Machine learning assisted Differential Evolution for the Dynamic Resource Constrained Multi-project Scheduling Problem with Static project Schedules

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

DOI
10.1016/j.ejor.2025.05.059

Publication date
2025

Document Version
Final published version

Published in
European Journal of Operational Research

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete optimization

Machine learning assisted Differential Evolution for the Dynamic Resource Constrained Multi-project Scheduling Problem with Static project Schedules

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
Stochastic optimization
Modular production
Machine learning

ABSTRACT

In large modular construction projects, such as shipbuilding, multiple similar projects arrive stochastically. At project arrival, a schedule has to be created, in which future modifications are difficult and/or undesirable. Since all projects use the same set of shared resources, current scheduling decisions influence future scheduling possibilities. To model this problem, we introduce the Dynamic Resource Constrained Multi-project Scheduling Problem with Static project Schedules. To find schedules, both a greedy approach and simulation-based approach with varying scenarios are introduced. Although the simulation-based approach schedules projects proactively, the computing times are long, even for small instances. Therefore, a method is introduced that learns from schedules obtained in the simulation-based method and uses a neural network to estimate the objective function value. It is shown that this method achieves a significant improvement in objective function value over the greedy algorithm, while only requiring a fraction of the computation time of the simulation-based method.

1. Introduction

Modular production is used in various industries to combine the benefits of product standardization with the ability to meet customer specific demands. This is usually done by defining a base product and optional modules, which can be selected to configure the product. In modular production for large construction products, such as shipbuilding (Agarwala, 2015), aircraft manufacturing (Buergin et al., 2018), or housing construction (Neelamkavil, 2009), this results in similar projects arriving sequentially. These projects have to be scheduled while satisfying resource and time constraints. A well studied problem in scheduling is the Resource Constrained Project Scheduling Problem (RCPSP). This problem consists of a set of activities that have to be scheduled, subject to resource and precedence constraints. The goal is to minimize the *makespan*: the total project duration.

At the moment of scheduling a project, there might be some indication about the arrival time of the next project. Furthermore, since large construction projects require communication and resource reservation across multiple stakeholders, modification of earlier made schedules can be undesired or even impossible. Therefore, it is desired to completely schedule a project, without postponed decisions or later modifications. Since all projects use the same set of shared resources, each schedule influences future scheduling capabilities.

To model these properties of project scheduling for modular production, the Dynamic Resource Constrained Multi-Project Scheduling Problem with Static project Schedules (DRCMPSP/SS) is introduced in this paper. This problem consists of a set of stages in which projects arrive sequentially. At each project arrival, we assume to have an estimate of the arrival time of the next project. As soon as a new project arrives, it has to be scheduled completely, without the possibility to reschedule. In reality, rescheduling is possible. However, by assuming the opposite, we find solutions that are robust against the uncertain arrival of future projects. Furthermore, the goal of the DRCMPSP/SS is to minimize the weighted average makespan of all projects.

As is discussed in Section 2, there are various studies on the dynamic arrival of new projects for the RCPSP. However, to the best of our knowledge, research on proactive resource constraint project scheduling where the uncertainty is in the complete structure of the arriving project and projects need to be completely scheduled immediately without later modifications, is limited. One possible reason for this might be that proactive scheduling of complete projects results in a computationally very expensive problem, for which finding good solutions takes very long.

The contribution of this paper is threefold. First, we formally introduce the DRCMPSP/SS. Secondly, we introduce a new solution

* Corresponding author.
E-mail addresses: T.vanderBeek@tudelft.nl (T.v.d. Beek), J.T.vanEssen@tudelft.nl (J.T.v. Essen), J.F.J.Pruyn@tudelft.nl (J. Pruyn), K.I.Aardal@tudelft.nl (K. Aardal).

https://doi.org/10.1016/j.ejor.2025.05.059
Received 22 June 2024; Accepted 30 May 2025
Available online 14 June 2025

representation that supports time gaps and present a simulation-based heuristic optimization algorithm. Finally, we propose a heuristic method based on objective function estimation by a neural network that is trained with data from the simulation-based method. These methods are compared against a greedy alternative: scheduling each project as well as possible without looking ahead.

In Section 2, we first give an overview of research related to the DRCMPSP/SS. Subsequently, we give a description of the problem in Section 3. Then, in Section 4, the solution methods are given. Finally, we present the results of the computational study in Section 5 and conclude the paper in Section 6.

2. Literature review

The RCPSP was introduced by Pritsker et al. (1969) and proven to be NP-hard by Blazewicz et al. (1983). It has been one of the most studied scheduling problems, which has resulted in many solution methods and variations. In this section, we first give a general introduction of the RCPSP under uncertainty, before presenting related research on the RCPSP with new project arrivals. Furthermore, since the methods in these papers do not seem suitable for the DRCMPSP/SS, we present research on estimating the objective function within heuristic algorithms.

Numerous researchers have studied versions of the RCPSP under uncertainty. Herroelen and Leus (2005) give an overview of different variants of the RCPSP without the assumption of complete information. They differentiate methods on how they react to disruptions or uncertainty. The first type is called predictive-reactive scheduling. Here, a baseline (or predictive) schedule is created before execution, and as soon as changes in the input are revealed this schedule is repaired or modified. The second type is dynamic scheduling, which does not change or repair a baseline schedule, but where a scheduling policy is decided upon that can deal with uncertainty.

As there are many different types of uncertainties for the RCPSP, we focus especially on the Resource Constrained Multi-Project Scheduling Problem (RCMPSP) with arriving projects. In this setting, baseline scheduling is usually done with the assumption that there is a penalty for modifying earlier defined schedules. This is done by Pamay et al. (2014), who present an RCMPSP problem with new project arrivals and weighted earliness and tardiness costs. At each project arrival time, a local search heuristic is used that minimizes the makespan of the new project plus the earliness and tardiness penalties for deviations of previously scheduled projects. A similar problem is investigated by Capa and Ulusoy (2015). They consider a problem that includes preemption, stochastic durations and new project arrivals, and use a genetic algorithm to minimize the makespan and the total sum of absolute deviations.

For the DRCMPSP/SS, modifications of earlier schedules are not allowed. Therefore, research on dynamic scheduling is presented here. Problems of this kind are usually modeled with **Markov Decision Processes** (MDP).

Choi et al. (2007) study an RCMPSP with uncertainty in duration, costs and task outcome and with new project arrivals, with the goal of cost minimization. They model this as an MDP where the possible actions at each timestep are whether to perform, not perform or cancel each task. They heuristically create state–action pairs by simulation and use Q-learning to find solutions to this problem. They present solutions for instances with up to 5 different project types. Another variant is given by Salemi Parizi et al. (2017). They consider an RCPSP with new project arrivals, where new projects are rejected if there are too many incomplete projects in the queue. At each time, the policy determines which tasks to start in order to minimize the infinite-horizon discounted expected profit. This is solved with a simulation-based approximate policy iteration method and computational results are given for instances with up to 15 different project types. Satic et al. (2020) solve a stochastic RCMPSP with new project arrivals with cost minimization based on early/late finish penalties. They provide exact solutions based on an MDP and dynamic programming, and compare this to a priority rule based reactive algorithm and a genetic algorithm. This is done for fairly small instances, with the largest containing 4 project types with all 2 tasks per project.

All these MDP-based approaches have certain characteristics in common. First of all, they handle instances with relatively few types of projects and relatively small projects in terms of number of tasks. Secondly, they provide policies to decide between projects at various given time steps, instead of making all decisions at the start of a project. Therefore, we broaden our view.

The field of *simulation optimization* has both the characteristics of handling very expensive to compute objective functions by simulating stochastic processes and making multiple decisions at one decision stage. More precisely, it deals with optimization problems where the objective function and/or constraints can be evaluated through a stochastic simulation. Since the DRCMPSP/SS has these characteristics as well, we further explore this method, instead of MDP based approaches. For more details on simulation optimization, we refer to various surveys (Amaran et al., 2016; Juan et al., 2015; Homem-de Mello & Bayraksan, 2014).

When considering simulation optimization, the main difficulty for the DRCMPSP/SS is that each simulation contains the scheduling process of newly arriving projects, and therefore, will be computationally very expensive. A method for dealing with computationally expensive objective functions is estimation with machine learning, which has been studied for various problems. One of these is a machine scheduling problem studied by Hao et al. (2016). They solve a problem consisting of machine assignment and sequencing decisions, where they use a so called *extreme learning machine* to estimate the value of a specific machine assignment. This is subsequently used by a differential evolution algorithm. Park and Kim (2017) present a general optimization algorithm where a particle swarm optimization algorithm uses a neural network to estimate the fitness function for each particle, based on the fitness of the parent. This is used to select promising solutions for full fitness function computation. This algorithm is used to optimize 10 benchmark functions. Another approach using objective function estimation can be seen in Zheng et al. (2020). Here, an assembly job shop problem is studied with optimization on makespan and its expected deviation. The expected deviation is estimated by a radial basis function network that uses data from previously ran Monte Carlo simulations. This estimator is then used within a tabu search heuristic to find good solutions to the problem.

