Scheduling methods for modular shipbuilding

van der Beek, T.

DOI
10.4233/uuid:af383529-1f15-463d-b18b-0ec3e1ed328a

Publication date
2023

Document Version
Final published version

Citation (APA)
van der Beek, T. (2023). *Scheduling methods for modular shipbuilding*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:af383529-1f15-463d-b18b-0ec3e1ed328a

SCHEDULING METHODS FOR MODULAR SHIPBUILDING

SCHEDULING METHODS FOR MODULAR SHIPBUILDING

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 12 december 2023 om 15:00 uur

door

Tom VAN DER BEEK

Master of Science in Offshore and Dredging Engineering,
Technische Universiteit Delft,
geboren te Bergen op Zoom, Nederland.

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. K.I. Aardal	Technische Universiteit Delft, promotor
Dr. ir. J.T. van Essen	Technische Universiteit Delft, copromotor
Dr. ir. J. Pruyn	Technische Universiteit Delft, copromotor
Onafhankelijke leden:	
Prof. dr. M.M. de Weerdt	Technische Universiteit Delft
Prof. dr. ir. B. van Arem	Technische Universiteit Delft
Prof. dr. E. Demeulemeester	Katholieke Universiteit Leuven
Prof. dr. S. Helber	Leibniz-Universität Hannover
Dr. ir. L.J.J. van Iersel	Technische Universiteit Delft, reservelid

Keywords: Modular shipbuilding, Project scheduling, Resource constrained project scheduling problem, optimization.

Printed by: Proefschriften.nl

Front & Back: Astrid Janse, Bureau Janse

ISBN 978-94-6473-319-8

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

CONTENTS

SUMMARY

In shipbuilding, there is a need for faster and more efficient production. Furthermore, quick adaptation of new technologies is desired. One method to potentially achieve these goals, is modular production. Modular production consists of developing a product family that consists of a base platform and several modules. Instead of designing and producing each product as a one-off product, it can be created by combining modules. Although research highlights the potential of this method, there is a lack of quantitative results. Furthermore, modular shipbuilding comes with many scheduling challenges. The solution to these challenges influences the usefulness of modular shipbuilding. Therefore, this dissertation focuses on identifying these challenges and finding good solution methods for them.

In this dissertation, three challenges for scheduling for modular production are considered. The first challenge is the definition and usage of modules. For this, three things have to be considered. First, the required resources, such as cranes, can differ based on whether modules are used. Second, module definition influences the rest of the project. An example is that using a larger module might require that the roof installation has to be postponed until the larger module is installed. Third, since shipbuilding consists of very large projects, each project has room for project-specific solutions, and thus, module usage can differ per project.

The second challenge is inventory management. Due to the reduced production time, the influence of items with long lead times increases. Furthermore, due to the increased standardization of components, inventory costs are spread out over multiple projects, increasing the economic feasibility of keeping components in inventory. Additionally, long-lead items or modules can be replenished in multiple ways, such as in-house production or outsourcing.

Finally, the third challenge is stochastic scheduling. Since modular production results in a product family of similar ships, there is some indication of the structure of the next arriving project. This information can be used to create schedules that also perform well on arriving future projects.

To handle these challenges, the **Resource Constrained Project Scheduling Problem with a flexible Project Structure** (RCPSP-PS) is studied. This scheduling problem consists of a set of activities of which a subset has to be executed. This allows the modeling of modularization choices, such as the use of pre-assemblies or outsourcing. For this problem, we introduce a **Mixed Integer Linear Programming** (MILP) model and develop a solution method based on finding sets of activities with certain execution properties. The first type of sets contains activities of which *at least one* activity is executed and the second type contains activities of which *at most one* is executed. Subsequently, these types of sets are used to add cutting planes and eliminate variables, which in turn is used in an exact solution method. This solution method is then compared against

a state-of-the-art method from the literature, which shows that the proposed solution method performs significantly better. However, as expected, due to the NP-hardness of the RCPSP-PS, many instances remain without a proven optimal solution.

In order to find good solutions for the unsolved instances, two heuristic methods are presented, along with an extension to the RCPSP-PS. This extension is the addition of nonrenewable resources with consumption and production, leading to the **Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources** (RCPSP-PS/CPR). This type of resources allows for modeling resources such as floor space, capital and inventory.

To quickly find feasible solutions to the scheduling problem, the concept of **group graphs** is introduced. With this concept, two heuristic algorithms are developed: A **hybrid differential evolution** algorithm and an **ant colony optimization** algorithm. Both algorithms are compared to a state-of-the-art algorithm from the literature and it is shown that they produce better solutions. Since these heuristic methods find good solutions, even for large instances, they can be used in practice for any case where the exact algorithm does not provide a solution, or where the exact solution method takes too long to provide a good solution. This can be the case due to instance size or due to the limited availability of computing time.

Next, the scope is broadened to the production of a product family instead of a single product. This is done by expanding the RCPSP-PS/CPR to the **Resource Constrained Project Scheduling Problem with Modular construction and new Project arrivals** (RCPSP-MP). This expanded problem includes the stochastic arrival of new projects, as well as the inventory allocation of nonrenewable resources. The latter can, in combination with the flexible project structure, be used to model the pre-assembly of certain modules.

The model consists of a **scenario tree**, which is a representation of multiple scenarios. To find solutions, an MILP model is developed that represents all projects in a scenario tree simultaneously. Furthermore, for larger instances, a heuristic **Progressive Hedging** (PH) algorithm is designed. This algorithm outputs individual solutions and then uses penalties to converge to a single common solution that can be used across all scenarios. To improve the convergence rate, two extensions to the PH algorithm are introduced. These extensions also improve the quality of the derived solutions. With this PH algorithm, inventory allocation and project structure decisions can now be made, while considering future project arrivals. This is an important step in scheduling for modular shipbuilding, as the final goal is to create a profitable product family, instead of a single product.

Finally, stochastic project arrivals are considered for the standard **Resource Constrained Project Scheduling Problem** (RCPSP). Here, instances of the RCPSP arrive sequentially, with overlapping project end and start times. Since these projects use a set of shared resources, the resource usage of a current project influences the next project. The goal here is to schedule each project, while also taking into account expected future projects. For this, **simulation optimization** is used: each objective function evaluation contains a simulation of projects arriving in the future. It is shown that although this creates better solutions than by not looking ahead, it also is very computationally expensive. Therefore, a second method is introduced. This method uses several neural

networks to create an estimator for the objective function value. This requires a lot of time to generate the data and time to train the neural networks, but all of this can be done at non-critical moments in time. Then, at project arrival, instead of using simulation, the estimator is used within an optimization algorithm. This results in computing times that are only a fraction of the computing times of simulation optimization.

Furthermore, when comparing this data-assisted method to the method of not looking ahead, a significant improvement is seen. Thus, it showcases the use of machine learning within optimization for the RCPSP. In practice, this algorithm can be used in a more developed stage of scheduling, where all modularization and outsourcing decisions are already made.

In conclusion, this dissertation presents methods for scheduling modular shipbuilding. These methods consists of flexible project structures, nonrenewable resources, resource allocation and stochastically arriving projects. Besides the use in modular shipbuilding, the abstraction of these methods allow them to be used in many other industries.

SAMENVATTING

In scheepsbouw is er een behoefte aan snellere en efficiëntere productie. Daarnaast is flexibiliteit, ten opzichte van nieuwe technieken, belangrijk. Een mogelijke methode om deze doelen te bereiken is modulaire productie. Dit bestaat uit het ontwerpen van een productfamilie, bestaande uit een basis platform en verschillende modules. Hiermee kan een product gecreëerd worden door modules te selecteren, in plaats van het ontwerpen en produceren van een alleenstaand product. Alhoewel meerdere onderzoekers het potentieel van deze methode aankaarten, is er een gebrek aan kwantitatieve resultaten. Daarnaast introduceert modulaire scheepsbouw meerdere uitdagingen in planning. De oplossingen voor deze uitdagingen beïnvloeden de bruikbaarheid van modulaire productie in scheepsbouw. Daarom richt dit proefschrift zich op het identificeren van deze uitdagingen en het creëren van oplossingsmethodes.

In dit proefschrift worden drie uitdagingen voor planning voor modulaire productie aangekaart. De eerste uitdaging is de definitie en het gebruik van modules. Hiervoor moeten drie factoren meegenomen worden. Ten eerste kunnen de benodigde middelen, zoals kranen, verschillen op basis van het gebruik van modules. Ten tweede beïnvloedt de moduledefinitie de rest van het project. Een voorbeeld is dat het gebruik van een grotere module kan vereisen dat de installatie van het dak moet worden uitgesteld totdat de module is geïnstalleerd. Ten derde, aangezien de scheepsbouw uit zeer grote projecten bestaat, is er bij elk project ruimte voor projectspecifieke oplossingen en kan het modulegebruik dus per project verschillen. De tweede uitdaging is voorraadbeheer. Door de kortere productietijd neemt de invloed van langlopende artikelen toe. Bovendien worden, door de toegenomen standaardisatie van componenten, voorraadkosten uitgesmeerd over meerdere projecten, waardoor het economisch haalbaarder wordt om componenten op voorraad te houden. Bovendien kunnen artikelen of modules met een lange doorlooptijd op verschillende manieren worden aangevuld, bijvoorbeeld door zelf te producren of door productie uit te besteden. Ten slotte is de laatste uitdaging stochastische planning. Aangezien modulaire productie resulteert in een productfamilie van vergelijkbare schepen, is er enige indicatie van de structuur van het volgende project dat binnenkomt. Deze informatie kan worden gebruikt om planningen te maken die ook goed presteren bij aankomende toekomstige projecten.

Om met deze uitdagingen om te gaan, wordt het **Resource Constrained Project Scheduling Problem with a flexible Project Structure** (RCPSP-PS) bestudeerd. Dit is een planningsprobleem dat bestaat uit een set activiteiten, waarvan een subgroep moet worden uitgevoerd. Hiermee kunnen modularisatie-keuzes worden gemaakt, zoals het gebruik van subassemblages of uitbesteding. We introduceren een **Mixed Integer Linear Programming** (MILP) model en een oplossingsmethode, gebaseerd op het vinden van subgroepen van activiteiten met bepaalde uitvoeringseigenschappen. De eerste subgroep bevat activiteiten waarvan *tenminste één* activiteit moet worden uitgevoerd. De

tweede subgroep bevat activiteiten waarvan *maximaal één* activiteit moet worden uitgevoerd. Deze subgroepen van activiteiten worden vervolgens gebruikt om snijvlakken toe te voegen en variabelen te elimineren. Dit wordt vervolgens gebruikt in een exacte oplossingsmethode. Deze oplossingsmethode wordt vervolgens vergeleken met een (state-of-the-art) methode uit de literatuur, waaruit blijkt dat deze beduidend beter presteert. Echter, zoals verwacht vanwege de NP-hardheid van het RCPSP-PS, blijven veel gevallen zonder een bewezen optimale oplossing.

Om goede oplossingen te vinden voor niet opgeloste instanties worden twee heuristische methodes gepresenteerd, samen met een uitbreiding van de RCPSP-PS. Deze uitbreiding is de toevoeging van niet-hernieuwbare middelen met consumptie en productie, wat resulteert in het **Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources** (RCPSP-PS/CPR). Met dit type middelen kunnen middelen zoals vloerruimte, kapitaal en inventaris gemodelleerd worden.

Om snel uitvoerbare oplossingen te vinden voor het planningsprobleem, wordt het concept van **groepsgrafen** geïntroduceerd. Met dit concept worden twee heuristische algoritmes ontwikkeld: een **hybrid differential evolution** algoritme en een **ant colony optimization** algoritme. Beide algoritmes worden vergeleken met een state-of-the-art algoritme uit de literatuur en er wordt aangetoond dat ze betere oplossingen opleveren. Aangezien deze heuristische methodes goede oplossingen vinden, zelfs voor grote gevallen, kunnen ze in de praktijk worden gebruikt voor elk geval waarin het exacte algoritme geen oplossing biedt, of waar de exacte oplossingsmethode te lang duurt. Dit kan het geval zijn vanwege de grootte van de instantie of vanwege de beperkte beschikbaarheid van rekentijd.

Vervolgens wordt de scope verbreed naar de productie van een productfamilie, in plaats van een enkel product. Dit wordt gedaan door het RCPSP-PS/CPR uit te breiden naar het **Resource Constrained Project Scheduling Problem with Modular construction and new Project arrivals** (RCPSP-MP). Dit uitgebreide probleem omvat de stochastische aankomst van nieuwe projecten, evenals de voorraadtoewijzing van niet-hernieuwbare bronnen. Dit laatste kan, in combinatie met de flexibele projectstructuur, gebruikt worden om de voormontage van bepaalde modules te modelleren.

Het probleem bestaat uit een **scenarioboom**, wat een weergave is van meerdere scenario's. Om oplossingen te vinden, wordt een MILP-model gemaakt dat alle scenario's in een scenarioboom tegelijkertijd representeert. Bovendien wordt voor grotere instanties een heuristisch **Progressive Hedging** (PH) algoritme gemaakt. Dit algoritme creëert individuele oplossingen en gebruikt vervolgens strafpunten om te convergeren naar één gemeenschappelijke oplossing die in alle scenario's kan worden gebruikt. Om de convergentiesnelheid te verbeteren, worden twee uitbreidingen op het PH-algoritme geïntroduceerd. Deze uitbreidingen verbeteren ook de kwaliteit van de gevonden oplossingen. Met dit PH-algoritme kunnen nu beslissingen over voorraadtoewijzing en projectstructuur worden genomen, rekening houdend met toekomstige projectaankomsten. Dit is een belangrijke stap in de planning voor modulaire scheepsbouw, aangezien het uiteindelijke doel is om een winstgevende productfamilie te creëren, in plaats van een enkel product.

Tenslotte worden stochastische aankomsten voor het standaard **Resource Constrained Project Scheduling Problem** (RCPSP) onderzocht. Hier arriveren realisaties van het RCPSP achtereenvolgens, met overlappende eind- en starttijden. Aangezien de projecten middelen delen, wordt het volgende project beïnvloed door het middelengebruik van het huidige project. Het doel is dus om het huidige project te plannen, terwijl de verwachte toekomstige projecten in acht worden genomen. Hiervoor wordt **simulatie-optimalisatie** gebruikt: elke doelfunctie evaluatie bestaat uit een simulatie van toekomstige projectaankomsten. Alhoewel deze methode betere oplossingen genereert dan de methode die niet vooruit kijkt, zijn de rekentijden erg lang. Daarom wordt er een tweede methode geïntroduceerd die eerder gesimuleerde data gebruikt. Deze tweede methode gebruikt meerdere neurale netwerken om een schatting te maken van de doelfunctie. Het genereren van deze data en het trainen van de neurale netwerken kost veel tijd, maar kan uitgevoerd worden op niet kritieke momenten. Hiermee kan, zodra een project arriveert, kan een schatting gemaakt worden in plaats van dat de volledige simulatie uitgevoerd moet worden. De rekentijden van deze methode zijn slechts een fractie van die van de volledige simulatiemethode.

Daarnaast laat een vergelijking tussen de methode met neurale netwerken en de methode zonder vooruitkijken een significante verbetering van gevonden planningen zien. Hieruit blijkt het nut van het gebruik van machine learning binnen optimalisatie van de RCPSP. In de praktijk kan deze methode gebruikt worden in een verdere planningsfase, waar alle modularisatie- en uitbestedingskeuzes al gemaakt zijn.

Samengevat, deze thesis presenteert methodes voor planning in modulaire scheepsbouw. Deze methodes bevatten flexibele projectstructuren, niet-hernieuwbare middelen, voorraadtoewijzing en stochastische aankomsten van projecten. Behalve het gebruik in modulaire scheepsbouw kunnen deze methodes, door het niveau van abstractie, ook in andere industrieën gebruikt worden.

ACKNOWLEDGEMENTS

The greatest benefit of doing a PhD at two departments, is having two offices. Although this does not add much practical benefits, it allowed me to feel included in two groups at the same time. It also meant that, even while staying at home for almost two years, I had a great variety of office mates and colleagues. This was a big help in getting my PhD, but even more in enjoying it. When starting my PhD, I only joined the Optimization group once a week. However, I immediately felt at home, except for two bullies called Jongeren van Balthasar, who kept unplugging my computer, adding stuff to my to do list, rearranging my keyboard and more. However, I am thankful for you guys for teaching me the importance of locking your pc and not leaving your belongings unattended. To the other PhDs: Remie, Josse, Ren-fei, Lara, Naqi, Qiaochu, Willem, Bram, Guillermo, Luigi, Nando: thanks for the great times, such as BBQs, parties, trips and more. Esther, thanks for the support when you already had more than enough to handle on your own plate. To Jacopo: thank you for making my PhD, and a lot of events outside of that, fun. You are always the life of the party and I completely understand why Theresia keeps referring to you as her second favorite PhD student. To Yuki: thanks for creating the good old days in the Balpol, it definitely helped getting me through my PhD and all lockdowns. Looking forward to the day that you gamble away your house and savings on stocks and will move in with me and Eline again. Furthermore, I want to thank everyone else in the optimization team, both the academic staff and the support staff.

During my PhD, I was less and less present at the 3ME office, but I still have some great memories there. I want to thank everybody from the maritime department who I shared an office, lunches, or any coffee breaks with. In particular, I want to thank the people who I built a closer connection with. George, thanks for all the advice and fun times, I hope you can build your group exactly as how you want it. Dimitris, my two-times office mate, I always enjoyed our time together. I am sure we will keep in touch and look back on our academic time. Nikos, you were quite often the person I could relate most to in all work related struggles. Thanks for all the fun times and good conversations. I'm finishing up my thesis right now, and I am sure you will soon do the same.

I also want to thank everyone outside of work who supported me or helped me get my mind of things. Football was always a great distraction for me, and I want to thank both teams I played in. I also want to thank everyone in the Krylov Tigers for all the post-match discussions: I am sure my defence will be quite relaxed in comparison.

I also want to thank all non-work friends who supported me during these last four years. Gijs, Gnoj, thanks for all the great halla moments. Rowan, thanks for all the great things you initiated for our group and my stays at Hotel de Nijs. Osta, unfortunately we connected last years on the feeling of stress, but luckily also in the good times that relieved it. Henk, teaching you how to play football was an amazing experience, hope we can continue on it soon. Ruben: you were always a great host at Café de Ijsvogel, looking

forward to many more BBQs. Sylvie, Joyce, Deirdre, thanks for the trips, outings, support and fun over the last 5 years and before. To the Almere group: Sjoerd, Martijn, Quinten, Martin, Tom, Marijke, and Renée thanks for all the great times, and especially thanks to everyone living in Utrecht for the fun times in the last year: Irene, Chris, Floor, and Demi. Casper: thanks for giving me some perspective from outside the academic bubble, not thanks for all the talks about birds. Furthermore, I want to thank Katmanbro for walking across my keyboard and attacking everything on my desk.

I am also very lucky to have had great support from my family, in work-related problems and anything else as well. This support came in the form of always being able to reach out, but also in just having a good time. To everyone that became my family later in life: Wouter, Thomas, Novia, Dorus, Cor, Robert and Denise, thanks for all the good times. To my favourite family member: even though we did not have conversations yet, you already brighten up the whole room. To my dad, thanks for the support. Niek: thanks for always being up for some fun at the family gatherings, such as playing games or having a skelter race. To Sabrina and Kelly, when I felt bad, I would call you two to talk and get support. I never once had the feeling that you were not there for me. Rob: thanks for support when you act serious and for making everyone laugh at all other times. Mom: thank you for everything, not just in the last 5 years, but every day before that.

Although other PhD students, friends and family can be a great support, as a PhD student you have the closest work-based relationship with your supervisors. Due to the interdisciplinary aspect of my thesis, I had two supervisor groups and two (potential) promotors. Hans, thank you for the short time that we worked together, and also thank you for letting me follow the direction that was closer to my passion. Karen, thank you for creating this amazing group with a very personal atmosphere and letting me be a part of it. Jeroen, when we had our first talk about me potentially joining for a PhD, I asked you if I was free to choose my direction. I don't know if you knew back then that you would spend years with someone who wasn't really that interested in ships at all. However, you always knew when to switch between helping me and letting me explore my own interests. Not only am I really thankful for this, I am also quite impressed.

Although my PhD started 5 years ago, my interest in optimization only appeared one year earlier, when I followed a side-course on it. I asked the lecturer, Theresia, to help me and later to supervise my master thesis. There were many reasons to not do this: being a busy professor having a lot of students, having two (and halfway during my thesis, three) children at home, not getting the hours for supervising me... However, you chose to help me and this will have a positive effect on nearly every day of my future life, something of which I will always be thankful. Even more, you were an even better supervisor during my PhD. Both on a personal and on a professional level, you were an amazing support.

Finally, there is one person which I need to thank the most. Eline, you chose exactly the wrong time to live with me: in the final year of my PhD when I was full of stress. However, this was also the time when I could use the most support, which I always got from you. From taking care of me to just being there to talk about things, I could always count on you, even if I sometimes had to tell you to stop supporting me and focus more on yourself. But besides this support, the main thing you did was making every day so much better. Bedankt dat je van elke dag een feestje maakt.

1

INTRODUCTION

Up to the 1970's, Europe was a global leader in all segments of the shipbuilding industry. Since then, low production costs, government subsidies and tax breaks, have caused a shift of the market to Asian shipyards. Specifically, China, South-Korea and Japan have become market leaders in the construction of bulk carriers and container ships. This caused European shipyards to focus on niche markets, such as cruise ships, ferries and workboats, where they could gain a competitive advantage by technological improvements (Petersson et al., 2019).

Many of these niche markets play an important role in the global energy transition, with various examples that can be found in different industries. Passenger transportation is a good example. Due to the short routes and the use of permanent docks, ferries have been touted as one of the ship types most suited for electrical propulsion. However, despite this potential, the European ferry fleet remains old and in need of newer and cleaner vessels. In 2016, it was estimated that the majority of European ferries are older than 20 years (Gagatsi et al., 2016). Considering that commercial ships usually retire after 25-30 years (Stopford, 2008), this means that a large fraction of European ferries are close to replacement. Furthermore, ferry routes are usually close to populated areas, therefore increasing the importance of lower emissions, such as CO_2, NO_x, SO_x and particulate matter. Unfortunately, there are many cases of polluting ferry transport. For example, Vierth et al. (2019) found several cases of comparisons between trucks and roll-on/roll-off transport, where shipping by truck is less polluting than shipping by ferry. In 2015, the EU has imposed a drastic restriction in the allowed emission of sulphur for the Baltic Sea, North Sea and English channel. Regulations like these are pushing the shipping industry to look for cleaner solutions.

Another example would be the offshore wind turbine sector. In order to create a sustainable supply of energy, offshore construction activities play a vital role. For example, it has been estimated that offshore wind energy will grow rapidly in the European Union, from 5.9 GW of installed capacity in 2015 to 95 GW in 2050. Most of this growth will take place between 2020 and 2030 (Commission et al., 2021). The major part of vessels for the offshore wind sector is used for construction purposes, and vessel supply has been

touted as a potential bottleneck for offshore wind farm construction (Athanasia et al., 2012). Furthermore, due to the large increase in size for offshore wind turbines, shipbuilding speed has to keep up with these developments (Poulsen and Lema, 2017).

Finally, the agriculture industry can be seen as a potential important part of the global energy transition. Seaweed agriculture is one of the fastest growing components of the global food production. Furthermore, seaweed farms can act as a CO_2 sink, help reduce emmissions from agriculture and can produce bio-fuels (Vierth et al., 2019). For this upcoming industry, workboats are essential, as they are needed for installation, maintenance and operation.

To keep the European maritime sector flexible enough to adapt to these changes, quick design and production of ships is desired. Furthermore, in light of the global competition, shorter lead times can create a competitive advantage. A promising method to achieve these goals is modular production. In modular production, products are built from smaller parts, called modules, that can be combined to quickly create a variety of products. This method of production has been successful in various industries, such as the automotive and aerospace industries. Therefore, this dissertation explores the concept of modular shipbuilding. Since the goal is to decrease production times and costs, this is done from a scheduling point of view.

In the remainder of this chapter, we give an introduction to scheduling for modular shipbuilding. First, in Section 1.1, we introduce the concept of modularity, discuss how this concept can be applied to production, and explore other industries that incorporate modular production. Next, in Section 1.2, the link to shipbuilding is made. Here, we first give a general overview of the traditional shipbuilding process, after which we give a more detailed description of hull construction and outfitting. After this, we evaluate the current state of modular shipbuilding. In Section 1.3, an introduction to project scheduling given. This is started with a general introduction to mathematical optimization. After this, we present the basic version of the scheduling problem that is studied in this dissertation. Finally, we compare this basic problem to the requirements for modular shipbuilding and identify the gaps in optimization theory. This chapter is concluded in Section 1.4, where an outline of this dissertation is given.

1.1. MODULARITY

The definition of product modularity varies across industry and literature. In this thesis, we use the definition as given in Garud et al. (2009), applicable to a variety of industries:

Modularity is a strategy for organizing complex products and processes efficiently. A modular system is composed of modules that are designed independently but still function as an integrated whole.

To achieve modularity, information is partitioned into **hidden information** and **visible information**. Hidden information is information that only affects the local module, and thus does not need to be communicated between modules. Conversely, visible information affects different modules. Therefore, this information needs to be available across the whole system. Visible information falls into three categories (Garud et al., 2009):

- An **architecture** that specifies which modules will be part of their system and what

their functions will be.
- **Interfaces** that describe how the modules will interact.
- **Standards** for testing and measuring the performance of modules.

The concept of visible and invisible information simplifies the design and management of complex systems. As long as the interfaces are defined concisely, the architecture of the system enables one to understand the system without having to know all the module-based details. Potential benefits from modularity include (Kusiak, 2002):

- Economy of scale.
- Increased feasibility of product or component change.
- Increased product variety.
- Lead time reductions.
- Decoupled risk.
- Easier product diagnosis, repair and disposal.

However, modularity can also incur costs. These include:

- Redundant physical architecture.
- Excessive capability due to designing for the most rigorous application.
- Potential for static product architectures and excessive product similarities.

Therefore, both the modular design and production process require careful consideration to balance these costs and benefits. This challenge is addressed by the New, Advanced and Value-added Innovative Ships (NAVAIS) project from the European Union. The goal of the NAVAIS project is to develop a modular based product family for shipbuilding for ferries and workboats. This dissertation is part of this project, and focuses on modular production to achieve *lead time reduction* and *increased product variety*. To understand how modular production can assist in achieving these goals, the next subsection will introduce the *customer order decoupling point*.

1.1.1. CUSTOMER ORDER DECOUPLING POINT

In order to explain the benefits of modular production, we first explain the concept of the **Customer Order Decoupling point** (CODP). This is defined as the point in the value chain for a product, where the product is linked to a specific customer order (Olhager, 2010). Generally, this points specifies the moment where product specifications get frozen and where inventory is held. As can be seen in Figure 1.1, we distinguish four stages in the process of creating a product: engineering, fabricating, assembling and delivering. The CODP can be located before any of these stages. The latest CODP is **make-to-stock**, where products are built completely to inventory. In the **assemble-to-order** strategy, products are partially produced, but still have to be assembled upon customer order. One step before this is **make-to-order**. Here, products are designed, but the production only starts after they are ordered. Finally, the earliest CODP is **engineer-to-order**. Product with this CODP are both engineered and produced after customer order. A later position of the CODP generally guarantees a shorter lead-time, but also carries a higher risk for the manufacturer. Before the CODP, actions are forecast-driven,

rather than customer order-driven. Naturally, forecasts include uncertainty, and therefore the risk of overstocking or understocking. Furthermore, a late CODP also increases the difficulty of maintaining a varied product portfolio.

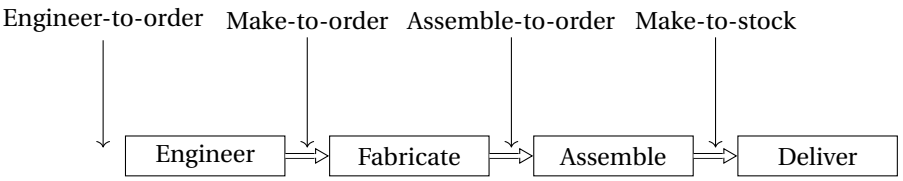


Figure 1.1: Possible customer order decoupling points.

Modular production can be seen as production with an assemble-to-order CODP. Furthermore, it is also possible for a supply chain with a make-to-order CODP to include **modular design**. Here, the designs are made, but do not comprise complete products. Instead, the products are designed consisting of one or more base platforms and multiple modules. Upon customer order, these are combined to quickly create the design of the complete product. This reduces the risk, as a smaller number of sub-assemblies is in stock, and can be used for multiple applications. This reduces stock and also increases turnover rate.

1.1.2. MODULARITY IN PRODUCTION

Usually, the first step in achieving modularity is modularity in design. This envolves creating a design with the use of modules, thus creating a product family that has high product variety with relatively few separate designs. Additionally, modularity can also be extended to the factory floor. This is defined as **modularity in production** (Sako and Said, 1999). In Sako and Said (1999), modularity in production is defined as follows:

Modularity in production is the ability to pre-combine a large number of components into modules and for these modules to be assembled off-line and then brought onto the main assembly line.

Thus, modularity in production implies a dispersed assembly process, distinguishing between pre-assembly and final assembly. Modular assembly allows for a later CODP. This is especially important when product variety is high and lead times are desired to be low (Fredriksson, 2006).

1.1.3. INDUSTRIES

Modular design and production has been introduced in various industries. One of the earliest and most successful industries is that of computers. In 1964, instead of creating a single computer, IBM introduced the system/360. This was a family of computers, along with design rules for the interfaces and architecture of the modular structure. This led to multiple advantages. Firstly, it allowed other manufacturers to design and produce modules, encouraging innovation. Secondly, it was now possible with a relatively small product portfolio to cover both commercial and scientific applications. Finally, it allowed customers to upgrade their product by replacing modules, allowing it to grow with them instead of risking the possibility of outgrowing their investment.

The modular structure became the standard in the computer industry, with various companies improving the concept further. A major step was made by Dell. By using modularity, Dell designed a line of technologically competitive, lower-priced PCs. This was done by creating an assemble-to-order production process, which, combined with outsourcing, 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. In this industry, different parts of the world use different modularization strategies. Western automakers that apply a modular production strategy, are mostly interested in outsourcing. Conversely, Japanese automakers are more inclined towards in-house modular production (Pandremenos et al., 2009). Furthermore, manufacturers in Brazil have implemented the **modular consortium model** (Pires, 1998). In this model, modules are delegated to specific module suppliers, which have the responsibility of assembling their modules directly on the assembly line.

A good showcase of the possibilities of modularity in the automotive industry, is the SMART car (Sako and Murray, 1999). This is a small city car, where most of the production has been outsourced to module suppliers. These suppliers assume responsibility for 40% of the total design cost and more than half of the infrastructure and production equipment costs. Furthermore, the modular design allows for clients to choose a particular variant at the dealership. This variant was then constructed at the dealer from modules in stock within 2 hours (Frigant and Lung, 2002).

Besides the automotive and computer industry, there are numerous other industries that apply modular design and/or production. One of these is the bus manufacturing industry. In this industry, Piran et al. (2021) investigated the reduction in time for a manufacturer after implementing modularity. They found significant reductions in corporate time (48 %), design time (55 %) and production time (30 %). In the construction industry, prefab houses apply modularity in order to obtain benefits of mass production and automation (Neelamkavil, 2009). This involves producing modules for housing off-site in factories and transporting them to the installation site for assembly (Ferdous et al., 2019). The benefits from this process include the reduction of material waste, improved safety, minimization of building time and improved quality (Innella et al., 2019). Besides housing, off-site modular construction has also been investigated for nuclear reactors, where a potential reduce of 38 % in capital costs has been suggested (Wrigley et al., 2021).

Although many benefits of modular design and production are presented, evaluating modularity in other industries also exposed certain risks. Kotabe et al. (2007) evaluates modular production in the Brazilian automobile industry. Although they show multiple obtained benefits, they also state that the hidden information of modules, from the perspective of the assemblers, has potential downsides: it can diminish learning capabilities for the assembler. Furthermore, it can make them dependent on exclusive suppliers. They note that this problem may arise, especially in environments where there is a lack of trust between parties. Furthermore, one must consider that what is best for the industry, is not always best for individual companies. Earlier, we introduced the modular personal computer from IBM: System/360. Although this had significant impact on the industry as a whole, they could not retain their position as market leader. The reason

for this is that although IBM tried to keep control of the production of the critical components, these components were replicated by Compaq. With this, they could order all other modules from IBM's own suppliers and build a similar system with minimal effort (Braha et al., 2006).

In conclusion, there are several cases in other industries where modularity is introduced with success. However, it was also shown that implementing these is not a trivial task and has certain downsides or risks. Therefore, careful analysis is required before implementing modular production. In the next section, we give a general overview of the shipbuilding process and of modularity in shipbuilding.

1.2. SHIPBUILDING

In this section, a description is given of the shipbuilding process and related work in modularization in this area. Initially, the traditional shipbuilding process is presented, starting from the pre-contract phase and ending at the ship delivery. Next, an elaboration is given on the production process and this section is concluded with a literature overview of research related to shipbuilding modularization.

1.2.1. TRADITIONAL SHIPBUILDING OVERVIEW

In this section, a basic overview of the complete shipbuilding process is given. The process starts with the **pre-contract design** (Wei, 2012). The goal of this design is to give sufficient information to both the shipbuilder and client for a technological and economic assessment of the proposed design. After this, the contract is signed and a more detailed design is made. This is made in the following iterative stages (Rose, 2017):

- **Basic design**: This design describes the ship as a total system. It creates an initial solution by a preliminary general arrangement and selecting materials and technology.
- **Functional design**: This design defines each system of the ship systematically.
- **Transition design**: In this stage, the design is reorganized from a design based on system schematics to a design based on physical locations.
- **Work instruction design**: This design has detailed cost estimates and construction drawings and is suitable for production.

After the design phase, the ship production process starts. The first step of producing the ship, is material procurement and component fabrication. In certain shipyards, this starts even before signing the contract for certain long-lead items. Subsequently, the production processes can start. This process follows two lines: that of the steel structure construction and that of the outfitting. The steel structure is built by constructing steel sections and joining these. During both steel structure construction and outfitting, verification work is done to control the quality of production. Later, both lines are described in more detail.

The scheduling of this production process is done on different levels. Initially, the **Master plan** is created. This plan is created during the pre-contract design. This plan

contains completion dates of key milestones, such as contract signing, launching and delivery. Usually, payments from the clients are associated to these milestones. The second planning phase is the **Erection plan**. This is usually created at the end of the transition design, or in the beginning of the work instruction design. The erection plan dictates the planning of when the steel structure sections are joined together. With this plan, the **Section building plan** can be made that determines how each section is created. Diving into further details, the **Outfitting plan** is made. This plan defines the outfitting and painting per section. After the production processes, the ship is commissioned and made ready for sea trials. Finally, the sea trials are executed, after which the ship can be delivered. In the next subsection, the hull construction process is discussed more elaborately.

1.2.2. HULL CONSTRUCTION AND OUTFITTING

The hull of the ship is created by iteratively joining together smaller sub-assemblies to create larger assemblies. This involves the following chronological stages:

- **Panel construction**: During this stage, steel panels are created by joining steel plates with profiles, girders and brackets.
- **Section assembly**: This stage create sections by joining the panels and individual parts. This is the basic construction unit of a ship.
- **Block building**: Assemblies are joined together to form larger blocks.
- **Erection**: In this stage, the blocks and assemblies are joined on a slipway/drydock to form the ship hull.
- **Launching**: In this stage, the ship is launched into the water, by either using a slipway or a drydock.

The outfitting of the ship involves placing all components, machinery and systems. Outfitting and painting can be done at various stages (Wei, 2012). **Pre-outfitting** is the process of mounting components while the sections are being assembled or just after assembly. Usually, sections are painted directly after this. **On board outfitting** involves outfitting that is done during or after hull erection. Finally, outfitting that is done after launching, is defined as **outfitting along the quay**. Pre-outfitting is generally the most effective method of outfitting, since it is performed inside specialized workshops. Outfitting on board can be required due to time constraints, as the erection schedule usually is leading to the outfitting schedule. Furthermore, it can be required due to the nature of the work, such as equipment alignment and connection works. Similarly, this is the case for outfitting along the quay. However, in this case, workers need to bring materials and equipment and often work in difficult positions. Therefore, it is preferred to do most outfitting work as early as possible. The choice of which work to perform at which stage, depends (amongst others) upon the following criteria:

- **Production schedule**: As the outfitting schedule is usually not leading, there is a limit to the work that can be done during each stage.

- **Space accessibility**: As the ship is further assembled, certain spaces become inaccessible. In these cases, it is important to perform the outfitting before this.
- **Lifting capacity**: The lifting capacity is an important constraint on the maximum weight of a ship part, and thus of the amount of outfitted machinery.
- **Material availability**: Delays in material procurement can cause outfitting work to be shifted to later phases.
- **Section building position**: The position of the section determines the suitability for certain outfitting components. For example, if a section is constructed upside-down, components that are close to the ceiling can be outfitted easier.

In modular outfitting, outfitting is done on sub-assemblies that are then installed as a whole in the ship. Example of outfitting systems, are the propulsion system, pumps and piping systems, and Heating, Ventilation and Air-Conditioning (HVAC) systems (Storch, 2007). The degree of work done by the shipyard itself varies. Many European shipyards (Wei, 2012) use *outsourcing* to shift work to subcontractors. This offers various potential advantages: it can alleviate shipbuilder workforce shortfalls, reduce the total costs of building a ship due to reduced overheads, lower wage rates and improved efficiency. Furthermore, it reduces the need of new capital investments. Schank et al. (2005) defines two types of outsourcing: **total outsourcing** and **peak outsourcing**. Total outsourcing subcontracts a complete system (for example, HVAC) or task (for example, painting). Peak outsourcing is capacity related, and occurs when a shipbuilder temporarily does not have enough resources or wants to reduce the workload.

