RC). The SP indicator is a measure of the shape of the network, where a value of zero indicates a project consisting of all activities in series and a value of one indicates a project where all activities can be executed in parallel (disregarding resource constraints). The RF reflects the average portion of *resource types* per activity, and the RC represents the *average amount of each resource* requested. For the exact formulation of these parameters, we refer to Demeulemeester et al. (2003). The values of these parameters are selected randomly from the values shown in Table A.1.

Table A.1
Parameters options for project sets.

	Base projects	Module projects
SP	[0.2, 0.4, 0.6, 0.8]	
RC	[0.5, 0.7, 0.9]	
RF	[0.25, 0.5, 0.75]	
$\|R^r\|$	4	
$\|N\|$	7-9	12-17

Subsequently, NM activities of this base project are replaced by MOs. These activities are randomly sampled from all activities, except the start and finish activity. The *Direct Construction Alternative* (DCA) is created by the instance generation algorithm from Demeulemeester et al. (2003), with the parameters sampled random from the values shown in Table A.1. The *Module Alternative* (MA) and the *Reconstruct Alternative* (RA) are created by scaling the durations of the DCA subproject to match a *target duration* (TD). For the MA, this target total duration is set to $TD \cdot TD^{MA}$, where $0 < TD^{MA} < 1$ is an input parameter. Similarly, the target total duration for the RA is $TD \cdot TD^{RA}$, where $0 < TD^{RA} < 1$ is an input parameter. Furthermore, the *Outsource Alternative* (OA) consists of a single activity with (rounded) duration chosen uniformly from the interval $TD \cdot (TD^{OA} \pm VAR^{OA})$. Here $TD^{OA} > 0$ and $VAR^{OA} > 0$ are input parameters.

Next, we describe the resource and cost structure of each project. For each module option, the amount of nonrenewable resource $r \in R^n$ used is chosen randomly in the interval $\{1, \dots, RU\}$, where RU is the resource usage parameter. The profit obtained for finishing a project at time t, is set equal to $-t$, meaning that the profit decreases constantly with an increase in makespan. The cost of each unit of nonrenewable resource $r \in R^n$ is chosen uniformly from the interval $(1 \pm VAR^{rc}) \cdot RC$, where VAR^{RC} and RC are input parameters. Finally, to determine the cost of executing the RA, the sum of the inventory costs of all nonrenewable

Table A.2
Parameter options for instance sets.

	Exact	Heuristic-only
TD^{MA}	[0.25, 0.5, 0.75]	
TD^{RA}	[0.25, 0.5, 0.75]	
TD^{OA}	[1, 1.5, 2]	
VAR^{OA}	0.25	
RU	5	
RC	$[1, 2, 3]$	
VAR^{RC}	0.2	
OC	[0.25, 0.5, 0.75]	
VAR^{OC}	0.2	
PT	[0, 5]	
ST	1.25	
VAR^{ST}	0.25	
NM	random from $[2, 3, 4]$	
$\|R^r\|$	4	
Shape	[[2, 1], [2, 2], [3, 1], [3, 2]]	[[2, 2, 1], [2, 3, 1], [3, 2, 1], [3, 3]]
$\|N\|$	random from 7-9	random from 12-17

resources required for the module option is defined as TC. Then, the cost for executing the RA is taken uniformly from the interval $TC \cdot (OC \pm VAR^{OC})$, where OC is the outsourcing cost parameter and VAR^{OC} its corresponding variance.

After creating a project for each non-root scenario tree node $\psi \in \Psi$, these projects are combined sequentially to create multi-project $\mathcal{P}^m_\omega$ for each scenario $\omega \in \Omega$, by creating finish to start precedence relationships between the finish and start nodes of the respective projects. Then, all multi-projects are combined to create the scenario tree. The earliest start time of each project, τ, is taken randomly from the interval $(ST \pm VAR^{ST}) \cdot critical\_path$, where $critical\_path$ is the critical path length of all preceding projects. Additionally, ST and VAR^{ST} are the starting time and starting time variance parameters, respectively. Furthermore, preparation time in between projects might be needed. Therefore, the duration of the starting activity of each project is set to a duration taken randomly from $[0, \dots, PT]$, where PT is the Preparation Time parameter.