In conclusion, there has been quite some research on the RCMPSP with new project arrivals, but these approaches seem unsuitable for the DRCMPSP/SS. Therefore, we have expanded our search to simulation optimization and objective function estimation to find different building blocks in order to handle the DRCMPSP/SS.

3. Problem description

In this section, we give a problem description of the DRCMPSP/SS. We start by describing the environment of the arriving projects and the scheduler. Subsequently, we present the structure of a single project, and finally, we explain the full optimization problem, consisting of multiple projects.

The DRCMPSP/SS environment consists of a *project generator* and a *project scheduler*, which operate sequentially for $|K|$ iterations, where K is the set of stages, i.e., $K = \{1, \dots, |K|\}$. In the first iteration, i.e. for $k = 1$, the project generator outputs the first project $\mathcal{P}^1$, its arrival time $\tau^1 = 0$ and the earliest arrival time for the next project τ^2_{min}. After this, the project scheduler schedules the first project $\mathcal{P}^1$. In each subsequent iteration $k > 1$, the project generator outputs a project $\mathcal{P}^k$, earliest next arrival time τ^{k+1}_{min}, and current arrival time $\tau^k = \tau'^k + \tau^k_{min}$, where $\tau'^k \geq 0$ is the deviation from the estimate τ^k_{min}. Therefore, the time is

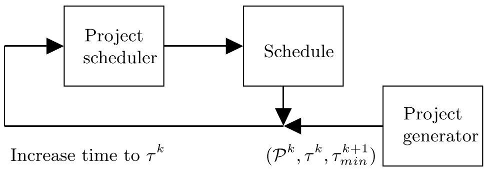

Project scheduler
Schedule
Project generator
Increase time to τ^k
$(\mathcal{P}^k, \tau^k, \tau^{k+1}_{min})$

Fig. 1. Scheduling environment with the project scheduler and the project generator.

incremented until τ^k, the arrival time of project $\mathcal{P}^k$, and project $\mathcal{P}^k$ is scheduled by the project scheduler. The process is illustrated in Fig. 1.

At each stage $k \in K$, we let Ξ^k be the random variable representing the generated project. Thus, each realization ξ^k of Ξ^k is a 3-tuple $(\mathcal{P}^k, \tau^k, \tau^{k+1}_{min})$. The shared project environment consists of a set of resources R that have to be shared across all projects. The resource availability λ_r defines the total capacity of resource $r \in R$. Furthermore, we define T as the set of time periods in which the projects can be scheduled.

At decision step k, we obtain a single project $\mathcal{P}^k = (N^k, P^k, \mathbf{d}^k, \mathbf{b}^k)$. For brevity, we omit the superscript k as long as it is clear from context or irrelevant. The project $\mathcal{P}$ consists of a set of activities N where each activity $i \in N$ has a deterministic duration d_i. Furthermore, a set of precedence relationships is given: precedence relationship $(i, j) \in P$ imposes that activity $i \in N$ has to finish before activity $j \in N$ can start. All activities have to be scheduled after the project arrival time τ^k, while satisfying all precedence relationships. Furthermore, each activity $i \in N$ requires a constant quantity of b_{ri} of resource $r \in R$ between the start and end time, and the total requirement of all activities, plus any resource requirements from previous projects, cannot exceed the resource limit λ_r at any time. A mixed integer linear program formulation of this problem is given in Pritsker et al. (1969).

In the standard RCPSP, the objective is to minimize the makespan of the project. However, in the DRCMPSP/SS, we are interested in minimizing the combined makespan of all projects instead. Simply taking the average of the makespan would unfairly focus more on larger projects. Therefore, based on the research of Chen et al. (2019), we minimize the average makespan divided by the critical path length per project. The critical path length is the duration of the project, while relaxing the resource constraints (Artigues et al., 2008). This can be calculated quickly and can be used as a measure for the size of the project. Thus, the goal is to minimize the expected sum of the makespan over the critical path for each project.

We denote X to be a solution to project $\mathcal{P}$. The objective function of this single project is called the **current objective** and is defined as the makespan divided by the critical path length:

$$obj(X) = \frac{\text{makespan}(X)}{\text{critical path length}(X)} \tag{1}$$

Furthermore, we introduce the notation $x[i] = \{x^1, \dots, x^i\}$ for any indexed variable x. Thus, $X[k]$ contains the solution vectors $\{X^1, \dots, X^k\}$ for projects 1 to k. Additionally, let $\mathcal{X}^k(X[k-1])$ be the solution space of project k, given solutions to earlier projects $X[k-1]$. With this notation, we define the **cost-to-go** function ctg^k for all $k \in K$:

$$ctg^k\left(X[k-1], \Xi^k\right) = \min_{X^k \in \mathcal{X}^k(X[k-1])} obj(X^k) + \mathbb{E}\left[ctg^{k+1}(X[k], \Xi^{k+1})\right]. \tag{2}$$

The first part of this function is the current objective of the considered project $k \in K$ and the second part is the expectation of all future projects. By defining this as iterative minimization functions, the minimization at each decision stage finds the minimal value, given that the future decisions also minimize the cost-to-go. Therefore, optimizing Eq. (2) for $k = 1$ captures all decision stages due to the iterative formulation. If we define $ctg = ctg^1$, the optimization problem can be expressed as:

$$ctg = ctg^1\left(\emptyset, \Xi^1\right) = \min_{X^1 \in \mathcal{X}^1} obj(X^1) + \mathbb{E}\left[ctg^2(X[1], \Xi^2)\right]. \tag{3}$$

Furthermore, all notation used throughout this paper is presented in Appendix.

4. Solution methods

In the previous section, the DRCMPSP/SS is described. An instance of the DRCMPSP/SS consists of a set of shared resources and an uncertain number of projects with unknown activity properties and project arrival time. This results in a multi-stage stochastic problem, where for each stage an NP-hard problem needs to be solved and the expected objective value of multiple future NP-hard problems needs to be determined. However, computing the expectation over multiple stages would be computationally too challenging. Therefore, we present three solution methods, which at each decision step, only consider the current project, and possibly an estimation of the project directly arriving after that.

The first method is the **Greedy Method** (GM). This method schedules each project by scheduling it as well as possible, without looking ahead. This is a fast and simple method that serves as a baseline to compare against the other algorithms. The second method is the **Full Method** (FM). Here, each objective function evaluation contains a simulation of projects arriving in the future. Thus, this method looks ahead and finds schedules that account for future arriving projects. The final method is the **Trained Method** (TM). This method uses data from earlier or simulated runs from the FM and trains a neural network based estimator. This estimator replaces the simulations in the objective function evaluation of the FM. Therefore, the TM takes much less computing time than the FM.

In the remainder of this section, the three methods are described.

4.1. Greedy method

An easy way to schedule multiple arriving projects in practice, is simply by not looking ahead and scheduling each project as well as possible at the time of arrival. We call this method the *Greedy Method* (GM) and use it as a benchmark algorithm. This allows us to answer the question: Can we improve scheduling for the DRCMPSP/SS by learning from data? Based on successful implementations for the RCPSP (Van der Beek et al., 2023; Quoc et al., 2020; Sallam et al., 2020; Zaman et al., 2021), we use a Differential Evolution (DE) algorithm to optimize each schedule. Since the focus of this paper is on the learning-from-data aspect, we only give a brief description of the algorithm and refer to Storn and Price (1995) for a more elaborate description. Furthermore, since the DE algorithm is used throughout this paper with varying details, we use a general notation.

The solutions in the algorithm are stored in solution matrix $\mathbb{X}$ that consists of γ solution vectors of length *sol_len*. In the GM, *sol_len* is equal to the number of activities. Each solution vector $\mathbf{x} \in \mathbb{X}$ is a priority vector: a vector with an entry for each activity that defines the priority to schedule this activity. This vector is converted to a schedule by a serial Schedule Generation Scheme (SGS). The serial SGS selects activities based on their priorities, and then sequentially schedules each one at its earliest resource and constraint feasible time. This is described in more detail in Artigues et al. (2008). This gives a schedule: an assignment of start times to the project activities. Furthermore, $\mathbf{x}^*$ keeps track of the best solution found so far. During each iteration, an improvement step is performed for each $\mathbf{x} \in \mathbb{X}$. In this improvement step, three solution vectors $\mathbf{a}^1, \mathbf{a}^2$ and $\mathbf{a}^3$ are randomly selected from $\mathbb{X}$ and used to create a trial solution $\rho = [\rho_1, \dots, \rho_{\text{sol\_len}}]$. This is done by creating $\mathbf{a}$ (line 6) and selecting entries from that vector with probability c. To ensure there is always at least one entry replaced, f is used. If the objective value of this trial solution ρ is better than or equal to the objective value of the currently considered solution $\mathbf{x}$, it replaces $\mathbf{x}$ in $\mathbb{X}$. Similarly, if it is better than the best solution $\mathbf{x}^*$ found so far, it replaces this one too. This gives the full solution algorithm, as given in Algorithm 1.

Solution matrix $\mathbb{X}$ is initialized by generating a matrix with uniformly distributed random values between 0 and 1. The algorithm is terminated when no new improved solution has been found in the last 25 iterations. This value is set manually, as it was found to give reasonable computing times.