1.2.3. SHIPBUILDING MODULARIZATION

The results of modular production in other industries have drawn attention from maritime researchers. Agarwala (2015) lists potential benefits of modular shipbuilding. These include construction benefits, such as a reduced construction period and higher efficiency from learning. Furthermore, the added value can also be found in ship maintenance (cheaper and faster to upgrade/repair/replace) and ship operation. Additionally, modular construction is not only promising for the shipbuilder and shipowner, but also for the environment (Ančić et al., 2019). Since modules can be designed for easy replacement, ships can be easy to upgrade. This can accelerate innovations such as propulsion by green energy. Furthermore, the option for upgrading certain parts of the ship can extend the lifetime of it, thus decreasing waste of obsolete ships.

The prospects of potential benefits have resulted in various research in modular shipbuilding. Henriksen and Røstad (2014) lists multiple paths for modularization. For the process-wise path, they suggest using standardization and outsourcing as a basis. In their view, this can be a cost effective way of distributing development cost. Outsourcing and standardization are commonly viewed by researches as promising methods for modularization in shipbuilding. However, both the module scope as the module details differ in literature. Rubeša et al. (2011) approach modularization from an outfitting point-of-view. They compare labor costs of outfitting through different phases of the shipbuilding process, such as onboard or in a workshop. Alternatively, Gunawan et al.

(2018, 2020) define modules based on piping arrangement. Based on a design structure matrix, indicating the connections between components, they use optimization algorithms to find modules with a low number of interconnections. Pero et al. (2015) evaluate the effects of modular construction on the supply chain for multiple industries. One case study involves the production of a cruise-ship, where key modules include upper walls, decks, living modules, mechanical systems (engines, navigations, pumps) and service systems (cooling, heating, water supply). They note that due to the lack of standardized interfaces, the integration with suppliers is quite high. Furthermore, modularity in small passenger ships is researched by Vladimir et al. (2018, 2022). They divide the ship into three modules: hull, power system and superstructure.

Despite various research, modularity is sparsely used in the maritime sector (Pfeifer et al., 2020). Some successful implementations are found in the naval sector (Vladimir et al., 2018; Alliff et al., 2016). However, especially in the latter case, the lack of publicly available information inhibits the transfer to other types of vessels. Furthermore, a major naval project to create the **Littoral Combat Ship**, a surface combatant with modular mission packages, has resulted in limited success. Due to various concerns over cost, survivability and effectiveness, the program has been controversial and can be seen as a warning that modular shipbuilding requires careful evaluation of the potential risks (O'Rourke, 2020).

Possible challenges for successful implementation are the increased complexity in planning and need for initial investments (Alliff et al., 2016). Since the size of series are much smaller than in industries such as automotive and aerospace, initial engineering costs are significantly higher per product. Therefore, to obtain the benefits of modular construction, module definition is critical (Agarwala, 2015). However, little is done to quantify the gains and losses of modular construction in shipbuilding. Rubeša et al. (2011) provide a method to estimate the decrease in cost and lead time, based on rules of thumb obtained from historical data. Although this provides a good starting point, these methods currently evaluate practices, instead of the benefits of future practices. Furthermore, due to the rather simplistic scheduling done in these methods, potential benefits due to modular construction can remain hidden.

To review the exact gaps in quantification, the process of implementing a modular production line is discussed in the next section.

1.2.4. IMPLEMENTATION OF MODULAR PRODUCTION

In the preceding text, the concept of modular production is described along with an overview of the shipbuilding industry and modularization efforts in this industry. It can be concluded that, although there is some research indicating the potential of modularization, it is not widespread in this industry. Furthermore, it is shown that there is very little research done in quantifying the benefits of modular production. Since the transformation to modular production requires considerable effort and investment, quantification is essential. Therefore, in this subsection, the requirements for introducing a modular production line in shipbuilding are evaluated.

First, before starting modular production, a product family is required. This product family can be viewed from two sides. From the client perspective, it consists of **design**

modules. These design modules must cover the desired market: a potential customer should have their demands fulfilled by selecting the right set of modules. Conversely, the shipyard uses **production modules** to actually construct the ships. Although the production modules have to be able to construct all possible designs that can follow from combining the design modules, the two types of modules can differ from each other. For example, production modules can consist of smaller sub-assemblies that can be combined into the design modules.

Naturally, the definition of production modules has influence on the project schedule. For example, larger modules require larger cranes, but smaller modules might require more on-board installation work. Therefore, in order to define modules for efficient production, one should consider the effects on project scheduling. Furthermore, due to the size and duration of ship production projects, there is time available for custom and ship-specific production solutions. For example, even though the use of a module would be beneficial, resource (un)availability might mean that for a specific project it is better to use direct construction. Therefore, while defining the modules, one should consider this flexibility during production.

After creating a modular design, the production yards should be prepared for constructing these products. Here, one should consider inventory allocation for the following reason: since modular design reduces design time, there is less time between the arriving product order and the start of building. Therefore, for material and/or components with long lead times, the risk of material procurement becoming a production bottleneck is increased. A potential method for dealing with this, is to keep components or assemblies in stock. In regular shipbuilding, where each product is unique, this carries high risks: if no order for a ship that requires the component/assembly comes in, the component/assembly still has to be stored, financed, and depricated. However, since a modular design has a larger degree of component commonality, this risk is reduced. Thus, because of the increased risk of delivery times becoming a bottleneck and because of the lower risk due to a larger degree component commonality, a modular shipbuilding process requires one to reconsider the inventory policy.

Finally, after the work preparation phase, the production process starts. Here, the definite decisions for scheduling have to be made: which modules to use, should certain work be outsourced, should the inventory be replenished, and so on. Furthermore, while scheduling a project, we have some expectation on the work required for the next project due to the increased standardization of products. To fully capture the benefits of modular production, it might be important to include this knowledge of the future into the current schedule.

In conclusion, both for implementing and executing modular production, there are a lot of specific choices and considerations to be made. The goal of these choices is to create an optimized production process. Although scheduling methods could aid in making these choices, to the best of our knowledge, scheduling theory for modular shipbuilding is still not well developed. However, it would fill an important gap for the successful implementation of modular shipbuilding for two reasons: first, better quantification of the benefits of modular production are needed to justify the large initial investment costs of the implementation. Secondly, smarter scheduling decisions can increase the ben-

efits, thus helping in the economic feasibility of modular production in shipbuilding. Therefore, in the next section, an introduction to optimization theory and, subsequently, scheduling theory is given.

1.3. PROJECT SCHEDULING

Scheduling is the process of constructing a schedule such that every activity or action aligns with the final objective. In mathematical optimization, this is done by transforming a real world problem to a mathematical model. This model consists of an **objective function** and a **solution space**. The solution space is the set of all feasible solutions and the objective function maps each solution to a value, which in turn is minimized or maximized.

If we look at our own calendar, we usually just have to schedule a few activities per day. In normal situations, this does not create many problems. However, scheduling events with a small group already can result in some difficulties. Now, consider a shipyard, where thousands of activities have to be scheduled. Furthermore, ships usually consists of hundred of thousands of parts and manufacturing can take up to several months or even years (Brett et al., 2022). It can easily be understood that this constitutes a difficult scheduling problem, with many possible schedules. In these cases, optimization can help us find the best schedule.

One might wonder if optimization theory and algorithms are really necessary: we have very fast computers that seem to get even faster every year. Can we not just evaluate every possible solution the find the best solution? Although this method, called **complete enumeration**, would definitely solve the problem, it often takes too long to be of any practical use. For example, consider a simple sequencing problem, where we have to find the best order to visit 30 houses. The number of combinations is $30! = 30 \cdot 29 \cdot 28 \cdots \approx 2.65 \cdot 10^{32}$ (30 options for the first house, 29 for the second, 28 for the third, and so on). To put this number in perspective: if we could do 10 billion objective function evaluations per second, it would take $8.41 \cdot 10^{14}$ years to calculate all objective function values. This is around 53,050 times the age of the universe.

Thus, better methods are required. In this section, we first give a general introduction to combinatorial optimization. With this theory in mind, we then present one of the main scheduling problems discussed in the optimization literature. As we will see, this scheduling problem is quite basic and only provides a starting point for the problems encountered in modular shipbuilding. Therefore, we subsequently discuss the shortcomings of this scheduling problem, and thus what is needed to use it for modular shipbuilding.

1.3.1. MATHEMATICAL OPTIMIZATION

As stated above, mathematical optimization involves finding an optimal solution. In this subsection, we give a general introduction to mathematical optimization. We start with the mathematical formulation of the models we consider. Subsequently, we provide some information on the complexity of problems that can be formulated with such a model. Finally, we present methods to find (optimal)) solutions to these problems. For a more elaborate description of some of the theory introduced in this subsection, we refer

to Schrijver (2008) and Cook et al. (1997).

FORMULATION

As stated earlier, mathematical optimization involves finding a solution within a **solution space** that optimizes a certain **objective function**. Therefore, in its most general form, an optimization problem $\mathscr{P}$ can be stated as:

$$(\mathscr{P}) = \begin{cases} \min f(\mathbf{x}), \\ \text{subject to } \mathbf{x} \in \mathscr{X}, \end{cases} \tag{1.1}$$

where $\mathscr{X}$ is the solution space and $f(x)$ is the objective function. A **feasible solution** is defined as any vector $\mathbf{x} \in \mathscr{X}$. Note that the minimization chosen here is without loss of generality, since every maximization function can be written as a minimization problem of the negation of the objective function.

Usually, the solution space is given by constraints. A constraint defines some bound on the solution vector $\mathbf{x}$, for example $g(\mathbf{x}) \leq 0$. If both the constraints and the objective function are linear, we call this **linear programming**. A linear program is in **canonical form** if it is written as:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x}, \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0. \end{aligned} \tag{1.2}$$

Here, $\mathbf{c}$ is the cost vector, A the constraint matrix and $\mathbf{b}$ the constraint vector.

Every constraint in a linear program forms a halfspace. Consequently, the solution space is a polyhedron if the number of halfspaces is finite. Then, for every linear objective function, the following property holds: 'If there is an optimal solution, at least one optimal solution lies at a vertex (corner point) of the polyhedron.' This property is used in the simplex method, which solves most linear programs quickly *in practice*.

In Figure 1.2, it is illustrated in two dimensions that (as long as there is one) there is always an optimal solution at a vertex. In this illustration, the constraints are represented by the black lines, and it can be seen that the feasible region (grey) is indeed a polyhedron. It can be seen that optimizing the linear objective function (red) results in aligning this objective function with the top vertex.

An additional constraint to the linear program can be the requirement that one or more variables have an integer value. If this is the case, we speak of a **Mixed Integer Linear Programming** (MILP) model. This model is very useful in practice, as many decision problems are discrete. Especially, binary variables can represent yes or no decisions, which is quite useful in decision-making. However, as useful as it may be, discreteness complicates the process of finding an optimal solution, as the solution space is no longer a polyhedron, and therefore, the optimal solution does not necessarily lay at a vertex of the polyhedron. This can be seen in Figure 1.2, where an integer grid is shown by the black dots. In the next subsection, we present some basic complexity theory to answer the question: why are some problems 'easy' and some 'hard'?

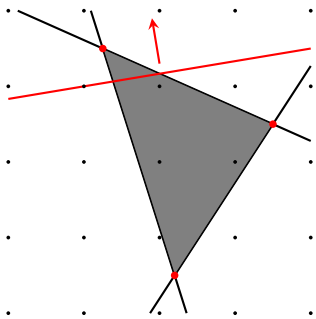


Figure 1.2: A linear objective function in a convex polyhedron always has an optimum at a vertex.

COMPLEXITY

Although the example given at the start of this subsection gives some intuition why certain optimization problems might be difficult to solve, the number of possible solutions does not necessarily define that a problem is difficult. An example is finding the shortest path between two points in a network. Although there are exponentially many different paths that can be taken, Dijkstra (1959) gives a fast algorithm that is guaranteed to find the shortest path. Therefore, this subsection presents an introduction to computational complexity theory, in order to answer the question: when is a problem considered 'easy' or 'hard'?

Simply said, the difficulty of a problem is related to the required resources, generally time and memory, to solve it. As any problem can vary in size, we are not interested in absolute computing times. Instead, we are interested in how the size of the problem scales with the required resources of the solution method. In particular, we are interested if the number of resources scale *polynomially* with the instance size, or not. If the *maximal* computing time needed to solve a problem is a polynomial function of the problem size, it is said that *the problem can be solved in polynomial time*. For example, if t is the maximal computing time and x and y are parameters representing the size of the problem, $t = x^3 + y^7$ means the problem can be solved to optimality in polynomial time, but $t = 4^x + 2^y$ (exponential function) does not. This concept is used to divide problems into complexity classes. Before describing these classes, we note that we consider **decision problems** instead of **optimization problems.** A decision problem is a problem with a yes-no question: "Is there a route that takes less than 20 km?" instead of "What is the shortest route?"

The class of decision problems solvable in polynomial time is denoted by P. An example is finding a path below a certain length between two points, which can be solved with Dijkstra's Algorithm in polynomial time (Dijkstra, 1959). The class NP, standing for *nondeterministic polynomial time*, contains all decision problems where each input with a positive answer has a '**certificate**' from which the correctness can be verified in polynomial time. For example, consider the problem: "Is there a route of less than 20 km that visits a given set of locations?". Then, given a sequence of all locations, we can calculate the length of the route in polynomial time. Therefore, the sequence is the certificate.

Furthermore, we have the class of NP-complete problems. By definition, these are

the hardest problems in NP. A problem in NP is said to be NP-complete if every problem in NP can be 'reduced' to this problem. Reducing a problem $\mathscr{A}$ to a problem $\mathscr{B}$ is defined as a polynomial transformation of an instance of problem $\mathscr{A}$ to an instance of problem $\mathscr{B}$, such that a yes-answer for the instance of problem $\mathscr{B}$ also gives a yes-answer to the instance of problem $\mathscr{A}$, and vice versa. This means that if problem $\mathscr{A}$ is reducible to problem $\mathscr{B}$, any algorithm that solves problem $\mathscr{B}$ in polynomial time does the same for problem $\mathscr{A}$. Therefore, it is said that problem $\mathscr{B}$ is 'at least as hard' as problem $\mathscr{A}$.

Finally, the last complexity class we introduce is the class of NP-hard problems. This group contains all problems, not necessarily in NP, that are 'at least as hard' as all NP-complete problems.

Since every solution to a problem is a certificate, it follows that all problems in P are also in NP. If the converse is true, and thus $P = NP$, is not known, and is a major open problem in theoretical computer science. Therefore, there are two possibilities of the relationship of all classes. In Figure 1.3, we see both the possibility for $P = NP$ and for $P \subsetneq NP$.

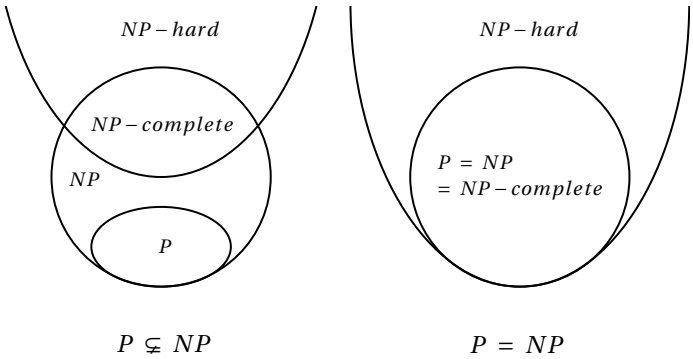


Figure 1.3: Complexity classes, depending on whether $P = NP$ or not (excluding the empty language and its complement, which belong to P but are not NP-complete).

EXACT SOLUTION METHODS

After discussing the computational complexity, we now present some basic **exact solution methods** to optimization problems. An exact solution method for an optimization problem is defined as a method that is guaranteed to find the optimal solution. This has two implications: first, we find the optimal solution and, second, we *know* that this indeed the optimal solution. The first method that we present is the **simplex method**.

The simplex method was designed by Dantzig (1951) and solves most LP problems quickly in practice. It uses two properties of LP problems. The first one is that if there is an optimal solution, there always is an optimal solution in one of the vertices of the solution space polyhedron. The second property is that if, given a solution at a vertex, none of the adjacent vertices has a solution with a better objective value, the solution at the vertex is optimal. By using this, the simplex algorithm traverses from vertex to vertex, until there is no improvement possible. In that case, the solution is optimal.

In practice, the simplex method can be used to quickly find an optimal solution to most LP problems. There are LP problems for which the simplex method requires exponential time, but there exist other algorithms that solve these problems in polynomial time. However, no polynomial time algorithm has been found yet to solve general MILP problems. Therefore, MILP problems are usually much harder to solve than most LP problems.

One of the most common methods for solving an NP-hard MILP problem is the **Branch and Bound** (B&B) method (Land and Doig, 1960), although this might take exponential time. In this method, a **search tree** is used to break up the solution space into smaller and smaller parts. A search tree is a graphical representation, consisting of **nodes** and **edges**. Each node represents (a part of) the solution space. The root node represents the complete solution space. Each node can be **branched** upon, which splits up the solution space represented by the considered node into two parts represented by two new nodes. Furthermore, **bounding** is used to discard nodes that can be proven to not contain an optimal solution. We now illustrate the algorithm for a minimization problem. In this problem, we assume that all variables have to be integer, resulting in an **Integer Linear Program** (ILP).

The B&B method iteratively selects a node that has not yet been investigated, and evaluates the problem represented by that node. For this problem, the **LP-relaxation** is solved. This is the same problem, without restricting variables to integer values. There are now three options:

1. The LP-relaxation is infeasible.
2. The solution of the LP-relaxation only has variables with integer values.
3. The solution of the LP-relaxation has variables with non-integer values.

If the relaxed problem does not have a feasible solution, neither has the integer problem. This means that the node cannot be branched upon any further. When a node is not branched upon further, this is called **pruning** a node. In the second option, the optimal solution to the relaxed problem is integer. This integer solution gives an **upper bound** on the objective value of the *complete* integer problem. The smallest upper bound is maintained globally, and is used to prune all nodes for which the objective function value of the LP-relaxation of the corresponding problem is equal to or larger than this upper bound, since in these nodes no better integer solution can be found than given by the globally best upper bound. Finally, the last option is that the LP-relaxation has a non-integer solution. If the objective function value of this solution is lower than the best found integer solution, further branching is required. Otherwise, this node can be pruned. Branching is done by selecting a variable x_i with a non-integer value in the LP-relaxation x_i^* to branch on. Then, two new nodes are created, one with the constraint $x_i \leq \lfloor x_i^* \rfloor$ and one with $x_i \geq \lceil x_i^* \rceil$. Since x_i is required to be integer, this does not remove any feasible integer solutions.

The B&B method starts this process at the root node and iteratively evaluates nodes until all leaves of the tree have been pruned. Since nodes are only pruned when further branching will not result in a better solution, the best found integer solution is thus the

optimal solution. If the algorithm terminates without finding a feasible integer solution, which happens if all leaf nodes are pruned because of infeasibility of the LP-relaxation, it means that the integer problem is infeasible.

For each node, the choice whether the solution space has to be branched on further, depends on the lower and upper bounds and on the feasibility of the linear relaxation of the resulting solution space. Therefore, the quality of the bounds are vital to the performance of the B&B algorithm. For this reason, a lot of research is done on how to improve these bounds. One method is to change the formulation of the ILP. Often, the same problem can be expressed in various forms of variables and constraints. Although the optimal solution value will be the same, the performance of the B&B algorithm might be significantly different. A specific method of modifying the ILP, is to add additional constraints. We define additional constraints, added with the purpose of improving the lower bound, as **valid inequalities**. These constraints are added to remove a part of the solution space of the linear relaxation, while not removing any integer solutions.

HEURISTIC SOLUTION METHODS

In the previous section, we described how the simplex method and B&B algorithm can solve LP's and ILPs. For MILPS, the same B&B algorithm can be used, while only branching on variables that are required to be integer. However, especially for (M)ILPs, many of these problems are NP-hard. This means that there is currently no known algorithm that solves these problems to optimality in polynomial time. Thus, even though improved lower bounds can decrease the computational requirements to solve these problems, there is currently no guarantee of finding an optimal solution within a reasonable time. Therefore, we often have to settle for finding *good feasible*, instead of *optimal*, solutions. This is done by **heuristic** methods. A heuristic algorithm for an optimization problem is defined as an algorithm that tries to find a good solution, but does not give a *guarantee of optimality*. Note that it is definitely possible for these algorithms to find the optimal solution. However, this will not always happen, and even if it happens, we are not able to prove optimality in general. Fortunately, giving up the guarantee of optimality usually results in significantly less computing time, often just a fraction of the computing time for exact methods.

We divide the heuristics into three categories: classic heuristics, metaheuristics and hyperheuristics. Classic heuristics are problem specific and often close to how an actual person might solve this problem. An example is the well known nearest neighbor algorithm to solve the Traveling Salesman Problem (TSP), as described in Cook et al. (1997). The TSP asks the question: "Given a list of locations and the distances between each pair of locations, what is the shortest possible route to visit all locations precisely once and return to the starting location?" The nearest neighbor algorithm, as its name implies, finds a solution by moving from one location to another, each time selecting the closest non-visited location. Classic heuristics have the benefit of being easy to implement and easy to explain to users. However, other types of heuristics often outperform these simpler types.

The second type of heuristics are metaheuristics. These heuristics are not problem-specific and, thus, can be used on many problems. These algorithms try to explore the

solution space in an efficient way, to find (near-)optimal solutions. Many of these heuristics are based on **local search**: a local search algorithm tries to find a good solution, by iteratively modifying an existing solution. Given a solution **x**, the set of solutions that can be reached by this modification step is called the **neighborhood**. If the objective value of solution **x** is better than the objective value of all solutions in the neighborhood, it is called a **local optimum**. If a solution is at least as good as all other solutions in the solution space, it is called a global optimum. This is illustrated in Figure 1.4.

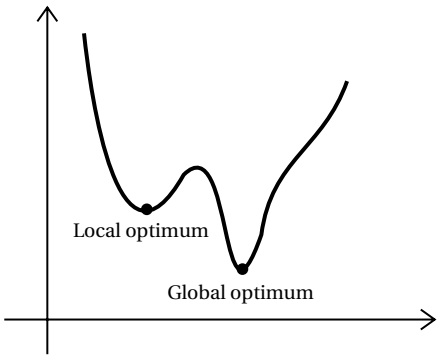


Figure 1.4: An example of a local optimum and a global optimum.

Reaching a local optimum is quite easy. However, unfortunately, a lot of combinatorial problems have solution spaces with many local optima that are not global optima. Therefore, one of the main goals of metaheuristic design is to escape these local optima. For example, the local search based algorithm **simulated annealing** (Kirkpatrick et al., 1983) occasionally accepts solutions even if they have worse objective values than the current one. In Figure 1.4, it is illustrated how this might work. If we are at the local optimum, there are no better neighboring solutions, and thus, we get stuck if we are only looking for those. However, if we temporarily would accept worse solutions, we might 'climb out' of the local optimum and find the global optimum.

Search based metaheuristics can either consist of a single agent or of a population of agents. **Single agent methods**, such as simulated annealing or tabu search (Glover, 1977), have a single agent exploring the solution space. Conversely, **population-based** heuristics keep a population of agents, each representing a solution. These agents simultaneously explore the solution space and influence each other. Often, these algorithms are inspired by nature. Examples are genetic algorithms (described in, e.g., (Katoch et al., 2021)) that simulate evolution through natural selection, inheritance and mutation, and Ant Colony Optimization (Colorni et al., 1991), where the foraging behavior of ants is mimicked.

The final category of heuristics is called **hyper heuristics**. In Gendreau and Potvin (2019), a hyper heuristic is defined as "an automated methodology for selecting or generating heuristics to solve computational search problems". By selecting, combining or adapting several components of heuristics, it is possible to create new algorithms. Often, this is done by incorporating machine learning techniques.

1

1.3.2. RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

There are many different optimization problems that involve scheduling and/or planning of activities. We now introduce a basic project scheduling problem to give an idea of the type of problems considered in this thesis. In shipbuilding, both precedence constraints and resource availability are important. These characteristics are captured in the **Resource Constrained Project Scheduling Problem** (RCPSP) (Pritsker et al., 1969). In this section, we present the standard version of this problem. The RCPSP is a well researched scheduling problem and will serve as the basis for the methods presented in this dissertation. It consists of a set of activities that have to be scheduled, according to precedence and resource constraints. The goal is to find a schedule of minimal **makespan**; the duration of the project.

PROBLEM FORMULATION

We now formulate the RCPSP. The RCPSP consists of a set of activities N that have to be scheduled. The first activity 1, and the last activity $|N|$, are dummy activities. They have a duration of zero and indicate the start and end of the project, respectively. Each activity $i \in N$ has a duration d_i and has to be scheduled according to the precedence relationships P: a precedence relationship $(i, j) \in P$ imposes that activity j can only start after activity i is finished. These precedence relationships can be represented by a **scheduling graph**, as shown in Figure 1.5. Here, each node represents an activity and each edge a precedence relationship. Thus, for example, the edge from node 0 to 2 represents that activity 2 has to start after activity 0.

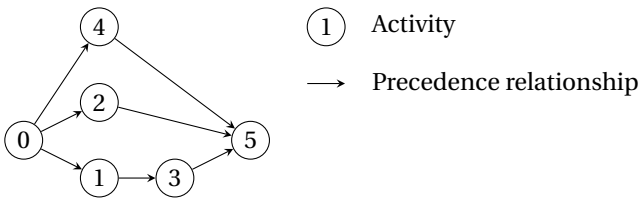


Figure 1.5: Scheduling graph for example instance of the RCPSP.

Furthermore, the activities have to be scheduled according to resource constraints. The set of resources is denoted by R. These resources can represent things such as machinery, workers, workshops etc. The total availability of each resource $r \in R$ is λ_r. This means that at any given time, the maximal resource usage of all running activities using resource $r \in R$ cannot exceed λ_r. Activity $i \in N$ uses k_{ri} units of resource $r \in R$ during its execution. Furthermore, we introduce the set of timesteps $T = \{1, \cdots, |T|\}$. The timesteps form the planning horizon: all possible times at which we can schedule the activities.

With this notation, we present an MILP formulation, as given in Pritsker et al. (1969). For this, we use the binary variable X_{it} for each activity $i \in N$ and timestep $t \in T$. If activity $i \in N$ starts at time $t \in T$, we set $X_{it} = 1$ and, otherwise, $X_{it} = 0$. Then, Objective function (1.3a) defines the objective. Since $X_{|N|t}$ is only equal to one if the final activity $|N|$ starts at time t, the sum in Objective function (1.3a) computes the starting time of final activity N. Thus, by minimizing this, the makespan of the project is minimized.

Next, we discuss the constraints, which define that each solution X represents a feasible schedule. The first set of constraints, Constraints (1.3b) impose that each activity starts execution at exactly one timestep. Secondly, the precedence relationships are imposed by Constraints (1.3c). Here, for each precedence relationship $(i, j) \in P$, the left-hand side of the equation increases by $t' + d_i$, where t' is the starting time of activity i. Thus, it represents the ending time of activity i. The right-hand side of the equation represents the starting time of activity j. Therefore, Constraints (1.3c) ensure that for each precedence relationship $(i, j) \in P$, activity j starts after or at the ending time of activity i. Constraints (1.3d) make sure that each schedule X satisfies the resource constraints. The left-hand side represents the total consumption of resource $r \in R$ at time $t \in T$. The outer sum sums over all activities, and the inner sum makes sure to select only activities that are still being executed. This is then set to be less than or equal to the total resource availability λ_r. Finally, Constraints (1.3e) define that the variables can only take on values of zero and one.

$$\min \sum_{t \in T} t X_{|N|t}, \tag{1.3a}$$

$$\sum_{t \in T} X_{it} = 1, \quad \forall i \in N, \tag{1.3b}$$

$$\sum_{t \in T} (t + d_i) X_{it} \leq \sum_{t \in T} t X_{jt}, \quad \forall (i, j) \in P, \tag{1.3c}$$

$$\sum_{i \in N} \sum_{u=1}^{d_i} k_{ri} X_{i(t-u+1)} \leq \lambda_r, \quad \forall r \in R, t \in T, \tag{1.3d}$$

$$X_{it} \in \{0, 1\}, \quad \forall i \in N, t \in T. \tag{1.3e}$$

EXAMPLE

We now give a small example of an RCPSP instance and two possible solutions, based on the scheduling graph given in Figure 1.5. We consider an instance with a single resource: $R = \{1\}$ with capacity $\lambda_1 = 2$. For each activity $i \in N$, the duration d_i, resource requirement k_{1i} and precedence successors are given in Table 1.1.

Table 1.1: Example instance of the RCPSP.

i	d_i	k_{1i}	Successors
0	0	0	$\{1, 2, 4\}$
1	1	2	$\{3\}$
2	2	1	$\{5\}$
3	3	1	$\{5\}$
4	1	1	$\{5\}$
5	0	0	$\emptyset$

Since activity one and two require three units of resource one in total, it follows that they cannot be scheduled simultaneously. This gives at least two feasible schedules, as

shown in Figure 1.6. Here, the total resource usage is shown on the y-axis, along with the maximum resource capacity. It can be seen that by scheduling activity one first, activity two and three can be carried out simultaneously, resulting in a makespan of four. Conversely, if activity two is executed first, the precedence relationship between activity one and three implies that activity three cannot start before activity one finishes. This results in a suboptimal schedule, with a makespan of six. Therefore, it can be seen here that by making different scheduling decisions, the same project can either be executed in four or six time units.

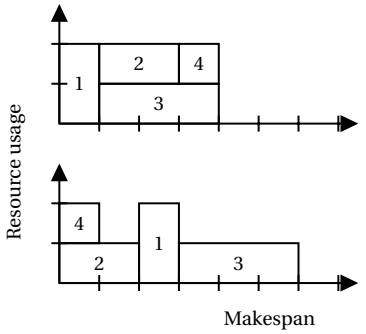


Figure 1.6: Solutions for the example instance of the RCPSP.

EXTENSIONS

Although the RCPSP captures the basics of a lot of scheduling problems quite well, it can be somewhat simple for a lot of real world applications. Therefore, a lot of research done for the RCPSP is done on extensions of the RCPSP. In this subsection, we give a very brief overview of research done in extensions of the RCPSP. Since this is such a well researched area, the overview that we present is not aimed to be exhaustive, but is given with the purpose of giving some idea of the kinds of extensions. Later in this thesis, extensions related to modular shipbuilding are given in more detail.

In Hartmann and Briskorn (2010), an overview is given of extensions and variants of the RCPSP. They divide these extensions into five categories: *Different activity concepts*, *different temporal constraints*, *different resource constraints*, *alternative objectives* and *multiple objectives*. For different activity concepts, they list, among others, preemptive scheduling and multiple modes. Preemptive scheduling is the possibility of interrupting activities at certain points in time. Multiple modes refers to the possibility of executing an activity in different ways, with varying durations and resource requirements. Examples of varying temporal constraints are the inclusion of minimal and maximal time lags between activities, or the addition of deadlines. Different resource constraints can, for example, include nonrenewable resources that do not automatically renew after using them. Another example is resource capacities varying with time. Furthermore, alternative objectives are a common extension to the RCPSP. For example, instead of minimizing the final makespan, cost based objectives are possible. Often, this is done on the net present value: the cost discounted for the time value of money. Finally, multiple projects can be considered, often with a combined objective function.

Besides variations to the *deterministic* RCPSP, many researchers have included uncertainty to the RCPSP (Herroelen and Leus, 2005). One source of uncertainty are stochastic activity durations. Other research includes rescheduling with the random arrival of new activities.

SOLUTION METHODS

Both exact and heuristics methods have been developed to (heuristically) solve the RCPSP. Exact methods usually consist of Branch and Bound algorithms, often based on an MILP model. To improve performance, different formulations can be used. In Koné et al. (2013), various MILP formulations are compared. These formulations can either be in **discrete time** or **continuous time**. In discrete time formulations (e.g., Constraint set (1.3)), the activities have to be scheduled in discrete units of time with a decision variable linking each activity to its timestep. In continuous time formulations, there are no discrete timesteps. Often, continuous variables are used to determine the starting time of activities, along with additional binary variables to impose the precedence sequence and/or resource constraints. Alternatively, an event based formulation is possible. Here, activities are modeled as pairs of events: a starting event and an ending event. These events are then ordered in time, and the activities are linked to the events with binary variables.

Another method to improve the performance of the solution algorithm, is to add constraints/valid inequalities that cut off infeasible solutions. A fast and effective method to impose bounds on the starting time of activities, is by using the **Critical Path Method** (CPM). We define the length of a path in the scheduling graph as the sum of the durations of all activities encountered on that path. The **critical path** to a given activity is then the longest path from the start to that activity. For example, in Figure 1.5, there are three paths to activity 5: $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$, $0 \rightarrow 2 \rightarrow 4 \rightarrow 5$ and $0 \rightarrow 4 \rightarrow 5$. The length of these paths is four, two and one, respectively. Therefore, the critical path length is four, and forms a lower bound on the starting time of activity 5.

Due to the NP-hardness of the RCPSP, heuristic algorithms are often needed in practice. An important consideration here is the schedule representation. Unfortunately, formulating a schedule by linking each activity directly to the executing times, as is done in most MILP formulations, is unpractical for use within (meta)heuristics. Therefore, **Schedule Generation Schemes** (SGSs) are usually used. An SGS converts a certain solution representation to a schedule.

Since most SGSs cannot represent every schedule possible, we now present a classification of schedules (a more elaborate description can be found in, for example, Artigues et al. (2008)). As introduced earlier, a feasible schedule is a schedule that satisfies the precedence and resource constraints. Now, we introduce the concept of a **left shift**. A left shift on an activity $i \in N$ of a *feasible* schedule S transforms it into a *feasible* schedule S', by setting the starting time for activity i at least one timestep earlier. If this shift can be done by sequential left shifts of *one timestep*, the shift is called a **local left shift**. If this is not true, since certain intermediate left shifts would result in an infeasible schedule, it is called a **global left shift**. In Figure 1.7, an example of both types of left shifts is shown. Here, activity 4 can be moved at most 2 timesteps earlier using only left shifts of one timestep, as shown in the top-right. After this, another left shift of one timestep

would result in an infeasible schedule, due to activity 1. Therefore, the schedule in the bottom-right can only be obtained by a *global left shift* on the initial schedule.

A schedule for which no local left shift is possible is called a **semi-active schedule** and a schedule for which no global left shift is possible is called an **active schedule**. There always exists an optimal schedule that is active. If for any activity, no local or global left shift would be possible even if **preemption** (splitting up activities in time) was allowed, the schedule is called a **non-delay** schedule, according to the classification for the RCPSP given in Sprecher et al. (1995). It follows that every non-delay schedule is active and that both non-delay schedules and active schedules are semi-active. An optimal solution is also active, although not necessarily non-delay.

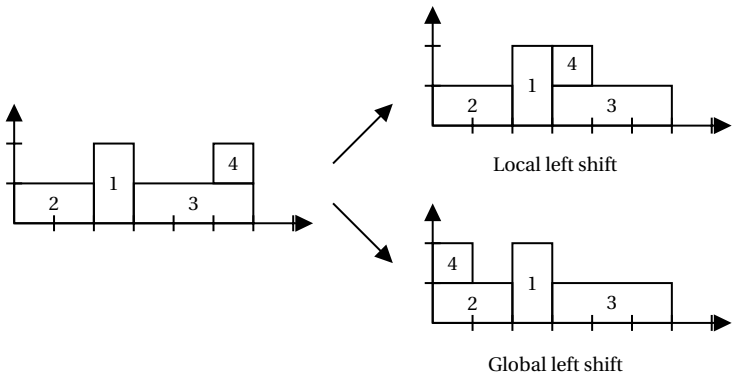


Figure 1.7: Local and global left shift for a schedule of the example instance given in Table 1.1.

The classification of schedules is useful, since a solution representation now only has to represent active schedules. Usually, this is done by representing the schedule by a sequence of activities, a so called **activity list**. This activity list is then converted to a schedule using an SGS. This is done by starting with an empty schedule, and iteratively setting the starting times for activities. The SGS decides, based on the activity list, the order of activities and how the starting times are set. There are two main types of SGSs: **serial SGSs** and **parallel SGSs**. In a serial SGS, in each iteration, the next activity in the activity list is scheduled at the earliest feasible time possible. A parallel SGS iterates over the timesteps in ascending order. In each iteration at time t, a second iteration is done over all unscheduled activities in the order of the activity list. Each activity is then scheduled at time t, unless it causes infeasibility. A serial SGS can represent any active schedule, whereas a parallel SGS can only represent non-delay schedules.

To construct an initial schedule to be used in a heuristic, a priority rule can be used (Kolisch and Hartmann, 1999). A priority rule is a mapping that assigns each activity $i \in N$ a priority value, where at each decision moment the activity with the highest (or lowest) priority value is selected. An activity list can then be constructed by iteratively considering all precedence feasible unscheduled activities, and adding the one with the highest (or lowest) priority value. Often, to further improve these schedules, improvement metaheuristics are used, such as local search or genetic algorithms.

1.3.3. SCHEDULING FOR MODULAR SHIPBUILDING

After having presented the basic RCPSP, we now discuss this problem in the context of modular shipbuilding. Specifically, we discuss the shortcomings of the RCPSP in order to handle the difficulties and opportunities that arise when considering modular shipbuilding.

DEFINING AND USING MODULES

One consideration is the design phase. For modular shipbuilding, this switches from designing a single product to designing a product family. This increases the significance of *design for production*: any investment made here is now returned for all ships in the product family, instead of a single ship. In this design, the definition of modules, and possibly submodules, plays a very important role.