A.2. Instance sets

The method described in Appendix A.1 is used to create two instance sets: *Exact* and *Heuristic-only*. The former is used to evaluate the performance of the ILP formulation given in Constraint set (3) and the latter is used to compare the various variants of the PH algorithm. In Table A.2, the parameters per set are given. We define each possible combination of parameters, except for the parameters NM, RC, OC and $|N|$, as an *instance combination*. Then, for each instance combination, NM and $|N|$ are selected randomly ($|N|$ is sampled for each individual base project) and an instance is created for each combination of RC and OC. This creates multiple instances. For each of these instances, the HDE algorithm is ran once per scenario, resulting in a resource inventory level for each scenario. Subsequently, from all instances that are created for one instance combination, the instance is selected with the highest standard deviation of resource inventory levels across all scenarios. This is done to assure computational challenging instances. If the resource inventory levels are zero for all scenarios, or if the MA is selected for each MO, the instance is considered not computationally challenging. Therefore, the instance generation algorithm restarts, and if no computationally challenging instance is created after 10 restarts, no instance is created. This happened for 7 combinations for the *Exact* instance set, resulting in $216 - 7 = 209$ instances (all combinations of parameters, except NM, $|N|$, RC, and OC). For the *Heuristic-only* set, each combination resulted in a feasible challenge, presumably due to the larger number of projects in each instance. Therefore, the *Heuristic-only* instance set contains 216 instances. All notation introduced in this section is listed in Table A.3.

Table A.3
Notation for Appendix A.

OC	Outsourcing cost parameter.
PT	Preparation time parameter.
RC	Resource constrainedness.
RU	Resource usage parameter.
ST	Starting time parameter.
TD	Target duration.
TD^{MA}	Module alternative duration parameter.
TD^{OA}	Outsource alternative duration parameter.
TD^{RA}	Reconstruct alternative duration parameter.
VAR^{OA}	Outsource alternative variation parameter.
VAR^{OC}	Outsourcing cost variation parameter.
VAR^{RC}	Resource constrainedness variation parameter.
VAR^{ST}	Starting time variation parameter.

References

Athanasia, A., Anne-Bénédicte, G., & Jacopo, M. (2012). The offshore wind market deployment: Forecasts for 2020, 2030 and impacts on the European supply chain development. *Energy Procedia*, *24*, 2–10.

Bakker, H., Dunke, F., & Nickel, S. (2020). A structuring review on multi-stage optimization under uncertainty: Aligning concepts from theory and practice. *Omega (United Kingdom)*, *96*, 102080.

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

Capa, C., & Ulusoy, G. (2015). Proactive project scheduling in an R&D department a bi-objective genetic algorithm. In *IEOM 2015 - 5th international conference on industrial engineering and operations management, proceeding* (pp. 1–6).

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

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

Carøe, C. C., & Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, *24*, 37–45.

Crainic, T. G., Fu, X., Gendreau, M., Rei, W., & Wallace, S. W. (2011). Progressive hedging-based metaheuristics for stochastic network design. in: *Networks* (pp. 114–124)

Dantzig, G. B. (1955). *Linear programming under uncertainty. Management science*, 1, 197–206.

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). Rangen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, *6*, 17–38.

Frigant, V., & Lung, Y. (2002). Geographical proximity and supplying relationships in modular production. *International Journal of Urban and Regional Research*, *26*, 742–755.

Guo, G., Hackebeil, G., Ryan, S. M., Watson, J. P., & Woodruff, D. L. (2015). Integration of progressive hedging and dual decomposition in stochastic integer programs. *Operations Research Letters*, *43*, 311–316.

Hasannia Kolaee, M., Mirzapour Al-e-Hashem, S.M.J. (2022). Stochastic medical tourism problem with variable residence time considering gravity function. *RAIRO - Operations Research*, *56*, 1685–1716.

Haugen, K. K., Løkketangen, A., & Woodruff, D. L. (2001). Progressive hedging as a metaheuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, *132*, 116–122.

Jiang, X., Bai, R., Wallace, S. W., Kendall, G., & Landa-Silva, D. (2021). Soft clustering-based scenario bundling for a progressive hedging heuristic in stochastic service network design. *Computers and Operations Research*, *128*, 105182.