Algorithm 1 Differential evolution

```
1:  $\mathbb{X} \leftarrow$ Initialization matrix of size $\gamma \times sol\_len$
2:  $\mathbf{x}^* \leftarrow \text{argmin}_{\mathbf{x}\in\mathbb{X}}(\text{objective } \mathbf{x})$
3:  while not terminated do
4:      for $\mathbf{x} \in \mathbb{X}$ do
5:          $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3 \leftarrow$ Pick randomly without replacement from $\mathbb{X} \setminus \{\mathbf{x}\}$
6:          $\mathbf{a} \leftarrow \mathbf{a}^1 + w(\mathbf{a}^2 - \mathbf{a}^3)$
7:          $f$ = random from $\{1, \cdots, N\}$
8:          $\rho_i \leftarrow \begin{cases} a_i & \text{with probability } c \text{ or if } i = f \\ x_i & \text{otherwise} \end{cases}$
9:          if objective $\rho \leq$ objective $\mathbf{x}$ then
10:             Replace $\mathbf{x} \in \mathbb{X}$ by $\rho$
11:             if objective $\rho <$ objective $\mathbf{x}^*$ then
12:                 $\mathbf{x}^* \leftarrow \rho$
13:             end if
14:         end if
15:     end for
16: end while
17: return $\mathbf{x}^*$
```

The term $\mathbf{a}^1 + w(\mathbf{a}^2 - \mathbf{a}^3)$ in line 6 ensures that activities for which the prioritization is similar in good solutions, these priorities will stay similar in the resulting $\mathbf{a}$. For activities for which the prioritization varies in good solutions, the term $\mathbf{a}^2 - \mathbf{a}^3$ keeps the same variety in prioritization in the resulting $\mathbf{a}$. Therefore, the algorithm creates solutions that vary the priority of the activities for which the correct prioritization is still unclear, and spends less time varying the priority of activities for which the correct prioritization is already clear.

4.2. Full method

This subsection presents the *Full Method* (FM), which is the most computationally expensive method presented in this paper. The FM also uses a DE algorithm (Algorithm 1) to search the solution space. However, instead of only looking at the current project, it includes a simulation to estimate the objective value for the subsequent project. This is done by modifying three parts of the algorithm: the solution representation, the objective function, and an optional part to store data for the TM. These are explained in the remainder of this subsection, after which the full description of the FM is given.

4.2.1. Solution representation

In the GM, a priority vector is used to represent solutions which can be converted to a schedule by a serial SGS. The serial SGS sequentially schedules each activity at the earliest possible starting time. However, when future arrivals are considered, it is possible that the best schedule contains activities that are scheduled later than their earliest possible starting time. This can be done to allow availability of resources for later arrivals. For this reason, we introduce a new solution representation by defining each solution vector $\mathbf{x}$ as a vector of length $2|N|$, where $|N|$ is the number of activities to be scheduled. Then, the first $|N|$ entries form a priority vector, as described in Quoc et al. (2020). The remaining entries form the **gap vector**. This vector defines the gap for each activity. When scheduling an activity according to the serial SGS, this gap value is rounded and added to the starting time. This allows the schedules to have spread out activities in order to decrease the resource usage at certain times. A similar concept is used in Hartmann (2015), where it is called *delay value*.

4.2.2. Objective function

The main difference between the FM and the GM, is that in the FM a solution is evaluated on both the current project and simulated future projects. This is based on the *Sample Average Approximation* (Kleywegt et al., 2002). This method finds solutions to stochastic problems by generating multiple scenarios and creating a deterministic equivalent of the stochastic problem using these scenarios. However, using a fixed set of scenarios risks finding solutions that perform well only on these specific scenarios (Homem-De-Mello, 2003). Secondly, the FM is designed with compatibility for the TM in mind: besides running the FM to obtain solutions, it should also be possible to use the FM to generate data for the TM. Using a fixed set of scenarios limits the variety in the training data. Therefore, the FM does not create a deterministic equivalent. Instead, it uses an iterative optimization approach where each objective estimation consists of optimizing multiple future candidate projects, with different scenarios per iteration. This allows us to consider a large set of scenarios and create varied training data, without having to optimize for all of these scenarios simultaneously.

In each iteration of the FM, ζ scenarios are generated, each containing a different second project. This set of projects is denoted by $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_\zeta\}$. Let $\mathcal{X}(X, \mathcal{P})$ be the set of feasible solutions for project $\mathcal{P} \in \mathbb{P}$, given solution X to the current project $\mathcal{P}^1$. Then, to evaluate a solution X in the algorithm, we take the current objective of the current project $obj(X)$ plus the average of the best solutions for the scenario projects. The objective values for these scenario projects are called the **scenario objectives**. Thus, the objective function for the FM becomes:

$$obj^{FM}(X) = obj(X) + \frac{1}{\zeta} \sum_{\mathcal{P}\in\mathbb{P}} \min_{X'\in\mathcal{X}(X,\mathcal{P})} obj(X') \tag{4}$$

This is called the **combined objective**, where $\min_{X'\in\mathcal{X}(X,\mathcal{P})} obj(X')$ represents an optimization problem. This means that for each objective function computation, ζ minimization problems have to be solved. In the FM, we estimate this by using the GM. This allows the FM to contain information on different simulated future scenarios. However, it also increases the computational time immensely: for every objective function computation, multiple other optimization algorithms are run.

4.2.3. Storing data

Each solution X, evaluated in the algorithm, results in a resource profile Y: a matrix, where each entry Y_{rt} represents the total usage of resource $r \in R$ at time $t \in T$. This resource profile is used to generate training data for the TM. This is done as follows: given a solution X and corresponding resource profile Y with shape $|R| \times |T|$, one datapoint is stored for each incoming scenario project $\mathcal{P} \in \mathbb{P}$. If project $\mathcal{P}$ has earliest arrival time τ_{min} and has *scenario objective* v, the datapoint is $(Y_{[:,\tau_{min}:|T|]}, v)$. Thus, it stores the resource profile, starting from the earliest arrival time of the incoming scenario project, and the scenario objective.

4.3. Full method algorithm

In this subsection, the FM algorithm is shown. In the current research, this is implemented as a DE algorithm, created by adapting the GM (Algorithm 1). However, it can be any population-based search algorithm. The algorithm starts by initializing the population of solution vectors and by setting the scenario count ζ to 2, meaning that in the first iteration, we evaluate 2 incoming scenario projects. Then, the algorithm starts the iterative process.

At each iteration, ζ scenario projects are generated, together with a new population of solutions. These solutions are generated as described in Section 4.1. The new solutions are compared against the previous solutions, and replace the old solutions if they are better. If the algorithm is used to generate data, it stores the datapoints of all solutions for both populations. Subsequently, it performs a *paired t-test* between the combined objectives of both populations. This idea was adopted from Homem-De-Mello (2003). If the p-value of the test is smaller than

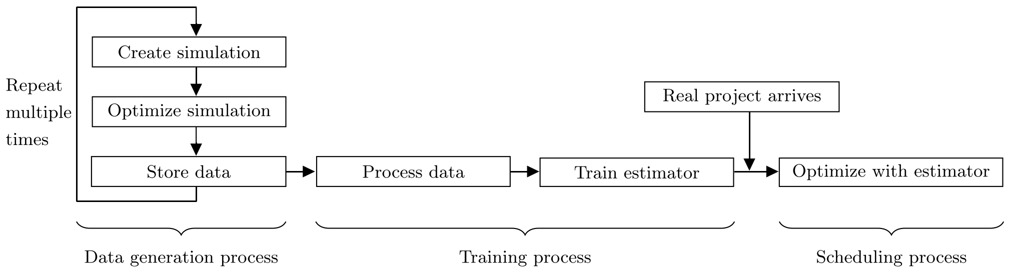


Fig. 2. Workflow of optimization approach.

the p-value threshold parameter μ, it cannot be concluded that $V_{q_1 1}$ and $V_{q_2 2}$ are statistically different and ζ is increased by 1.

The iterative process is repeated until the stopping criterion is met: when the average value of all combined objectives has not reached a new minimum in a certain number of iterations. In that case, the algorithm is terminated and the best solution in the final population is returned.

4.4. Trained method

The FM can be used to find schedules for arriving projects. However, this requires a lot of time, usually multiple days per project. Therefore, we use the stored data $\mathcal{D}$ to create an objective estimator. This objective estimator replaces the simulation step in the FM and thus reduces computing time significantly.

The workflow of the TM is visualized in Fig. 2. This consists of three stages. First, the **data generation process** generates all data required for training the estimator. This is done by using the project generator Ξ to create multiple *simulations*, where each simulation consists of K projects. Then, the FM method is executed on each of these projects, while taking the resource usage of the solutions of all previous projects into account. During this process, the resource profiles and corresponding scenario objectives are saved to $\mathcal{D}$. Next, the **training process** starts. This process uses stored data $\mathcal{D}$ to train an objective estimator. Finally, the real project arrives and the **scheduling process** starts. This process uses the objective estimator within the DE algorithm to schedule any incoming projects.