When defining several candidate modules, two things have to be considered. Firstly, the use of a module has influence on the rest of the production process. For example, if we choose to construct a large module A in a workshop, we have to make sure that there is still enough physical space to place this module at the location in the ship. Therefore, we have to make sure that this location is not yet enclosed, for example by delaying the roof construction. Consequently, modules also influence each other. To continue the example, if we have to hold off constructing the roof of the location of module A, we might even gain more benefit if we also use module B located at the same space. However, if A and B require similar workshop resources, using them together can introduce delays.

Secondly, even though modular shipbuilding has the potential of gaining certain benefits of serial production, each ship still requires a large construction network with a lot of room for project-specific solutions. Therefore, it should be considered that even if a ship is designed with modules, it might be beneficial to not use a certain module at certain times. For example, this can be due to capacity limitations in workshops, delivery time of ordered components, or due to outsourcing availability. Not considering these project-specific deviations would result in unrealized potential of reductions in makespan and/or costs. Additionally, even when it is decided that a module is used, there are various options on how to use this. For example, the construction could be outsourced or even built to stock before the customer order arrives.

To evaluate certain designs for production, it is important to perform simulations to gain an insight into the results of modularization decisions. One method of using the RCPSP in these simulations is by creating multiple projects, each representing a certain configuration of modularization choices. This configuration consists of which module to use, but also how to exactly use it (outsource, build in house, construct of pre-ordered sub-assemblies). Therefore, the number of possible configurations increases very quickly. This makes evaluating all configurations unrealistic. However, calculating the schedules for only some configurations has the risk of missing potential benefits. The same happens when an actual ship order comes in. Although in this case, human experts might give some indication of which modules to use and how to use them, it still constitutes a difficult scheduling problem. Using intuition or expert judgment on this carries the risk of executing sub-optimal schedules. Therefore, instead of defining many variants of the RCPSP, realizing the full potential of modular shipbuilding requires a method of optimally choosing the modularization configurations and scheduling these.

INVENTORY MANAGEMENT

The quickest way of incorporating any module, is by not having to construct it when needed, but by simply taking a pre-assembled module from inventory. Similarly, by having certain long-lead items (for example, the ship engine) in inventory, ordering times can be eliminated. Unfortunately, having these items in inventory incurs significant costs, and is therefore not always financially feasible. First, there is the storage cost of these items. Storage halls have to be rented, the items have to be insured, employees are required for managing the storage facilities, etc. Secondly, there are the financing costs: keeping an item in inventory requires this item to be purchased earlier than required, which binds up capital. Finally, there are the depreciation costs: holding an item in the inventory, carries the risk of the value of this item to decrease over time and even of the item becoming obsolete.

Thus, although keeping items in the inventory potentially decreases makespan, it also carries certain costs. In modular shipbuilding, however, a higher level of component commonality across products causes these costs per ship to be lower. Since ships are more similar and thus often use the same component, the expected time in inventory for a component is relatively low. However, the exact inventory amount is not simple to determine: not only does it depend on the components required for future project arrivals, it also depends on the scheduling decisions made in these future projects. For example, consider a component that is usually required in the 6th month of the ship production process. If the delivery time is 5 months, there is no need of having this item in the inventory. However, if, due to other scheduling decisions, the process is sped up and the component is required after 4 months, it would be beneficial to have this item in the inventory. Therefore, inventory management is linked to scheduling decisions. The basic RCPSP only considers a fixed set of resources that automatically become available after use. Therefore, the RCPSP would have to be extended to consider inventory allocation decisions.

STOCHASTIC SCHEDULING

The goal of modular shipbuilding is not to quickly or cheaply produce a single ship, but to quickly or cheaply produce a series of ships. As these ships consists of a similar basic platform with varied modules, there is a lot of commonality between projects. This creates potential for improvements in scheduling. Since a lot of costs, such as fixed contracts for workers or rent for the facilities, are constant, having an (partially) unused shipyard is not cost effective. Therefore, there is usually some project overlap in the shipyard: the next project starts before the current project is completely finished. Therefore, resource and modularization decisions influence future projects. Potentially, this can result in better schedules: if we know that certain resources are critical at the start of a project, it makes sense to consider this while scheduling other projects. However, the basic RCPSP does not support uncertainty: all information is fixed at the beginning of the project. Therefore, by including stochasticity, schedules can be made that are just not well-performing on the current project, but are also beneficial to other projects.

1.4. DISSERTATION OUTLINE

In Section 1.3.3, the shortcomings of the RCPSP for modular shipbuildings are introduced. This thesis is built on the premise that a successful modular design has to consider production, and therefore, scheduling. Therefore, this thesis aims to bridge the gap between the current state of scheduling research and the state required to evaluate modular designs and schedule these in practice.

In Chapter 2, we focus on the different modularization choices. We introduce an extension to the RCPSP that handles flexible project networks, in order to represent the various construction paths that have to be taken due to different choices in modularization. With this new model, we also give an exact solution method, based on decreasing the solution space. We evaluate the performance of this model and compare it against a model from the literature.

Subsequently, in Chapter 3, we further extend this method to include *nonrenewable resources with consumption and production*. This allows the modeling of characteristics such as inventory, floor capacity and capital requirements. Furthermore, we further analyse the flexible project structure introduced in Chapter 2 and give two metaheuristics to find good solutions for this extended problem.

Then, in Chapter 4, uncertainty is introduced. In this chapter, we do not consider a single project, but a series of projects where it is uncertain which project arrives at what time. In this stochastic problem, we optimize the total costs, while determining the inventory allocation, modularization, outsourcing and scheduling decisions.

Chapter 5 also introduces uncertainty, but in a different setting. It considers the basic RCPSP with arriving projects and tries to schedule these such that the resource usage is beneficial to projects arriving in the future. The aim of this chapter is not only to evaluate if this optimization is possible: we introduce a data-assisted method that uses data from earlier optimization processes to create a machine learning assisted optimization algorithm that uses a fraction of the time of the original algorithm.

Finally, this thesis is concluded in Chapter 6. Here, we reflect on the methods introduced, both from an optimization and a shipbuilding point of view. We summarize the achievements, discuss the shortcomings and indicate the work to be done to use these methods in practice.

2

EXACT SCHEDULING WITH A FLEXIBLE PROJECT STRUCTURE[1]

In Chapter 1, the concept of modular shipping is introduced along with its potential benefits. One of these benefits is a reduction in project makespan: using a smart modularization in production potentially can result in shorter production times. However, it was also discussed that this modularization is not trivial. In order to capture the benefits of modular production, it is required that the modularization choices are made carefully and that production is accounted for while evaluating different options. Therefore, in this chapter, we introduce an extension to the RCPSP that allows for better modeling of modular shipbuilding, in order to realize the potential benefits.

This extension is called a **flexible project structure**. In the basic RCPSP, all activities have to be executed. When considering which modules to use, the selection of modules differs per chosen modularization: using a module requires different activities that use different resources. For example, installing components in situ might be done by simply carrying these components manually to the location. However, when they are preassembled, a crane is required. Furthermore, the precedence relationships also might vary, based on modularization choices. If a module is used, roof installation might have to be delayed until after module installation.

The most important decision is whether to use a module. However, there are decisions to make within each module. Should it be outsourced? If so, it requires less capacity at the yard, but one might have to consider longer delivery times. If transportation is done by resources such as a ship, additional outsourcing might use the same transportation resource and thus improve efficiency. Furthermore, the decisions of what exactly to use as a sub-assembly and what not might have impact on the total makespan of the project. As these decisions are often yard and project specific, we do not aim to give an answer to the question of how to create a modulary design for production. Instead, a

[1] This chapter is reproduced from the paper submitted for peer-review to the European Journal of Operations Research (ISSN 03772217).

scheduling model is given that allows for the optimization of decisions such as modularization and/or outsourcing. The aim of this model is to capture flexibility as general as possible. As is discussed in the remainder of this chapter, this is done by modifying an existing model in literature.

After a scheduling problem is created, the goal is to find good solutions for it. Therefore, in the second part of this chapter, a solution method is given to find optimal solutions. This is done by analyzing the problem, proving mathematical properties of it, and using these properties to decrease the solution space. At the end of this chapter, the proposed methods are evaluated.

2.1. INTRODUCTION

The Resource Constrained Project Scheduling Problem (RCPSP) is an optimization problem aimed at scheduling activities. It comprises a list of activities that have to be scheduled, while satisfying a list of precedence constraints and resource availability constraints. The problem aims to minimize the makespan of the project and is used in various applications, such as assembly scheduling or employee scheduling (Artigues et al., 2008). The problem was proven to be *NP*-hard by Blazewicz et al. (1983) and much research has been done in finding heuristic methods (Pellerin et al., 2019) to solve it. Additionally, generalizations of the RCPSP have been studied by many in great detail (Hartmann and Briskorn, 2010).

While the RCPSP assumes that all activities have to be executed, this is not always required. In many applications, like housing construction (Servranckx and Vanhoucke, 2019), highway project construction (Wu et al., 2010), modular shipbuilding (Rubeša et al., 2011) and aircraft turnaround scheduling (Kellenbrink and Helber, 2015), there are multiple ways of completing a project. This results in an RCPSP that can be completed by executing only a subset of all activities. This is called the **Resource Constrained Project Scheduling Problem with a flexible Project Structure** (RCPSP-PS). The RCPSP-PS consists of two sub-problems. First, the decision has to be made which subset of activities to execute. This is called the **activity selection problem**. Secondly, a schedule has to be made with these selected activities, which gives rise to the **activity scheduling problem**.

In large assembly projects, the choice of which activity to execute next is often associated with flow of components and/or material. If these components, e.g. a ship hull, cannot be split up, the choices can be exclusive; only one alternative can be selected from a subset of activities. This is what we call the **exclusivity criterion**, which complicates the activity selection problem as is shown in this chapter. Furthermore, to better represent project scheduling in reality, two properties are often required. The first is the **separation of scheduling and selection logic**. Furthermore, we introduce the concept of **independent succession** (IS): a choice to execute a certain activity, can be forced by multiple other activities independently instead of a single activity.

An example can be given from modular shipbuilding, where a ship is produced by combining multiple construction modules. For each module, we have two options: construct it locally with available materials, or ship it from another yard. Shipping of modules can be combined, such that we only need to execute one shipping activity for all modules. Therefore, if at least one module is shipped, the activity 'shipping modules'

has to be executed and finished before installing the modules. To model this, we require both the separation of scheduling and selection logic, and IS.

Although multiple papers introduce various models for the RCPSP-PS, there has not been a combination of the exclusivity criterion, independent succession and separation of scheduling and selection logic. Furthermore, there is little research on cutting planes and related exact methods for the RCPSP-PS. To fill this gap, this chapter presents a new model for the selection logic, based on modifying the model of Kellenbrink and Helber (2015) to add independent succession. This allows for a simplification of the model, such that no distinction has to be made anymore between optional activities and activities that are always executed. Furthermore, an exact solution algorithm is given. This algorithm uses a variable reduction method based on the **Critical Path Method** (CPM) (Zhou et al., 2013) and two types of cutting planes. The proposed solutions methods are then evaluated on restricted instances against the model of Kellenbrink and Helber (2015), and against each other on non-restricted instances.

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

2.2. LITERATURE OVERVIEW

The RCPSP is a classical optimization problem, introduced by Pritsker et al. (1969) and proven to be NP-hard by Blazewicz et al. (1983). Since then, numerous studies have been focused on developing heuristic and exact solution methods. An overview of this research can be found in, for example, Pellerin et al. (2019) and Lombardi and Milano (2012). These two review papers discuss heuristic and exact methods, respectively. In this section, we focus on exact solution methods for and generalizations of the RCPSP.y, we de

The standard RCPSP, without generalizations, is often solved by MILP solvers or constraint programming solvers (Herroelen and Leus, 2005). One way of reducing the solution space in both methods is variable reduction. For the RCPSP, this is often based on the *Critical Path Method* (CPM) (Rayward-Smith, 2001), which defines the earliest time an activity can start, based on precedence constraints, while ignoring resource constraints. Another way of reducing the solution space is by finding better lower bounds. Stronger lower bounds for MILP problems can reduce the number of branch-and-bound nodes needed to be explored. For constraint programming, optimization can be done by iteratively proving (in)feasibility for varying makespans. Stronger lower bounds can decrease the number of iterations in this process.

Various researchers developed cutting planes for the RCPSP. Brucker and Knust (2000) provide bounds based on constraint propagation and linear programming. The constraint propagation method keeps track of the minimal durations between activities and iteratively updates this based on the precedence constraints. The linear programming method relaxes the precedence and non-preemption constraints. Baptiste and Demassey

(2004) use the same relaxation and provide additional bounds based on so-called energetic reasoning and weak versions of preemption and precedence. Energetic reasoning for the RCPSP is defined by comparing the demand of resources within a certain time interval with the supply of resources. Hardin et al. (2008) provide a class of valid inequalities for the RCPSP with a single resource. These are based on covers; subsets of activities that cannot be executed all simultaneously due to limited resource availability. Furthermore, they give sufficient conditions under which these inequalities are facet-defining and provide lifting procedures. Other lower bounds for the RCPSP are given by Haouari et al. (2012), who provide three classes of lower bounds based on energetic reasoning and an efficient way of generating these.

A generalization of the RCPSP that enables different ways of executing activities is the *Multi-Mode Resource Constrained Project Scheduling Problem* (MRCPSP) (Talbot, 1982). In this problem, each activity has multiple execution modes with varying durations and/or resource usage. An overview of variations and solution methods can be found in Węglarz et al. (2011). For the MRCPSP, various exact methods are developed. Zhu et al. (2006) provide a branch-and-cut algorithm that uses both reduction of variables based on constraint-propagation, and cutting planes based on resource conflicts and precedence relations. Araujo et al. (2020) propose two new models for the MRCPSP and introduce preprocessing cuts based on feasible subsets: sets of job-mode combinations that can be executed simultaneously. Furthermore, they give a branch-and-cut algorithm that uses five types of cutting planes in parallel.

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

Kellenbrink and Helber (2015) separate the scheduling requirements from the precedence requirements, and give a model based on a set of choices and a distinction between optional and mandatory activities. They call this model the RCPSP-PS and solve it heuristically using a genetic algorithm. Each activity has a set of activities it can cause to be selected. Furthermore, the model includes a time-indexed MILP formulation that imposes some restrictions on the selection logic: it is not possible for multiple activities to have the same activity in the set of activities it can cause to be selected. This restricts the modeling process. Tao and Dong (2017) introduce the *RCPSP with alternative activity chains*, where they give a single network that defines both the precedence constraints and the selection constraints based on AND-activities and OR-activities. An AND-activity is an activity for which all successors have to be executed and an OR-activity is an activity for which at least one successor has to be executed. By using a single network for both the precedence and selection constraints, separation of precedence and selection logic is not possible; every precedence relationship is equal to a

Table 2.1: Overview of models with a flexible project structure

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

selection relationship and vice versa. This means that problems where the precedence and selection constraints do not coincide cannot be modelled with this approach. They solve this model heuristically using a simulated annealing based algorithm. In Tao and Dong (2018), they extend this to multiple objectives. Furthermore, Servranckx and Vanhoucke (2019) present the *RCPSP with alternative subgraphs*. This is a model based on alternative subgraphs and branches; a branch consists of a set of activities, and an alternative subgraph is a collection of branches of which exactly one has to be executed. This model also has a single network for both precedence and selection logic and is solved heuristically by a tabu search procedure. Finally, Hauder et al. (2020) use a network with different types of activities (OR, AND and OUT) to model both the selection and scheduling problem, while adding the extension of supporting multiple projects. Besides the standard objective of makespan minimization, they also consider time balance and resource balance as objective functions. They provide time indexed MILP formulations for these models and a constraint programming method.

An overview of all models is given in Table 2.1. It can be seen that there is not yet a model combining separation of scheduling and selection, independent succession and the exclusivity criterion. Furthermore, most of the focus is on heuristic methods. Although three papers include an exact method, for two of these it comprises of only an MILP formulation. Although there are differences in these MILP formulations for the flexible project structure, they all share a similar time-indexed basic RCPSP model.

2.3. RCPSP-PS

In this section, the problem is formalized. First, Section 2.3.1 gives a description of the problem, which includes a description of the selection graph with selection groups. Finally, Section 2.3.2 gives an MILP formulation for the RCPSP-PS.

2.3.1. PROBLEM DESCRIPTION

The RCPSP-PS consists of a set of activities N of which a subset has to be executed while minimizing the makespan of executing the selected activities. Let $n = |N| - 2$ be the number of non-dummy activities. Then, the starting activity is activity 0 and the final activity is activity $n+1$. Both of these activities have a duration and resource requirement of zero and the final activity can only be executed after all other executed activities. The time horizon during which these activities are scheduled is represented by a set of discrete time periods T. Each activity $i \in N$ has a duration of d_i time periods. Activities have to be scheduled while satisfying resource, precedence and selection constraints. The set of resources is denoted by R. Each resource $r \in R$ has a capacity of λ_r, and each activity $i \in N$ uses k_{ri} units of resource r across the whole duration. The precedence relationships are defined by a set of tuples $\mathscr{P}$. For each $(i, j) \in \mathscr{P}$, it is required that activity i is finished before the start of activity j. These relationships can be represented in a graph by creating a node for each activity and a directed edge for each precedence relationship. This graph is called the **precedence graph**. Furthermore, the set $\mathscr{P}_j$ contains all predecessors of activity j and the set $\mathscr{S}_i$ contains all successors of activity i in the precedence graph.

In the RCPSP-PS, only a subset of activities has to be executed. To define the choices on the selection of activities, the concept of **selection groups** (denoted by set G) is introduced. A selection group $g \in G$ consists of an **activator activity** a_g and a set of one or more **successor activities** S_g. If an activator activity is executed, exactly one of the successor activities has to be executed, which means that a selection group defines an 'exclusive or'-relationship. Additionally, we define the set of **selection groups with full precedence** $H \subseteq G$. This set contains all selection groups with a time precedence relationship between the activator and all successors, i.e., $H = \{g \in G : (a_g, j) \in \mathscr{P}, \forall\, j \in S_g\}$. Furthermore, we define a **unit selection group** as a selection group $g \in G$ with only one successor (i.e., $|S_g| = 1$). This defines a direct consequential relationship; if activator activity a_g is executed, the single successor activity in S_g will have to be executed as well. An 'and'-relationship can be modeled by multiple unit selection groups.

The selection groups can be represented by a **selection graph**. This is a graph in which each activity is represented by a node and the selection groups are represented by directed edges. A single edge denotes that the source and target activity belong to a unit selection group. Multiple edges with an arc between them indicate a non-unit selection group, where the source activity is the activator and the target activities the successor activities. This is illustrated in Figure 2.1.

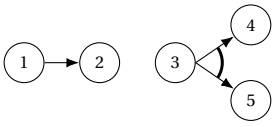


Figure 2.1: A unit selection group $(a_g, S_g) = (1, \{2\})$ (left) and a non-unit selection group $(a_g, S_g) = (3, \{4, 5\})$ (right).

The precedence and selection relationships split up the RCPSP-PS into two problems. The first one is selecting which activities will be executed. This is called the **se-**

lection problem. The next question is when to schedule the executed activities. This is defined as the **scheduling problem**.

An example of a precedence graph and a selection graph is given in Figure 2.2. The selection graph in this example contains 5 selection groups, with $(0,\{1,2\})$ being the only non-unit selection group. This imposes that if activity 0 is executed, either activity 1 or activity 2 has to be executed. Furthermore, if activity 0 is executed, activity 3 has to be executed as well. Finally, for activities 1, 2 and 3, it follows that if one of them is executed, activity 4 has to be executed as well.

The precedence graph is very similar, and thus, activities 1, 2 and 3 can only start after the end of activity 0, and activity 4 can only start after the end of activities 1, 2 and 3, only considering executed activities. Furthermore, since there is a directed edge between activity 2 and 3, activity 3 can only start after activity 2 is finished, if both are executed. If activity 2 is not executed, activity 3 can start directly after finishing activity 0. Since for every selection relationship, there is a time precedence relationship as well, all selection groups have full precedence ($G = H$).

In Kellenbrink and Helber (2015), the selection logic was mainly modeled by **choices**, **activities causing a choice**, and **optional activities per choice**. Respectively, these are analogous to selection groups, activators and successor activities. The difference between these concepts is that selection groups support IS. Furthermore, to simplify the model, the concept of **activities caused by another activity** in Kellenbrink and Helber (2015) is not included the presented model, but can be modeled by unit-selection groups instead. This was done to simplify both modeling and mathematical analysis for cutting planes, which are presented in Section 2.4. Finally, this chapter does not consider nonrenewable resources. This was done since the focus of the proposed solution method is the flexible project structure. Since nonrenewable resources are not a standard component of the RCPSP, we exclude this extension from the model.

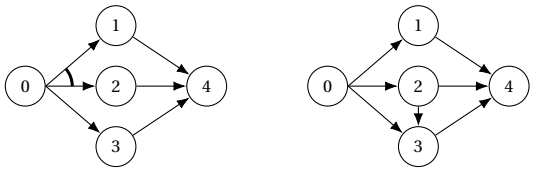


Figure 2.2: Example of graphs.

2.3.2. MODEL

To model the problem, we introduce binary decision variables X_{it} that are equal to one if activity $i \in N$ starts at time $t \in T$, and zero otherwise. Objective function (2.1a) minimizes the completion time of the final activity, and thus, the total project makespan. The first and final activities are always executed due to Constraints (2.1b) and (2.1c), respectively. Note that Constraints (2.1c) can also follow from the selection groups. Constraints (2.1d) impose that each activity can only be executed once. Furthermore, Constraints (2.1e) make sure that if activator activity a_g of selection group $g \in G$ is executed,

at least one successor activity $i \in S_g$ has to be executed. Constraints (2.1f) impose that if activator a_g of selection group $g \in G$ is executed, at most one successor activity is executed. The precedence constraints are set by Constraints (2.1g). These constraints define that for each $(i, j) \in \mathscr{P}$, if both are executed, the starting time of activity i plus its duration d_i cannot be larger than the starting time of activity j. Furthermore, Constraints (2.1h) define that for each resource $r \in R$ and time $t \in T$, the total resource usage is smaller than or equal to the resource capacity λ_r. Finally, Constraints (2.1i) specify that the decision variables X_{it} are binary.

$$\min \sum_{t \in T} t X_{(n+1)t}, \tag{2.1a}$$

$$\sum_{t \in T} X_{0t} = 1, \tag{2.1b}$$

$$\sum_{t \in T} X_{(n+1)t} = 1, \tag{2.1c}$$

$$\sum_{t \in T} X_{it} \leq 1, \quad \forall i \in N, \tag{2.1d}$$

$$\sum_{t \in T} X_{a_g t} \leq \sum_{i \in S_g} \sum_{t \in T} X_{it}, \quad \forall g \in G, \tag{2.1e}$$

$$\sum_{j \in S_g} \sum_{t \in T} X_{jt} \leq |S_g| - \big(|S_g| - 1\big) \sum_{t \in T} X_{a_g}, \quad \forall g \in G, \tag{2.1f}$$

$$\sum_{t \in T} (t + d_i) X_{it} \leq \sum_{t \in T} t X_{jt} + M\big(1 - \sum_{t \in T} X_{jt}\big), \quad \forall (i, j) \in \mathscr{P}, \tag{2.1g}$$

$$\sum_{i \in N} \sum_{u=1}^{d_i} k_{ri} X_{i(t-u+1)} \leq \lambda_r, \quad \forall r \in R, t \in T, \tag{2.1h}$$

$$X_{it} \in \{0, 1\}, \quad \forall i \in N, t \in T. \tag{2.1i}$$

In the next subsection, we further discuss the activity selection problem.

2.3.3. ACTIVITY SELECTION PROBLEM

After introducing the model, we now give a formal definition for the activity selection problem. Recall that the exclusivity criterion is the criterion that per selection group with an executed activator activity, exactly one successor has to be executed. With this, the **selection problem** (imposed by Constraints (2.1d) to (2.1f)) is defined as follows: given a selection graph, find a selection of activities including activity 0, such that for each selection group $g \in G$, exactly one activity j from the set of successor activities S_g is selected if activator activity a_g is selected. This problem is proven to be NP-hard by Barták et al. (2007).

Although the exclusivity criterion of the selection group is important for certain real-world applications, there are also cases in which the exclusivity criterion does not hold and where more than one successor can be executed per selection group. In these cases, there is an 'at least one'-requirement instead of an 'exactly one'-requirement. This can

be modeled by adding dummy activities (activities with no duration and no resource requirements) before each successor activity, as is demonstrated in Figure 2.3.

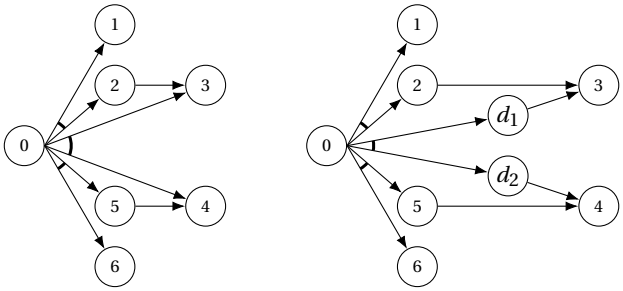


Figure 2.3: Instances where from activities 3 and 4, exactly one (left) and at least one (right) activity has to be executed.

On the left side in Figure 2.3, selection group $(0, \{3,4\})$ is shown with the exclusivity criterion; either 3 or 4 can be executed. However, activity 3 and 4 can also be activated by either activity 2 or 5. In some applications, it would be prefered to allow either activity 3 or 4, or both. This is the case in the right-hand side of Figure 2.3. Here, the exclusivity criterion is moved to the dummy activities d_1 and d_2. This means that for activity 3 and 4, at least one has to be executed instead of exactly one.

Note that dummy activities only have to be added for successor activities that are also successor activities of other groups. To demonstrate this, consider a selection group $g \in G$, which should be transformed such that from the successor activities S_g, at least one has to be executed instead of exactly one. Then, a dummy activity only has to be added before each successor activity $j \in S_g$, if j is also a successor of another selection group. In Figure 2.3, this means that if there is no arc from activity 5 to activity 4, dummy activity d_2 is not required. The proof for this is given in Lemma 1.

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

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

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

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

Step 4 *Remove selection group g.*

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

First, apply the algorithm with $S'_g = S_g$. Now, Constraints (2.1e) and (2.1f) impose for selection group h that if a_h is executed, *exactly* one dummy variable has to be executed. Consequently, Constraints (2.1e) impose that *at least* one activity $i \in S_g$ has to be executed, since activity $i \in S_g$ can also be executed when the activating dummy activity D_i is not executed.

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

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

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

Firstly, we consider the case where activity $i \in S_g \setminus S'_g$ is not executed in solution X. By Constraints (2.1e), it follows that dummy activity D_i is also not executed. Therefore, removing D_i and selection group h_i does not cause any infeasibilities and Y remains a feasible solution for $\mathscr{B}$.

Secondly, consider the case where activity $i \in S_g \setminus S'_g$ is executed. If dummy activity D_i was executed, the number of executed successor activities for selection group h stays the same in solution Y, which therefore remains feasible for problem $\mathscr{B}$. If D_i was not executed, which is possible considering Constraints (2.1e), there is an infeasibility in problem $\mathscr{B}$ for group h in Constraints (2.1f). However, since activity i is not the successor of any other group, it can be set to not executed. This does not cause any infeasibilities because Constraints (2.1e) impose in the direction of activator to successor and not in the reverse direction.

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

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

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

Now, consider the case where activity $i \in S'_g \setminus S_g$ is executed. In this case, none of the dummy activities are executed and dummy activity D_i can be set to executed in solution

X to obtain a feasible solution. Similar as for the reverse case, the objective activity $n+1$ is not changed, and therefore, the objective value remains equal.

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

2.4. SOLUTION METHOD

In this section, we present an exact solution method for the RCPSP-PS based on preprocessing and subsequently using an MILP solver. The preprocessing procedure uses a variable reduction method and cutting planes. The variable reduction method is based on the *Critical Path Method* (CPM) and is described in Section 2.4.1. This method sets earliest start times and latest finish times for the individual activities. Subsequently, in Section 2.4.2, we identify properties of the selection problem. With these properties, we give two types of cutting planes. Finally, in Section 2.4.3, the solution algorithm is given. The solution algorithm uses a combination of variable reduction and cutting planes to increase the earliest start times. Subsequently, it solves this preprocessed problem by using a MILP solver.

2.4.1. VARIABLE REDUCTION

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

The variable reduction method is based on what we call a **Non-Empty Execution Set** (NEES): a set of activities of which at least one has to be executed. Variable reduction can then be done based on the following principle: if each activity in a NEES A has the same successor activity j in the precedence graph, then the earliest start time s_j of activity j is equal to $\min_{i\in A}\{s_i + d_i\}$. Similarly, if each activity in a NEES A has the same predecessor activity i, the latest finish time f_i of activity i is equal to $\max_{j\in A}\{f_j - d_j\}$

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

For the variable reduction method, we need to know whether a candidate set $N' \subseteq N$ contains a NEES. For intuition, consider all activities as the candidate set ($N' = N$) in the selection graph as shown in Figure 2.4. We traverse the selection graph, starting in the root activity r. Then, a set of activities A is a NEES if, regardless of the choices we make in the selection groups, we always end up in an activity in A. Therefore, if we reach a certain group g that is on the path to A, there should still be a path to A no matter what

successor we pick. In Figure 2.4, no choice can be made at selection group g_1 such that no activity in A will be executed.

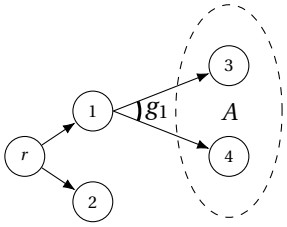


Figure 2.4: Selection graph example of a NEES. Since either activity 3 or activity 4 always has to be executed, $\{3,4\}$ is a NEES.

To identify a NEES in a subset of activities $N' \subseteq N$, we solve the MILP given by Constraint set (2.2). This MILP selects selection groups and activities, where all selected selection groups form a set of paths, starting at activity 0. As long as each path ends in a selected activity, the set of selected activities form a NEES.

To formulate this, we introduce the set G_i that contains all groups $g \in G$ with activator $a_g = i$. Furthermore, we introduce binary variables U_g for each $g \in G$ and V_i for each $i \in N'$, which are equal to one if a selection group or activity is selected, respectively, and zero otherwise.

With this, Constraints (2.2a) imposes that at least one selection group has to be selected at starting activity 0, which is the start of all paths. Then, the paths are continued due to Constraints (2.2b) and (2.2c). These impose that if a selection group g is selected, for each successor $i \in S_g$ either a succeeding group $h \in G_i$ is selected (path continues), or activity i itself has to be selected (path ends in activity i). If activity $i \in N'$, this is imposed by Constraints (2.2b). Otherwise, binary variable V_i does not exists and one group $h \in G_i$ has to be selected. This is imposed by Constraints (2.2c). Note that, since Constraint set (2.2) is used to *identify* a NEES, V does not have to be bounded from above, since adding any activity to a NEES still constitutes a NEES.

$$\sum_{g \in G_0} U_g \geq 1, \tag{2.2a}$$

$$U_g \leq V_i + \sum_{h \in G_i} U_h, \quad \forall g \in G, i \in S_g \cap N', \tag{2.2b}$$

$$U_g \leq \sum_{h \in G_i} U_h, \quad \forall g \in G, i \in S_g \setminus N', \tag{2.2c}$$

$$U_g \in \{0,1\}, \quad \forall g \in G, \tag{2.2d}$$

$$V_i \in \{0,1\}, \quad \forall i \in N'. \tag{2.2e}$$

We can now show that if V is a feasible solution to Constraint set (2.2), it constitutes a NEES.

Lemma 2
Let $N' \subseteq N$ be a subset of activities and U, V be the solution of Constraint set (2.2). Then, A given by $\{i \in N' : V_i = 1\}$ is a NEES.

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

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

Start with $B = \{0\}$. Since the source activity 0 always has to be executed, it is a NEES. Furthermore, due to Constraints (2.2a), it also satisfies Property 2.

For the induction step, assume that set B satisfies both properties. For each activity $i \in B$ with $V_i = 0$ (and thus $\sum_{g \in G_i} U_g \geq 1$ by Property 2), let set C_i contain all successor activities of i that satisfy Property 2; $C_i = \{j : \exists g \in G_i | j \in S_g \wedge V_j + \sum_{h \in G_j} U_h \geq 1\}$. Since $\sum_{g \in G_i} U_g \geq 1$ by assumption, there is at least one group $g \in G_i$ with $U_g = 1$. Then, by Constraints (2.2b) and (2.2c), it follows that all successors of this group satisfy Property 2 and are therefore included in set C_i. Thus, if i is executed, at least one activity in C_i is executed. We now create $\bar{B}$ by replacing i by C_i; $\bar{B} = B \cup C_i \setminus \{i\}$. Since B is a NEES, so is $\bar{B}$. Thus, $\bar{B}$ satisfies both properties. Finally, take $\bar{B}$ as the new B.

For each iteration, each activity is replaced by its successors unless $V_i = 1$. Since the graph is acyclic, the iterative process will terminate, in which case $V_i = 1$ for every $i \in B$, since at some point G_i will be empty. As shown, both properties are maintained in this process. Therefore, the final set B is a NEES. Because A contains all activities $i \in N'$ with $V_i = 1$, this means that $B \subseteq A$. Since B is a NEES, which means that at least one activity in B has to be executed, then any superset of B is also a NEES. Therefore, A is a NEES as well and since A results from a solution to Constraint set (2.2), this solution constitutes a NEES. □

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

The first loop in Algorithm 1 loops over all activities in topological order. This ensures that s_i is defined for every predecessor $i \in \mathscr{P}_j$. Furthermore, it defines the sequence $B = \{b_1, \cdots, b_{|B|}\}$. This sequence contains all activities $i \in \mathscr{P}_j$, ordered by earliest finishing time $s_i + d_i$ in descending order. Loop 2 then takes incremental subsets B' of the ordered set of predecessors $i \in \mathscr{P}_j$ and tries to solve Constraint set (2.2) with $N' = B'$.

For each iteration of Loop 2, an element k is added to $N' = B'$ until Constraint set (2.2) becomes feasible. If adding element k to N' results in Constraint set (2.2) becoming feasible, it follows that $V_k = 1$. Otherwise, Constraint set (2.2) would also be feasible in the previous iteration. Since B' is ordered in non-increasing order of $s_i + d_i$, in the previous iteration, B' contained all activities $i \in B$ with $s_i + d_i \geq s_j + d_j$. Since solving Constraint set (2.2) in the previous iteration did not give a feasible solution, B' maximizes $\min_{i \in B'}\{s_i + d_i\}$.

Something similar can be done for the latest finish time f_i for all activities $i \in N$. The difference is that f_i is initially given a value of $|T|$ for every activity and the algorithm runs backwards. This means that Loop 1 is reversed, B contains all successors of j and

Algorithm 1 Preprocessing algorithm

```
N^(s) ← topological sorting of N on precedence graph
s_i ← 0  ∀  i ∈ N
for all j ∈ N^(s) do                                   ▷ Loop 1
    B ← P_j, sorted by non-increasing s_i + d_i.
    stop ← False
    n ← 1
    while stop = False AND n ≤ |B| do                  ▷ Loop 2
        B' ← first n elements of B
        Solve Constraint set (2.2) for N' = B'
        if Constraint set (2.2) is feasible then
            s_j ← min_{i∈B'}{s_i + d_i}
            stop ← True
        end if
        n ← n + 1
    end while
end for
```

is ordered in non-decreasing order of $f_i - d_i$. Furthermore, instead of updating s_j, f_j is updated to $\max_{i \in B'}\{f_i - d_i\}$. Finally, after both preprocessing steps, all variables X_{it} with $t < s_i$ or $t > f_i - d_i$ can be set to zero.

2.4.2. CUTTING PLANES

Due to Constraints (2.1g), the linear relaxation of the proposed MILP is very weak. To strengthen the formulation, we present a simple type of valid inequalities and two types of cutting planes. The valid inequality is added for each selection group with full precedence $g \in H \subseteq G$ and is shown in Constraints (2.3), which simply states that if a selection group has full precedence, each successor activity has to be executed later than the activating activity, if both are executed.

$$\sum_{t \in T} (t + d_{a_g}) X_{a_g t} \leq \sum_{j \in S_g} \sum_{t \in T} t X_{jt}, \quad \forall g \in H, \tag{2.3}$$

Furthermore, two types of cutting planes are presented. The first type is based on groups of activities of which *at least* one has to be selected. For the second type, groups of activities of which *at most* one can be selected are used.

The procedure is the same for both cutting plane types: first, the linear relaxation is solved to obtain the relaxed solution. Subsequently, the separation problem is solved to obtain a cutting plane that cuts off the relaxed solution. This cutting plane is added to the original problem and a new solution is obtained. This is repeated until no more cutting planes can be found or when the objective has not increased for a fixed number of iterations.

NON-EMPTY EXECUTION CUTTING PLANES

In this subsection, a method is presented for which cutting planes are generated based on **Non-Empty Execution Sets** (NEES). Recall that these are sets in which at least one activity has to be executed. The separation problem is given by Objective function (2.4) subject to the constraints of Constraint set (2.2), in order to find a NEES with less than one executed activity in the relaxed solution. Here, X^*_{it} is the solution obtained by solving the LP relaxation and V_i is the decision variable from Objective function (2.4) indicating whether an activity $i \in N$ is in a NEES.

$$\min \sum_{i \in N} V_i \sum_{t \in T} X^*_{it}. \tag{2.4}$$

By Lemma 2, the index set of V forms a NEES. Thus, if the value of Objective function (2.4) is smaller than 1, the fractional solution X^* contains a NEES that has less than one activity executed. Therefore, we add Constraints (2.5) as a cutting plane, where A is the index set of V; $A = \{i \in N : V_i = 1\}$,

$$\sum_{i \in A} \sum_{t \in T} X_{it} \geq 1. \tag{2.5}$$

MAX-ONE CUTTING PLANES

The second type of cutting planes is based on **Max-One Execution Sets** (MOES). We call cutting planes generated by this method '**max-one cutting planes**'. This method requires selection groups with full precedence. A MOES $N' \subseteq N$ is a set of activities of which at most one will be executed in the optimal solution. To identify these sets, **Common Rooted Paths** (CRP) are used as defined in Definition 2.4.1. For this, we introduce the notation of the **vertex sequence** of a path P; the sequence of all vertices on a path P, denoted by $\mathcal{V}(P)$.