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

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

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

Lamghari, A., & Dimitrakopoulos, R. (2016). Progressive hedging applied as a metaheuristic to schedule production in open-pit mines accounting for reserve uncertainty. *European Journal of Operational Research*, *253*, 843–855.

Løkketangen, A., & Woodruff, D. L. (1996). Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. *Journal of Heuristics*, *2*, 111–128.

Melchiors, P., Leus, R., Creemers, S., & Kolisch, R. (2018). Dynamic order acceptance and capacity planning in a stochastic multi-project environment with a bottleneck resource. *International Journal of Production Research*, *56*, 459–475.

Pamay, M. B., Bülbül, K., & Ulusoy, G. (2014). Dynamic resource constrained multi-project scheduling problem with weighted earliness/tardiness costs. *International Series in Operations Research and Management Science*, *200*, 219–247.

Pandremenos, J., Paralikas, J., Salonitis, K., & Chryssolouris, G. (2009). Modularity concepts for the automotive industry: A critical review. *CIRP Journal of Manufacturing Science and Technology*, *1*, 148–152.

Peng, Z., Zhang, Y., Feng, Y., Rong, G., & Su, H. (2019). A progressive hedging-based solution approach for integrated planning and scheduling problems under demand uncertainty. *Industrial and Engineering Chemistry Research*, *58*, 14880–14896.

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.
Rockafellar, R. T., & Wets, R. J. B. (1991). Scenarios and policy aggregation in optimization under uncertainty, *16*, 119–147.
Sako, M., & Said, F. M. (1999). Modules in design , production and use : Implications for the global automotive industry 1. In *International motor vehicle program (IMVP) annual sponsors meetin*.
Satic, U., Jacko, P., & Kirkbride, C. (2022). Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem. *International Journal of Production Research*, *60*, 1411–1423.
Satic, U., Jacko, P., & Kirkbride, C. (2024). A simulation-based approximate dynamic programming approach to dynamic and stochastic resource-constrained multi-project scheduling problem. *European Journal of Operational Research*, *315*, 454–469.
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.
Ship&Offshore (2024). Damen to build support vessels for stock. https://www.shipandoffshore.net/news/shipbuilding/detail/news/damen-to-build-support-vessels-for-stock.html.
Shirzadeh Chaleshtarti, A., Shadrokh, S., Khakifirooz, M., Fathi, M., & Pardalos, P. M. (2020). A hybrid genetic and lagrangian relaxation algorithm for resource-constrained project scheduling under nonrenewable resources. *Applied Soft Computing Journal*, *94*, 106482.
Storch, R. L., & Sukapanpotharam, S. (2003). Common generic block: Mass customization for shipbuilding. *Proceedings of the Institution of Mechanical Engineers Part M: Journal of Engineering for the Maritime Environment*, *217*, 79–94.
Tao, S., & Dong, Z. S. (2017). Scheduling resource-constrained project problem with alternative activity chains. *Computers and Industrial Engineering*, *114*, 288–296.
Tao, S., & Dong, Z. S. (2018). Multi-mode resource-constrained project scheduling problem with alternative project structures. *Computers and Industrial Engineering*, *125*, 333–347.
Torres, J., Li, C., Apap, R., & Grossmann, I. (2019). A review on the performance of linear and mixed integer two-stage stochastic programming algorithms and software. *Optimization Online*, *15*, 1–30.
Valls, V., Ballestín, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *185*, 495–508.
Van der Beek, T. (2022). Instances and computational results for the resource constrained project scheduling problem with modular production. https://doi.org/10.4121/21774530
Van der Beek, T., Souravlias, D., Van Essen, J. T., Pruyn, J., & Aardal, K. (2023). Hybrid differential evolution algorithm for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources. *European Journal of Operational Research*, *313*, 92–111.
Van der Beek, T., van Essen, J., 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*, 56–69.
Van Slyke, R. M., & Wets, R. (1969). L -shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, *17*, 638–663.
Watson, J. P., & Woodruff, D. L. (2011). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, *8*, 355–370.
Zhu, F. W., Sun, X. X., Miller, J., & Deng, Z. J. (2014). Innovations in knowledge management: Applying modular design. *International Journal of Innovation*, *6*, 83–95.