Since the data generation process consists of using the FM on simulated data, the description can be found in Section 4.2. Therefore, the remainder of this subsection explains the training and scheduling processes. The training process is split up into data processing and estimator training.

4.4.1. Data processing

The goal of the objective estimator is to evaluate a schedule on the estimated scheduling performance for future projects. This is done by taking the resource profile of a solution as input and returning a scalar value, called the **profile score**. This is a measure for the quality of a resource profile, with a higher value indicating a resource profile of higher quality. However, dataset $\mathcal{D}$ obtained by the FM consists of a set of resource profiles, each with a corresponding set of scenario objectives. In this subsection, it is explained how to convert these scenario objectives to a single scalar value per resource profile, on which the estimator can be trained.

Before presenting the data processing method, three observations are made. The first one is that, since the goal of the estimator is to compare solutions, it is not required that the value given by the objective estimator resembles the combined objective. Instead, for an ideal objective estimator $f(x)$, it only is required that $f(X) < f(X')$ when $\mathbb{E}\,(\text{combined objective of } X) < \mathbb{E}\left(\text{combined objective of } X'\right)$ for any two solutions X and X'. Secondly, a possible method to create the estimator is to train it directly on the tuples of a resource profile and scenario objectives in $\mathcal{D}$. However, this does not give a reliable estimate, since the objective values in $\mathcal{D}$ are obtained for different scenarios. Finally, since the number of scenarios varies in the FM, we note that the number of objective values in $\mathcal{D}$ also varies per iteration. It follows that solutions evaluated on more scenarios provide more certainty about the expected combined objective value.

To obtain the scalar values for each resource profile, called the *profile scores*, the **profile network** is introduced. This is a network that contains a node for each resource profile and an edge based on comparisons between these profiles. Each directed edge from resource profile Y_i to resource profile Y_j has a weight, representing the probability that resource profile Y_j has better scenario objective values than resource profile Y_i. Then, resource profiles with good scenario objectives can be found by performing random walks in this network, as is explained later.

To create the profile network, we let $\mathcal{Y} = \{Y_1, \dots, Y_{|\mathcal{Y}|}\}$ contain all unique resource profiles in $\mathcal{D}$. The objective values are stored in U, where U_{ni} is the vector of scenario objectives for resource profile Y_i in iteration n the FM. Furthermore, we let C_{ij} be the set of all iterations in $\mathcal{D}$ that contain both resource profile $Y_i \in \mathcal{Y}$ and $Y_j \in \mathcal{Y}$. With this, we define matrix Δ_{ij} containing the difference of all scenario objectives for iterations with both resource profile Y_i and Y_j:

$$\Delta_{ij} = \bigcup_{n \in C_{ij}} \left(U_{ni} - U_{nj}\right), \tag{5}$$

where we use the ∪-operator to *concatenate* vectors. With this, we require a measure for the confidence that profile Y_i should be chosen over profile Y_j and define this measure as Q_{ij}, where a low value of Q_{ij} indicates that profile Y_i should be chosen over Y_j. For this measure, we use the *t-distribution*. Usually, this distribution is used for normally distributed data. However, even though Δ_{ij} might not be normally distributed, we only require a confidence measure and not an exact probability. Using a *t-distribution* has the following beneficial properties:

1. More samples result in a higher confidence.
2. No samples or no difference between samples result in $Q_{ij} = Q_{ji}$.
3. The values are symmetrical; $Q_{ij} = 1 - Q_{ji}$.

Therefore, we define the confidence value Q_{ij} as follows:

$$Q_{ij} = F\left(\frac{\overline{\Delta_{ij}}\sqrt{|\Delta_{ij}|}}{std\left(\Delta_{ij}\right)}, |\Delta_{ij}| - 1\right), \tag{6}$$

with $std()$ being the sample standard deviation with Bessels correction and $F(x, n)$ the cumulative distribution function of the *t-distribution* with n degrees of freedom. Next, we apply a method similar to Negahban et al. (2012) to convert these values to a network usable for random walks. First, let C^I_{ij} be the indicator value, equal to 1 if there is at least one iteration where both resource profile $Y_i \in \mathcal{Y}$ and $Y_j \in \mathcal{Y}$ are found ($|C_{ij}| > 0$) and zero otherwise. We only create an edge between profile Y_i and Y_j, if there is at least one comparison in the same iteration ($C^I_{ij} = 1$). We define δ_i as the number of outgoing arcs from profile Y_i:

$$\delta_i = \sum_{j=1, i \neq j}^{|\mathcal{Y}|} C^I_{ij}. \tag{7}$$

Now, we define for each pair of profiles $Y_i, Y_j \in \mathcal{Y}$ the value A_{ij} that represents the probability of moving to profile $Y_j \in \mathcal{Y}$, while located in profile $Y_i \in \mathcal{Y}$, in the random walk:

$$A_{ij} = \begin{cases} \frac{1}{\delta_i} Q_{ij} & \text{if } i \neq j \text{ and } C^I_{ij} = 1 \\ 1 - \frac{1}{\delta_i} \sum_{k \neq i} Q_{ik} C^I_{ik} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

Thus, we obtain a profile network $\mathcal{G} = (\mathcal{Y}, E)$, where each unique resource profile $Y_i \in \mathcal{Y}$ is a node, and with an edge $(i,j) \in E$ if resource profiles Y_i and Y_j have been compared at least once in the same iteration of the FM. Each edge $(i,j) \in E$ has a value of A_{ij}. This network is used to calculate the **profile score vector** $\mathbf{s} = \{\mathbf{s}_1, \dots, \mathbf{s}_{|\mathcal{Y}|}\}$: A vector where each entry $\mathbf{s}_i$ is the *profile score* of resource profile $Y_i \in \mathcal{Y}$. The resource profile vector is calculated as the stationary distribution of a random walk, which is done by initializing the profile score vector to $\mathbf{1}$ and repeatedly matrix-multiplying $\mathbf{s}$ by A until the change in $\mathbf{s}$ is below a certain threshold η. This is shown in Algorithm 2. After convergence, a large entry s_i in score vector $\mathbf{s}$ indicates a high quality resource profile $Y_i \in \mathcal{Y}$.

Algorithm 2 Random walk to obtain profile score vector $\mathbf{s}$ from matrix A.

```
1: $\mathbf{s} \leftarrow \mathbf{1}$
2: $diff \leftarrow \infty$
3: while $diff > \eta$ do
4:     $\mathbf{s}' \leftarrow \mathbf{s} \cdot A$
5:     $diff \leftarrow |\mathbf{s} - \mathbf{s}'|$
6:     $\mathbf{s} \leftarrow \mathbf{s}'$
7: end while
```

A requirement for obtaining useful scores, is that $\mathcal{G}$ is connected. This is always the case if $\mathcal{D}$ only contains data from a single run of the FM, since for each generation a comparison is added containing both the old and new population. However, since $\mathcal{D}$ can contain data from multiple runs, it is possible that $\mathcal{G}$ is not connected. In this case, each score $\mathbf{s}_i$ for $Y_i \in \mathcal{Y}$ only represents a score relative to nodes within the connected component of Y_i. To tackle this problem of $\mathcal{G}$ being possibly disconnected, we apply the following procedure: after generating the data, we use Algorithm 2 to obtain profile score vector $\mathbf{s}$. If $\mathcal{G}$ is connected, we terminate the algorithm. Otherwise, for each separate connected component in $\mathcal{G}$, we select the resource profile with the highest score within that respective component. This gives us a set of resource profiles $\mathcal{Y}^{max}$, one for each connected component. Then, a set of new scenarios is created and the scenario objectives are calculated for each resource profile $Y \in \mathcal{Y}^{max}$. The number of scenarios is equal to the maximum number of scenarios evaluated in one generation, of all runs of the FM in the data generation process (i.e.: largest ζ encountered in all runs of the FM). These new evaluations are added to $\mathcal{D}$ and then used to generate new edges in $\mathcal{G}$, between all resource profiles in $\mathcal{Y}^{max}$. Since each connected component has one resource profile in $\mathcal{Y}^{max}$, $\mathcal{G}$ is now connected. Therefore, we subsequently compute the profile scores $\mathbf{s}$ on this updated graph $\mathcal{G}$ by executing Algorithm 2. This is summarized below:

1. Generate data $\mathcal{D}$ using the FM.
2. Create *profile network* $\mathcal{G}$ and compute profile scores $\mathbf{s}$.
3. If $\mathcal{G}$ is connected, terminate. Otherwise, go to the next step.
4. Get connected components and create set $\mathcal{Y}^{max}$ consisting of the resource profiles with the highest score in each connected component.
5. Create objective evaluation for profiles $\mathcal{Y}^{max}$ and add this data to $\mathcal{D}$.
6. Repeat step 2 and terminate.

This generates a profile score for each unique resource profile. These scores are used to train the objective estimator, as discussed in the next section.