In the remainder of this section, we give a definition of a CRP and show that if there is no CRP between two activities, at most one of these activities will be executed. Subsequently, we show how to identify all CRPs in the selection graph. Finally, these CRPs are used to generate a new type of cutting planes.

Definition 2.4.1 (Common rooted path (CRP))
For two activities i and j, it is said that they have a common rooted path (r, P, Q) if there is another activity r with a path P from r to i and a path Q from r to j with the following properties: the first activities p_1 and q_1 on P and Q after r, respectively, do not belong to the same selection group of activator r. Furthermore, after splitting at r, paths P and Q are disjoint; $\mathcal{V}(P) \cap \mathcal{V}(Q) = \{r\}$.

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

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

Proof. Consider two activities i and j that are both executed in the solution. This is illustrated in Figure 2.5. There has to be a path of executed activities from the start activity

0 to both i and j. Call these paths S_1 and S_2, respectively. If these paths split at an activity r to successor activities u and v ($u \neq v$), it follows that u and v cannot be in the same selection group due to Constraints (2.1f). Since paths can merge after splitting, take activity r as the activity immediately before the last split, and the remaining paths as P and Q, which give a CRP (r, P, Q). Let $\ell(P)$ be the number of activities on path P. If there is no split, assume w.l.o.g, $\ell(P) < \ell(Q)$. Then, j lies in the extension of i, and $(i, \{i\}, \{i\} \cup \{\mathcal{V}(Q) \setminus \mathcal{V}(P)\})$ gives a CRP between i and j.

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

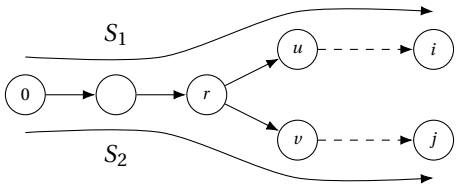


Figure 2.5: Example for CRP. There is a CRP $(r, (r, u, \cdots, i), (r, v, \cdots, j))$.

All MOES can be found by using a **rooted path graph** (RPG), as defined in Definition 2.4.2.

Definition 2.4.2 (Rooted Path Graph (RPG))
A rooted path graph $A_G = (N, E)$ is a graph with the same activities N as the selection graph and with an edge $(i, j) \in E$ if and only if there is a common rooted path between i and j.

It then follows that an independent set in the RPG represents a set of activities without any CRP's between them, and thus, a MOES. We distinguish between two types of CRP's: **splitted** CRP's and **extended** CRP's. A splitted CRP (r, P, Q) splits up at activity r and has $\ell(P) > 1$ and $\ell(Q) > 1$ (note that paths P and Q both include activity r), where $\ell(P)$ is the number of activities on path P. In an extended CRP there is no split and one path is an extension of the other, i.e., the root is i, $P = \{i\}$ and Q is a path from i to j. In Figure 2.5, there is a splitted CRP between i and j. However, there also is an extended CRP between, for example, r and i: $(r, (r), (r, u, \cdots, i))$.

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

Given these definitions, we now present Algorithm 2, which creates an RPG. Three sets of edges are introduced: Final edges $E^{(f)}$, active edges $E^{(a)}$ and new edges $E^{(n)}$. There are four steps which add edges to these sets:

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

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

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

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

These steps are illustrated in Figure 2.6.

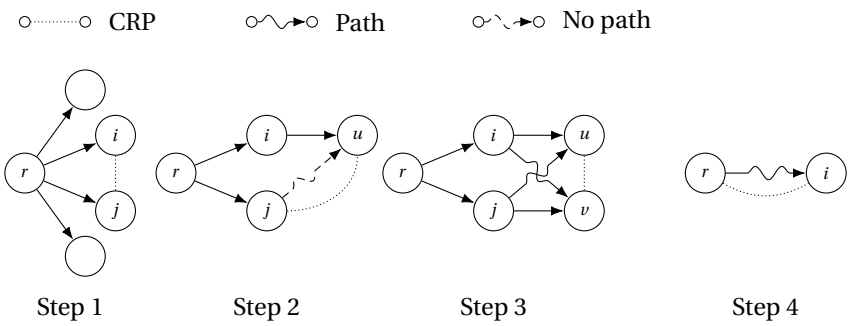


Figure 2.6: Steps in RPG algorithm.

Algorithm 2 Rooted Path Graph

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

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

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

Theorem 1 states that Algorithm 2 creates an RPG.

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

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

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

1. Edge (i,j) created by Step 1: since $Q = (r,j)$ and u is on path Q, it follows that $u = r$ or $u = j$. The former would result in a cycle $r \rightarrow i \rightarrow u = r$, which is not possible since we have an acyclic graph. The latter contradicts $u \notin \Theta_j$, so (i,j) cannot be created by Step 1.

2. Edge (i,j) created by Step 2. Let $(i,j) = (a_0,b_0)$, where the index 0 stands for the number of iterations, counting backwards. There are now two cases, (a_0,b_0) was created by extending edge (a_{-1},b_0) to a_0 on a_{-1}, or by extending edge (a_0,b_{-1}) to b_0 on b_{-1}. Consider the last case. By assumption, $u \in \mathcal{V}(Q)$ and u is a successor of

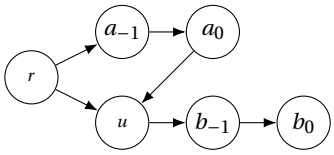


Figure 2.7: (i, j) created by Step 2, with $u \in \mathcal{V}(Q)$.

a_0. Therefore, as shown in Figure 2.7, there is a path $a_0 \to u \to b_0$. This means that b_0 is reachable from a_0, which is a contradiction since Step 2 only extends if b_0 is not reachable from a_0.

This means that if (a_0, b_0) is created by Step 2, it has to be extended to a_0 from input edge (a_{-1}, b_0). The same logic holds for this input edge (a_{-1}, b_0); it cannot be created by extending (a_{-1}, b_{-1}) to b_0 on b_{-1}. Thus, (a_0, b_0) is created by iteratively extending edge (a_{-n}, b_0) to a_{-n+1} on a_{-n}. Taking n as large as possible, we get edge (a_{-n}, b_0), which is not created by Step 2. Therefore, (a_{-n}, b_0) has to be created by either Step 1 or Step 3.

Consider the case that (a_{-n}, b_0) is created by Step 1. This means that both activities a_{-n} and b_0 are successor activities of the root activity $r \in N$. Since u cannot be reachable from b_0 and an activity is always reachable by itself, $b_0 \in \Theta_{b_0}$ and thus $u \neq b_0$. Therefore, since $u \in \mathcal{V}(Q)$ and $Q = (r, b_0 = j)$, we have that $u = r$, and thus, there is a path $a_0 \to u = r \to a_{-n} \to \cdots \to a_0$, which is a cycle.

If (a_{-n}, b_0) is created by Step 3, call the input edge (a_{-n-1}, b_{-1}). Since $u \in \mathcal{V}(Q)$ and $u \neq b_0$, it follows that b_{-1} is reachable from u, i.e., $b_{-1} \in \Theta_u$. By construction in Step 3, a_{-n} is reachable from b_{-1}. Therefore, there is a path $u \to b_{-1} \to a_{-n} \to a_0 \to u$, which is also a cycle. This means that there is a cycle in both cases. This contradicts the fact that we have an acyclic graph. Therefore, edge (i, j) cannot be created by Step 2.

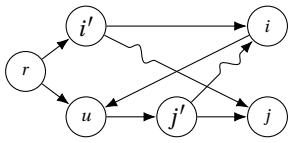


Figure 2.8: (i, j) created by Step 3, with $u \in \mathcal{V}(Q)$.

3. Edge (i, j) created by Step 3. Let (i', j') be the input edge which created (i, j), with i' and j' in P and Q, respectively. This is illustrated in Figure 2.8. Since $u \in \mathcal{V}(Q)$ and $u \neq j$, it follows that j' is reachable from u. This gives a path $i \to u \to j' \to i$. This creates a cycle, which is a contradiction.

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

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

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

If there is a CRP, there is an edge created by Algorithm 2:
Consider activities i and j with a CRP $\{r, P, Q\}$, $P = (r, p_1, \cdots, p_n)$ and $Q = (r, q_1, \cdots, q_m)$. Let $p_n = i$ and $q_m = j$. If either i is reachable from j or vice versa ($(i, j) \in \Theta$), Step 4 creates an edge. Therefore, assume that $(i, j) \notin \Theta$.

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

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

We now give the separation problem for maximum execution cutting planes in Constraint set (2.6). These cutting planes are based on a MOES and a group of activities for which the MOES has full precedence. As stated earlier, a selection group with full precedence has a precedence relationship between the activator and each successor. The idea of the cutting planes is that if one activity i from the MOES is executed, then there is always an activity j executed that has to be executed after finishing activity i.

We define E as the set of all edges in the RPG. The MILP formulated by Constraint set (2.6) uses the optimal relaxed solution of Constraint set (2.1), denoted by X^*_{it}. The activities of the MOES are captured by binary variables W_i, which are equal to one if activity $i \in N$ is selected for the MOES and zero otherwise. The set of selected successor activities of the MOES is defined by binary variables Z_j for all $j \in N$, where $Z_j = 1$ if activity j is selected and zero otherwise. These two sets of activities are linked by selected selection groups. If a selection group $g \in G$ is selected, then binary variable $Y_g = 1$ and $Y_g = 0$ otherwise.

Constraints (2.6b) only select full-precedence groups for which the activators have no CRP. Constraints (2.6c) define that activities can only be selected within the MOES, if they are the activator of a selected full-precedence group. Furthermore, Constraints (2.6d) select only successors of selected full-precedence groups. The first term in Objective function (2.6a) represents the finishing times of the MOES, the second term represents the starting times of the successor activities of the activities in the MOES. By maximizing the difference between these two terms, a violation of the precedence relations can be

found.

$$\max \sum_{i \in N} W_i \sum_{t \in T} (t + d_i) X^*_{it} - \sum_{j \in N} Z_j \sum_{t \in T} t X^*_{jt}, \tag{2.6a}$$

$$Y_g + Y_h \leq 1, \quad \forall g \in H, h \in H, (a_g, a_h) \in E, a_g \neq a_h, \tag{2.6b}$$
$$Y_g \geq W_{a_g}, \quad \forall g \in H, \tag{2.6c}$$
$$Z_j \geq Y_g, \quad \forall g \in H, j \in S_g, \tag{2.6d}$$
$$W_i \in \{0,1\}, \quad \forall i \in N \text{ for which } |\{g : i = a_g, g \in H\}| \geq 1, \tag{2.6e}$$
$$Y_g \in \{0,1\}, \quad \forall g \in H, \tag{2.6f}$$
$$Z_j \in \{0,1\}, \quad \forall j \in N. \tag{2.6g}$$

Proposition 2. *Let X^* be the linear relaxed solution of Constraint set (2.1). Furtermore, let W^*, Y^* and Z^* be the solution of Constraint set (2.6) and let the value of Objective function (2.6a) be larger than 0. Then, Constraints (2.7) is a cutting plane for the RCPSP-PS that cuts of the current solution X^*:*

$$\sum_{i \in N} W^*_i \sum_{t \in T} (t + d_i) X_{it} \leq \sum_{i \in N} Z^*_i \sum_{t \in T} t X_{it}. \tag{2.7}$$

Proof. Constraints (2.6b) imposes that groups can only be selected if there is no CRP between the activators. Therefore, in combination with Constraints (2.6c), there is no CRP between the set of activities for which $W^*_i = 1$. Therefore, for an integer solution, the left hand side of Constraints (2.7) contains *at most one* non-zero summation term $\sum_{t \in T}(t + d_i)X_{it}$, for which $\sum_{t \in T} X_{it} = 1$. Consider the case that one activity i' for which $W^*_{i'} = 1$ is executed. Then, there exists at least one selection group $g \in H$ with activator i' for which $Y^*_g = 1$ and thus there exists at least one executed activity j' that is a successor of group g and for which $Z^*_{j'} = 1$. Due to the full precedence, we obtain Equation (2.8). If no activity i' for which $W^*_{i'} = 1$ is executed, the first term of Constraints (2.7) is zero and, therefore, it is a cutting plane.

$$\sum_{i \in N} W^*_i \sum_{t \in T} (t + d_i) X_{it} = \sum_{t \in T} (t + d_{i'}) X_{i't} \leq \sum_{t \in T} t X_{j't} \leq \sum_{i \in N} Z^*_i \sum_{t \in T} t X_{it}. \tag{2.8}$$

Therefore, as long as Objective function (2.6a) has a value larger than 0, Constraints (2.7) cuts of the current solution X^*. □

Both cutting plane types are used as an initial step in solving Constraint set (2.1). First, the LP-relaxation of Constraint set (2.1) is solved. Secondly, the separation problems are solved and the cuts are added to the LP-relaxation. This is repeated until no more cuts are found, or when the objective increase is lower than a certain treshold for a fixed number of iterations.

2.4.3. CONSTRAINT PROPAGATION ALGORITHM

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

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

The algorithm initializes by creating an empty set of cutting planes for each activity and by generating a set of **forced activities** $\mathscr{F}_i$ for each activity $i \in N$. A set of forced activities $\mathscr{F}_i$ for activity $i \in N$ is defined as the set of activities that always have to be executed if activity i is executed. To determine whether activity $j \in N$ is in the set of forced activities $\mathscr{F}_i$, we use a modified graph as input for Constraint set (2.2). First, we remove all activities $k \in N$ without a CRP between k and i. Since i is executed, any activity without a CRP to i is not executed. Then, if there is a feasible solution to Constraint set (2.2) for activity set N' with $V_j = 1$ and $V_p = 0$ for every $p \in N', p \neq j$, $\{j\}$ is a NEES and is thus always executed. In this case, activity $j \in \mathscr{F}_i$.

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

The first function, *find_cutting_planes*(i,$\mathscr{C}_i$,$\mathbf{s}$), generates a set of cutting planes as follows. First, we replace activity $n+1$ in Objective function (2.1a) by activity i and adding the constraint $\sum_{t \in T} X_{it} = 1$, which we refer to as **setting the objective function to** i. This gives an MILP where activity i is always executed, while minimizing the starting time of activity i, thus, providing a lower bound on the starting time s_i of activity i. The solution obtained by solving the relaxation of this MILP, with starting times $\mathbf{s}$ and cutting planes $\mathscr{C}_i$, is then used to generate additional cutting planes as given by Constraints (2.5) and (2.7), which together with the already given cutting planes form the new set $\mathscr{C}_i$. The function call is aborted as soon as more than 10 consecutive cutting planes did not improve the linear relaxation value, as it was discovered experimentally that increasing this number did not increase performance significantly, while increasing the computing time.

The second function, *linrelax*($i, \mathscr{C}_i, \mathbf{s}$), solves the linear relaxation of Constraint set (2.1) combined with Constraints (2.3) while setting the objective function to i, adding cutting planes $\mathscr{C}_i$ and setting the lower bound on activity starting times to $\mathbf{s}$ for all activities in N. The latter is done by setting $X_{jt} = 0$ for $t < s_j$ for each activity $j \in N$. The function returns the objective function value, rounded up to the nearest integer, which is a lower bound on the starting time of activity i.

The function *variable_reduction*($\mathbf{s}$) calls Algorithm 1, with the modification of using $\mathbf{s}$ as initial lower bound instead of setting it to zero in line 2. It then returns lower bounds for all activities. Finally, the function *solve*($\mathbf{s}$) first calculates the latest finishing time f_i for each node $i \in N$ as described in Section 2.4.1 and then solves the MILP while adding valid inequalities from Constraints (2.3) and setting $X_{it} = 0$ for all $i \in N$ with $t < s_i$ or $t + d_i > f_i$.

Algorithm 3 Solution algorithm

```
1: 𝒞 ← {∅ : i ∈ N^(s)}
2: s ← variable_reduction(0)
3: ℱ ← forced activities
4: N^(s) ← topological sorting of N on precedence graph
5: improved ← True
6: while improved = True do
7:     improved ← False
8:     for all i ∈ N^(s) do
9:         𝒞_i ← 𝒞_i ∪ {𝒞_j : j ∈ ℱ_i}
10:        𝒞_i ← 𝒞_i ∪ find_cutting_planes(i, 𝒞_i, s)
11:        v ← linrelax(i, 𝒞_i, s)
12:        if v > s_i then
13:            improved ← True
14:            s_i ← v
15:            s ← variable_reduction(s)
16:        end if
17:    end for
18: end while
19: solve(s)
```

With these functions, the solution algorithm is given in Algorithm 3. Initially, the lower bounds $\mathbf{s}$ are set by calling the *variable_reduction*() function. Subsequently, the sets of forced activities $\mathscr{F}_i$ for every activity $i \in N$ are calculated. The algorithm will now loop over all activities in topological order $N^{(s)}$. For each activity $i \in N^{(s)}$, cutting planes are calculated by setting the objective to i, and by adding cutting planes from all forced activities. The latter is done because a cutting plane for an activity j is valid when this activity is executed. Next, for activity i, a lower bound will be calculated by using *linrelax*(i,$\mathscr{C}_i$,$\mathbf{s}$). If this improves the current lower bound for i, the *variable_reduction*($\mathbf{s}$) is called to possibly propagate this improvement to other activities. If there is any improvement for at least one of the activities, the loop over all activities will be repeated. If not, the while loop will terminate. Subsequently, to get an optimal solution, the *solve*($\mathbf{s}$) function is called to solve the MILP.

2.5. COMPUTATIONAL RESULTS

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

2.5.1. INSTANCES

In this section, we give a brief overview of the instance generation algorithm and give the instance sets we evaluate. The instances are generated by creating a simple network with

placeholder activities and replacing these placeholder activities by **subnetworks**. For each instance, subnetworks are generated by the generation procedure of Vanhoucke et al. (2008). This procedure generates instances of the RCPSP based on the number of activities, **Serial/Parallel indicator** (SP), **Resource Factor** (RF) and **Resource Constrainedness** (RC). The description of these parameters can be found in Appendix 2.B and the specific details can be found in Vanhoucke et al. (2008). Each instance contains two types of subnetworks, called **Phase-1 subnetworks** and **Phase-2 subnetworks**. The parameters $N1$ and $N2$ define the number of activities of these respective networks. All other parameters are equal for all subnetworks of an instance.

The network of placeholder activities is created from a shape array, for example [3,3], indicating two sequential selection-groups with each three successors. Each successor is replaced by a phase-1 subnetwork, and in the resulting network, the **Replace Number** (RN) determines the number of activities replaced by multiple parallel phase-2 subnetworks. Finally, additional precedence links are placed according to the **Additional Links** (AL) parameter. If **Independent Succession** (IS) is allowed, then the same number of selection links as precedence links is placed as well. Details of the instance generation algorithm can be found in Appendix 2.B.

With these parameters, we create three sets of instances: literature instances, sensitivity instances and comparison instances. For all sets, the parameters RF and SP are constant (RF = 0.75 and SP = 0.5), as we capture the variance in resources and shape by the RC parameter and shape array, respectively. Furthermore, each instance has 4 resources. For all other parameters, the values are shown in Table 2.B.1. Here, each list shows the considered values per parameter and an instance is generated for each combination, with certain combinations to be removed after computational results indicated these instance to be unsolvable within a solving time of 6 hours. Furthermore, the full sets of instances can be found Van der Beek (2022c).

The largest instance set, *Literature*, has no IS, and therefore, can be solved by the model in Kellenbrink and Helber (2015). This set is thus used to compare the proposed method in this chapter with a model from literature. The smaller instance set *Sensitivity* neither allows IS and is used to evaluate the solution methods to variations in the initial time horizon T. Therefore, any instance in this set that cannot have T reduced by 10% without becoming infeasible is removed. Finally, the instance set *Comparison* includes IS, and therefore, can only be solved with the proposed methods in this chapter. This instance set is used to gain insight in the improved performance by various parts of the proposed solution method.

2.5.2. RESULTS

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

1. **Basic**: Solve the MILP given by model Constraint set (2.1) for final activity $n+1$.

2. **Variable reduction (VR)**: Basic method combined with Constraints (2.3) and the

variable reduction method given in Algorithm 1; i.e., calling function *solve*(**s**) with **s** from *variable_reduction*(**0**).

3. **Constraint propagation (CP)**: Algorithm 3.
4. **Literature**: MILP model proposed in Kellenbrink and Helber (2015). Notice that their variable reduction method does not reduce variables on our instances, since all non-dummy activities are optional.

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

LITERATURE

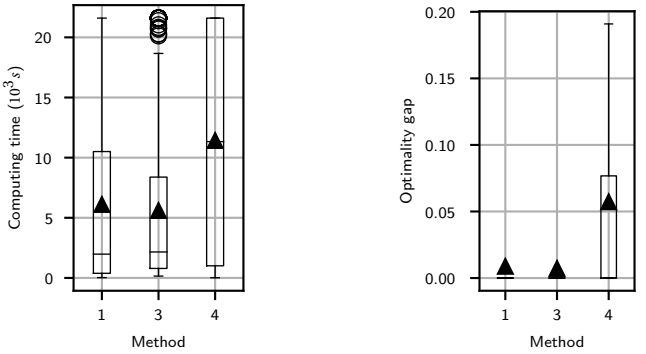
First, the instance set *Literature* is evaluated. To compare both basic MILP models, each instance is solved by methods Basic and Literature. Furthermore, each instance is solved by the CP method to evaluate the improvement. Due to the large size of this instance set and the computational effort required, we evaluate this set only on the full (CP) method and leave the analysis for the intermediate step (VR) for the *Sensitivity* and *Comparison* instance sets.

A summary of the results is given in Table 2.3. Here, it can be seen that the number of optimal solutions obtained by the methods Basic and CP is significantly higher than by method Literature. Furthermore, the CP method is, on average, about 32 minutes faster than the *Literature* method. However, this is skewed due to the non-solvable instances. When comparing only the **interesting instances**, a decrease in average computation time of 50% is achieved. We define an interesting instance as an instance that has at least one method solving it to optimality, and at least one method not solving it to optimality within 5 minutes. This can be seen in Figure 2.9a, where the computing time of all 976 *interesting instances* are shown.

Furthermore, we define the optimality gap as $(ub - lb)/ub$ where lb and ub are the lower and upper bound found by the MILP solver, respectively. In Table 2.3 and Figure 2.9b, this shows a similar trend as for computing time. Furthermore, to gain more

Table 2.2: Properties of instance sets

	Literature	Sensitivity	Comparison
RC	[0.5,0.7,0.9]	[0.5,0.7,0.9]	[0.5,0.7,0.9]
N1	[3,4,5]	[3,4]	[3,4,5]
N2	[3,4,5]	[3,4]	[3,4,5]
Shape	[[2,2], [2,3], [3,2], [2,2,3], [2,3,2], [3,2,2],[3,3]]	[[2,3],[2,3]]	[[2,2],[2,3],[3,3], [2,3,2]]
RN	[2,3]	3	[2,3]
AL	[0,2,4,6]	[0,3]	[0,3,6]
IS	No	No	Yes
# instances	1008	34	576



(a) Computing time.

(b) Optimality gap (without fliers).

Figure 2.9: Computing time and optimality gap for instance set *Literature* for methods 1 (Basic), 3 (CP) and 4 (Literature).

insight in this optimality gap, we evaluate the best lower and upper bounds found during the MILP solver process. Similar to the optimality gap, we normalize this to the lowest upper bound found for each instance. Furthermore, we evaluate the number that a certain method reached the best bound, compared to other methods, excluding all **ties**. A tie is defined as an instance where all three methods reach the same bound. It can be seen that on both bounds and optimality gap, the Basic and CP method outperform the Literature method. Furthermore, the Basic method has a larger number of best lower bounds, but the CP method has a higher average lower bound. From this, it can be deduced that if CP has the best bound, on average it is with a larger difference than if Basic has the best bound. Furthermore, it can be seen that the Basic method performs better than the CP method on all aspects of the upper bound. Therefore, the better average optimality gap of the CP method compared to the Basic method can be attributed to the better performance of the lower bounds.

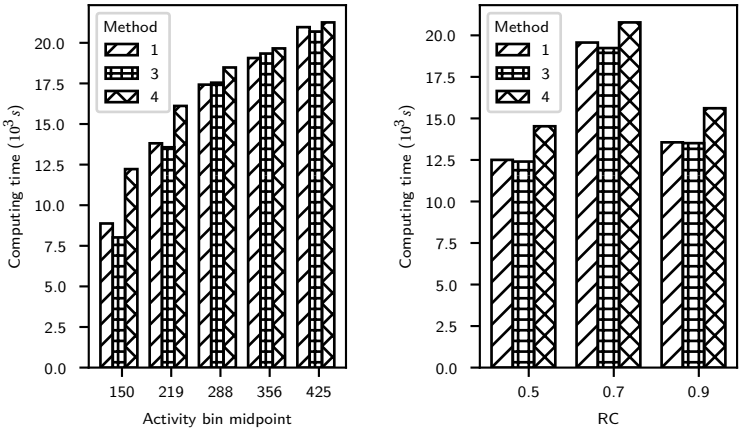
When comparing the size of the problems, it can be seen that the Basic model is con-

Table 2.3: Summary of results for instance set *Literature*. The numbers of best lower bounds and best upper bounds have 260 and 867 ties, respectively.

	Basic	CP	Literature
# Optimal solutions	356	364	256
Average computing time ($10^3 s$)	15.11	14.95	16.85
Average optimality gap	0.130	0.126	0.215
# Best lower bound	452	420	4
Average normalized lower bound	0.872	0.876	0.793
# Best upper bound	104	99	5
Average normalized upper bound	1.0029	1.0032	1.0116
Average number of variables	35048	15275	35048
Average number of constraints	2018	2340	1093

siderably larger than the Literature model. Both have the same number of variables, but the Basic model has about twice as much constraints. However, the constraint propagation reduces the size significantly. Although some more constraints (and thus rows) are added due to Constraints (2.3), the number of variables is more than halved.

Upon evaluating computing times, we found two major trends: with the number of activities and with the RC value. The average computing times for different values of these parameters are plotted in Figures 2.10a and 2.10b. It can be seen that the solving difficulty increases with the number of activities. We see that the difference between the methods decreases with the number of activities, presumably due to the influence of unsolved instances (which are capped at 6 hours). Furthermore, while evaluating the dependency on the RC parameter, it can be seen that values around 0.7 are significantly more difficult. A possible explanation could be that since resource constraints are difficult to solve, a lower constrainedness results in an easier instance, while a high value would significantly decrease the feasible region, and thus, the number of schedules that have to be evaluated.



(a) Per activity bin

(b) Per RC value

Figure 2.10: Average computing times for instance set *Literature* for methods 1 (Basic), 3 (CP) and 4 (Literature).

SENSITIVITY

An important part in any time-based scheduling formulation of the RCPSP is the size of time horizon T. Decreasing this can significantly decrease the number of variables and often speed up the computation. To determine our value of T, we create a feasible solution using a simple heuristic: the selection problem is solved by iteratively adding nodes, starting at the root node, until a feasible selection is obtained. In the case of multiple candidate nodes, the selection is made randomly. Then, a schedule is made by in a similar way: iteratively adding random selected nodes in a precedence feasible way, starting at the root node. The now sorted selected nodes form an activity list, and

a serial generation scheme is used to create a schedule (Kolisch and Hartmann, 1999). Note that this simple heuristic works for the structure of the instances considered in this chapter, but not necessarily for all instances of the RCPSP-PS. In case of other structures, more sophisticated heuristics should be used. If no heuristic is available, the sum of all durations form an (very bad) upper bound that can be used.

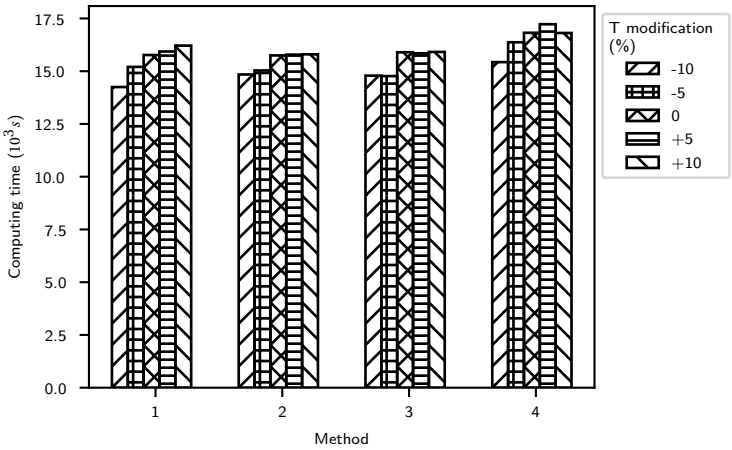


Figure 2.11: Sensitivity to modifications in T for methods 1 (Basic), 2 (VR) , 3 (CP) and 4 (Literature).

To evaluate the influence of the time horizon T, the instance set $Sensitivity$ is evaluated while modifying T for each instance. These modifications are done in percentages from the initial T value, after which they are rounded. In Figure 2.11, the average computing times per method and per T modification values are shown. For the Basic and the Literature method, the computing time seems to vary similarly, although the largest modification for the Literature method has a smaller average computing time. However, we assume this is due to randomness in the MILP solving process. For the VR and CP methods, we can see that increasing T has very little effect on the average computing time. Furthermore, reducing it improves the performance, although this trend does not seem to continue for further reduction from -5% to -10%.

COMPARISON

Finally, we evaluate the instance set *Comparison* to gain insight in the relative improvements of different parts of the CP method. A summary of the results is given in Table 2.4 and the total computation time per method for *interesting* instances are shown in Figure 2.12a. It can be seen that, although the CP method has the lowest average computing time, the largest part of this decrease in computing time can be attributed to the VR method. However, when considering the number of optimal solutions obtained, the improvement due to the CP method is considerably larger than the improvement due to the VR method, indicating that the CP method is especially useful on more difficult instances.

The optimality gaps show counter-intuitive behavior, with the Basic method having

the smallest gap. When evaluating the average normalized lower and upper bounds, it can be seen that this optimality gap is mainly due to the better upper bounds found by the Basic method. Looking at the number of best lower and upper bounds, it seems that the Basic method performs better than both the VR and CP method. However, this is not necessarily the case for the lower bounds, since this comparison is slightly skewed due to CP and VR performing well on the same instances. If only Basic and CP would be compared on lower bounds, Basic would have 122 best lower bounds and CP 125 (excluding ties between Basic and CP). However, since on multiple instances where CP has a better lower bound, VR has an even lower lower bound, CP performs relatively worse when comparing all three methods. For the upper bounds, however, Basic performs better than both the VR and the CP method.

Furthermore, Table 2.4 lists the average sizes of the MILPs. It can be seen that the largest part of the reduced variables is due to the VR method, averaging to a reduction of 44%. The CP method in turn has a further reduction of 4%. Both the VR and CP method use the same valid inequalities, resulting in the same number of constraints: 15% more than the Basic method.

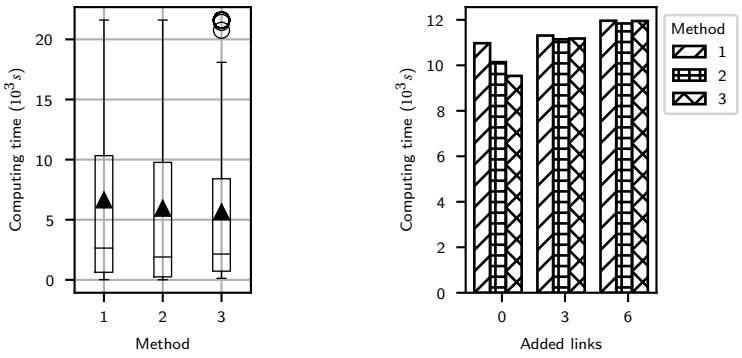
Finally, we evaluate the performance of the methods compared to the AL parameter, which gives an indication of the similarity between the selection graph and the precedence graph. The average computing time per AL value is shown in Figure 2.12b.

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

In Figure 2.13a, the relative linear relaxation values are shown. It can be seen that the

Table 2.4: Summary of results for instance set *Comparison*. The number of best lower bounds and best upper bounds have 298 and 512 ties, respectively.

	Basic	VR	CP
# Optimal solutions	323	325	336
Average computing time ($10^3 s$)	11.41	11.04	10.89
Average optimality gap	0.057	0.059	0.058
# Best lower bound	127	88	113
Average normalized lower bound	0.9449	0.9452	0.9453
# Best upper bound	38	24	34
Average normalized upper bound	1.0027	1.0049	1.0036
Average number of constraints	1535	1767	1767
Average number of variables	21253	11871	11000

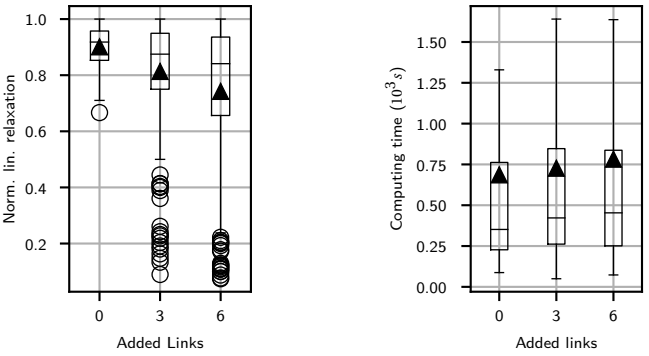


(a) Interesting instances.

(b) Average computing time.

Figure 2.12: Computing times for instance set *Comparison* for methods 1 (Basic), 2 (VR) and 3 (CP).

relative performance of the CP method increases with the AL value. However, by plotting the computing times of the preprocessing part of the CP method in Figure 2.13b, it can be seen that the computing time also increases with AL values. As seen in Table 2.5, there is an decrease in MILP solver time due to the CP method for each AL value. However, for AL values larger than 0, this decrease is smaller than the increased computing time due to preprocessing. Therefore, no average computing time improvement is achieved. However, when evaluating the number of solved instances for an AL value of 3 (108 for VR and 111 for CP) and 6 (100 for VR and 102 for CP), the use of the CP method can still be valuable for these instances.



(a) Normalized linear relaxation of VR compared to CP.

(b) Computing time of preprocessing step for CP (without outliers).

Figure 2.13: Analysis on AL value.

Table 2.5: Partial computing times ($10^3 s$) for instance set *Comparison* for different values of AL.

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

2.6. CONCLUSIONS

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

Computational tests show that the basic MILP model performs significantly better than the most similar model from literature. Furthermore, an improvement on both computing time and the number of optimal solutions found is achieved by both the *Variable reduction* and the *Constraint propagation* method. Additionally, although for the final instance set the average optimality gap did not improve, the best lower bound found did. Furthermore, it was shown that for all methods, a lower initial time horizon decreased the average computing time, therefore indicating the usefulness of good heuristics, even for instances that can be solved to optimality.

Besides the direct computational improvements, the methods presented in this chapter decrease the solution space. Therefore, they can be directly implemented in other exact approaches, such as constraint programming. Furthermore, the mathematical proofs in this chapter can be used as building blocks for other theoretic or computational improvements.

APPENDIX

2.A. NOTATION

Sets

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

Variables

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

Parameters

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

2.B. INSTANCE GENERATION METHOD

In this section, we give the instance generation algorithm. The instance generation algorithm works by creating a simple sequential network and replacing activities. First, we give a list of all input parameters. Subsequently, we give the instance generation algorithm, and finally, we describe the three instance sets.

Table 2.B.1: Properties of instance sets

Parameter	Description
Resource constrainedness (RC)	Resource Constrainedness parameter from Vanhoucke et al. (2008).
Resource Factor (RF)	Resource Factor from Vanhoucke et al. (2008).
Serial/Parallel indicator (SP)	Serial/Parallel indicator from Vanhoucke et al. (2008).
Shape	Vector with initial shape of the network, for example [2,3,2].
Network size 1 (N1)	Size of networks of Phase 1.
Network size 2 (N2)	Size of networks of Phase 2.
Alternatives Number (AN)	Number of alternative options of network Phase 2.
Replacement Number (RN)	Number of activities in a Phase 1 network that is replaced by a set of Phase 2 networks.
Additional Links (AL)	Number of additional links (precedence and/or selection) added.
Independent Succession (IS)	Boolean, true if selection links are added.

All input parameters are given in Table 2.B.1. The algorithm is based on multiple functions that either replace or create a part of the network. Since we are mostly interested in the activities of the network, we use a $N^{(i)}$ notation to refer to the *activities* of a (partial) network. Using an index j, the j^{th} activity is denoted by $N^{(i)}_j$. Furthermore,

unless specified otherwise, every created activity is a dummy activity with zero duration and resource requirements.

Next, we introduce the functions used by the algorithm. The first function is *create_sequential_network*(a). This function creates a network of a serial activities. This means that for each activity i with $i < a$, a *precedence relationship* is created from activity i to activity $i+1$, as well as a *selection relationship* from i to j. Creating a selection relationship from i to $i+1$ is defined as creating a selection group $(i, \{i+1\})$ (i is the activator and $\{i+1\}$ is the successor set).

The second function is *replace_group*(n, a), where n is an activity and a an integer. This function creates $a+2$ activities, consisting of a starting activity, a successor activities and an ending activity. A selection group with full precedence is created between the activator activity and the successor activities. Subsequently, both a precedence and a selection relationship are created from each successor to the ending activity. The function replaces activity n by this partial network and returns the *a successor* activities for future reference.

The third function, *replace_network*($n, N1, RC, RF, SP$), is the only function creating non-dummy nodes. The input is an activity n and the parameters $N1, RC, RF$ and SP. It uses these parameters as input for the method described in Vanhoucke et al. (2008) to create an RCPSP instance. This instance is converted to an RCPSP-PS instance by creating a selection link for each precedence link. Then, activity n is replaced by this instance and all activities, except the root and final activities, are returned for future reference.

Both functions *replace_network*() and *replace_group*() modify the input network *in place* and return a subset of activities (as explained above) for future reference. For example, consider the set of activities N with activity $n \in N$. Then, by calling $M \leftarrow replace\_group(n, a)$, a selection group is created with a successors and this selection group replaces activity $n \in N$. The successors are stored under M, and since n is part of N and the modification is done in place, the set of activities N now contains this group: $M \subset N$.

Finally, when the complete network with precedence and selection links is required, the function *network*(N) is used. This function returns the full networks, instead of just the activities.

With these functions, the first part of the algorithm is described. In Line 1, the network is initialized by creating a sequential network of the same length as the length of the *Shape* vector. Then, in Line 3, each h^{th} activity in the sequential network is replaced by a selection group with $Shape[h]$ successors. Each of these successors is then replaced by a network generated by the *replace_network* function. We call these networks: **Phase 1** networks. Subsequently, in the loop starting at Line 6, *RN* activities within this network are replaced by a selection group with *AN* successors in Line 8. Then, each successor of this selection group is replaced by a network at Line 10. These networks are called **Phase 2** networks.

This creates the basic network. The final part of the algorithm adds additional selection and precedence links. Before describing this part, two more functions are introduced to create the additional links. The first is *get_prec_candidates*(N), where N is a set

of activities. This function returns all potential pairs for added *precedence relationships*, within N. Two activities n and m are a potential pair if all following criteria are satisfied:

1. Creating a precedence link from activity n to activity m does not create a cycle.
2. Activities n and m are part of the same Phase 1 network.
3. Activities n and m are part of a Phase 2 network.
4. Activities n and m are not dummy-activities.

Similarly, the function *get_sel_candidates*(N) takes a set of activities N as input and returns all potential pairs for added *selection* relationships, within N.

1. Creating a selection link from activity n to activity m does not create a cycle.
2. Activities n and m are part of the same Phase 1 network or activity n was created in an earlier iteration of Line 2 in Algorithm 4.
3. Activities n and m are part of a Phase 2 network.
4. The Phase 2 networks of activities n and m are not part of the same group in Line 8.
5. Activities n and m are not dummy-activities.

With these functions, the remainder of the algorithm can easily be described. In Line 15, up to AL precedence links are added from the candidates. Subsequently, if IS is enabled, the same is done for selection links. Finally, the algorithm returns the complete network in Line 35.

Algorithm 4 Instance generation algorithm

$N^{(1)} \leftarrow$ Create_sequential_network($|Shape|$)
for h in range($|Shape|$) **do**
  $N_h^{(2)} \leftarrow$replace_group($N_h^{(1)}, Shape[h]$)
  for i in range($Shape[h]$) **do**
    $N^{(3)} \leftarrow$ replace_network($N_i^{(2)}, N1, RC, RF, SP$) ▷ Phase 1
    for j in range(RN) **do**
      $n \leftarrow$Select random w/o replacement from $N^{(3)}$.
      $N^{(4)} \leftarrow$replace_group($N_n^{(3)}, AN$)
      for k in range(AN) **do**
        $N^{(5)} \leftarrow$ replace_network($N_k^{(4)}, N2, RC, RF, SP$) ▷ Phase 2
      end for
    end for
  end for
end for
for i in range(AL) **do**
  $prec\_candidates \leftarrow$ get_prec_candidates($N^{(1)}$)
  if $|prec\_candidates| = 0$ **then**
    break
  else
    $n, m \leftarrow$Select randomly from $prec\_candidates$
    Create precedence link between n and m
  end if
end for
if IS **then**
  for i in range(AL) **do**
    $sel_candidates \leftarrow$get_sel_candidates($N^{(1)}$)
    if $|sel\_candidates| = 0$ **then**
      break
    else
      $n, m \leftarrow$select randomly from $sel\_candidates$
      Create selection group $(n, \{m\})$
    end if
  end for
end if
return network($N^{(1)}$)

3

HEURISTIC SCHEDULING FOR THE RCPSP-PS WITH CONSUMPTION AND PRODUCTION OF RESOURCES [1]

In the previous chapter, the RCPSP-PS is given to model modularization and outsourcing decisions for modular shipbuilding. Although the flexible project structure is very general, certain characteristics of modular shipbuilding cannot be modeled due to limitations in resource types. This chapter, therefore, gives a generalization of the resources introduced in the previous chapter.

First, we present the resources that cannot be modeled with the RCPSP-PS. The first is *capital*. Shipbuilding is a capital-intensive industry. During the building process, a lot of money is tied up in the ships being constructed. To maintain a healthy financial situation, there can be a limit on the maximum amount of capital that is tied up in a project. This is related to project scheduling: activities such as material procurement 'lock down' capital. Conversely, payments are usually associated with project milestones, such as finishing the steel work, installing the engine or finishing the vessel. By considering this while scheduling, it can be assured that the capital requirements do not exceed a certain limit.

Secondly, floor space at the ship yard is limited. Although 2D fitting of floor space is an NP-hard problem by itself (Türk et al., 2022), a simplified model can be created by considering floor space as a one dimensional entity, possibly considering various types of floor space. Then, activities such as 'creating an assembly' require the resource floor space: floor space is needed to store the assembly. Subsequently, the activity 'install an assembly in the ship' frees up floor space, as the assembly is taken and moved to the inside of the ship.

[1] This chapter is reproduced from the paper published in the European Journal of Operational Research (Van der Beek et al., 2024).

Finally, it is possible to pre-construct certain assemblies. This will especially be useful when considering multiple projects at once. These assemblies can be modeled as resources that are generated by activities that construct these assemblies and depleted by activities that use the assemblies.

As can be noted, all resources introduced above do not generate automatically after use. Instead, they are produced by certain activities and consumed by others. Thus, we call this **nonrenewable resources with production and consumption**. In this chapter, these types of resources are added to the model.

Furthermore, in Chapter 2, an exact solution method is given. Although exact solution methods should be used whenever possible, due to the NP-hardness of the RCPSP-PS, these methods often take too long to be of practical use. Therefore, in this chapter, we also present two heuristic methods to quickly find good solutions. The computational study, among other things, then showcases why exact methods are useful even if they would not be used in practice: by evaluating against exact solutions, an indication of the performance of the heuristic methods is given.

3.1. INTRODUCTION

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

However, in reality, these assumptions are not always valid, due to multiple reasons. First of all, in many construction projects, there are multiple ways of completing a project. For example, in shipbuilding, assemblies can be produced beforehand in workshops, or directly on the ship. After production in a workshop, an assembly has to be installed as a whole in the ship, requiring specialized resources such as cranes. Conversely, direct production on the ship does not require this, although the less ideal working conditions might require better trained workers. Secondly, not all resources are renewed automatically. For larger products, like machines or aircrafts, factory space becomes only available after a moving/shipping activity is done. Similarly, capital might only be freed up after a certain milestone activity has been reached. In this case, these resources are modeled as *nonrenewable resources*. For example, consider capital as one of these resources. An activity that requires purchasing components consumes capital. This capital can then be produced again by activities that generate income, such as reaching a project milestone. Similarly, completing a sub-assembly and moving it to some other location will free up (produce) the resource 'floor space'.

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

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

This chapter has three contributions. Firstly, we adapt the **Mixed Integer Linear Programming** (MILP) formulation of Chapter 2 by adding nonrenewable resources. Secondly, group orderings are introduced; a fast method to find feasible solutions to the selection problem, to be used within heuristics. This method is only applicable to instances with certain characteristics, which are discussed in this chapter. However, these characteristics are present in many practical instances of the RCPSP-PS/CPR. Finally, we present a **Hybrid Differential Evolution** (HDE) algorithm and evaluate its performance against special cases from the literature, optimal solutions and a **Ant Colony Optimization** (ACO) benchmark algorithm.

In Section 3.2, we present an overview of the literature related to the RCPSP-PS/CPR. Subsequently, we formulate the problem in Section 3.3. Then, we present a method to create feasible selections and use this to create the HDE algorithm and the ACO algorithm in Section 3.4. This is followed by a comparison of computational results in Section 3.5 and conclusions in Section 3.6.

3.2. LITERATURE REVIEW

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

We now consider **scheduling with nonrenewable resources**. We use this term to denote the general case of a set of resources that are consumed and not *automatically* renewed. Furthermore, we consider the generalization of **consumption and production of resources**, where a resource cannot only be consumed but also produced.

When considering scheduling with nonrenewable resources with consumption and production, two types of closely related problems have to be studied: **activity-based**

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

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

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

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

The flexible project structure is firstly introduced by Barták et al. (2007). However, they do not consider the RCPSP, but a different scheduling problem: temporal networks where resources are not considered and the goal is to satisfy maximum and minimum

time lags between activities. For this problem, they introduce the problem of a flexible structure and, amongst other things, show that the selection problem is NP-hard. One of the earliest RCPSP variants with a flexible project structure is the *Extended RCPSP*, introduced by Kuster et al. (2009). They study a disruption management problem, which they model as an RCPSP with an initial activation state and substitution criteria. These substitution criteria define what changes are allowed to the initial state. 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, Chapter 2 introduces an MILP model where the choices are based on **selection-groups**; a group consisting of an activator activity and a set of successor activities. If an activator activity is executed, exactly one successor activity has to be executed. A solution method is given that uses cutting planes and constraint propagation for preprocessing, after which the problem is solved to optimality by a commercial MILP solver.

We now compare the RCPSP-PS/CPR to the problems introduced in this literature review. For this, we use three concepts from Chapter 2. The first concept is separate scheduling and selection. This means that the graph representing all selection constraints can be separate from the graph representing all precedence constraints. The second concept is *Independent Succession* (IS). A model with IS allows for multiple activities to simultaneously cause the execution of the same other activity. If a model does not feature IS, each activity can only be executed based on at most one other activity. Finally, the *exclusivity criterion* (EC) is introduced. If a model includes the EC, choices are *exclusive*; only one alternative can be selected from a set of candidate activities. With these terms, the different models can be compared. This is done in Table 3.2.1 and shows that the models of Chapter 2, Kuster et al. (2009); Kellenbrink and Helber (2015); Servranckx and Vanhoucke (2019); Tao and Dong (2017) are special cases of the RCPSP-PS/CPR. However, the problem considered in Čapek et al. (2012) contains additional extensions, and therefore, is not a special case of the RCPSP-PS/CPR.

In conclusion, although there is various research on the different elements of the RCPSP-PS/CPR, this has not been combined yet. Furthermore, the research on heuris-

Table 3.2.1: Overview of models with a flexible project structure

Paper	Separate scheduling and selection	Independent succession	Exclusivity criterion	Exact method	Heuristic method	Nonrenewable resources	Production of resources
Kuster et al. (2009)		✓	✓		✓		
Čapek et al. (2012)	✓		✓	✓	✓		
Kellenbrink and Helber (2015)	✓		✓	✓	✓	✓	
Tao and Dong (2017)		✓		✓	✓	✓	
Servranckx and Vanhoucke (2019)		✓	✓		✓		
Chapter 2	✓	✓	✓	✓			
This chapter	✓	✓	✓	✓	✓	✓	✓

tic methods considering nonrenewable resources with consumption and production is limited. To the best of our knowledge, no implementation of this exists.

3.3. RCPSP-PS/CPR

In this section, a formal description of the RCPSP-PS/CPR is given. First, Section 3.3.1 describes the problem and introduces all required notation. Subsequently, Section 3.3.2 gives an MILP formulation for the RCPSP-PS/CPR. This model is based on the model for the RCPSP-PS from Chapter 2. Like nearly all formulations for the RCPSP with a flexible project structure, it uses a time-indexed formulation. Therefore, we use the constraints from the time-indexed formulation of Sahli et al. (2016) to model the nonrenewable resources.

3.3.1. PROBLEM DESCRIPTION

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

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

$k^-_{ri} = k^+_{ri}$. Furthermore, each resource has a total capacity of λ_r.

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

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

Finding a set of executed activities such that this holds for each group is called the **selection problem**, as given in Definition 3.3.1. A representation of the selection problem is given by the **selection graph**. This graph consists of a node for each activity and an arc for each activator-successor relationship. When the selection graph is illustrated, a circular arc is drawn to denote that multiple activator-successor relationships belong to the same group. Figure 3.3.1 displays a selection graph with one selection group with multiple successors and four selection groups with one successor. The selection group with multiple successors has activator 1 and successors $\{2,3\}$.

Definition 3.3.1 (Selection problem)
The selection problem consists of finding a set of executed activities $N' \subseteq N$ with $1, |N| \in N'$, such that for each selection group $g \in G$ it holds that if activator a_g is executed ($a_g \in N'$), there is exactly one executed successor: $|S_g \cap N'| = 1$.

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

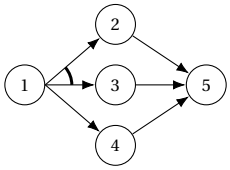


Figure 3.3.1: Example of a selection graph.

3.3.2. PROBLEM FORMULATION

The problem is modeled with binary decision variables X_{it} for each activity $i \in N$ and for each time $t \in T$. This variable is equal to one if activity i is started at time t and zero otherwise. Objective function (3.1a) minimizes the starting time of final activity $|N|$, and therefore, the time of the total project. The first activity is always executed due to Constraint (3.1b). Furthermore, each activity can only be started once, as imposed by Constraints (3.1c).

The selection problem is captured by Constraints (3.1d) and (3.1e). The former impose that if an activator a_g of selection group $g \in G$ is executed, at least one successor activity $i \in S_g$ is executed. The latter ensure that per selection group $g \in G$ with executed

activator a_g, at most one successor activity $i \in S_g$ can be executed. Furthermore, the precedence relations are imposed by Constraints (3.1f) for each precedence relationship $(i, j) \in \mathscr{P}$. Due to these constraints, activity i has to be finished before or at the start of activity j, if both are executed. Here, M is a sufficiently large number ($M \geq |T| + \max_{i \in N} d_i$).

For each resource $r \in R$, Constraints (3.1g) ensure for each time $t \in T$ that the total consumption minus the total production up to time t is smaller than the total resource capacity λ_r. Finally, Constraints (3.1h) impose that decision variables X_{it} are binary. In Table 3.3.1, all notation is given for the model. Furthermore, in Appendix 3.A, all notation used throughout this chapter is listed.

Table 3.3.1: Notation used in Constraint set (3.1).

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

$$\min \sum_{t \in T} t X_{|N|t}, \tag{3.1a}$$

$$\sum_{t \in T} X_{1t} = 1, \tag{3.1b}$$

$$\sum_{t \in T} X_{it} \leq 1, \quad \forall i \in N, \tag{3.1c}$$

$$\sum_{t \in T} X_{a_g t} \leq \sum_{i \in S_g} \sum_{t \in T} X_{it}, \quad \forall g \in G, \tag{3.1d}$$

$$\sum_{j \in S_g} \sum_{t \in T} X_{jt} \leq |S_g| - \big(|S_g| - 1\big) \sum_{t \in T} X_{a_g t}, \quad \forall g \in G, \tag{3.1e}$$

$$\sum_{t \in T} (t + d_i) X_{it} \leq \sum_{t \in T} t X_{jt} + M\big(1 - \sum_{t \in T} X_{jt}\big), \quad \forall (i, j) \in \mathscr{P}, \tag{3.1f}$$

$$\sum_{i \in N} \left(\sum_{\tau=1}^{t} k^-_{ri} X_{i\tau} - \sum_{\tau=1}^{t-d_i} k^+_{ri} X_{i\tau} \right) \leq \lambda_r, \quad \forall r \in R, t \in T, \tag{3.1g}$$

$$X_{it} \in \{0, 1\}, \quad \forall i \in N, t \in T. \tag{3.1h}$$

3.4. SOLUTION METHOD

In this section, a *Hybrid Differential Evolution* (HDE) algorithm is presented to find feasible solutions for the RCPSP-PS/CPR. Naturally, a search-based algorithm requires a fast way of exploring different solutions to the selection problem. This poses a challenge, since the selection problem is NP-hard (Barták et al., 2007). Therefore, Section 3.4.1 gives a special case of the selection problem and a polynomial-time algorithm to find feasible solutions to this special case. This algorithm is used within the HDE algorithm that is given in Section 3.4.2. Furthermore, since the RCPSP-PS/CPR is introduced in this chapter, there is no heuristic algorithm for comparison for general instances. Therefore, in Section 3.4.3, we briefly introduce an *Ant Colony Optimization* (ACO) algorithm for comparison purposes.

3.4.1. GROUP ORDERINGS

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

By applying Algorithm 5 on a group ordering J, a selection of activities is obtained. As it is explained below, this selection may or may not be feasible, depending on certain instance properties. The algorithm keeps track of a list of executed activities N^e and a list of forbidden activities N^f. Then, following the order of group ordering J, each selection group $g \in J$ is evaluated. First, it is checked if the activator a_g is executed. If this is the case, a successor has to be selected. The set of candidate activities N_g^c consists of the set of successor activities S_g, minus the set of forbidden activities. From these candidate activities, one activity n is selected. This selection is based on the algorithm in which Algorithm 5 is used, as is explained in Sections 3.4.2 and 3.4.3. Finally, activity n is added to the set of executed activities and all other successors $S_g \setminus \{n\}$ are added to the set of forbidden activities.

As stated in Definition 3.4.1, a group ordering is *feasible* if Algorithm 5 always results in a feasible selection of activities, regardless of the selection made in line 7. From the selection problem, as defined by Constraints (3.1d) and (3.1e), there are two ways in which a selection can be infeasible for group g: the activator a_g is selected and either none of the successors in S_g are selected, or more than one successor is selected. Since Algorithm 5 keeps track of forbidden activities, the latter case will not happen. The first case can happen in two ways: the set of candidate activities is empty, or an activator a_g is selected for execution after group g has been processed. Based on this, Observation 1 gives a condition for feasibility of a group ordering.

Algorithm 5 Using a group ordering J

1: $N^e \leftarrow \{1\}$ ▷ Executed activities
2: $N^f \leftarrow \emptyset$ ▷ Forbidden activities
3: **for** $g \in J$ **do**
4: **if** $a_g \in N^e$ **then**
5: $N_g^c \leftarrow S_g \setminus N^f$ ▷ Candidate activities
6: **if** $N_g^c \neq \emptyset$ **then**
7: $n \leftarrow$ Select from N_g^c
8: $N^e \leftarrow N^e \cup \{n\}$
9: $N^f \leftarrow N^f \cup (S_g \setminus \{n\})$
10: **end if**
11: **end if**
12: **end for**
13: **return** N^e

Definition 3.4.1 (Feasible group ordering)
*A feasible group ordering J is an ordered list of selection groups such that applying Algorithm 5 always returns a feasible solution to the selection problem, regardless of the selection choice in line 7. If J does not contain all selection groups G, it is called a **partial** group ordering.*

Observation 1
A (partial) group ordering $J = [j_1, \cdots, j_{|J|}]$ is feasible (i.e., Algorithm 5 always returns a feasible activity selection) if it satisfies the following properties:

Property 1 *There exist no groups $j_a, j_b \in J$ with $a < b$, such that*

$$a_{j_a} \in S_{j_b}. \tag{3.2}$$

Property 2 *There does not exist a group $j_a \in J$ such that*

$$S_{j_a} \subseteq \bigcup_{i \in \{1,\cdots,a-1\} | S_{j_i} \neq S_{j_a}} S_{j_i}. \tag{3.3}$$

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

Furthermore, due to Property 2, for each group j_a, there is a successor activity $j \in S_{j_a} \setminus \cup_{i=1}^{a-1} S_{j_i} \neq \emptyset$ that is not a successor activity of an earlier group and therefore can be chosen without causing a conflict. If any earlier group has exactly the same successors, making a choice here automatically results in no conflicts arising when making a choice for j_a. Therefore, these can be excluded from the right-hand side of Equation (3.3). □

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

Definition 3.4.2 (Strict successor containment)
A selection group $g \in G$ has strict successor containment on selection group $h \in G$, if the successors of selection group h are a strict subset of the successors of selection group g; i.e., $S_h \subsetneq S_g$.

Definition 3.4.3 states how to create a group graph.

Definition 3.4.3 (Group graph)
A group graph $\mathcal{H}$ is a mixed (containing directed and undirected edges) graph based on the selection groups. It is created in the following way:

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

We will show that if the group graph does not contain a cycle, the corresponding RCPSP-PS/CPR instance has a feasible group ordering. For this, we need the following lemma:

Lemma 3
If an instance of the selection problem has an acyclic group graph, every group $g \in G$ with $|S_g| = 1$ satisfies Property 2 in Observation 1 for any (partial) group ordering.

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

Thus, if there is a group $h \in G$ with $i \in S_h$, it follows that $S_h = \{i\} = S_g$. In this case, group h is not included in the right-hand side of Equation (3.3). Since there does not exist a group $h \in G$ with $i \in S_h$ and $S_h \neq S_g$, the right-hand side will never include activity i and Equation (3.3) will never hold for $j_a = g$. □

Subsequently, we show how to construct partial feasible group orderings in Lemma 4.

Lemma 4
Consider a group graph $\mathcal{H}$ that is acyclic. Then, for each connected component of undirected edges, a breadth-first search ordering starting in any node satisfies both properties in Observation 1 and thus is a partial feasible group ordering.

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

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

We now prove Property 2. For this, we introduce

$$\mathscr{F}_J(a) = \bigcup_{i \in \{1,\cdots,a-1\} | S_{j_i} \neq S_{j_a}} S_{j_i}, \tag{3.4}$$

which is the right-hand side of Equation (3.3) and where j_i is the i^{th} entry of (partial) group ordering J. Furthermore, we consider group $j_a \in J$ where a is the index of group j_a in J. Now, Property 2 is satisfied if for each $j_a \in J$ it holds that $S_{j_a} \nsubseteq \mathscr{F}_J(a)$.

Due to Lemma 3, we know that Property 2 is satisfied if $|S_{j_a}| = 1$, thus we only need to consider the case that $|S_{j_a}| > 1$. Since undirected edges are created between groups with successor overlap if at least one group has more than one successor, we know that if two groups in G' have successor overlap, there is an undirected edge between them.

Since J is ordered breadth-first, there is at most one group $j_c \in J$ with successor overlap and $c < a$. An example of this is shown in Figure 3.4.1. We now consider two cases: the case that $S_{j_a} \neq S_{j_c}$ and the case that $S_{j_a} = S_{j_c}$.

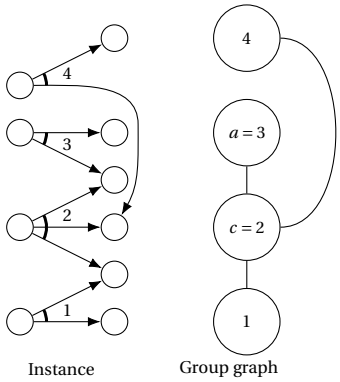


Figure 3.4.1: A breadth-first ordering J on a connected component in the group graph. This illustrates that for each group $j_a \in J$, there can be only one group $j_c \in J$ with $c < a$.

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

$$S_{j_a} \cap \mathscr{F}_J(a) = S_{j_a} \cap S_{j_c}. \tag{3.5}$$

For contradiction to Property 2, assume that $S_{j_a} \subseteq \mathscr{F}_J(a)$. Then, it follows that $S_{j_a} \cap \mathscr{F}_J(a) = S_{j_a}$. Combining with Equation (3.5) gives $S_{j_a} = S_{j_a} \cap S_{j_c}$, from which it follows that $S_{j_a} \subseteq S_{j_c}$. Since, by assumption, $S_{j_a} \neq S_{j_c}$, we get $S_{j_a} \subsetneq S_{j_c}$. However, due to strict successor containment, this means that there is a directed edge from j_a to j_c. This edge forms a cycle with the undirected edge, and therefore contradicts the assumption of an acyclic group graph. Thus, in the case of $S_{j_a} \neq S_{j_c}$, the assumption $S_{j_a} \subseteq \mathscr{F}_J(a)$ cannot be true, and therefore, $S_{j_a} \nsubseteq \mathscr{F}_J(a)$. Thus, Property 2 holds.

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

□

Subsequently, we can combine all partial feasible group orderings. This is stated in Lemma 5.

Lemma 5
If a group graph $\mathscr{H}$ is acyclic, there is a feasible group ordering.

Proof. We can construct a feasible group ordering J by combining all partial group orderings from the connected components of undirected edges in Lemma 4 by *topological sort*; if there is a directed path from connected component of undirected edges G' to connected component of undirected edges H', the partial group ordering of H' has to appear after the partial group ordering of G'. Note that we define nodes without undirected edges as connected components of size 1. By assumption of an acyclic group graph, it is always possible to sort the partial group orderings topologically.

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

As the base step for induction, Lemma 4 states that J'_1 satisfies both properties of Observation 1. For the induction step, we take the feasible partial group ordering $J(n) = (J'_1, \cdots, J'_n)$ with $n < |\mathscr{C}|$ and append the partial group ordering J'_{n+1} to obtain $J(n+1)$. We now show that $J(n+1)$ satisfies the properties from Observation 1.

First, we show that for each combination of groups $g \in J(n)$ and $h \in J'_{n+1}$, Property 1 holds. Assume, for contradiction, that it does not hold, and thus, g is a successor of h ($a_g \in S_h$). Then, by Step 2 in Definition 3.4.3, there would also be a directed edge from component J'_{n+1} to $J(n)$. This contradicts the topological sorting in combination with an acyclic group graph, and therefore, Property 1 is satisfied.

To prove Property 2, we consider group $h \in J'_{n+1}$ and show that Equation (3.3) is never satisfied for $h = j_a$. For this, we define K as the preceding part of the group ordering $\{j_1, \cdots, j_{a-1}\}$ and split this up into $K_1 = \{j_1, \cdots, j_b\} = J(n)$ and $\{j_{b+1}, \cdots, j_{a-1}\} = K_2 \subset J'_{n+1}$.

First, we consider $K_1 = J(n)$. Since Property 2 is satisfied if $|S_h| = 1$ due to Lemma 3, we only need to consider the case where $|S_h| > 1$. For this case, we know that there is

no group $g \in J(n)$ with successor overlap between g and h. Otherwise, there would be an undirected edge between g and h and they would be in the same connected component. This means that there is no overlap between all successors of $J(n)$ and S_h : $\left(\bigcup_{g \in J(n) | S_g \neq S_h} S_g\right) \cap S_h = \emptyset$.

Now, consider $K_2 \subset J'_{n+1}$. Property 2 holds by Lemma 4, which results in $S_h \nsubseteq \bigcup_{g \in K_2 | S_g \neq S_h} S_g$. Combining this with the equation for K_1 gives $S_h \nsubseteq \bigcup_{g \in K | S_g \neq S_h} S_g$. This means that Property 2 is satisfied for each $h \in J_{n+1}$.

3

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

An example of the process of obtaining a feasible group ordering can be seen in Figure 3.4.2. First, the selection graph is given. Subsequently, in Step 2, the group graph is presented. The connected components based on undirected edges are evaluated in Step 3. Each connected component is ordered by breadth-first. In Step 4, a final ordering is obtained, respecting both the topological sorting and the partial orderings of Step 3. It can be seen that this group ordering satisfies both properties of Observation 1, and therefore, using Algorithm 5 always results in a feasible selection. Theorem 2.

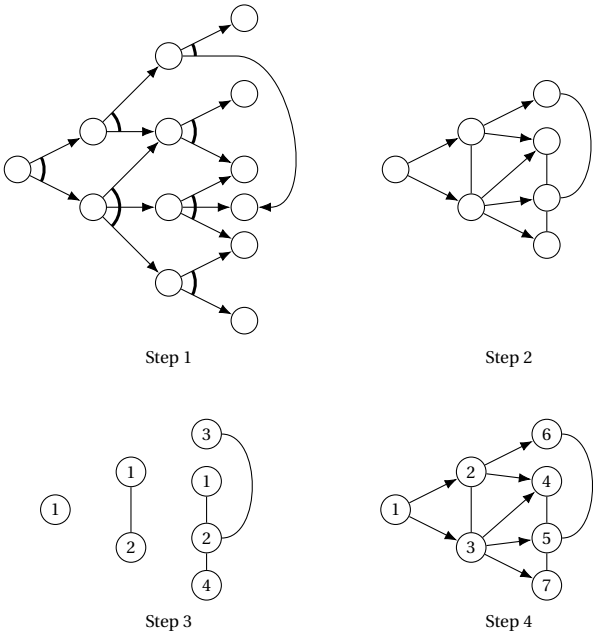


Figure 3.4.2: Creating a feasible group ordering.

In conclusion, we can create a feasible group ordering, given an instance of the RCPSP-PS/CPR with an acyclic group graph, by executing the steps introduced above. This gives Theorem 2.

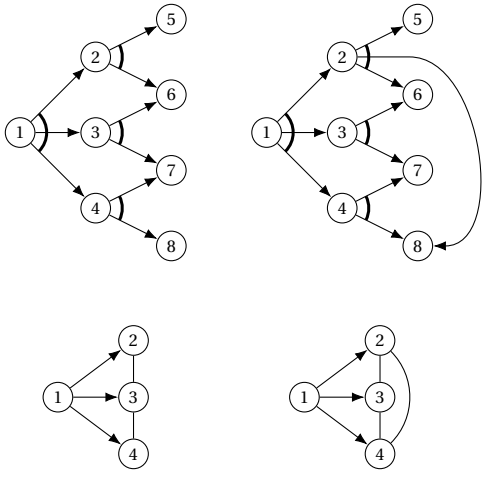
Theorem 2
If an instance of the RCPSP-PS/CPR has an acyclic group graph, a feasible group ordering can be found in polynomial time.

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

1. Construct a group graph $\mathscr{H}$, as described in Definition 3.4.3.
2. Find a breadth-first ordering for each connected component in $\mathscr{H}$, as described in Lemma 4.
3. Topologically sort the connected components, as described in Lemma 5.

Step 1 consists of 4 operations, described in Definition 3.4.3. The last two of these 4 operations have the highest time complexity, namely $O(|G|^2|N|)$. Step 2 and 3 can both be solved by a breadth-first ordering algorithm, which has a time complexity of $O(|G|^2)$. Therefore, the combined time complexity is $O(|G|^2|N|)$. Furthermore, Lemma 5 shows that this group ordering is feasible. □

The heuristics introduced in this chapter require a group ordering to create feasible solutions to the selection problem. As Theorem 2 states, this group ordering can be found if the group graph is acyclic. An example is given in Figure 3.4.3. Here, Figure 3.4.3a shows an instance with an acyclic group graph. Therefore, this instance can be solved by the methods of this chapter. Conversely, Figure 3.4.3b shows an instance with a cyclic group graph: groups $2 \rightarrow 3 \rightarrow 4 \rightarrow 2$ form a cycle. Therefore, we cannot use the breadth-first ordering to find a feasible group ordering. This means that Algorithm 5 is not guaranteed to find a feasible selection, and thus, this instance cannot be solved by the methods proposed in this chapter.



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

Figure 3.4.3: Instances with corresponding group graphs.

3.4.2. HYBRID DIFFERENTIAL EVOLUTION ALGORITHM

The algorithm introduced is a *Hybrid Differential Evolution* (HDE) algorithm with so-called *Forward Backward Improvement* (FBI). The Differential Evolution algorithm, originally introduced by Storn and Price (1995), was chosen based on multiple successful implementations in the literature (Quoc et al., 2020; Sallam et al., 2020; Zaman et al., 2021) for RCPSP variants and on good preliminary results.

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

SOLUTION REPRESENTATION

Since DE is an algorithm for continuous variables, a method to convert a DE solution into a schedule is required. Each agent in the DE algorithm is represented by a $2 \times |N|$ **priority matrix** A that contains two priorities (scalars) per activity. The **selection priority** for activity $i \in N$ is denoted by A_{1i}. This is used to make a selection of executed activities. Furthermore, A_{2i} is the **scheduling priority**, which is used to schedule the selected activities. To convert a priority matrix A into a feasible schedule, we first determine the executed activities. This is done by using a feasible group ordering J and applying Algorithm 5. In this algorithm, on line 7, the successor with the highest *selection* priority is selected. This results in a set of executed activities N^e.

Next, these executed activities are scheduled according to Algorithm 6. Here, the activities are scheduled iteratively. In each iteration, at line 5, the available activity with the highest *scheduling* priority A_{2i} from all available activities is chosen. An activity $j \in N$ is available if all executed predecessors $P_j \cap N^e$ are scheduled, where P_j is the set of time-based predecessors of activity j.

Algorithm 6 Scheduling activities N^e

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

If an activity $i \in N$ has to be scheduled, it will be at the earliest time possible. This is as soon as all predecessors are finished and the resource availability is sufficient. However, due to nonrenewable resources not automatically regenerating, it might happen that it is not possible to schedule activity i in a resource feasible way. If this is the case, then activity i is scheduled at the first possible time when all required *renewable* resources are available. This causes resource infeasibility. To guide the search towards the feasible

solution region, we penalize the objective function. Let p_{rt} be the resource availability of nonrenewable resource $r \in R^n$ at time t of a solution. Then, the objective value *used within the heuristic algorithm* is the makespan (which we want to minimize) plus a penalty of $M \cdot \sum_{r \in R^n} \sum_{t \in T} -\min(0, p_{rt})$. Here, M is a sufficiently large number such that a larger resource deficit always results in a larger objective value.

DIFFERENTIAL EVOLUTION

The core of the HDE algorithm is the differential evolution algorithm. Like many population-based algorithms, this method explores the search space by iteratively creating *offspring* solutions from the current population. Let γ be the size of the population $\mathscr{A}$. Then, we initialize $\mathscr{A}$ as the set of γ $2 \times |N|$ matrices containing random values between 0 and 1.

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

$$B = B^1 + w\left(B^2 - B^3\right), \tag{3.6}$$

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

This process creates a new priority matrix, the trial matrix T. If the objective value of this solution is lower than or equal to the objective value of the base matrix A, the base matrix is replaced by the trial matrix. This process repeats itself, until no improvement of the best solution has been found for ω generations. Then, the best-found solution is returned. The complete algorithm is shown in Algorithm 7.

Algorithm 7 Differential evolution

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

COMBINE DIFFERENTIAL EVOLUTION WITH FORWARD BACKWARD IMPROVEMENT

Finally, the DE algorithm is combined with a forward backward improvement (Valls et al., 2008). The complete process in shown in Figure 3.4.4. After the DE algorithm terminates, each solution is subjected to the forward-backward improvement. If this improves the best-found solution, the DE algorithm restarts. If it does not find a new best solution, the algorithm terminates.

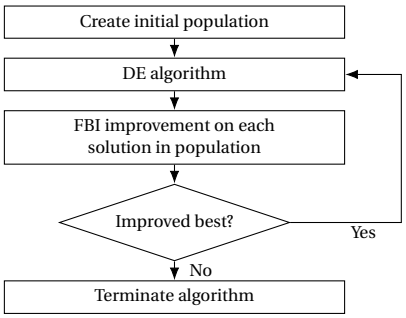


Figure 3.4.4: Hybrid Differential Evolution algorithm.

EXAMPLE

We now show a small example to illustrate the solution presentation. For this, we consider an instance with a single renewable resource r with $\lambda_r = 1$ and where each non-dummy activity $i \in N \setminus \{1,7\}$ has a duration of $d_i = 1$ and a resource requirement $k_{ri} = 1$. Furthermore, the selection and scheduling graphs are shown in Figure 3.4.5. Here, the groups are displayed as the numbers *outside of* the activity nodes. Based on the selection graph, the group graph can be created, which is also shown in Figure 3.4.5. It can be seen that the group graph is acyclic, and by following the procedure in Lemma 5 we obtain group ordering $[1,2,3,4,5,6,7]$. Note that this group ordering is constant for an instance and is not modified during the execution of the heuristic algorithm. We let priority matrix A be

$$A = \begin{bmatrix} 0.8 & 0.4 & 0.9 & 0.5 & 0.4 & 0.3 & 0.4 \\ 0.4 & 0.6 & 0.2 & 0.8 & 1.1 & 1.6 & 0.4 \end{bmatrix} \tag{3.7}$$

and show how to obtain a schedule.

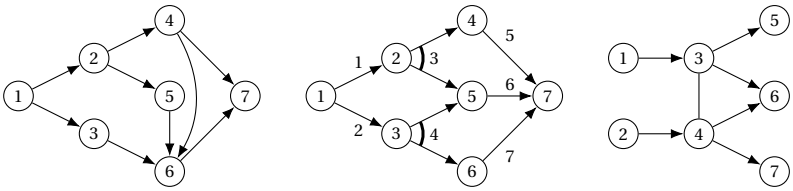


Figure 3.4.5: Precedence, selection and group graph of an example instance.

We first consider the selection problem. This problem uses the priorities given in the top row of A. Let N^e be the set of executed activities and start by selecting the root node: $N^e = \{1\}$. Now, we evaluate the first group in the group ordering: $g = 1$. Since activity 2 is the only successor, it will be selected: $N^e = \{1,2\}$. Similarly, group $g = 2$ selects activity 3. Next, group $g = 3$ is evaluated. The two candidate activities are activity 4 and 5. Since $A_{14} > A_{15}$, we select activity 4 and add activity 5 to the list of forbidden activities. This gives $N^e = \{1,2,3,4\}$ and $N^f = \{5\}$. The next group, $g = 4$, also has 2 successors. However, since activity 5 is forbidden, only activity 6 remains and has to be selected. Subsequently, the remaining selection groups only have one successor. Thus, this results in $N^e = \{1,2,3,4,6,7\}$.