4.4.2. Estimator training

The profile score vector $\mathbf{s}$ from the previous section forms a measure to compare the resource profiles $\mathcal{Y}$, encountered in the data generation process. However, in the scheduling process, we require a method to compare any resource profile to other resource profiles. Therefore, we use the resource profiles $\mathcal{Y}$ and profile score vector $\mathbf{s}$ to train an objective estimator, which in turn can create an estimate of the score of any resource profile. This objective estimator is a function that takes a resource profile Y as input and outputs a scalar estimation of the quality, i.e., a profile score. This estimator is trained during the training process and, subsequently, used within the optimization algorithm during the scheduling process. The core of this estimator consists of a neural network that is trained several times.

The used neural network is a dense feed-forward network, which has as input a resource profile, consisting of a set of sequences, where a sequence denotes the resource usage for a single resource. The length of each sequence is the makespan of the project, minus the earliest arrival time of the next project. Then, the output of the neural network is a scalar: the estimation of the profile score. In order to make estimations for sequences, one might think of different neural network structures, such as recurrent neural networks or transformers (Lim & Zohren, 2021). However, even though the input sequence in theory can be of infinite length, very long sequences correlate to schedules with a very high makespan, and thus of low-quality. This permits us to cut off sequences after a certain length, presuming that this length is sufficiently large. Furthermore, the absolute location in the sequence is important, a property that suits a feed-forward network instead of a recurrent network. Nevertheless, preliminary tests were performed with recurrent neural networks, long short-term memory networks and transformer based networks, but the best performance was found with the simple feed-forward network.

We now describe the neural network architecture. The input layer has a size of $l|R|$, where l is the maximal considered sequence length and $|R|$ the number of resources. l is set to be the length of the longest profile, encountered in the data generation process. After the input layer, we have s^h fully connected hidden layers, each with s^w nodes. After the final hidden layer, there is one output node, which is fully connected to the last hidden layer. The rectangular shape of the hidden layers was chosen for tuning efficiency, since it can be described by only 2 variables. When the input, encountered in the scheduling process, is longer than l, the final part is truncated.

Furthermore, we use an ADAM optimization process (Kingma & Ba, 2015) with a *weight decay* of p^w and a *learning rate* of p^l. The parameter values were tuned with the hyperparameter optimization method from Bergstra et al. (2013).

4.4.3. Trained method algorithm

In the previous section, it is explained how to process the data $\mathcal{D}$ to obtain profile scores and how to train a neural network on these profile scores. In the remainder of this section, it is described how to create the objective estimator and how it is used within an optimization algorithm.

Using a neural network as an objective estimator has the following major problem: In a neural network, there are some areas of the input space bound to have lower accuracy, and thus, some profiles are estimated to be high quality while they are not. When an optimization algorithm uses the neural network as an objective function, *it actively searches for these low-accuracy areas*, since they often hold good objective values due to the variation. To remedy this, a technique encountered in reinforcement learning research is used (Levine et al., 2020). Here, we train a neural network several times with different seeds. Then, instead of using one estimated value, the objective consists of the mean value of all predictions plus a penalty term based on the variance of the estimations. The idea behind this is that if the variance is high, the area can be seen as *low-accuracy*. By adding a penalty based on this, the search is guided away from these low-accuracy areas.

Let $\mathcal{M} = \{\mathcal{M}_1, \dots, \mathcal{M}_{|\mathcal{M}|}\}$ be the set of trained neural network estimators, where each $\mathcal{M}_i \in \mathcal{M}$ is a function taking a resource profile $Y \in \mathcal{Y}$ as input and outputting a scalar value. Then, calling $\mathcal{M}(Y)$ returns the set of values for each neural network function $\mathcal{M}_i \in \mathcal{M}$. Furthermore, let $obj^{est}(X, \tau^{k+1}_{min})$ be the estimated objective value of solution X with next earliest arrival time τ^{k+1}_{min} and let $p^m > 0$ and $p^s > 0$ be scaling parameters. Then, we define the estimated objective obj^{est} as:

$$obj^{est}(X, \tau^{k+1}_{min}) = current\_objective(X) - p^m \overline{\mathcal{M}\left(RP\left(X, \tau^{k+1}_{min}\right)\right)} + p^s std\left(\mathcal{M}\left(RP\left(X, \tau^{k+1}_{min}\right)\right)\right). \quad (9)$$

This objective function thus evaluates a potential schedule X based on three parts. The first part consists of its current objective. The second part consists of an average over all quality predictions multiplied by scaling parameter p^m. Since we are minimizing the objective, this term is subtracted. Finally, a penalty term consisting of scaling parameter p^s and the standard deviation of all quality predictions is added, in order to penalize low-accuracy estimations. Thus, $obj^{est}(X)$ gives an estimate of the combined objective. The calculation exists of using an SGS to calculate the resource profile given solution X, and then using the trained neural networks on this profile. This objective function is then used within Algorithm 1 to create the TM.

5. Computational study

In this section, the methods are evaluated and the computational results are presented. First, the instances are described. After this, the tests setup is described, including data processing and parameter tuning. With this, we present the actual tests results.

5.1. Problem instances

The presented methods are evaluated on multiple instances. An instance consists of a project generator Ξ and a set of resources R. The project generator generates multiple simulations that consist of K realizations, where each realization is an arriving project. The instances are created to replicate characteristics from modular production. The first characteristic is that projects are similar. When considering projects from a single product family, each project has some base activities that occur in each project, and some activities that result from the modularization choices of the customer. Secondly, there is an estimate of when the next project will arrive. In practice, for large construction projects, there is usually some contact with the customer before a project arrives. Although this does not give any exact information, it can give a rough estimate. Finally, production facilities aim to have some overlap in project execution times. Therefore, it is imposed that the next arriving project arrives before the current project is finished.

With these conditions, the instance generation method is now given. The instances are created by using a *base network*, generated by the method described in Vanhoucke et al. (2008). This method uses as input the **Serial/Parallel** (SP) parameter, the **Resource Factor** (RF) and the **Resource Constrainedness** (RC). For a description of these parameters, we refer to Vanhoucke et al. (2008). The base network has n^{base} activities. From these activities, we randomly select n^{opt} activities to be *optional*. Then, for each realization of an arriving project, we randomly pick n^{sel} from these optional activities and exclude the rest of the optional activities. This means that there are $\binom{n^{opt}}{n^{sel}}$ different projects in the distribution Ξ, with each configuration sampled uniformly. Furthermore, the minimum arrival times of a realization of Ξ are set by scheduling each previous project by the GM. This gives a finishing time, and the arrival time of the next project is set halfway the previous arrival and finishing time. Finally, we set the varying additional arrival time, τ', to be taken uniformly between 0 and input parameter τ'^{max}.

Before applying stochastic optimization methods, it is recommended to test the potential of stochastic optimization by evaluating lower and upper bounds. A commonly used upper bound on the objective function value can be found by creating a naive approach and calculating the

Table 1
Characteristics of the considered instances.

#	n^{base}	n^{opt}	n^{sel}	SP	RF	RC	$\lvert R\rvert$	τ'^{max}
1	30	8	3	20	30	40	2	10
2	30	8	3	80	40	60	2	10
3	40	12	4	20	30	40	2	10
4	40	12	4	80	50	80	2	10
5	50	15	5	40	50	40	2	10
6	50	15	5	80	50	90	2	10

expected result of using this model (Birge & Louveaux, 1997). For two stage continuous problems, this naive model is the *expected value* solution, which can be created by taking the mean value of each stochastic variable. For our multi-stage integer problem, we define this naive model to be the greedy method: at each stage, the optimal solution that does not look ahead is chosen. The expected result is then defined as the **Expected result of the Greedy Solution** (EGS). As a lower bound for the objective function value, the expected value of the **Wait-and-See** (WS) solution (Birge & Louveaux, 1997) can be used. The WS solution is the optimal solution of the problem that assumes that it is possible to wait for all stochastic variables to be realized before making any decision. The difference between the WS value and the EGS then forms an upper bound on the **Value of Stochastic Solution** (VSS); the price one pays for using the naive model rather than the stochastic model.

Due to the very long computing time of generating data, parameter tuning for the training process, and training the neural networks, it is not feasible to evaluate many different instances. Therefore, we generate 6 instances and give detailed results for these. The selection of these instances is done based on an estimate of the VSS to focus on instances with a high potential for improvement by any stochastic method. To estimate the VSS, instead of creating and optimizing many scenarios per instance, only one scenario is generated. This estimate consists of taking the base project and creating a sequence of projects by copying this project. Subsequently, both the EGS and the WS solution values are approximated by the DE algorithm. Then, six instances were selected based on the number of activities and on the estimated VSS value. The characteristics of these instances are shown in Table 1.

5.2. Tests setup

In this subsection, the data creation, data processing and parameter tuning are discussed. The parameter tuning for all parameters, except the neural network related parameters, is carried out by a local search algorithm that iteratively varies a single parameter. Initially, this is done to determine the parameters w and c in Algorithm 1, by creating a set of projects from realizations of each instance and running the parameter tuning algorithm. This results in $w = 0.8$ and $c = 0.1$, meaning that 10% of variables are replaced in every iteration.