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

3.4.3. ANT COLONY OPTIMIZATION ALGORITHM

Since optimal solutions for general instances of the RCPSP-PS/CPR are unknown, we also present an Ant Colony Optimization (ACO) algorithm for comparison purposes. The

choice for this algorithm is made based on the following two points: firstly, the ACO algorithm was successfully implemented for the classic RCPSP (Merkle et al., 2002). Secondly, due to consumption and production of resources, not every schedule is feasible to the RCPSP-PS/CPR. The ACO algorithm allows for problem specific heuristic rules to find feasible solutions.

We now give a general description and pseudo code of the ACO algorithm for the RCPSP-PS/CPR. Since this algorithm is only for comparison purposes on general instances that cannot be solved to optimality, the details are given in Appendix 3.B.

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

We give a brief overview of the algorithm in Algorithm 8. Here, we use the population parameter γ and the iteration threshold parameter ω. Furthermore, we define $\mathbf{p}$ as the vector containing all pheromone trails and store the activity starting times of the best schedule found under $\mathbf{t}^*$.

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

After all ants create their solution, the new pheromone contribution $\mathbf{p}'$ is calculated based on the solutions found. The new pheromone trails are then determined by a convex combination of the old pheromone trails and the new contribution, where evaporation parameter ρ determines the influence of the new contribution. Finally, the iteration treshold counter θ is incremented by one. If there has been an improvement in the last ω iterations, a new iteration starts. Otherwise, the algorithm terminates and returns the best schedule found represented by $\mathbf{t}^*$.

3.5. COMPUTATIONAL RESULTS

In this section, we present the computational results to evaluate the HDE algorithm. This is done by running a series of tests on instances, both from the literature and newly generated instances. All tests were performed on a single core of a 3.0 GHz Intel XEON CPU with 4 GB RAM. The parameters used for the HDE algorithm are $\gamma = 300$, $\omega = 275$, $\mathbf{c} = [0.15, 0.25]$ and $w = 0.1$. For the ACO algorithm, the parameters are $\gamma = 300$, $\omega = 75$, $q = 1$, $\rho = 0.35$, $\alpha = 0.18$, $\beta = 0.6$, $\mu = 5$ and $\eta = 0.2$. These parameters are the result of a parameter tuning process, which is presented in Appendix 3.C.

Servranckx and Vanhoucke (2019) present both an algorithm to create instances for the RCPSP-AS (without nonrenewable resources) and a tabu search algorithm for solving these. They model the flexible project structure as networks with multiple **stages**. Each stage is represented by an activity, and possibly one or more **alternatives**. Then,

Algorithm 8 Ant Colony Optimization

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

the activities and alternatives across stages are linked to each other to create **branches** of activities that represent an option to execute a part of the project. The algorithm to create instances depends on several parameters, as listed below:

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

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

For further evaluation, two smaller, more uniformly distributed subsets are taken from the *Literature* instance set. These sets are named *Optimal* and *Non-optimal* and contain instances for which the optimal solutions are known and unknown, respectively. The optimal solutions were obtained according to the methods presented in Chapter 2.

These smaller instance sets are created by taking a subset of the set *Literature*, such that they have an equal number of instances for all possible combinations of values for the properties *linked*, *nested* and *flexibility*. Furthermore, a new instance was created for each selected instance by adding nonrenewable resources, as described in van der Beek (2021). Since there are more instances without the optimal solution known, the *Non-optimal* instance set contains more instances (2000) than the *Optimal* instance set

(900). Furthermore, the number of activities of all instances is between 102 and 702, the number of renewable resources is 5 and the number of nonrenewable resources is 2. The instance sets are summarized in Table 3.5.1 and the complete instance sets are given in van der Beek (2021).

Table 3.5.1: Instance sets.

Name	# of instances	Nonrenew. resources	Optimal solutions
Literature	3207	No	Mixed
Optimal	900	in 450 instances	Known
Non-optimal	2000	in 1000 instances	Unknown

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

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

$$BOG(obj) = \frac{obj - obj^*}{obj^*} \cdot 100\%, \tag{3.8}$$

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

3.5.1. INSTANCE SET: LITERATURE

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

For the results in the literature, only objective values of a single run are available. Therefore, for a fair comparison, each algorithm is run once per instance. In Table 3.5.2 is shown how each algorithm performs against the TS algorithm. It can be seen here that in more than half of the cases, the ACO and HDE algorithms tie with the TS algorithm.

We obtain Figure 3.5.1 by omitting all ties and plotting the *relative objective* $obj^r = obj/obj^{TS}$, where obj is the objective of the ACO/HDE algorithm and obj^{TS} is the objective of the TS algorithm. Here, it can be seen that the boxes of the boxplot are com-

pletely below 1, indicating that both ACO and HDE perform better on most non-tied instances. Furthermore, the average relative objective of ACO and HDE is shown to be at 0.982 and 0.976, respectively. This shows that the HDE algorithm performs better than the literature algorithm, and it suggests that the ACO is a reasonable algorithm for comparison purposes.

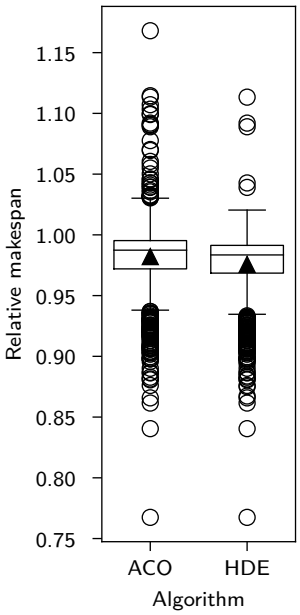


Figure 3.5.1: Comparison of relative makespan $obj^r = \frac{obj}{obj^{TS}}$ of ACO and HDE algorithms, omitting all ties.

Secondly, we compare the BOG for the HDE and literature algorithm over the *nested* parameter. This parameter was chosen since it was found to have the most significant correlation with the performance. This is shown in Figure 3.5.2. Here, the outliers show an increasing trend for the HDE algoritm of the BOG with the *nested* parameter value. However, both the boxes as the outliers show that the performance of the HDE algorithm is better than the performance of the algorithm from literature, even for the maximum *nested* parameter value. Since no clear trend was found for the other instance parameters, $linked$ and $flexibility$, these results are placed in Appendix 3.D.

3.5.2. INSTANCE SET: OPTIMAL

The next set of instances is called *Optimal* and contains instances for which the optimal solution is known. To create this set, a new instance was created for each instance in the set *Literature* by adding nonrenewable resources according to van der Beek (2021). Subsequently, up to 5 instances (depending on how many optimal solutions are known) were randomly selected for each combination of parameters (*linked*, *flexibility*, *nested*).

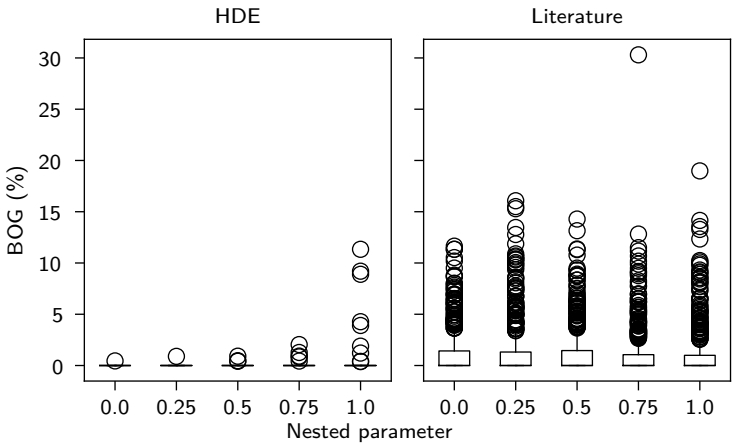


Figure 3.5.2: BOG of HDE and literature algorithm against the *nested* parameter.

This resulted in a set of 900 instances of which half contains both types of resources and the other half contains only renewable resources.

For this instance set, both algorithms are run 10 times. Thus, this resulted in 4500 runs per category (with and without nonrenewable resources). In Table 3.5.3, it is shown how many runs resulted in optimality and how many instances have at least one optimal solution in ten runs. It can be seen that in most runs, the optimal value was reached for both methods. Furthermore, it can be seen that the HDE obtains the optimal value at least once for nearly every instance. When comparing between instances with and without nonrenewable resources, it is shown that both methods perform better for instances with nonrenewable resources. A possible explanation is that the parameter tuning process contained challenging instances on the aspect of resource feasibility.

Next, we evaluate the optimality gaps for the HDE algorithm. The optimality gap is defined as $\frac{obj-obj^*}{obj} \cdot 100\%$, where obj is the found objective value and obj^* the optimal objective value. The optimality gap indicates the difference between the found objective and the optimal objective. Since the HDE algorithm is the main algorithm of this chapter, we further evaluate these optimality gaps for this algorithm only. This is shown in Figure 3.D.4, where the optimality gaps are shown for different *linked* parameter values. Here, it can be seen that in general the optimality gap is quite low (mostly below 2 %). Additionally, an increasing trend can be seen of the optimality gap against the *linked* parameter. This indicates, especially for the highest value, that the HDE algorithm has more difficulty of finding the optimal solution as the *linked* parameter increases. However, since all boxes are reduced to lines of value zero, this trend is only present in the outliers. Since no clear trend was found for the other instance parameters, $nested$ and $flexibility$, these results are placed in Appendix 3.D.

3.5.3. INSTANCE SET: NON-OPTIMAL

Finally, we consider the instance set *Non-optimal*. This set is created similarly to the instance set *Optimal*, however, it contains 10 instances per combination of parameter values and only includes instances for which the optimal value is not known. This results in a set of 2000 instances, of which half contain nonrenewable resources. Since we only have literature results for a part of the instance set and no optimal solutions, the ACO is used as a baseline algorithm for comparison. For each instance, each algorithm is run 10 times.

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

Furthermore, we compare the found objective values per algorithm. In Table 3.5.4, it is shown per algorithm how many times the best value of all 10 runs is lower than the other algorithm and how often they are tied. Similarly, it compares the average values of all 10 runs and shows how often this value is lower for HDE, ACO and how often it is tied. It can be seen that HDE performs clearly better than ACO, although the majority of instances result in a tie.

We also evaluate the BOGs. The average values are shown in Table 3.5.5 and the average values with spread are shown in Figure 3.5.4, categorized by the presence of nonrenewable resources. It can be seen that ACO performs significantly worse for instances with nonrenewable resources. Note that comparison on the BOGs show *relative* performance against other runs for the same instance. Therefore, they might contradict the *absolute* performance, indicated in Table 3.5.3. Furthermore, we evaluate the BOG compared to the number of activities, as shown in Figure 3.5.5. It shows, not surprisingly, that the difficulty of solving instances increases with the number of activities. Additionally, it can be seen that for each bin of activity numbers, HDE outperforms ACO on both spread and average value.

To evaluate the effects of nonrenewable resource availability, we use the concept of *resource strength* from Neumann and Schwindt (2003). There, a value of zero indicates that resource profiles of resource-feasible schedules are constant over time and a value of one indicates that the *earliest start schedule* (see Neumann and Schwindt (2003)) is feasible. However, this definition does not hold for the RCPSP-PS/CPR, since not all activities are executed. Nevertheless, the resource strength can still be used as an indication of the level of nonrenewable resource availability. In Figure 3.5.6, the BOGs are shown against the values of the resource strength. In the ACO algorithm, a decreasing trend can be seen, indicating better performance for instances with a larger resource strength. For the HDE, however, no clear trend can be seen.

Finally, the computing times are shown in Figure 3.5.7. As expected, the computing times increase with the number of activities. It also can be seen that the computing times of the HDE algorithm are significantly longer, even though the parameter tuning processes of both algorithms had the same computing time penalty. A possible explana-

tion for this is the high value of $\omega = 275$ for the HDE algorithm, which defines that after each improvement in objective function value, at least 275 iterations are executed.

3.6. CONCLUSION

In this chapter, a model is given for the RCPSP-PS/CPR by taking the model from Chapter 2 and adding nonrenewable resources. Finding a selection of activities that satisfies the selection constraints of the RCPSP-PS/CPR is NP-hard, so in order to find feasible selections quickly in metaheuristics, we restricted the problem. This is done by introducing the concept of group graphs and feasible group orderings.

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

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

Table 3.5.2: Comparison of ACO and HDE algorithms against the TS algorithm from the literature.

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

Table 3.5.3: Number of runs (out of 4500 per category) resulting in the optimal value and number of instances (out of 450 per category) resulting in at least one optimal solution.

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

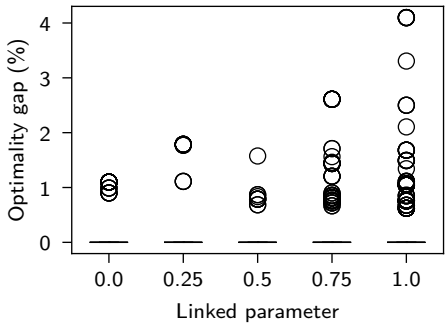


Figure 3.5.3: Boxplots of optimality gaps for the HDE algorithm against the *linked* parameter.

Table 3.5.4: For how many instances each algorithm performs better than the other, evaluated on best out of 10 runs and on average of 10 runs.

	Best	Average
ACO	1	6
HDE	147	758
Tie	1852	1236

Table 3.5.5: Average values of BOGs, categorized on instances with and without nonrenewable resources.

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

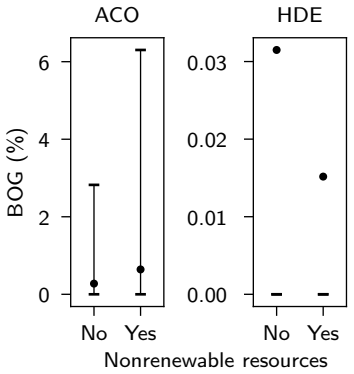


Figure 3.5.4: Mean, 2.5 and 97.5 percentile for BOGs against the resource type for both algorithms.

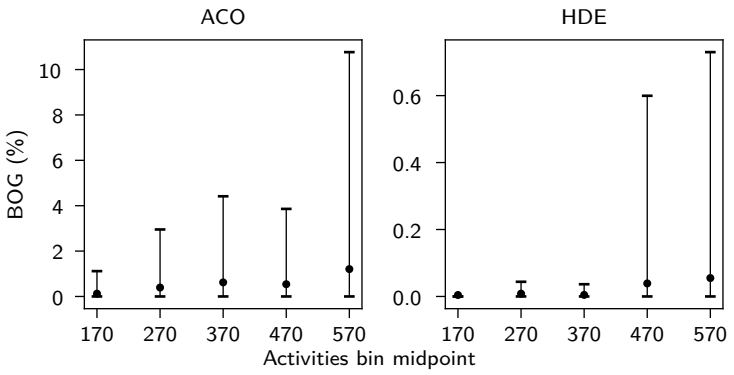


Figure 3.5.5: Mean, 2.5 and 97.5 percentile for BOGs against the activities per instance for both algorithms. Instances are grouped into bins. Each bin has a range of 100 activities.

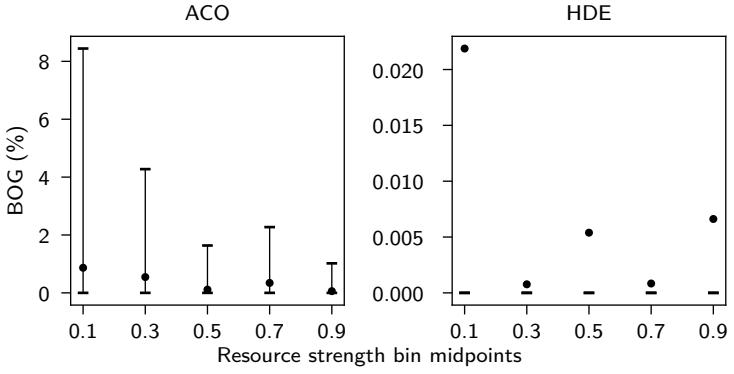


Figure 3.5.6: Mean, 2.5 and 97.5 percentile for BOGs against the resource strength. Resource strength values are grouped into bins. Each bin has a range of 0.2.

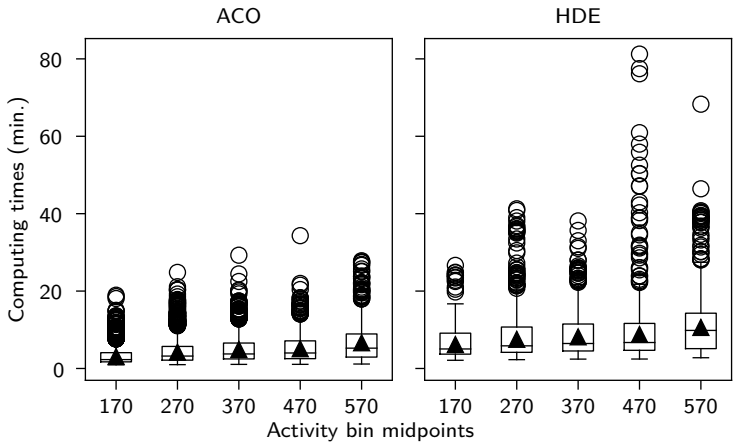


Figure 3.5.7: Durations against the activities per instance for both algorithms. Instances are grouped into bins. Each bin has a range of 100 activities.

APPENDIX

3.A. NOTATION

Sets

G	Selection groups.
N	Activities.
P_j	Predecessors of activity $j \in N$ in the precedence graph.
R	Resources.
R^r	Renewable resources.
R^n	Nonrenewable resources.
S_g	Successor activities of selection group $g \in G$.
T	Time periods.
$\mathscr{P}$	Time-based predecessor-successor pairs.

Variables

X_{it}	1 if activity $i \in N$ is executed at time $t \in T$, zero otherwise.

Parameters

a_g	Activating activity of selection group $g \in G$.
d_i	Duration of activity $i \in N$.
k_{ri}	Net resource production of resource $r \in R$ for activity $i \in N$.
k^+_{ri}	Production of resource $r \in R$ for activity $i \in N$.
k^-_{ri}	Consumption of resource $r \in R$ for activity $i \in N$.
M	Sufficiently large number.
q	Parameter for ACO algorithm setting the number of considered solutions.
w	Weight factor HDE algorithm.
α	ACO parameter indicating influence of selection pheromone values.
β	ACO parameter indicating influence of scheduling pheromone values.
γ	Population size in ACO and HDE algorithms.
η	ACO parameter indication probability of adding heuristic contribution.
λ_r	Capacity of resource $r \in R$.
μ	Resource tightness.
ρ	Evaporation coefficient for ACO algorithm.
ω	Threshold parameter for ACO and HDE algorithms.

Vectors

$\mathbf{c}$	Crossover probability for HDE algorithm.
$\mathbf{p}$	Pheromone trails.
$\mathbf{p}^{cs}$	Cumulative selection pheromone trail.
$\mathbf{p}^{sc}$	Scheduling pheromone trail.
$\mathbf{p}^{se}$	Selection pheromone trail.

Functions

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

3.B. ANT COLONY OPTIMIZATION ALGORITHM

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

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

3.B.1. PHEROMONE TRAILS

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

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

$$\mathbf{p} := \mathbf{p}(1-\rho) + \mathbf{p}' \cdot \rho. \tag{3.9}$$

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

pheromone contributions of selection, scheduling and cumulative selection, respectively.

Let $\mathscr{A}$ be the set of solutions found by all ants in the current iteration and let A^* be the best solution of all previous iterations. Then, we take $\mathscr{B}$ as the best q solutions from $\mathscr{A} \cup \{A^*\}$, where q is an algorithm parameter. Thus, we have $\mathscr{B} = \{B_1, \cdots, B_q\}$. Each solution $B_m \in \mathscr{B}$ is then a priority matrix. Here, the top row contains $(B_m)_{1i} = 1$ if activity $i \in N$ is executed and $(B_m)_{1i} = 0$ otherwise. The bottom row contains a priority vector for all selected activities, as to be used in Algorithm 6. Now, we calculate the selection pheromone contribution $\mathbf{p}'^{se}$. This is done by the following equation:

$$p_i'^{se} = \sum_{m=1}^{q} \frac{(B_m)_{1i}}{m}, \quad \forall i \in N. \tag{3.10}$$

Thus, in Constraints (3.10), the m^{th} best solution gives a contribution of $1/m$ to every activity that is executed. For the scheduling pheromone update, the following equation is used:

$$p_{ij}'^{sc} = \sum_{m=1}^{q} \left(\begin{cases} 1/m \text{ if } (B_m)_{2i} > (B_m)_{2j} \text{ and } i,j \in N^e \\ 0 \text{ otherwise} \end{cases} \right), \quad \forall i,j \in N. \tag{3.11}$$

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

$$p_i'^{cs} = \sum_{m=1}^{q} (B_m)_{1i}, \quad \forall i \in N. \tag{3.12}$$

For each node, a contribution of 1 is added if this node is executed in the selected solution. Combining $\mathbf{p}'^{se}$, $\mathbf{p}'^{sc}$ and $\mathbf{p}'^{cs}$ gives $\mathbf{p}'$ that can be used in Constraints (3.9) to calculate the new pheromone values.

3.B.2. HEURISTIC RULE

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

As for HDE, the scheduling of a set of executed activities N^e is done by Algorithm 6. In each iteration of this algorithm, there is a set of *scheduled activities* N^s and a set of *candidate activities* N^c, where one candidate activity $i \in N^c$ has to be selected to be added to the scheduled activities. Furthermore, we introduce the net resource production of resource $r \in R$ for activity $i \in N$ as $k_{ri} = k_{ri}^+ - k_{ri}^-$.

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

The resource supply $RS_r(N^c, i)$ represents the amount of nonrenewable resource $r \in R^n$ generated by candidate node $i \in N^c$, and thus, the preference of selecting this activity.

The RT is then defined as follows:

$$RT_r(N^s) = \frac{1}{2} erf\left(2 - \frac{\lambda_r + \sum_{j \in N^s} k_{rj}}{\mu \lambda_r}\right), \quad \forall r \in R^n. \tag{3.13}$$

The top side of the fraction in Constraints (3.13) represents the net amount of resource r generated by all scheduled activities N^s plus the initial available amount of resource r. Thus, this is a measure for the total available amount of resource r divided by the initial amount of resource r times the resource tightness parameter μ. Then, the Gauss error function erf is used together with the constants $\frac{1}{2}$ and 2 to create Constraints (3.13). This equation is close to zero when the available amount of resource r is equal to or higher than $\mu\lambda_r$, where λ_r is the original resource availability. If the resource availability decreases, the RT increases, and thus, more priority is given to the generation of this resource.

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

$$RS_r(N^c, i) = \begin{cases} \dfrac{k_{ri} - \min_{j \in N^c} k_{rj}}{\max_{j \in N^c} k_{rj} - \min_{j \in N^c} k_{rj}}, & \text{if } \min\limits_{j \in N^c} k_{rj} < \max\limits_{j \in N^c} k_{rj} \\ 0, & \text{otherwise.} \end{cases} \tag{3.14}$$

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

$$HC(N^s, N^c, i) = \frac{1}{|R^n|} \sum_{r \in R^n} RT_r(N^s) \cdot RS_r(N^c, i). \tag{3.15}$$

Each term in the sum of Constraints (3.15) consists of the resource tightness times the resource supply. This means that the heuristic contribution for activity $i \in N^c$ is increased the most, if it generates a nonrenewable resource (high resource supply) that is near to depletion (high resource tightness).

3.B.3. CREATING SOLUTIONS

Each ant creates a new solution at each iteration. This is done by combining the information from the pheromone trails with the heuristic contribution. First, the ant creates a selection of executed activities N^e. This is done by executing Algorithm 5. Here, at line 7, the next activity i from selection group $g \in G$ is selected from all candidates N_g^c with probability $PR^{se}(N_g^c, i)$:

$$PR^{se}(N_g^c, i) = \frac{(p_i^{se})^\alpha}{\sum_{j \in N_g^c} (p_j^{se})^\alpha}, \tag{3.16}$$

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

Subsequently, the executed activities are scheduled according to Algorithm 6. On line 5, candidate activity $i \in N^c$ is selected from candidates $N^c \subseteq N$, given that activities in set N^s are already scheduled, with probability $PR^{sc}(N^s, N^c, i)$. To calculate this probability, we first calculate the *Scheduling Score* $SC(N^c, i)$:

$$SC(N^c, i) = \left(\frac{\sum_{j \in N^c \setminus \{i\}} p_{ij}^{sc}}{|N^e|} \right)^{\beta} \cdot \frac{1}{p_i^{cs}}. \tag{3.17}$$

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

$$PR^{sc}(N^s, N^c, i) = \begin{cases} \dfrac{SC(N^c, i) + HC(N^s, N^c, i)}{\sum_{j \in N^c} SC(N^c, j) + HC(N^s, N^c, j)} & \text{, with probability } \eta, \\ \dfrac{SC(N^c, i)}{\sum_{j \in N^c} SC(N^c, j)} & \text{otherwise.} \end{cases} \tag{3.18}$$

3.B.4. FULL ALGORITHM

Finally, we combine the above to present the ACO algorithm, as given in Algorithm 9. For this algorithm, we use the population parameter γ and the iteration threshold parameter ω from the HDE algorithm. The algorithm initializes by setting all pheromone trail values to 1.

At each iteration, each ant creates a new schedule by first selecting activities according to Algorithm 5 and subsequently scheduling these according to Algorithm 6. If this new schedule represented by $\mathbf{t}$ is better than the best solution represented by $\mathbf{t}^*$, $\mathbf{t}$ is stored as best solution. After all ants have created a new solution, the pheromone values are updated. This process is repeated until there has not been an improvement for ω iterations.

Algorithm 9 Ant Colony Optimization

$\mathbf{p} \leftarrow \mathbf{1}$
$\mathbf{t}^* \leftarrow \infty$
$\theta \leftarrow 0$
while $\theta < \omega$ **do**
  for $i \in [1, \cdots, \gamma]$ **do**
    $N^e \leftarrow$ Executed activities by Algorithm 5 and Constraints (3.16)
    $\mathbf{t} \leftarrow$ Schedule by Algorithm 6 and Constraints (3.18)
    if $\mathbf{t}_{|N|} < \mathbf{t}^*_{|N|}$ **then**
      $\mathbf{t}^* \leftarrow \mathbf{t}$
      $\theta \leftarrow -1$
    end if
  end for
  $\mathbf{p}'^{se} \leftarrow$ by Constraints (3.10)
  $\mathbf{p}'^{sc} \leftarrow$ by Constraints (3.11)
  $\mathbf{p}'^{cs} \leftarrow$ by Constraints (3.12)
  $\mathbf{p}' = [\mathbf{p}'^{se}, \mathbf{p}'^{sc}, \mathbf{p}'^{cs}]$
  $\mathbf{p} := \mathbf{p}(1-\rho) + \mathbf{p}' \cdot \rho$
  $\theta \leftarrow \theta + 1$
end while
return $\mathbf{t}^*$

3.C. PARAMETER TUNING

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

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

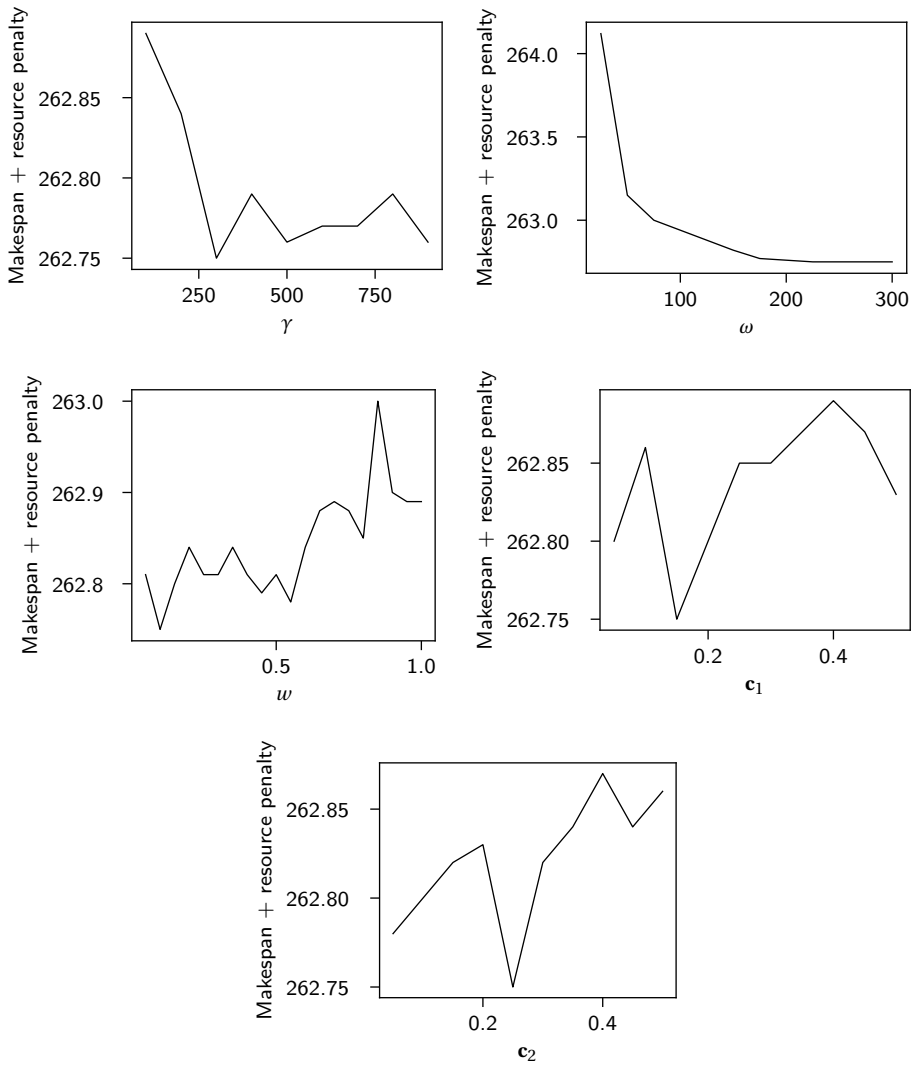


Figure 3.C.1: Average makespan plus resource penalties for the final iteration of the HDE parameter tuning process.

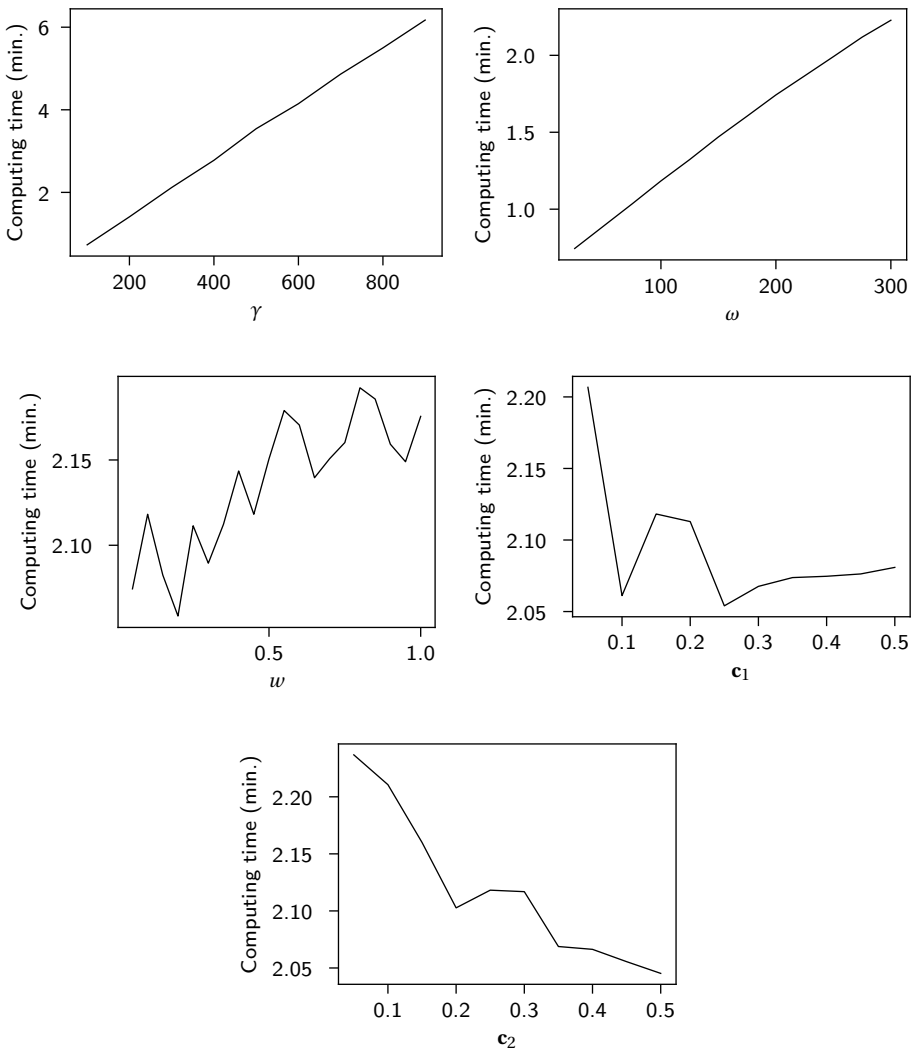


Figure 3.C.2: Average computing times for the final iteration of the HDE parameter tuning process.

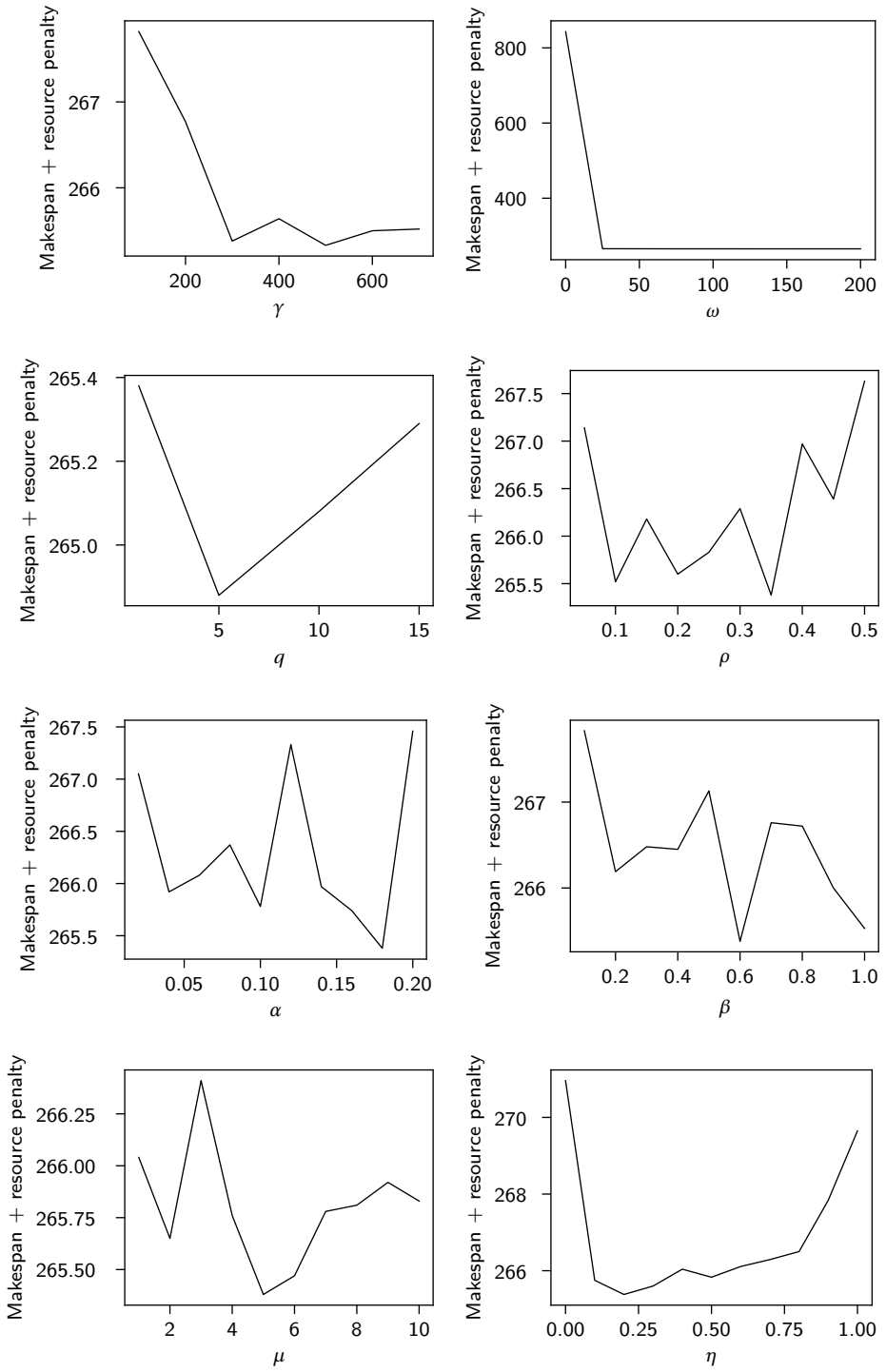


Figure 3.C.3: Average makespan plus resource penalties for the final iteration of the ACO parameter tuning process.