Since each instance represents a production scenario with a corresponding project generator, both the data generation process and the training process in Fig. 2 are executed for each instance. Additionally, the parameter tuning process for the trained method is also executed per instance, since this is also recommended in practice. For each of the six instances, 10 simulations are created, with each simulation consisting of $|K| = 5$ sequentially arriving projects. Then, the FM is sequentially executed on each arriving project, given the solutions of the previous projects in the same simulation. After this, the data is processed to create resource profiles and corresponding profile scores. Next, parameter tuning for the neural networks is started for each instance, as described in Section 4.4.2, to determine the parameters s^h, s^w, p^w and p^l. Subsequently, the neural network is trained multiple times for each instance. Then, the local search parameter tuning is started on each instance to determine the number of trained neural networks ($|\mathcal{M}| \le 15$, where 15 is chosen due to computational resource

Table 2
Parameters per instance.

#	l	s^w	s^h	p^l	p^w	p^m	p^s	$\|\mathcal{M}\|$
1	2542	400	4	1×10^{-4}	1×10^{-5}	1.0	0.7	10
2	1792	500	7	1×10^{-4}	1×10^{-5}	7.0	0.7	14
3	1149	500	6	1×10^{-4}	1×10^{-2}	1.0	0.2	5
4	1316	500	6	1×10^{-4}	1×10^{-5}	5.0	0.4	5
5	1124	500	7	1×10^{-4}	1×10^{-2}	1.0	0.1	10
6	1296	400	6	1×10^{-4}	1×10^{-5}	4.0	0.7	12

Table 3
Average computing times of processes.

Data creation	20.3 days
Parameter tuning	6.8 days
Neural network training (per network)	3.1 h

limitations) and the estimator parameters p^m and p^s. All parameters per instance are given in Table 2 and the complete instances are given in Van der Beek (2022). With these parameters and instances, the tests are performed, as explained in the next subsection.

The average durations of these steps are shown in Table 3. Here, it can be seen that the data creation part takes very long: 20.3 days. However, this process is additive, meaning that data from new runs can be added to the previous data. Therefore, in practice, it is possible to start with less generated data, and generate more data at a later time. Furthermore, this process can be easily parallelized. The parameter tuning also takes several days, because it involves training the neural network multiple times. After all this is done, training a neural network takes relatively short: around 3 h.

5.3. Computation results

The presented instances are used to evaluate the solution methods. As presented earlier, we consider three solution methods: *Greedy Method* (GM), *Trained Method* (TM) and *Full Method* (FM). This subsection describes the processing and tests done to evaluate these methods. These tests are divided into two categories: comparison between GM and TM and comparison between all methods. For the comparison between GM and TM, many simulations can be evaluated, since both of these methods are relatively fast. The goal of these tests is to evaluate whether the TM performs better than the GM, and thus, if the algorithm can learn from earlier optimization runs. The purpose of comparing all methods, so including FM, is evaluating the decrease in computing time due to learning from data and the cost, in terms of solution quality, of this. Since the FM is considerably slower, less tests are performed in this category.

To compare the results, we introduce the notion of *average relative makespan (arm)*:

$$arm = \frac{1}{|K|}\sum_{k\in K} obj(X^k) \tag{10}$$

which can be seen as the realized value of the ctg (Eq. (3)), averaged over all projects.

The neural network training and corresponding parameter tuning is performed on single cores of a 2.80 GHz GPU with 32 GB RAM. All other computations are performed on a single core of a 3.0 GHz Intel XEON CPU with 4 GB RAM.

5.3.1. Comparison between greedy method and trained method

The first category of tests are comparisons between the GM and the TM. The goal of these tests is to evaluate the improvement that can be obtained by training an objective estimator. For each of the six instances, 100 simulations are created, thus having 600 simulations in total. Then, each simulation is optimized with both the TM and the GM.

In Table 4, the results are summarized for these tests. Note that the minimal value for the arm is 5, since sequences of $|K| = 5$ arriving projects are generated in each of the 600 simulations. It can be seen that both the mean and the median arm are lower for the TM. However, the GM is slightly more stable, since the standard deviation of the arm is around 7% lower. Finally, we evaluate the number of times that either method *exclusively* has the lowest arm. Here, it can be seen that the majority of simulations has the best arm found by the TM. Deducting these values from the total number of simulations gives $600-403-173 = 24$, which shows that there are few simulations for which both methods reach the same value.

Table 4
Results of comparison between GM and TM.

	GM	TM
Mean arm	1.536	1.436
Median arm	1.527	1.365
Standard deviation arm	0.226	0.244
# lowest arm	173	403

Furthermore, the tests are evaluated in more detail in Fig. 3. Here, the instances are separated by the number of activities in the base network. The arm values are shown in Fig. 3(a). For comparison between the two methods, Fig. 3(b) shows the ratio of the arm values for the TM arm^T and the GM arm^G. This reveals a number of trends. Firstly, in Fig. 3(a), it can be seen that the deviation of the arm values increases for the GM with the number of activities. Secondly, in Fig. 3(b), it can be seen that with more activities, the increased performance of the TM becomes smaller. However, as both the mean and the median are slightly below one, the TM still performs better than the GM on the instances with 50 activities.

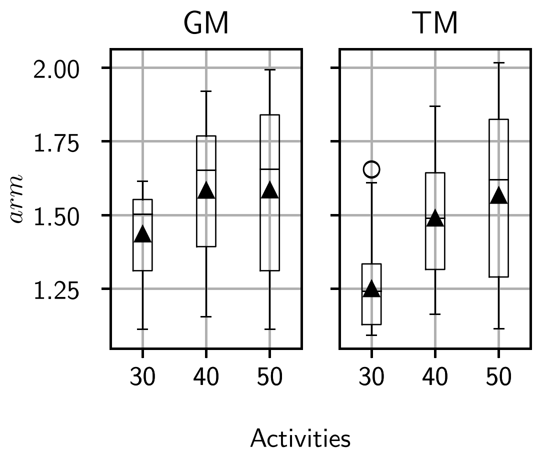
Additionally, the performance difference between the number of stages $|K|$ is evaluated. For this, we define $pobj_k = \sum_{t\in T^k}(t-\tau^k)X^k_{|N^k|t}$ as the partial objective: the makespan of the project at stage K. We use superscript to denote the method, such that $pobj^G_k$ and $pobj^T_k$ refer to the values for the GM and TM, respectively. Then, Fig. 4 shows the ratio between both methods, for each stage. Here, it can be seen that the first stage has all values greater than or equal to 1, meaning that the GM performs better than the TM. This is logical, as the TM introduces some delays in order to create better resource profiles for later stages. In the remaining stages, it can be seen that the TM performs better than the GM, with the difference in performance slightly increasing with the stage number.

Finally, the effect of using multiple trained neural networks in the estimator is evaluated. In Fig. 5, the average ratio between arm per method is shown, while varying the number of trained neural networks in the estimator. It can be seen that the performance rapidly increases with the first 6 trained neural networks, after which the performance increase flattens out somewhat. However, the average slope remains slightly negative, indicating a benefit of adding more neural networks.

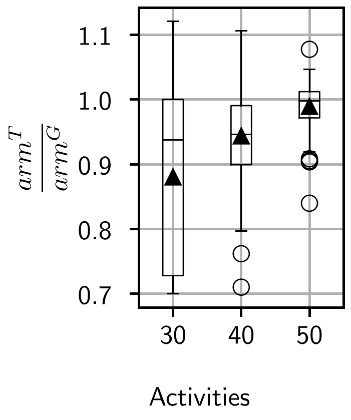
5.3.2. Comparison with full algorithm

The second category of tests compares the FM to the GM and TM. The goal of these tests is to evaluate the cost, in terms of solution quality, paid for the reduction in computing time. This is done by creating 20 simulations for each of the six instances and executing all methods on these simulations.

The summarized results of all three methods are shown in Table 5. Here, it can be seen that the quality of solutions found by the FM is superior: The mean arm, median arm, standard deviation of the arm and (non-exclusive) number of lowest arm found are better for the FM than for the other methods. However, it can also be seen that the FM has an average duration of more than three days, where the GM and TM have average computing times of less than 1 and 11 min, respectively. Thus, considering the GM as the base, the TM achieves 64% of the improvement of the FM, while only requiring around 0.33% of the computing time at the time of project arrival.



(a) Value of arm per number of activities.



(b) Ratio of arm per number of activities.

Fig. 3. Comparison between the TM and the GM on *arm*.

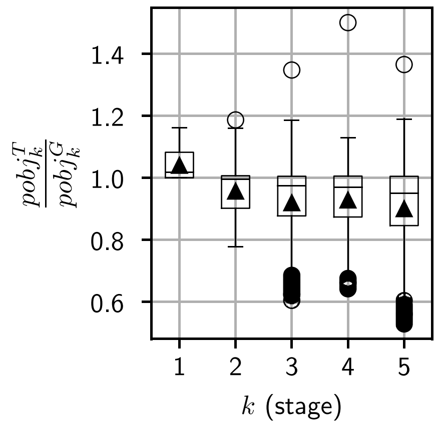


Fig. 4. Partial objective ratios per stage.

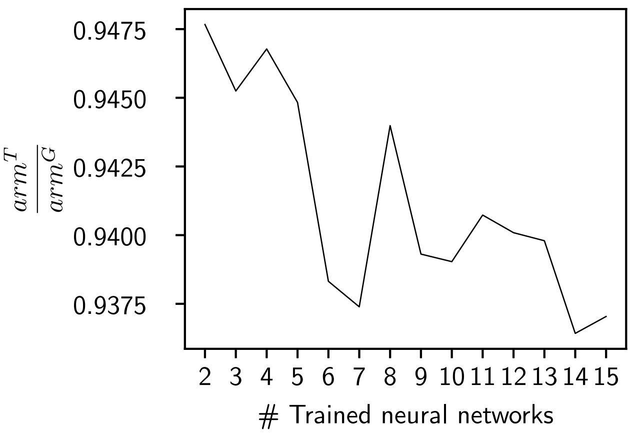


Fig. 5. Value of *arm* ratio while varying the number of trained neural networks in the estimator.

In Fig. 6, the *arm* values are shown. Here, it can be seen that the improvement against the TM and FM have a stronger correlation with the number of activities than the GM. For 50 activities, the GM and TM perform similar, although the results for the FM indicate that there is still some room for improvement in the TM. This holds especially for the median *arm*.

Table 5
Summarized results for comparison with FM.

	GM	TM	FM
Mean *arm*	1.572	1.457	1.392
Median *arm*	1.552	1.410	1.337
Standard deviation *arm*	0.220	0.249	0.233
Mean computational time (h)	0.014	0.174	52.253
# lowest *arm*	11	29	80

When evaluating the ratio between the *arm* of the TM and FM, as shown in Fig. 7(a), it can be seen that the TM is closest to the FM for the instances with 30 and 50 activities in the base network. A possible explanation is that the TM performs relatively well on the small instances, and that the FM performs relatively poor on the largest instances. This can be seen in Fig. 6, where there is a relatively small improvement for the FM, compared to the TM, on the largest instances. Furthermore, there is more variety in the distribution of larger projects. This can increase the difficulty of creating schedules that perform well on expected future arrivals.

Additionally, the ratio of partial objectives per stages are shown in Fig. 7(b), where $pobj_k^F$ refers to the partial objective of the FM at stage k. It can be seen that the relative performance of the TM decreases with the stage number. A possible explanation for this is the following: the resource profiles in $\mathcal{D}$, used to train the TM, are created from different simulations than the ones being evaluated in each test. In the first stage, there are no resource profiles from the earlier projects. Therefore, the resource profiles encountered in the training process are similar to the resource profiles in the evaluation process. In each subsequent stage, an extra project enters, and thus a potential deviation in resource profiles. Therefore, it follows that for later stages, the resource profiles encountered in the training phase are less similar to the profiles in the evaluation stages.

Finally, the computing times are shown in Fig. 8. Here, no clear trend can be seen for the GM and the TM, which might be a result of the low number of instances. However, for the FM, which has considerably longer computing times (notice the difference in y-axis), a clear increasing trend can be seen between the number of activities and the computing time.

6. Conclusions

In this paper, the stochastic optimization problem DRCMPSP/SS that schedules arriving projects under uncertainty is introduced. Furthermore, three solution methods are introduced: the greedy method,

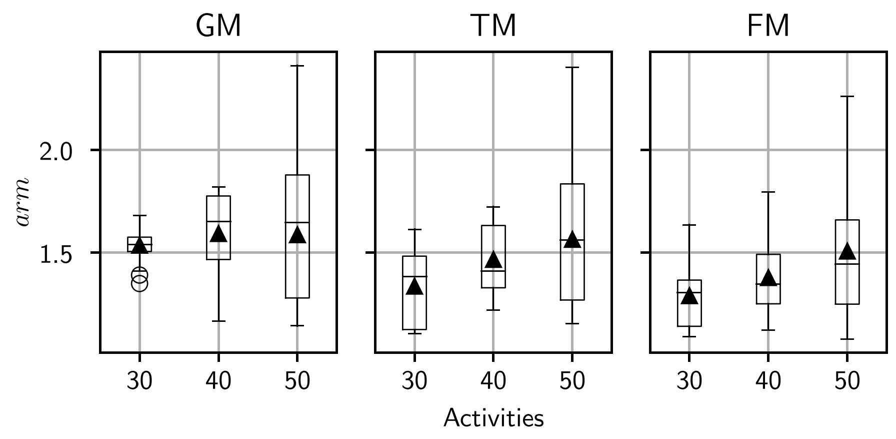
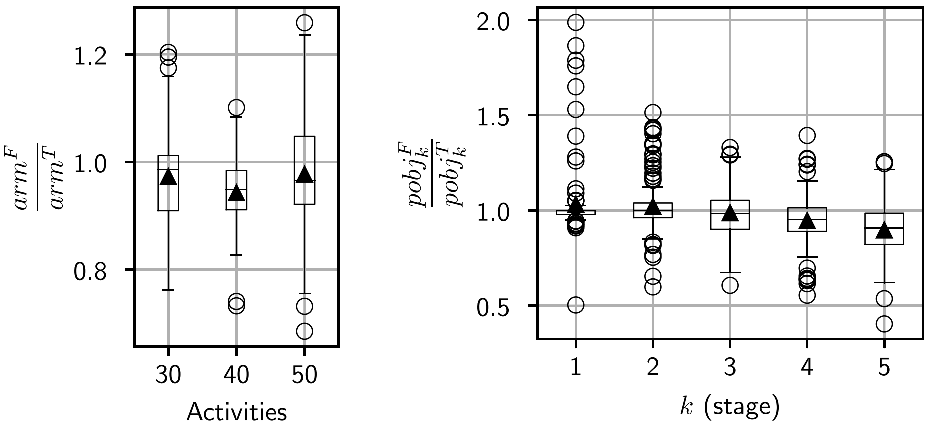


Fig. 6. Value of *arm* per method.



(a) Per number of activities. (b) Per stage.

Fig. 7. Ratios of partial objectives between FM and TM.

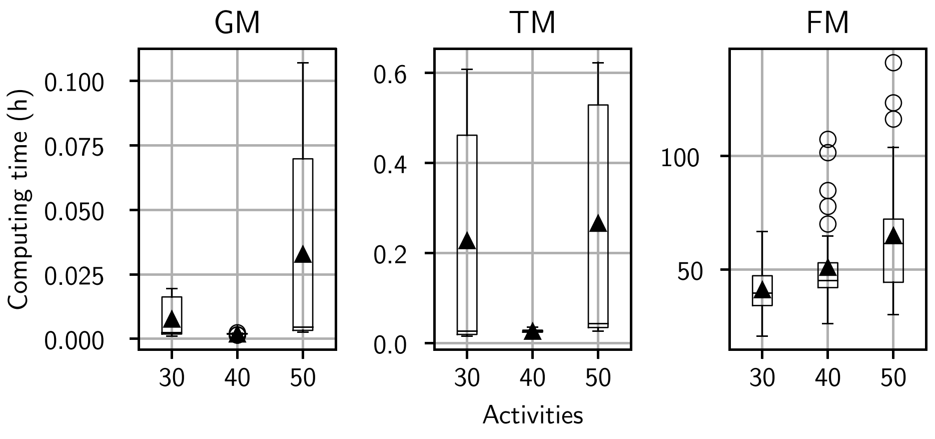


Fig. 8. Computing times per number of activities.

the full method, and the trained method. The greedy method does not look ahead and is used as a baseline method. The full method uses a sample average approximation approach with varying scenarios. The trained method learns from the full method to look ahead, while decreasing the computing time needed. When comparing the three methods, it can be seen that the trained method achieves a significant improvement in objective function value compared to the greedy method, while only requiring a fraction of the computing time of the full method. However, looking only at the solutions found, the full method still performs better. Therefore, the recommended use of these algorithms depends on the use case. Since the projects considered span several months, it often is recommended to run the full method for a few days to obtain the best schedule. However, if the size of the instances and the variation in scenarios becomes larger, the computing time of the full method might become too high and the trained method is preferred. Furthermore, quick preliminary schedules might be needed for discussion and estimates. For these use cases, the trained method is recommended as well. However, these conclusions do assume that a detailed enough simulation is possible. In many real-life cases, this will not be true, and thus one will have to revert to the GM.

From a computational point of view, it is shown how to use data from a heuristic optimization algorithm for the RCPSP. The process of learning from data to estimate the stochastic objective function value can be used in different algorithmic approaches. First of all, the data collection process is the same for any population-based search algorithm, and thus, other heuristics can be easily used. Therefore, it could be investigated whether the use of other heuristics might improve the overall performance. For example, the DE algorithm can be improved by adding forward–backward improvement (Li & Willis, 1992). Also other heuristics could be considered, such as genetic algorithms. Secondly, the data processing converts any set of comparisons to an objective estimator. Thus, this can be used with any simulation that uses resource profiles as input. Even more so, it can be converted easily to include other characteristics of the solution, as long as the corresponding neural networks are adapted as well.