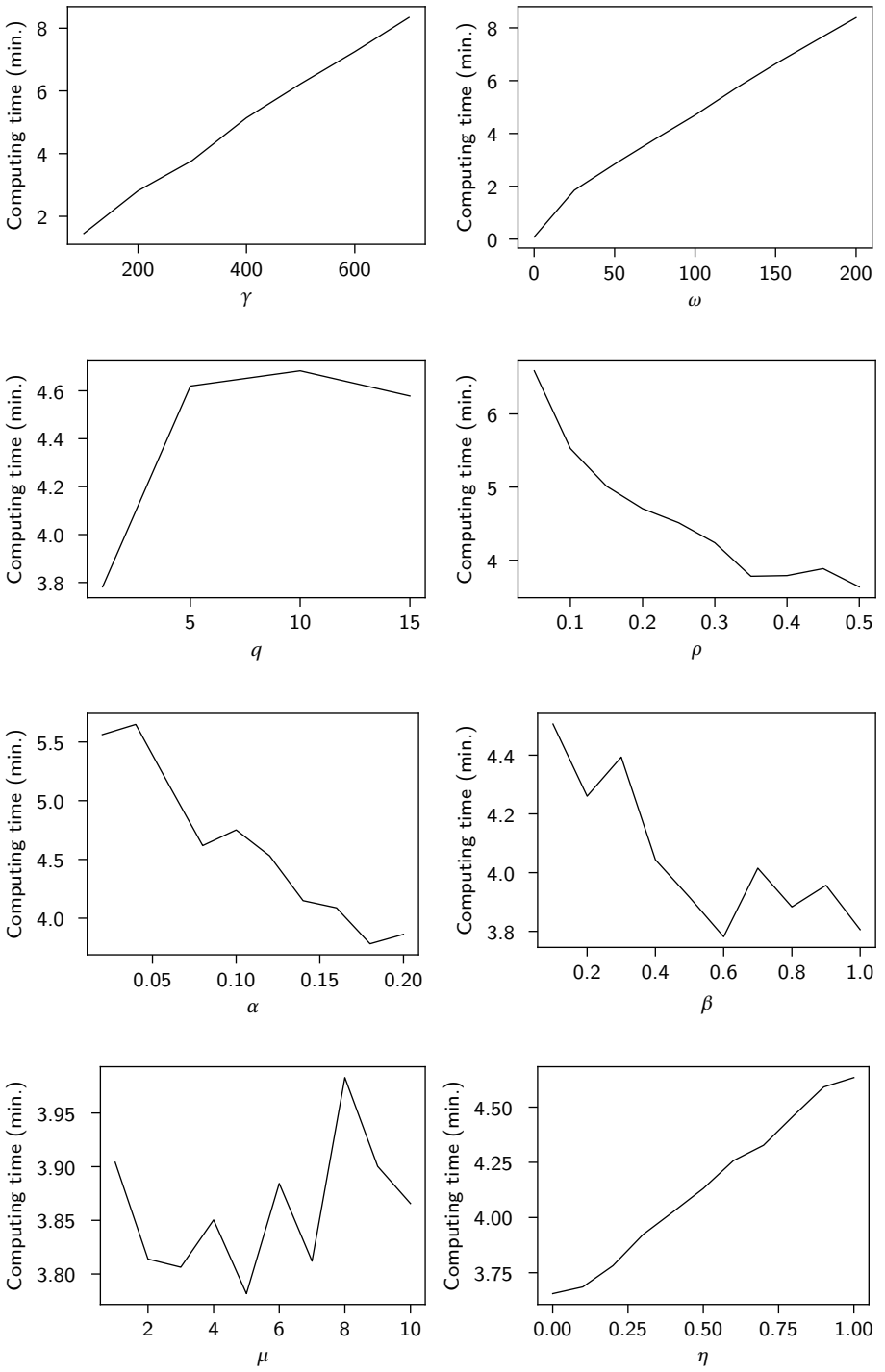


Figure 3.C.4: Average computing times for the final iteration of the ACO parameter tuning process.

3.D. ADDITIONAL COMPUTATIONAL RESULTS

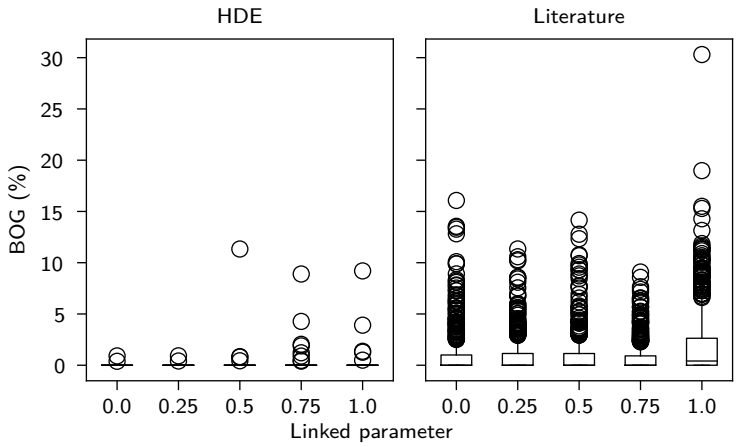


Figure 3.D.1: BOG of HDE and literature algorithm against the *linked* parameter.

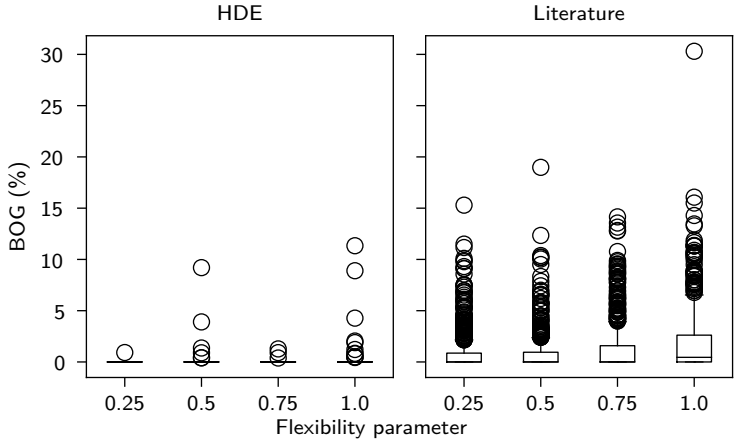


Figure 3.D.2: BOG of HDE and literature algorithm against the *flexibility* parameter.

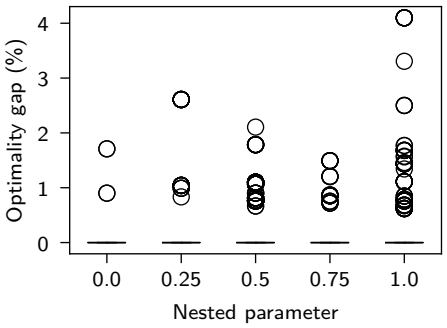


Figure 3.D.3: Boxplots of optimality gaps for the HDE algorithm against the *nested* parameter.

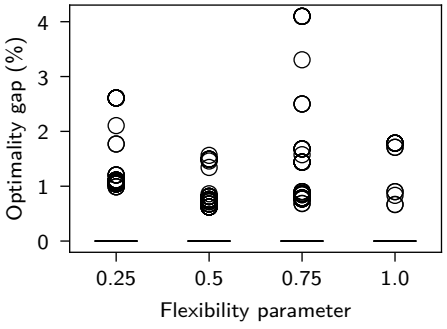


Figure 3.D.4: Boxplots of optimality gaps for the HDE algorithm against the *flexibility* parameter.

4

PROFIT MAXIMIZATION WITH STOCHASTIC PROJECT ARRIVALS

So far, in this thesis, the optimization goal is to minimize the makespan for a single project. However, in modular shipbuilding, the final product consists of a product family, instead of a single product. Consequently, the goal is to produce all products within this family as efficiently as possible. Therefore, this chapter extends the model to support two important characteristics of modular shipbuilding: inventory allocation and the stochastic arrival of new projects. It should be noted that these characteristics are strongly correlated: inventory allocation is done based on the expectation of arriving projects. If it was known exactly which project will arrive, having the right items in the inventory would be much easier. Unfortunately, this is not the case: if we want to fully capture the benefits of having pre-built assemblies, this decision might have to be made before the arrival of the product order.

Choosing the correct inventory allocation for minimal makespan is easy: simply keep the inventory levels as high as possible. However, since there are also costs incurred by having items in inventory, makespan minimization is not a realistic optimization objective in this setting. Instead, one should compare the additional costs due to inventory allocation to the benefits obtained by shorter project makespans. Then, it might be revealed that constructing a certain assembly as soon as the ship order comes in might be a better decision than having this assembly always on stock, even if it slightly delays the project finish date. Therefore, the model in this chapter maximizes the expected *profit* instead of minimizing the makespan.

With stochastic project arrivals, choices made for a current project influence the yard capacity in the future, and thus, the scheduling options for the next arriving project. For example, it might be beneficial for the current project to use a pre-built assembly and reconstruct this after the project is finished. However, this will require yard capacity that can interfere with the next project. Considering this yard capacity, a safer option would be to outsource building the assembly. However, this might incur additional costs.

To model these additional choices, this chapter introduces a model with profit maximization and stochastic project arrival. Furthermore, although the instances that we use to test our models and algorithms are not taken from real life projects, the structure of these instances shows how to model decisions such as when to outsource, when to construct in-house, and which modules to keep in the inventory. This model uses a scenario-based approach, where a set of scenarios are generated. The decisions are then made such that the average outcome over all scenarios is optimized. For this, both an exact and a heuristic method are given and the computational results of both methods are compared.

4.1. 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 be competitive, improvements are made on both the technical aspect and the process aspect. Improving the shipbuilding process is done for multiple reasons, one of which 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 decreasing lead time is to build ships to stock. Although this is done in some specific cases, 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 and 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 a resource inventory level has been decided upon, 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 projects.

Finally, using modules modifies the structure of the project schedule. 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 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.

Thus, 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, it can be beneficial to make different decisions per project. This can be modeled by adapting the **Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources** (RCPSP-PS/CPR) from Chapter 3 to include resource inventory levels, stochastic project arrivals and profit maximization. Furthermore, decisions have to be made during the arrival of a project, which influences scheduling capabilities in future projects. This results in a *multi-stage stochastic optimization problem*. To find solutions for this problem, we present a modified version of the **Progressive Hedging** (PH) algorithm that uses a heuristic algorithm to solve the scenario subproblems. This algorithm consists of a basic version and two extensions that improve the performance.

Thus, the contributions of this chapter are as follows: first, we introduce and present a formulation for the **Resource Constrained Project Scheduling Problem with Modular Production** (RCPSP-MP). Second, we present a PH algorithm for this problem and compare its performance against optimal solutions. Finally, we compare different variations of the algorithm.

In Section 4.2, an overview is presented of related research for both scheduling with stochastic project arrivals and multi-stage optimization. Subsequently, in Section 4.3, the RCPSP-MP is described and formulated. However, without assumptions on probability distributions, this model cannot be optimized directly. After this, in Section 4.4, the solution method is given. Finally, this solution method is evaluated in Section 4.5 and the chapter is concluded in Section 4.6.

4.2. LITERATURE REVIEW

In this section, we review the literature related to the RCPSP-MP, which can be categorized as a Resource Constrained Multi-Project Scheduling Problem (RCMPSP) with uncertain arrivals. Unfortunately, relevant research in this area is sparse, since most research is focused on baseline scheduling (Pamay et al., 2014; Capa and Ulusoy, 2015). Here, an initial baseline schedule is created, for which the number of later modifications done has to be minimized. However, in the RCPSP-MP, the goal is to schedule without later modifications. Therefore, 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 and Wets, 1969), and scenario based methods, such as **dual decomposition** (DD) (Carøe and Schultz, 1999) and progressive hedging (Rockafellar and 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 at-

tractive for multi-stage decomposition methods. Finally, there are several PH methods that find solutions to the decomposed subproblems by a meta-heuristic (Løkketangen and Woodruff, 1996; Haugen et al., 2001; Hasannia Kolaee and Mirzapour Al-e-Hashem, 2022). 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 and 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 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 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. They use a dynamic programming algorithm that finds good solutions for the subproblems. Although dynamic programming has an optimality guarantee for the standard subproblem, PH alters the objective function, which removes this guarantee and makes dynamic programming a heuristic. 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.

In conclusion, it can be stated that the current research on the RCPSP with uncertain arrivals does not give useful research directions for the RCPSP-MP. Therefore, we considered different SP methods, of which the PH algorithm seemed most promising. 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.

4.3. 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, the scheduling decisions have to be made. 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. 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 is scheduled that replenishes the same amount. 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.

An example of a scenario tree with 3 types of candidate projects can be seen in Figure 4.3.1. 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 $\tau = 120$.

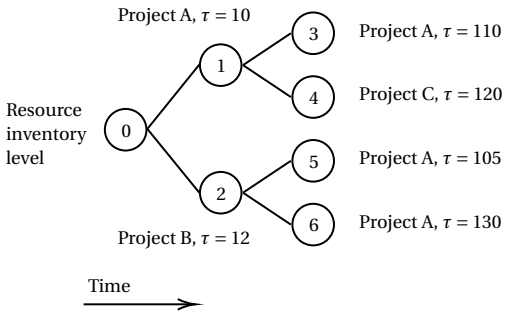


Figure 4.3.1: Example of a scenario tree for 3 stages (2 arriving projects).

We now give a formal description of the scenario tree and the projects contained in it. Let Ω be the set of all scenarios, 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 $\mathscr{P}_\psi$ be the project arriving at non-root node $\psi > 0$ and τ_ψ the arrival time of project $\mathscr{P}_\psi$.

The project $\mathscr{P}_\psi$ consists of a set of possible activities N_ψ. 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 executed. 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 executed, exactly one successor in S_g has to be executed. The set of executed activities has to be scheduled in the set of discrete time periods $T = \{1, \cdots, |T|\}$. Each activity $i \in N_\psi$ has a duration of d_i time periods. 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 $\mathscr{P}_\psi$ arrives. Therefore, a scenario $\omega \in \Omega$ then has a set of sequentially arriving projects $\{\mathscr{P}_\psi | \psi \in \Psi_\omega \setminus \{0\}\}$. 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 $\mathscr{P}^m_\omega$ 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 $\mathscr{P}^m_\omega$ 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 $\mathscr{P}_\psi$ 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 $\mathscr{P}^m_\omega$ and let P be the total set of all precedence relationships. With this, we can give the **Mixed Integer Linear Program** (MILP) of the RCPSP-MP.

The RCPSP-MP has two types of decision variables. 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, equal to 1 if activity $i \in N$ starts at time $t \in T$ and zero otherwise. The objective is to schedule the project such that the profit is maximized. Each nonrenewable resource $r \in R^n$ has a fixed cost of c_r per unit. For each activity $i \in N$ scheduled at time $t \in T$, the profit is p_{it}, which is a non-increasing function for t. 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. With this, we get Objective function (4.1a).

Furthermore, Constraints (4.1b) specify that for each scenario, the starting activity has to be executed. Constraints (4.1c) define that activities can only be scheduled after the project has arrived. Subsequently, Constraints (4.1d) impose that each activity is scheduled at most once. The selection groups are handled by Constraints (4.1e) and (4.1f). For each selection group $g \in G$, the former define that if the activator activity a_g is executed, at least one successor activity is selected. The latter define that if the activator activity a_g is executed, at most one successor is executed. Furthermore, Constraints (4.1g) 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 (4.1h) to (4.1j). The first constraint set imposes the constraints of renewable resources, while the latter do this for the nonrenewable resources. Constraints (4.1i) 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 (4.1j) define that the total resources consumption cannot be negative. Subsequently, Constraints (4.1k) 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 decision. Finally, Constraints (4.1l) and (4.1m) define the variable domains. All notation used throughout this chapter is presented in Appendix 4.A.

$$\max \sum_{\omega\in\Omega} \sum_{i\in N_\omega} \sum_{t\in T} \frac{p_{it} X_{it}}{|\Omega|} - \sum_{r\in R^n} c_r Y_r \tag{4.1a}$$

$$\sum_{t\in T} X_{s_\omega t} = 1, \quad \forall \omega \in \Omega, \tag{4.1b}$$

$$\sum_{t\in T, t<\tau_\psi} X_{it} = 0, \quad \forall \psi \in \Psi, i \in N_\psi \tag{4.1c}$$

$$\sum_{t\in T} X_{it} \leq 1, \quad \forall i \in N, \tag{4.1d}$$

$$\sum_{t\in T} X_{a_g t} \leq \sum_{i\in S_g} \sum_{t\in T} X_{it}, \quad \forall g \in G, \tag{4.1e}$$

$$\sum_{j\in S_g} \sum_{t\in T} X_{jt} \leq |S_g| - \big(|S_g| - 1\big) \sum_{t\in T} X_{a_g t}, \quad \forall g \in G, \tag{4.1f}$$

$$\sum_{t\in T} (t + d_i) X_{it} \leq \sum_{t\in T} t X_{jt} + M\big(1 - \sum_{t\in T} X_{jt}\big), \quad \forall (i,j) \in P, \tag{4.1g}$$

$$\sum_{i\in N_\omega} \sum_{t'=t-d_i+1}^{t} k_{ri} X_{it'} \leq \lambda_r, \quad \forall \omega \in \Omega, r \in R^r, t \in T, \tag{4.1h}$$

$$\sum_{i\in N_\omega} \left(\sum_{t'=1}^{t} k_{ri}^- X_{it'} - \sum_{t'=1}^{t-d_i} k_{ri}^+ X_{it'} \right) \leq Y_r, \quad \forall \omega \in \Omega, r \in R^n, t \in T, \tag{4.1i}$$

$$\sum_{i\in N_\omega} \left(\sum_{t'=1}^{t} k_{ri}^- X_{it'} - \sum_{t'=1}^{t-d_i} k_{ri}^+ X_{it'} \right) \geq 0, \quad \forall \omega \in \Omega, r \in R^n, t \in T, \tag{4.1j}$$

$$X_{it} = X_{jt}, \quad \forall \omega \in \Omega, i \in N_\omega, j \in N_{\omega i}, t \in T \tag{4.1k}$$

$$X_{it} \in \{0,1\}, \quad \forall i \in N, t \in T, \tag{4.1l}$$

$$Y_r \geq 0, \quad \forall r \in R^n. \tag{4.1m}$$

4.4. 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, a **Hybrid Differential Evolution** (HDE) algorithm creates the full schedules. 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.

4.4.1. 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. This means that we can consider two types of optimization processes: the **complete optimization process** that finds an implementable solution for all scenarios in the scenario trees, and the **subproblem optimization processes**, where separate scenarios are optimized.

These two optimization processes both have a unique solution representation. The subproblem optimization algorithms use the **subproblem solution vectors**. Here, each 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. For the details to convert these vectors to a schedule, we refer to Chapter 3.

Additionally, the complete optimization process uses **compressed solution vectors**. This vector contains only the nonrenewable resource availabilities and information on the selection of executed 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 activity that is a successor of successor group $g \in G$, with 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 $\mathbf{x}_\omega$ 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 executed or not.

The penalization of each compressed solution vector $\mathbf{x}_\omega$ for scenario $\omega \in \Omega$ is done by comparing $\mathbf{x}_\omega$ to the average solution vector $\overline{\mathbf{x}}_\omega$ over all scenarios. To calculate $\overline{\mathbf{x}}_\omega$, we note that each scenario $\omega \in \Omega$ consists of multiple scenario tree nodes Ψ_ω. Since $\mathbf{x}_\omega$ consists of the resource inventory vector (set in the root scenario tree node) and activity selection entries (related to projects in non-root nodes), each entry in $\mathbf{x}_\omega$ can be mapped surjectively to a scenario tree node. With this, we can create $\overline{\mathbf{x}}_\omega$: for each entry in $\mathbf{x}_\omega$ that maps to scenario tree node $\psi \in \Psi_\omega$, we average the corresponding entries in $\mathbf{x}_{\omega'}$ for all scenarios $\omega' \in \Omega_\psi$ that also contain scenario tree node ψ.

To formalize this, we introduce the **solution indicator vector**

$$U\left(\omega, \omega', \psi\right) = \left[U\left(\omega, \omega', \psi\right)_1, \cdots, U\left(\omega, \omega', \psi\right)_{|\mathbf{x}_\omega|}\right] \tag{4.2}$$

for every combination of $\omega \in \Omega$, $\omega' \in \Omega$ and $\psi \in \Psi_\omega$. If node $\psi \notin \Psi_{\omega'}$, we get $U\left(\omega, \omega', \psi\right) = \mathbf{0}$. Otherwise, if $\psi \in \Psi_{\omega'}$, we let the j^{th} entry of $U\left(\omega, \omega', \psi\right)$ be 1 if the j^{th} variable in $\mathbf{x}_\omega$

corresponds to a selectable activity in scenario tree node ψ, and zero otherwise. With this, we can express $\bar{\mathbf{x}}_\omega$ as

$$\bar{\mathbf{x}}_\omega = \sum_{\psi \in \Psi_\omega} \sum_{\omega' \in \Omega_\psi} \frac{1}{|\Omega_\psi|} U\left(\omega, \omega', \psi\right)^T \cdot \mathbf{x}_{\omega'}. \tag{4.3}$$

4.4.2. HYBRID DIFFERENTIAL EVOLUTION

The individual scenario subproblems are solved by the **Hybrid Differential Evolution** (HDE) algorithm, introduced in Chapter 3. 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. In each iteration, a **trial** solution vector is created for each solution $\mathbf{x}$ in the population. This is done by linearly combining 3 randomly selected solutions, based on the scaling parameter SC and replacing entries from this new solution in the original solution $\mathbf{x}$, based on replacement parameter RP. This new solution replaces the original solution $\mathbf{x}$ 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. If this does not improve any solutions, the process is terminated. For the complete details of this algorithm, we refer to Chapter 3.

4.4.3. PROGRESSIVE HEDGING

We now present the structure of the PH algorithm. This algorithm uses the, in PH literature called, **Lagrangian multiplier** $\mathbf{w}_\omega$ for each scenario $\omega \in \Omega$ and **penalty ratio** $\mathbf{r}_\omega$, to alter the objective functions of the subproblems to enforce implementability. Initially, $\mathbf{r}_\omega$ is set to its initialization value $\mathbf{r}_\omega^{init} > 0$ and each $\mathbf{w}_\omega$ is set to a zero vector with the same length as $\mathbf{x}_\omega$. Furthermore, the PH algorithm initializes a compressed solution $\mathbf{x}_\omega$ 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** $obj(\mathbf{x}_\omega)$.

After this initialization, the iterative process starts. In each iteration, the average solution $\bar{\mathbf{x}}_\omega$ is calculated for each scenario $\omega \in \Omega$ by Constraints (4.3). An implementable set of solutions correlates to $\mathbf{x}_\omega = \bar{\mathbf{x}}_\omega$ for every scenario $\omega \in \Omega$: each decision value is equal across all scenarios. Therefore, $\bar{\mathbf{x}}_\omega$ can be used to steer the solution set to be implementable. This is done by first modifying the Lagrangian vector $\mathbf{w}_\omega$, by adding the penalty ratio times the difference between current solution $\mathbf{x}_\omega$ and average solution $\bar{\mathbf{x}}_\omega$: $\mathbf{w}_\omega \leftarrow \mathbf{w}_\omega + \mathbf{r}_\omega(\mathbf{x}_\omega - \bar{\mathbf{x}}_\omega)$. If an entry of $\mathbf{x}_\omega$ is larger than the average solution vector $\bar{\mathbf{x}}_\omega$, the corresponding entry in $\mathbf{w}_\omega$ is increased. Conversely, entries lower than the average result in a decreased entry in the Lagrangian vector $\mathbf{w}_\omega$. This is then used, together with $\mathbf{r}_\omega$ and $\bar{\mathbf{x}}_\omega$, to alter the cost of the objective function. In the standard PH implementation, the altered objective function is given by:

$$alt\_obj\left(\mathbf{x}_\omega, \bar{\mathbf{x}}_\omega, \mathbf{w}_\omega, \mathbf{r}_\omega\right) = obj(\mathbf{x}_\omega) - \mathbf{w}_\omega^T \mathbf{x}_\omega - \mathbf{r}_\omega \|\mathbf{x}_\omega - \bar{\mathbf{x}}_\omega\|^2, \tag{4.4}$$

where the last two terms are penalization terms. The first one is based on the Lagrangian vector $\mathbf{w}_\omega$. Recall that positive entries of $\mathbf{w}_\omega$ are the result of larger than average

values in $\mathbf{x}_\omega$, since we start with $\mathbf{w}_\omega$ equal to zero and increase $\mathbf{w}_\omega$ when $\mathbf{x}_\omega$ is larger than average. Therefore, this term penalizes high values if the solution from the previous iterations (or initialization) was above the average solution vector. Similarly, negative values of $\mathbf{w}_\omega$ correspond to entries in $\mathbf{x}_\omega$ that are smaller than average. Therefore, this term guides the solution towards the average value. The second penalty term penalizes the squared distance between $\mathbf{x}_\omega$ and $\bar{\mathbf{x}}_\omega$, times the penalty ratio $\mathbf{r}_\omega$. 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, $\mathbf{x}_\omega$ represents only the resource inventory level and activity selection decisions. Therefore, after terminating the iterative process, this still needs to be converted to a schedule. First, if $\mathbf{x}$ is not implementable, the repair function repair($\mathbf{x}$) 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 Chapter 3.

After the repair process, a schedule is created for each project $\mathscr{P}_\psi$ at scenario tree node $\psi > 0$, $\psi \in \Psi$. This is done by ordering Ψ breadth-first from the root node. Then, all resource inventory levels and activity selection decisions are fixed according to $\mathbf{x}$. Subsequently, the HDE algorithm is ran for each scenario tree node while taking into account the schedules of all ancestor nodes. We denote this procedure by get_final_solution($\psi, \bar{\mathbf{x}}_\omega$), where ω is any scenario in Ω_ψ (note that entries in $\bar{\mathbf{x}}_\omega$, corresponding to node $\psi \in \Psi$, are equal across all scenarios $\omega \in \Omega_\psi$).

This gives the PH algorithm, as given in Algorithm 10, that is guaranteed to create feasible schedules for the RCPSP-MP. In the remainder of this section, we present improvement techniques for this algorithm.

4.4.4. EXTENSIONS

In order to improve the performance of the PH algorithm, two improvement techniques are presented here. They are either based on adaptations from improvement techniques in the literature, or newly created.

The first improvement technique introduced is **Overshoot limitation**, which is one of the contributions of this chapter. As shown in Constraints (4.4), the Lagrangian penalty is $\mathbf{w}_\omega^T \mathbf{x}_\omega$. Consider entry $x_i \in \mathbf{x}_\omega$ that has been higher than average for some iterations, such that the corresponding entry in $\mathbf{w}_\omega$ has become positive. This gives an incentive for x_i to become lower. 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 $\mathbf{x}_\omega$ by $\mathbf{x}'_\omega$ in the second term of Constraints (4.4), with for each entry $x'_i \in \mathbf{x}'_\omega$:

$$x'_i = \begin{cases} \bar{x}_i & \text{if } \big(w_i > 0 \text{ and } x_i < \bar{x}_i\big) \text{ or } \big(w_i < 0 \text{ and } x_i > \bar{x}_i\big), \\ x_i & \text{otherwise,} \end{cases} \tag{4.5}$$

Algorithm 10 Progressive Hedging algorithm

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

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

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

for $\psi \in \Psi$ **do** ▷ Create complete solution
  $\omega \leftarrow$ Select any from Ω_ψ
  $X_\omega \leftarrow$get_final_solution($\psi, \bar{\mathbf{x}}_\omega$)
end for

using corresponding entries $x_i \in \mathbf{x}_\omega$, $\bar{x}_i \in \bar{\mathbf{x}}_\omega$ and $w_i \in \mathbf{w}_\omega$. Thus, entries of $\mathbf{x}_\omega$ 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 $\mathbf{x}_\omega'$ is capped at the average value to prevent overshoot. This gives us the altered cost when overshoot limitation is used:

$$alt\_obj(\mathbf{x}_\omega, \bar{\mathbf{x}}_\omega, \mathbf{w}_\omega, \mathbf{r}_\omega) = obj(\mathbf{x}_\omega) - \mathbf{w}_\omega^T \mathbf{x}_\omega' - \mathbf{r}_\omega \|\mathbf{x}_\omega - \bar{\mathbf{x}}_\omega\|^2. \tag{4.6}$$

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 for binary values, variable bounding is the same as 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.

4.5. 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.

4.5.1. 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. Second, the instance sets are described. A summary of this is given here, while the detailed description can be found in Appendix 4.B and the instances are given in van der Beek (2022a). 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 $\mathscr{P}^m_\omega$, 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 Figure 4.5.1, an example of a single project is shown. Here, the base network consists of five activities. From this, we replace activity one by a module option. It can be seen that this creates 4 alternatives: DCA, MA, RA and

OA. The DCA and MA replace activity one, while the RA and OA are placed after the root activity in the precedence graph and after the MA in the selection graph.

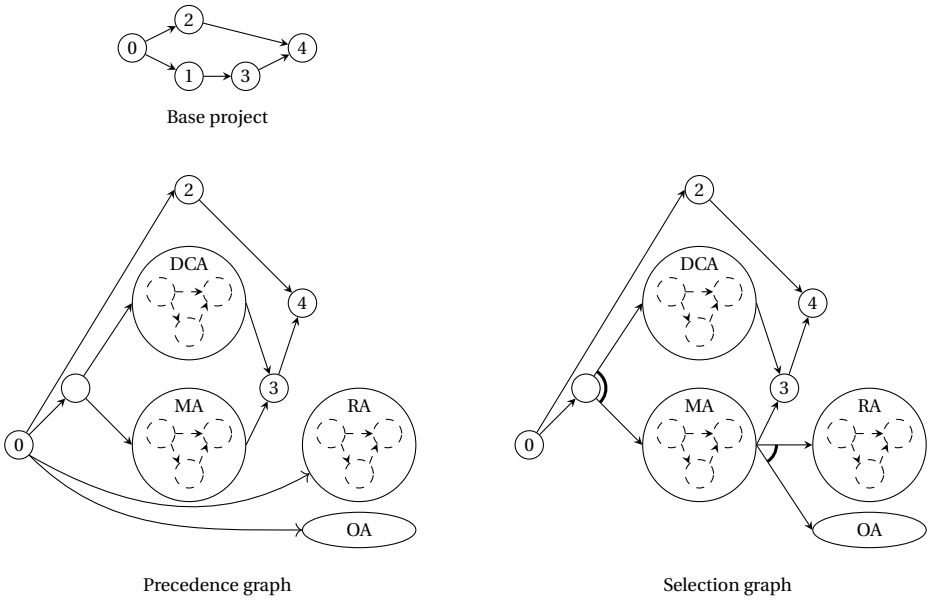


Figure 4.5.1: Insertion of a single module option in a base project.

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 with the MILP given in Constraint set (4.1). The goal of this instance set is to evaluate the performance of the MILP formulation. The latter instance set, *Heuristic-only*, contains larger instance that such that Constraint set (4.1) cannot be used to find feasible solutions. The goal of this instance set is to compare the various variants of the PH algorithm.

4.5.2. 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 MILP model in Constraint set (4.1) is used to find feasible (but not necessarily optimal) solutions. The goal of these instances is to evaluate the MILP 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 a single core of a 3.0 GHz Intel XEON CPU with 4 GB RAM.

The 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 other parameters are decided upon.

This resulted in the parameters $\gamma = 100$, $SC = 0.1$, $RP = 0.1$, $TP = 100$, $\mathbf{r}_{\omega}^{init} = 1.6$, $PM = 1.001$, $LB = 8$ and $LS = 7$.

During these computational tests, the following methods are used:

1. **MILP Method** (MM): Running the GUROBI optimizer on the MILP model in Constraint set (4.1) 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 4.4.3, without any extensions.
3. **Bounded Method** (BM): The BPH with the *variable bounding* extension from Section 4.4.4.
4. **Overshoot-limited Method** (OM): The BPH with the *overshoot limitation* extension from Section 4.4.4
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 MM, however, does not have this guarantee. Based on time limitations for the computational study, the maximal computing duration for the MM is set to 18 hours and the maximum number of iterations of 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:

$$BOG(obj) = \frac{obj - obj^*}{obj^*} \cdot 100\%. \tag{4.7}$$

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

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 MILP method. Table 4.5.1 shows a summary of the tests results. Here, it can be seen that for nearly half of the instances, a feasible solution 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 duration being close to this time limit. Furthermore, the optimality gaps (OG) are evaluated . This is defined as

$$OG = \frac{lb - obj^*}{obj^*} \cdot 100\%, \tag{4.8}$$

where lb is the highest lower bound found by the MILP-solver and obj^* is the best found feasible solution, either from the MILP-solver or from the PH heuristics. Using both the

feasible solution from the MILP-solver and the PH methods is done to obtain a meaningful OG, even when no feasible solution is found by the MILP-solver. In Figure 4.5.2a, 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.

Table 4.5.1: Summary of tests for all 209 instances for instance set *Exact*.

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

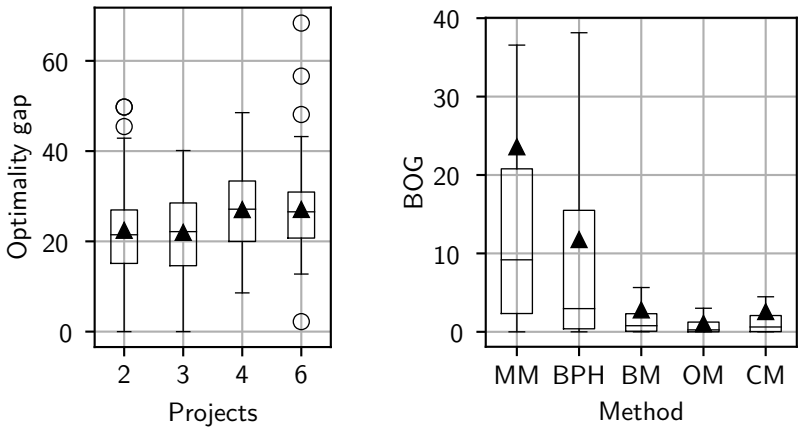
Furthermore, a comparison of all instances, for which feasible solutions are found for all methods, is given in Table 4.5.2. Here, it can be seen that both the mean BOG and the computing time is significantly better for all PH methods. In Figure 4.5.2b, the BOGs are shown for each method. It can be seen that the BPH has a significant improvement compared to the MM. Furthermore, all extensions result in an even larger reduction of BOG. However, when evaluating the number of instances for which a method reaches the best found solution (including ties), the BPH performs slightly worse than the MM. Here, it can be seen that the improvements are required to outperform the MM. Evaluating only the performance and not the computing time, it can be seen that the OM performs best.

Table 4.5.2: Comparison for all methods for all 97 instances in the instance set *Exact*, where a feasible solution is found by the MM.

Method	MM	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

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 4.5.3. Furthermore, Figure 4.5.3 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.



(a) OG per number of projects in scenario. (b) BOG per method (without outliers).

Figure 4.5.2: Results for instance set *Exact*.

Additionally, Table 4.5.3 shows the number of solutions that were implementable without the repair step in Algorithm 10. 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.

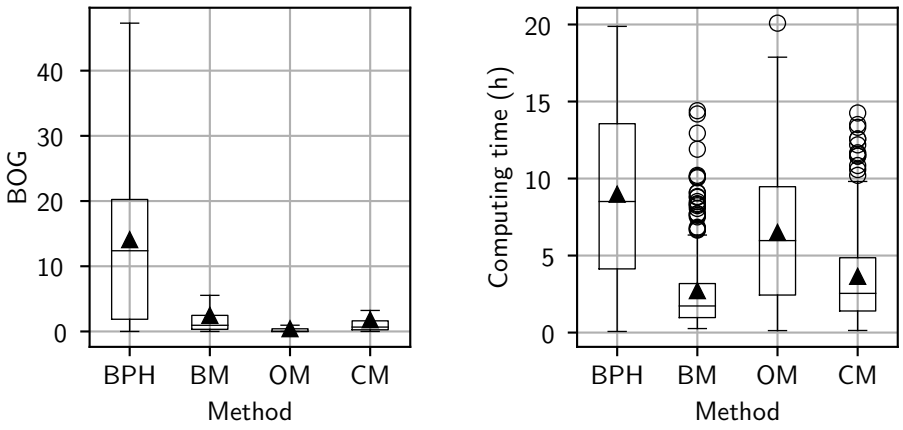
Furthermore, we evaluate the use of lag in variable bounding for the BM and the CM. To indicate the variations including lag of these methods, the suffic 'L' is used. Figure 4.5.4 shows the BOGs and computing times for all methods. It can be seen that introducing lag reduces the mean BOG and increases 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, Figure 4.5.5 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.

Table 4.5.3: 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



(a) BOG per method (without outliers).

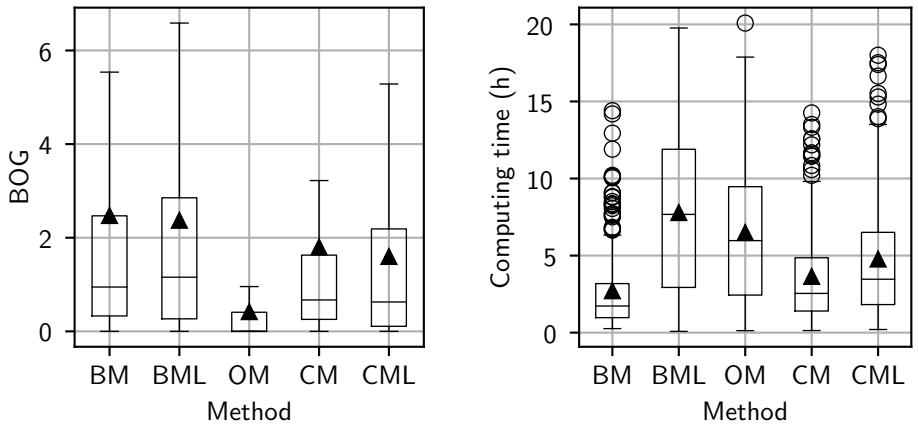
(b) Computing times.

Figure 4.5.3: Results for all PH methods for instance set *Heuristic-only*.

4.6. CONCLUSION

In this chapter, 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 scheduling, and activity selection decisions, while optimizing the mean of the objective functions over all scenarios.

An MILP 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 MILP model



(a) BOG per method (without outliers).

(b) Computing times.

Figure 4.5.4: Results for PH methods, while including lag on variable bounding, for instance set *Heuristic-only*.