Due to the high computational demands, the number of evaluated instances is limited. Although our results already reveal that the trained method achieves better results than the greedy method, and thus, demonstrates the value of learning from data, more computational tests are recommended to further verify this. In order to increase the computational testing efficiency, various options can be considered. For example, reducing the number of neural networks will reduce the neural network training time, but might reduce the performance. Similarly, one might introduce early stopping in the **scenario objectives** evaluation, sacrificing some accuracy for computational improvement. Finally, the computational burden can also be handled by parallelizing the data generation process and using more CPUs simultaneously.

For future research, one might focus on the computational evaluation of the trained method. One possibility is to evaluate the use of profile networks and study the correlation of neural network parameters to the performance of the trained method. Secondly, instead of generating new scenarios in each iteration of the full method, one could use a fixed set of scenarios. This allows, in every computation of the scenario objective, to use the previous solution as a starting point. This might achieve a significant reduction in computational time. However, this will also reduce the variety in data for the trained method, which will likely result in performance loss.

Furthermore, although the instances used resemble the characteristics of modular production, they are also fairly simplified. Therefore, creating more elaborate instances by using expert opinions or historical data can show the potential benefit of the proposed methods in practice. This can also give insight in the required size of the instances and the computational demands for this. Similarly, creating more general instances from other applications can indicate whether the presented methods are also applicable in other fields.

In conclusion, future research can focus on bringing the methods closer to applicability and by evaluating them with more computational resources. However, as shown by the difference between the trained method and the greedy method, this paper demonstrates that training from data is possible for the DRCMPSP/SS, and possibly for other variants of the Resource Constrained Project Scheduling Problem.

CRediT authorship contribution statement

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

Acknowledgments

The authors would like to thank all partners of the NAVAIS project for assistance during this research. The project has received funding from the European Union's Horizon 2020 research and innovation programme (Contract No.: 769419).

Appendix. Notation

A.1. Variables, parameters and vectors

arm	Average relative makespan.
b^k_{ri}	Resource requirement of resource $r \in R$ for activity $i \in N$ of project $\mathcal{P}^k$.
c	Replacement parameter in Algorithm 1.
$\mathbf{d}^k$	Duration vector of project $\mathcal{P}^k$.
l	Maximum sequence length in each neural network.
n^{base}	Number of activities in base network.
n^{opt}	Number of optional activities.
n^{sel}	Number of optional activities to be selected.
s^h	Number of hidden layers in neural network.
s^w	Width of hidden layers in neural network.
p^m, p^s	Scaling parameters.
p^l	Learning rate.
p^w	Weight decay.
$\mathbf{s}$	Profile score vector.
sol_len	Length of solution in Algorithm 1.
w	Weight parameter in Algorithm 1.
γ	Population size.
Δ_{ij}	Vector of differences between objectives of resource profiles Y_i and Y_j.
δ_i	Outgoing arcs of resource profile $Y_i \in \mathcal{Y}$ in the profile network.
ζ	Scenario counter.
η	Threshold parameter for profile score convergence.
λ_r	Resource capacity for resource $r \in R$.
μ	Threshold parameter for p-value.
Ξ^k	Distribution of (project, arrival time) at decision step $k \in K$.
τ^k	Arrival time of project $\mathcal{P}^k$.
τ^k_{min}	Earliest arrival time of project $\mathcal{P}^k$.
τ'	Deviation from earliest arrival time.
ω	Iteration threshold parameter.
SP	Serial/parallel parameter.
RC	Resource constrainedness.
RF	Resource factor.

A.2. Sets and matrices

A	Profile networks edge values.
B	Resource requirement matrix.
C_{ij}	Set of iterations in $\mathcal{D}$ that contain both profile Y_i and Y_j.
C^I_{ij}	Indicator value for C_{ij}.
$\mathcal{D}$	Stored data.
E	Arcs in profile network.
$\mathcal{G}$	Profile network.
K	Decision steps.
$\mathcal{M}$	Set of trained neural networks.
N^k	Set of activities of project $\mathcal{P}^k$.
P^k	Precedence relationships of project $\mathcal{P}^k$.
$\mathcal{P}^k$	Project at decision step $k \in K$.
$\mathbb{P}^k$	Scenario projects for decision step $k \in K$.
Q_{ij}	Comparison measure between resource profiles Y_i and Y_j.
R	Set of resources.
T	Set of time periods of all projects.
T^k	Set of time periods of project $\mathcal{P}^k$.
X	Binary solution matrix.
$\mathcal{X}$	Feasible region for project $\mathcal{P}^k$.
Y_{rt}	Resource usage of resource $r \in R$ at time $t \in T$.
$\mathcal{Y}$	Unique resource profiles.

A.3. Functions

$ctg^k(X[k-1], \Xi^k)$	Cost to go at decision step $k \in K$, given schedules $X[k-1]$ and distribution Ξ^k, for project $\mathcal{P}^k$.
$F(x, n)$	Cumulative distribution function of t-distribution for x with n degrees of freedom.
$obj^{est}(X, \tau)$	Estimated objective of solution X, with earliest next arrival time τ.
$RP(X[k], t)$	Resource profile of solutions $X[k]$, starting from time $t \in T$.
$std(x)$	Standard deviation of x.
$\mathcal{X}^k(X[k-1])$	Feasible schedules for project $\mathcal{P}^k$, given solutions $X[k-1]$.

References

Agarwala, N. (2015). Modular construction and ihop for increased productivity in shipbuilding. In *International seminar on nation building through ship building*.

Amaran, S., Sahinidis, N. V., Sharda, B., & Bury, S. J. (2016). Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, *240*(1), 351–380.

Artigues, C., Demassey, S., Néon, E., & Sourd, F. (2008). *Resource-constrained project scheduling*. London, UK: ISTE.

Van der Beek, T. (2022). Instances and file format for the dynamic resource constrained project scheduling problem with static project schedules.

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*(1), 92–111.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta, & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning, volume 28 of proceedings of machine learning research* (pp. 115–123). Atlanta, Georgia, USA: PMLR.

Birge, J. R., & Louveaux, F. (1997). *Introduction to stochastic programming*. New York, NY, USA: Springer-Verlag.

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

Buergin, J., Belkadi, F., Hupays, C., Gupta, R. K., Bitte, F., Lanza, G., & Bernard, A. (2018). A modular-based approach for just-in-time specification of customer orders in the aircraft manufacturing industry. *CIRP Journal of Manufacturing Science and Technology*, *21*, 61–74.

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).

Chen, H. J., Ding, G., Zhang, J., & Qin, S. (2019). Research on priority rules for the stochastic resource constrained multi-project scheduling problem with new project arrival. *Computers & Industrial Engineering*, *137*(August), Article 106060.

Choi, J., Realff, M. J., & Lee, J. H. (2007). A Q-learning-based method applied to stochastic resource constrained project scheduling with new project arrivals. *International Journal of Robust and Nonlinear Control*, *17*(13), 1214–1231.

Hao, J. H., Liu, M., Lin, J. H., & Wu, C. (2016). A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages. *Computers and Operations Research*, *66*, 215–224.

Hartmann, S. (2015). *Time-varying resource requirements and capacities* (pp. 163–176). Cham: Springer International Publishing.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306.

Homem-De-Mello, T. (2003). Variable-sample methods for stochastic optimization. *ACM Transactions on Modeling and Computer Simulation*, *13*(2), 108–133.

Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., & Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, *2*, 62–72.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd international conference on learning representations, ICLR 2015 - conference track proceedings* (pp. 1–15).

Kleywegt, A. J., Shapiro, A., & Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, *12*(2), 479–502.

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. (pp. 1–43).

Li, K. Y., & Willis, R. J. (1992). An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, *56*(3), 370–379.

Lim, B., & Zohren, S. (2021). Time-series forecasting with deep learning: A survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Article 3792194.

Homem-de Mello, T., & Bayraksan, G. (2014). Monte Carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, *19*(1), 56–85.

Neelamkavil, J. (2009). Automation in the prefab and modular construction industry. In *2009 26th international symposium on automation and robotics in construction* (pp. 299–306). ISARC 2009.

Negahban, S., Oh, S., & Shah, D. (2012). Iterative ranking from pair-wise comparisons. *Advances in Neural Information Processing Systems*, *3*, 2474–2482.

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.

Park, J., & Kim, K. Y. (2017). Meta-modeling using generalized regression neural network and particle swarm optimization. *Applied Soft Computing Journal*, *51*, 354–369.

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

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

Salemi Parizi, M., Gocgun, Y., & Ghate, A. (2017). Approximate policy iteration for dynamic resource-constrained project scheduling. *Operations Research Letters*, *45*(5), 442–447.

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

Satic, U., Jacko, P., & Kirkbride, C. (2020). Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem. *International Journal of Production Research*, 1–13.

Storn, R., & Price, K. (1995). *Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces*: *Technical Report, (TR-95-012)*, (pp. 1–12).

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

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

Zheng, P., Zhang, P., Wang, J., Zhang, J., Yang, C., & Jin, Y. (2020). A data-driven robust optimization method for the assembly job-shop scheduling problem under uncertainty. *International Journal of Computer Integrated Manufacturing*, *00*(00), 1–16.