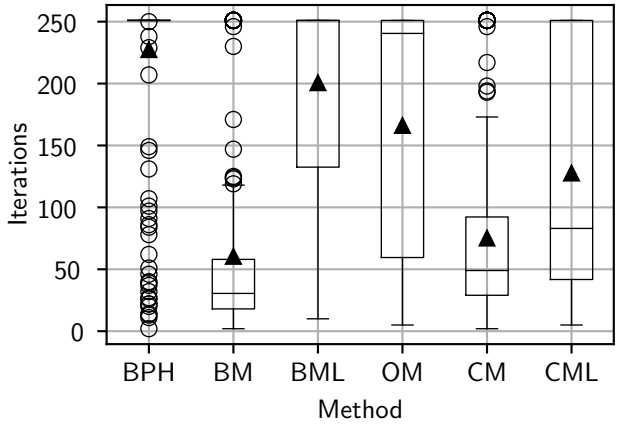


Figure 4.5.5: Number of iterations per method for instance set *Heuristic-only*.

is rarely solved to optimality, even for small instances. Thus, a PH algorithm is introduced that converges on 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 MILP-based solution method. However, the basic algorithm relies heavily on the repair step at the end of the algorithm and does not usually converge 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 computation 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.

APPENDIX

4.A. NOTATION

Sets

G	Selection groups.
G_ω	Selection groups in scenario $\omega \in \Omega$.
$G_{\omega g}$	Selection groups identical to selection groups $g \in G_\omega$ for scenario $\omega \in \Omega$ in all other scenarios.
N	Activities.
N^s	Selectable activities.
N_ω	Activities in multi-project $\mathscr{P}^m_\omega$ 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.
N_ψ	Activities of project $\mathscr{P}_\psi$ in scenario tree node $\psi \in \Psi$.
P	Predecessor-successor pairs.
P_j	Predecessors of activity $j \in N$ in the precedence graph.
R	Resources.
R^r	Renewable resources.
R^n	Nonrenewable resources.
S_g	Successor activities of selection group $g \in G$.
T	Time periods.
Ψ	Scenario tree nodes.
Ψ_ω	Scenario tree nodes in scenario $\omega \in \Omega$.
Ω	Scenarios.
Ω_ψ	Scenarios that contain scenario tree node $\psi \in \Psi$.

Variables

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 $r \in R^n$.

Other

a_g	Activating activity of selection group $g \in G$.
$alt\_obj(\cdots)$	Altered objective function.
c_r	Unit cost of initial availability of nonrenewable resource $r \in R^n$.
d_i	Duration of activity $i \in N$.
k_{ri}	Net resource production of resource $r \in R$ for activity $i \in N$.
k^+_{ri}	Production of resource $r \in R$ for activity $i \in N$.
k^-_{ri}	Consumption of resource $r \in R$ for activity $i \in N$.
$obj(\mathbf{x}_\omega)$	Unaltered objective function value of solution $\mathbf{x}_\omega$.
p_{it}	Profit obtained by scheduling activity $i \in N$ at time $t \in T$.
$profit(X)$	Profit obtained by executing schedule X.
$\mathbf{r}_\omega$	Penalty ratio.
s_ω	Starting activity of multi project $\mathscr{P}^m_\omega$ of scenario $\omega \in \Omega$.
SC	Differential evolution scaling parameter.

LR	Lag parameter for resource inventory level variables.
LS	Lag parameter for selection variables.
M	Sufficiently large number.
OC	Outsourcing cost parameter.
PM	Penaly multiplicator.
PT	Preparation time parameter.
RC	Resource constrainedness.
RF	Resource factor.
RU	Resource usage parameter.
SP	Serial/parallel indicator.
ST	Starting time parameter.
TC	Sum of all inventory costs of nonrenewable resources for a module option.
TD	Target duration.
TD^{MA}	Module alternative duration parameter.
TD^{OA}	Outsource alternative duration parameter.
TD^{RA}	Reconstruct alternative duration parameter.
$U(\omega,\omega',\psi)$	Solution indicator vector for scenarios $\omega,\omega' \in \Omega$ and scenario tree node $\psi \in \Psi$.
VAR^{OA}	Outsource alternative variation parameter.
VAR^{OC}	Outsourcing cost variation parameter.
VAR^{RC}	Resource constrainedness variation parameter.
VAR^{ST}	Starting time variation parameter.
$\mathscr{P}^m_\omega$	Multi-project representing all projects of scenario $\omega \in \Omega$.
$\mathscr{P}_\psi$	Project arriving at scenario $\psi \in \Psi$.
$\mathbf{w}_\omega$	Lagrangian multiplier for scenario $\omega \in \Omega$.
$\mathbf{x}_\omega$	Compressed solution vector of scenario $\omega \in \Omega$.
$\bar{\mathbf{x}}_\omega$	Average compressed solution vector of scenario $\omega \in \Omega$.
γ	HDE population size.
λ_r	Capacity of renewable resource $r \in R^r$.

4.B. INSTANCES

This section describes the instances used in this chapter. First, in Appendix 4.B.1, the instance generation method is presented. Second, Appendix 4.B.2 describes the instance sets created by this method.

4.B.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 Figure 4.3.1 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 ac-

tivities $|N|$, 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 4.B.1.

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 4.B.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 $\{1, \cdots, RU\}$, 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 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 $\mathscr{P}_\omega^m$ 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 $(ST \pm$

Table 4.B.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]	
$\lvert R^r \rvert$	4	
$\lvert N \rvert$	7-9	12-17

$VAR^{ST}) \cdot critical\_path$, 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 $[0, \cdots, PT]$, where PT is the Preparation Time parameter.

4.B.2. INSTANCE SETS

The method described in Appendix 4.B.1 is used to create two instance sets: *Exact* and *Heuristic-only*. The former is used to evaluate the performance of the MILP formulation given in Constraint set (4.1) and the latter is used to compare the various variants of the PH algorithm. In Table 4.B.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.

Table 4.B.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

5

STOCHASTIC MAKESPAN MINIMIZATION FOR THE RCPSP

In Chapter 4, we considered resource allocation and scheduling with stochastic project arrivals. However, the focus of the algorithm was on activity selection and resource allocation, not on scheduling. The reason for this was to reduce the number of choices to be made, and thus to reduce computational effort. Even with this limited focus, computing times were still in the order of hours or days, thus indicating that scheduling decisions might have been too much to include as well. However, it is not to say that scheduling decisions are not important while considering future arrival. As multiple sequential projects require the same resources, choosing another schedule for the same set of activities might result in less resource conflicts upon the arrival of the next project. This is investigated in this chapter.

Due to the computational complexity, the model used is the standard RCPSP instead of the DRCMPSP/SS. The context of the methods presented in this chapter is therefore later in the scheduling process as the previous models and methods. The methods of this chapter are used when all resource allocation and module decisions are made, resulting in a project with a fixed set of activities that *all* have to be executed. Thus, the projects are represented by the standard RCPSP.

As is shown in the remainder of this chapter, even the standard RCPSP, but with stochastic project arrivals, forms a computationally demanding optimization problem. Even standard heuristics from literature might take very long to finish. In practice, we might need a schedule quickly, since waiting for a schedule to be generated will delay other steps in the planning process. On the other hand, there is a lot of time before project arrival. Therefore, it would be beneficial if calculations could be shifted in time, such that most work is done at non-critical moments. This is done in this chapter, by using machine learning to learn from optimizing simulations. With this, an optimization algorithm is created that uses a learned objective function to quantify the quality of a schedule, based on an estimation of the performance of expected future arrivals.

This is then compared to the full optimization method, and to the method of scheduling without looking ahead.

5.1. 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 sequentially arriving similar projects. These projects have to be scheduled, while satisfying resource and time constraints. A common method of scheduling is using 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 modifications made later. Since all projects use a 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 chapter. This problem consists of a set of stages in which problems arrive sequentially. At each moment of project arrival, we assume to have an estimate of the arrival time of the next project. Each project is created by a *generator*, which draws projects from a certain distribution. As soon as a new project arrives, it has to be scheduled completely, without the possibility to reschedule. Although this is a simplification of reality, it serves to find solutions that deal with uncertain future projects. Furthermore, the goal of the DRCMPSP/SS goal is to minimize the weighted average makespans of all projects.

As is discussed in Section 5.2, there are various studies on the dynamic arrival of new projects for the RCPSP. However, to the best of our knowledge, these papers do either not consider completely scheduling a project at each decision point, or have a purely reactive approach, meaning that a decision maker only reacts and does not look ahead. One possible reason for this might be that proactive scheduling of complete projects results in a computationally very expensive problem, which can take very long to solve for more traditional optimization methods.

The contribution of this chapter is threefold. First, we introduce the DRCMPSP/SS. Secondly, we introduce a new solution 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 former method. These methods are compared against a greedy alternative:

scheduling each project as well as possible without looking ahead.

In Section 5.2, we first give an overview of research related to the DRCMPSP/SS. Subsequently, we give a description of the problem in Section 5.3. Then, in Section 5.4, the solution methods are given. Finally, we present the results of the computational study in Section 5.5 and conclude the chapter in Section 5.6.

5.2. 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 this schedule is repaired or modified as soon as uncertainties arise. The second type is dynamic scheduling, where no baseline schedule is present, but where a scheduling policy is decided upon that handles random events.

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 is not allowed as is done in the work discussed above. Therefore, research on dynamic scheduling policies 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 minization. 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 a 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. Secondly, they give 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 hard 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. As these characteristics are closer to the DRCMPSP/SS than MDP based approaches, we further explore this method. For more details on simulation optimization, we refer to various surveys (Homem-de Mello and Bayraksan, 2014; Amaran et al., 2016; Juan et al., 2015).

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 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.

5.3. 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. In the first iteration, the project generator outputs a project $\mathscr{P}^1$ and an earliest arrival time for the next project: τ^2_{min}. After this, project $\mathscr{P}^1$ has to be scheduled. In each subsequent iteration $k > 1$, it outputs a project $\mathscr{P}^k$, 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 incremented until τ^k, the arrival time of project $\mathscr{P}^k$, and project $\mathscr{P}^k$ is scheduled. This is repeated $|K|$ times, where K is the set of stages: $K = \{1, \cdots, |K|\}$. The process is illustrated in Figure 5.3.1. At each stage

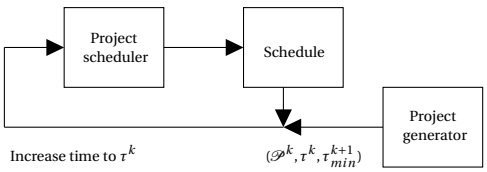


Figure 5.3.1: Scheduling environment with the project scheduler and the project generator.

$k \in K$, we let Ξ^k be the random variable representing the generated project. Thus, each realization ξ^k of Ξ^k is a 3-tuple $(\mathscr{P}^k, \tau^k, \tau^{k+1}_{min})$. 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 a set of feasible timesteps T in which the projects can be scheduled.

At decision step k, we obtain a single project $\mathscr{P}^k = (N^k, P^k, \mathbf{d}^k, \mathbf{b}^k)$. For brevity, we omit the superscript k as long as it is clear from context or irrelevant. The project $\mathscr{P}$ consists of a set of activities N, of which the first one is the starting activity and the last one is the ending activity. All activities N have to be scheduled, while subjected to the precedence relationships P. A precedence relationship $(i, j) \in P$ imposes that activity $i \in N$ has to finish before activity $j \in N$ can start. Furthermore, the vector $\mathbf{d}$, consisting of entries d_i for $i \in N$, defines the duration of each activity i. Finally, the $|R| \times |N|$-matrix $\mathbf{b}$, with entries b_{ri} for $r \in R$ and $i \in N$, defines the resource requirement for each resource r and activity i.

For notational purposes, we introduce the notation $x[i] = \{x^1, \cdots, x^i\}$ for any indexed variable x. Furthermore, each project $\mathscr{P}^k$ can only be scheduled after the arrival time τ^k. Therefore, we define the set $T^k = \{\tau^k, \cdots, |T|\}$ as the set of timesteps considered for project $\mathscr{P}^k$. With this, we denote a schedule for project $\mathscr{P}^k$ by binary matrix X^k, with $X^k_{it} = 1$ if activity $i \in N^k$ starts at time $t \in T^k$ and 0 otherwise. Since the projects use the same shared resources, the feasible region of all schedules for project $\mathscr{P}^k$ depends on the schedules of all previous projects. Therefore, we notate the feasible region of all schedules for project $\mathscr{P}^k$ as $\mathscr{X}^k(X[k-1])$ and define this as follows:

$$\begin{aligned}\mathscr{X}^k(X[k-1]) = \{X^k \in [0,1]^{|N^k|\times|T^k|},\\ \sum_{t\in T^k} tX^k_{it} + d_i \leq \sum_{t\in T^k} tX^k_{jt}, \quad \forall (i,j)\in P^k\\ \sum_{t\in T^k} X^k_{it} = 1, \quad \forall i \in N^k\\ \sum_{k'=1}^{k}\sum_{i\in N^{k'}} b^{k'}_{ri} \sum_{t'=t-d^{k'}_i, t'\in T^{k'}}^{t} X^{k'}_{it'} \leq \lambda_r, \quad \forall r\in R, t\in T^k\}.\end{aligned} \tag{5.1}$$

Since this is a basic formulation of the RCPSP, we refer to Pritsker et al. (1969) for an explanation and only focus on the final constraint, where $X^{k'}_{it}$ with $k' < k$ is input given by $X[k-1]$. This constraint imposes that the resource capacities are not exceeded. The modification made here, with respect to the standard RCPSP, is that we consider resources from all projects up to $\mathscr{P}^k$.

5

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 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 $cp(\mathscr{P})$ 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 define this iteratively, by introducing the **cost-to-go** function ctg^k:

$$\begin{aligned}&ctg^k\left(X[k-1], \Xi^k\right) =\\ &\min_{X\in\mathscr{X}^k(X[k-1])} \frac{\sum_{t\in T^k}(t-\tau^k)X^k_{|N^k|t}}{cp(\mathscr{P}^k)} + \mathbb{E}\left[ctg^{k+1}(X[k], \Xi^{k+1})\right], \quad \forall k\in K.\end{aligned} \tag{5.2}$$

The first part of this function is the objective function per project and the second part is the expectation of all future projects. Thus, the value of the objective function is the average value of all weighted makespans. 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 Equation (5.2) 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:

$$ctg = ctg^1\left(\emptyset, \Xi^1\right) = \min_{X\in\mathscr{X}^1} \frac{\sum_{t\in T^1}(t-\tau^1)X^1_{|N^1|t}}{cp(\mathscr{P}^1)} + \mathbb{E}\left[ctg^2(X[1], \Xi^2)\right]. \tag{5.3}$$

Furthermore, all notation used throughout this chapter is presented in Appendix 5.A.

5.4. 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 a project generator that generates projects and corresponding arrival times. This section presents three solution methods that schedule these arriving projects.

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.

5.4.1. GREEDY METHOD

One of the most common methods of scheduling 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 (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 chapter 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 chapter 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. Furthermore, $\mathbf{x}^*$ keeps track of the best solution so far. During the iterative improvement step, three solution vectors $\mathbf{a}^1, \mathbf{a}^2$ and $\mathbf{a}^3$ are taken and used to create a trial solution $\rho = [\rho_1, \cdots, \rho_{\text{sol\_len}}]$. This is done by using algorithm parameters w and c. Subsequently, the trial solution vectors are converted to schedules by a serial Schedule Generation Scheme (SGS), as described in Artigues et al. (2008). If this trial schedule is better than or equal to the currently considered solution, it replaces this one. Similarly, if it is better than the best solution found so far, it replaces this one too. This gives the full solution algorithm, as shown in Algorithm 11.

The initialization is done by creating a matrix with uniformly distributed random values between 0 and 1. Finally, the algorithm is terminated as soon as in the last 25 iterations no new improved solution has been found. This value is set manually, as it was found to give reasonable computing times.

Algorithm 11 Differential evolution

$\mathbb{X} \leftarrow$ Initialization matrix of size $\gamma \times sol\_len$
$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x}\in\mathbb{X}}(\text{objective } \mathbf{x})$
while not terminated **do**
  for $\mathbf{x} \in \mathbb{X}$ **do**
    $\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3 \leftarrow$ Pick randomly without replacement from $\mathbb{X} \setminus \{\mathbf{x}\}$
    $\mathbf{a} \leftarrow \mathbf{a}^1 + w(\mathbf{a}^2 - \mathbf{a}^3)$
    $f =$ random from $\{1, \cdots, N\}$
    $\rho_i \leftarrow \begin{cases} a_i & \text{with probability } c \text{ or if } i = f \\ x_i & \text{otherwise} \end{cases}$
    if objective $\rho \leq$ objective $\mathbf{x}$ **then**
      Replace $\mathbf{x} \in \mathbb{X}$ by ρ
      if objective $\rho <$ objective $\mathbf{x}^*$ **then**
        $\mathbf{x}^* \leftarrow \rho$
      end if
    end if
  end for
end while
return $\mathbf{x}^*$

5.4.2. FULL METHOD

This subsection presents the **Full Method** (FM), which is the most computationally expensive method presented in this chapter. The FM also uses a DE algorithm to find solutions. However, instead of computing only the objective of the current project, it also contains a simulated objective of future projects.

The problem presented in Section 5.3 is a multi-stage stochastic problem, where each stage contains an NP-hard problem and the expectation of multiple future NP-hard problems. However, computing the expectation over multiple stages would be too challenging computationally. Therefore, the FM uses a rolling horizon approach: At each decision step, we only consider the current project and one or more simulations for the next project.

In the remainder of this subsection, first the solution representation used in the FM is presented. Subsequently, the FM itself is given.

SOLUTION REPRESENTATION

In Solution set (5.1), an exact solution representation is given by using X_{it}^k for project $\mathscr{P}^k$, with $X_{it}^k = 1$ if activity i starts at time t and zero otherwise. This representation, however, is unpractical for use in meta-heuristics. Therefore, most heuristic approaches use either an activity list or a priority list in combination with a serial SGS (as in Section 5.4.1) or a parallel SGS (Pellerin et al., 2019). The main idea of these methods is to sequentially schedule each activity at the *earliest time available*. However, when future arrivals are considered, it is possible that the best schedule contains activities that are scheduled later than their earliest time available. This can be done to allow availability of resources for later arrivals. For this reason, we introduce a new schedule representa-

tion by defining each schedule **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.

FULL METHOD ALGORITHM

With the solution representation introduced above, we now present the FM. Along with this method, we explain how the data of the search process is stored. Although storing data is not necessary for the FM, it is required for the TM, as is explained later.

The FM is based on **Sample Average Approximation** (Kleywegt et al., 2002). A standard way of finding solutions for stochastic problems is by generating multiple scenarios and creating a deterministic equivalent. However, using a fixed set of scenarios risks finding solutions that perform well only on these specific scenarios (Homem-De-Mello, 2003). Therefore, we vary the scenarios during the execution of our data generation algorithm, as seen in Homem-De-Mello (2003). This allows us to consider a large set of scenarios, without having to optimize for all of these scenarios simultaneously.

Secondly, since one of the goals of the FM is to generate data for the TM, we are also interested in the quality of schedules found during the search process, instead of only being interested in the final schedule. Therefore, instead of generating a deterministic equivalent, the FM uses an iterated optimization approach where each objective estimation consists of optimizing multiple future candidate projects.

The FM optimizes a schedule and stores data that can later be used to train the TM. The goal is to create an algorithm that minimizes the **complete objective**: the value of Equation (5.3). To achieve this, we require some objective terminology. The **current objective** refers to the scaled makespan of a schedule for the current project; the project that we are scheduling in the FM. Looking one step into the future, one can create a set of scenarios for the next project. We call the set of scaled makespans for these scenarios the **scenario objectives**. Combining the current objective and the scenario objectives results in a measure for how good the current schedule is while taking into account future arrivals. This is called the **combined objective**.

The FM can be based on any search-based heuristic that explores many different schedules. The main modifications that have to be made, are the computation of the objective function and the storage of data. For this, we introduce the concept of resource profiles. A resource profile Y is a matrix, where each entry Y_{rt} represents the total usage of resource $r \in R$ at time $t \in T$. We define the resource profile function $RP(X[k], t)$, which gives the resource profile of all solutions $X[k]$, starting at time $t \in T$.

Furthermore, the function $scenario\_objectives(X^k, X[k-1], \xi[k], \mathbb{P}^{k+1})$ is introduced as given in Algorithm 12, where $\mathbb{P}^{k+1} = \{\mathbb{P}_1^{k+1}, \cdots, \mathbb{P}_{|\mathbb{P}^{k+1}|}^{k+1}\}$ is a set of candidate projects for decision step $k+1$; realizations of Ξ^{k+1}. The *scenario_objectives* function returns a list of objectives to evaluate solution X^k against projects $\mathbb{P}^{k+1}$. It does this by using a heuristic to schedule each project $\mathscr{P} \in \mathbb{P}^{k+1}$ as well as possible without looking ahead, given previous solutions $X[k-1]$. In our implementation, we use Algorithm 11

for this. Finally, we scale each objective value with the inverse of the critical path length of project $\mathscr{P} \in \mathbb{P}^{k+1}$.

Algorithm 12 Calculating a list of objectives for solution X^k with previous solutions $X[k-1]$, realization $\xi[k]$ and project scenarios $\mathbb{P}^{k+1}$.

```
1: v ← Vector of size |P^{k+1}|
2: for i ∈ [1, ··· , |P^{k+1}|] do
3:     X' ← Find a schedule for P_i^{k+1} given X[k], ξ[k]
4:     v_i ← (makespan of X')/cp(P_i^{k+1})
5: end for
6: return v
```

With this, we present the algorithm of the FM. This algorithm takes a set of k projects $\mathscr{P}[k]$, k arrival times $\tau[k]$, all previous solutions $X[k-1]$ and the project generation process Ξ^{k+1}, and returns a solution for the k^{th} project $\mathscr{P}^k$. The algorithm initializes by creating a population of γ **agents** with solutions $\mathbb{X}^1$ for project $\mathscr{P}^k$. Here, $\mathbb{X}^1$ is a matrix of dimension $\gamma \times 2|N|$, where each row represents a solution. In our implementation, we run Algorithm 11 on project $\mathscr{P}^k$ without considering future projects and storing the best γ unique solutions. Furthermore, it sets the variable ζ that represents the current number of scenario evaluations (realizations of Ξ^{k+1}) to 2. Besides this, the initialization process sets the iteration threshold counter variable θ to 0 and creates placeholders for the best solution indexes $\mathbf{q}$, scenario objective values V and combined objective values W. Both V and W are indexed by agent i and **population index** j, where population index j defines whether a property belongs to the old population ($j = 1$) or the new population ($j = 2$). Finally, it initializes empty set $\mathscr{D}$ where the resource usage and objective data will be stored and sets iteration counter m to zero.

The algorithm then performs iterative improvement steps until a termination criterion holds. In this iterative improvement step, a new set of schedules $\mathbb{X}^2$ is created. In our implementation, this is done by the DE modification step, as described in Algorithm 11. Secondly, we create ζ new projects by realizing Ξ^{k+1} and denote this set of projects by $\mathbb{P} = \{\mathbb{P}_1, \cdots, \mathbb{P}_\zeta\}$. Then, for each agent i in the population, we evaluate both $\mathbb{X}^1$ and $\mathbb{X}^2$. For each agent i and population index j, the scenario objective values are stored under V_{ij}. Then, the combined objective W_{ij} is defined as the sum of the average of the scenario objectives $\overline{V_{ij}}$ (the average over all ζ entries in V_{ij}) and the objective value of the current project. Furthermore, the algorithm adds a tuple of the resource profile and the scenario objectives under $\mathscr{D}_m$.

Then, we store the population index (1 for old population, 2 for new population) of the best solution under j^* and the agent index under $\mathbf{q}_2$. If the population index is 2 (and thus the best solution is part of the new population) and we are not in the first iteration, we perform a **paired t-test** between $V_{q_1 1}$ and $V_{q_2 2}$. This idea was adopted from Homem-De-Mello (2003). If the p-value of the test is smaller than 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. Subsequently, at the end of each iteration, the average objective value of the whole new population $\overline{W_2}$ is compared to the average objective of the best

population. Finally, both counters θ and m are incremented by one and $\mathbf{q}$ and $\mathbb{X}$ are updated, before starting a new iteration. If the average objective value has not reached a new minimum in the last ω iterations, which is an input parameter, the solution data $\mathscr{D}$ is saved and the best found solution is returned. A short overview of this algorithm is given in Algorithm 13 and the full algorithm is presented in Appendix 5.B.

Algorithm 13 Full method

```
Initialize population of γ solutions
ζ ← 2                                              ▷ Number of scenarios
while Not terminated do
    ℙ ← Scenario projects
    for i ∈ [1, ⋯, γ] do
        Create new solution.
        Compute ζ scenario objectives of old and new solution.
        Store scenario objectives and resource profiles of both solutions to 𝒟.
        Calculate combined objective of both solutions.
        If combined objective is improved, replace old solution by new solution.
    end for
    if best combined objective belongs to a new solution then
        p ←paired t-test between scenario objectives of old and new solution.
        if p < μ then
            ζ ← ζ + 1                                  ▷ Increase number of scenarios
        end if
    end if
    Compute average of combined objectives of new solutions.
    if did not find new lowest average in last ω iterations then
        Terminate while loop.
    end if
end while
Save data 𝒟.
Return best solution
```

5.4.3. 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 $\mathscr{D}$ 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 Figure 5.4.1. 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 $\mathscr{D}$. Next, the **training process** starts. This process uses stored data $\mathscr{D}$ 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.

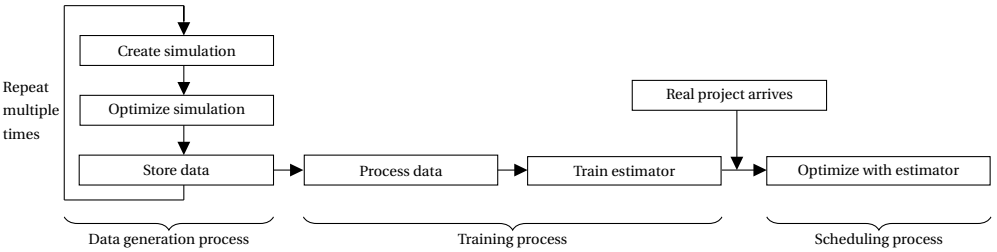


Figure 5.4.1: Workflow of optimization approach.

Since the data generation process consists of using the FM on simulated data, the description can be found in Section 5.4.2. Therefore, the remainder of this subsection explains the training and scheduling processes. The training process is split up into data processing and estimator training.

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**. However, the data $\mathscr{D}$ 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 $\mathbb{E}\big(\text{combined objective of } X\big) < \mathbb{E}\big(\text{combined objective of } X'\big)$ for any two solutions X and X'. Secondly, a possible method to create the estimator is to train it directly on combinations of resource profiles and the average scenario objectives in $\mathscr{D}$. However, this does not give a reliable estimate, since the objective values in $\mathscr{D}$ are obtained over different scenarios. Finally, since the number of scenarios varies in Algorithm 13, we note that the number of objective values in $\mathscr{D}$ also varies per iteration. It follows that solutions with more objective values 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 $\mathscr{Y} = \{Y_1, \cdots, Y_{|\mathscr{Y}|}\}$ contain all unique resource profiles in $\mathscr{D}$. The objective values are stored in U, where U_{ni} is the vector of scenario objectives for resource profile Y_i in iteration n of the while loop in Algorithm 13. Furthermore, we let C_{ij} be the set of all iterations in $\mathscr{D}$ that contain both resource profile $Y_i \in \mathscr{Y}$ and $Y_j \in \mathscr{Y}$. With this, we define vector Δ_{ij} containing the difference of all scenario objectives for iterations with both resource profile Y_i and Y_j:

$$\Delta_{ij} = \bigcup_{n \in C_{ij}} \left(U_{ni} - U_{nj}\right), \tag{5.4}$$

where we use the $\cup$-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}, with a low value for Q_{ij} indicating 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:

$$Q_{ij} = F\left(\frac{\overline{\Delta_{ij}}\sqrt{|\Delta_{ij}|}}{std\left(\Delta_{ij}\right)}, |\Delta_{ij}| - 1\right), \tag{5.5}$$

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 $Y_i \in \mathscr{Y}$ and $Y_j \in \mathscr{Y}$ 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:

$$\delta_i = \sum_{j=1, i \neq j}^{|\mathscr{Y}|} C^I_{ij}. \tag{5.6}$$

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

$$A_{ij} = \begin{cases} \frac{1}{\delta_i} Q_{ij} & \text{if } i \neq j \text{ and } C^I_{ij} = 1 \\ 1 - \frac{1}{\delta_i} \sum_{k \neq i} Q_{ik} C^I_{ik} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

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 Algorithm 13. Each edge $(i, j) \in E$ has a value of A_{ij}. Then, we can calculate the stationary distribution of a random walk by initializing the profile score row vector $\mathbf{s} = \{\mathbf{s}_1, \ldots, \mathbf{s}_{|\mathcal{Y}|}\}$ to $\mathbf{1}$ and repeatedly matrix-multiplying $\mathbf{s}$ by A until the change in $\mathbf{s}$ is below a certain treshold η. This is shown in Algorithm 14. After convergence, a large entry s_i in score vector $\mathbf{s}$ indicates a high quality resource profile $Y_i \in \mathcal{Y}$.

Algorithm 14 Random walk to generate profile scores $\mathbf{s}$

1: $\mathbf{s} \leftarrow \mathbf{1}$
2: $diff \leftarrow \infty$
3: **while** $diff > \eta$ **do**
4: $\quad \mathbf{s}' \leftarrow \mathbf{s} \cdot A$
5: $\quad diff \leftarrow |\mathbf{s} - \mathbf{s}'|$
6: $\quad \mathbf{s} \leftarrow \mathbf{s}'$
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 Algorithm 13, 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. Therefore, we use Algorithm 14 to obtain score vector $\mathbf{s}$. With this, we select the set of resource profiles with the highest score within its respective connected component. For each of these resource profiles, we perform a new objective evaluation to obtain a new edge with corresponding value. This is done by creating a set of new scenarios and calculating the scenario objectives, given each respective resource profile. Although Algorithm 12 requires the full previous solutions as input to calculate the scenario objectives, only the resource profiles of these solutions are used. Therefore, the scenario objectives can also be created from only the resource profiles. The number of scenarios is equal to the maximum number of scenarios evaluated in one generation, of all runs of Algorithm 13 in the data generation process (i.e.: largest ζ encountered in all runs of Algorithm 13). These new edges result in a connected graph $\mathcal{G}$, on which Algorithm 14 is executed to obtain scores $\mathbf{s}$. This is summarized below:

1. Generate data $\mathcal{D}$ using Algorithm 13.
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 profile 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.

ESTIMATOR TRAINING

The profile score vector $\mathbf{s}$ from the previous section forms a measure to compare the resource profiles $\mathscr{Y}$, 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 $\mathscr{Y}$ 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.

This 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 and 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. Then, the shape consists of nn^h hidden layers, each with nn^w nodes. After the final layer, there is one output node. This rectangular shape 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 and 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).

TRAINED METHOD ALGORITHM

In the previous section, it is explained how to process the data $\mathscr{D}$ to obtain profile scores and how to train a neural network on these profile scores. In the remainder of this sec-

tion, 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 $\mathcal{M} = \{\mathcal{M}_1, \cdots, \mathcal{M}_{|\mathcal{M}|}\}$ 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 $obj^{est}(X, \tau_{min}^{k+1})$ be the estimated objective value of solution X with next earliest arrival time τ_{min}^{k+1} and let $p^m > 0$ and $p^s > 0$ be scaling parameters. Then, we define the estimated objective obj^{est} as:

$$\begin{aligned} obj^{est}(X, \tau_{min}^{k+1}) = current\_objective(X) - p^m \overline{\mathcal{M}\left(RP\left(X, \tau_{min}^{k+1}\right)\right)} \\ + p^s std\left(\mathcal{M}\left(RP\left(X, \tau_{min}^{k+1}\right)\right)\right). \end{aligned} \tag{5.8}$$

This objective function thus evaluates a potential schedule X based on three parts. The first parts consists of its current objective. The second part consists of an average of all quality predictions multiplied by scaling paramater p^m. Since we are minimizing the objective, this term is subtracted. Finally, a penalty term consisting of scaling paramater 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 11 to create the TM.

5.5. 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.

5.5.1. 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, each arriving with equal probability. 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 τ'^{max}.

Before applying stochastic optimization methods, it is recommended to test the potential of stochastic optimization by exploring the bounds. A commonly used upper bound can be found by creating a naive model and calculating the expected result of using this model (Birge and 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 model: 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, the expected value of the **Wait-and-See** (WS) solution (Birge and 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 by 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 on the number of activities and on the estimated VSS value. These instances are shown in Table 5.5.1.

5.5.2. 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 11, 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 Figure 5.4.1 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 instance, 10 simulations are created, with each simulation consisting of $|K| = 5$ sequentially arriving projects. Then, Algorithm 13 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 5.4.3, to determine the parameters p^l, nn^h, nn^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 ($|\mathscr{M}| \leq 15$, where 15 is chosen due to computational resource limitations) and the estimator parameters p^m and p^s. All parameters per instance are given in Table 5.5.2 and the complete instances are given in van der Beek (2022b). 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 5.5.3. Here, it can be seen that the data creation parts 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

Table 5.5.1: Instances.

#	n^{base}	n^{opt}	n^{sel}	SP	RF	RC	$\|R\|$	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

all this is done, training a neural network takes relatively short: around 3 hours.

5.5.3. COMPUTATION RESULTS

The presented instances are used to evaluate the solution methods. As presented earlier, there are 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. The former compares only the GM and the TM. Since both of these methods are relatively fast, it is possible to evaluate many simulations. 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 latter category, comparison with all methods, also includes the FM. The purpose of these tests 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 **total relative makespan**:

$$trm = \sum_{k \in K} \left(\frac{\sum_{t \in T^k} (t - \tau^k) X^k_{|N^k|t}}{cp(\mathscr{P}^k)} \right), \tag{5.9}$$

which can be seen as the realized value of the ctg (Equation (5.3)): the sum of the makespan over all projects, relative to the critical path length.

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.

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 instance, 100 simulations are created by realizing Ξ for each $k \in K$, thus having 600 simulations in total. Then, each simulation is optimized with both the TM and the GM.

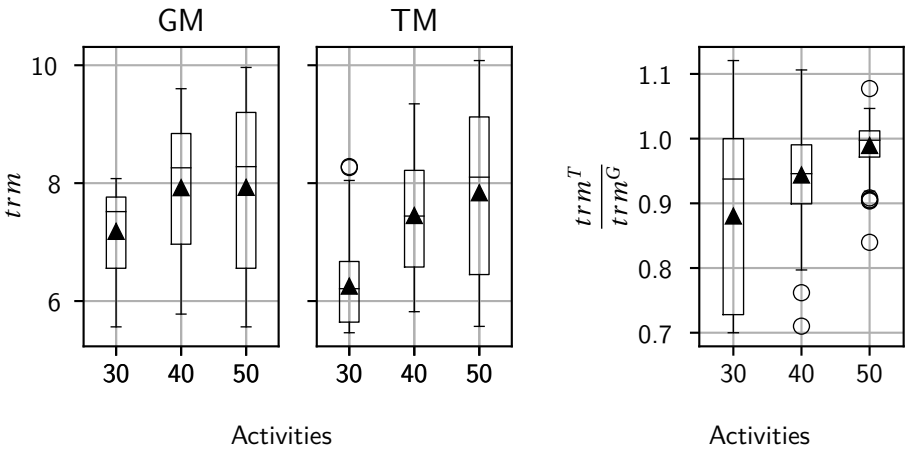
In Table 5.5.4, the results are summarized for these tests. Note that the minimal values for the trm is 5, since sequences of 5 projects are considered. It can be seen that both the mean and the median trm are lower for the TM. However, the GM is slightly more

Table 5.5.2: Parameters per instance.

#	l	nn^w	nn^h	p^l	p^w	p^m	p^s	$\lvert\mathscr{M}\rvert$
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

stable, since the standard deviation of the trm is around 7% lower. Finally, we evaluate the number of times that either method *exclusively* has the lowest trm. Here, it can be seen that the majority of simulations has the best trm 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.

Furthermore, the tests are evaluated in more detail in Figure 5.5.1. Here, the instances are separated by the number of activities in the base network. The trms are shown in Figure 5.5.1a. For comparison between the two methods, Figure 5.5.1b shows the ratio of the trms for the TM trm^T and the GM trm^G. This reveals a number of trends. Firstly, in Figure 5.5.1a, it can be seen that the deviation of the trms increases with number of activities for the GM. Secondly, in Figure 5.5.1b, 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 zero, the TM still performs better than the GM on the instances with 50 activities.



(a) Value of trm per number of activities.

(b) Ratio of trm per number of activities.

Figure 5.5.1: Comparison between the TM and the GM on trm.

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 $pobj_k^G$ and $pobj_k^T$ refer to the values for the GM and TM, respectively. Then, Figure 5.5.2 shows

Table 5.5.3: Average computing times of processes.

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

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.

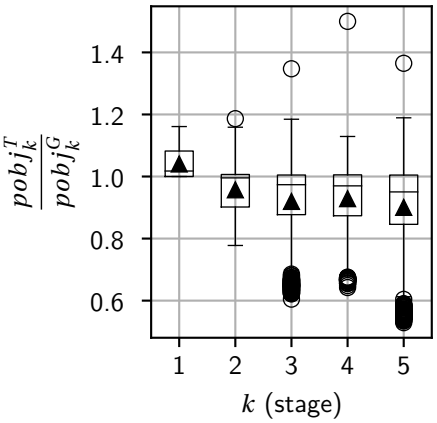


Figure 5.5.2: Partial objective ratios per stage.

Finally, the effect of using multiple trained neural networks in the estimator is evaluated. In Figure 5.5.3, the average ratio between trm 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.

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 per instance and executing all methods on these simulations.

Table 5.5.4: Results of comparison between GM and TM.

	GM	TM
Mean trm	7.678	7.181
Median trm	7.636	6.826
Standard deviation trm	1.132	1.222
# Lowest trm	173	403

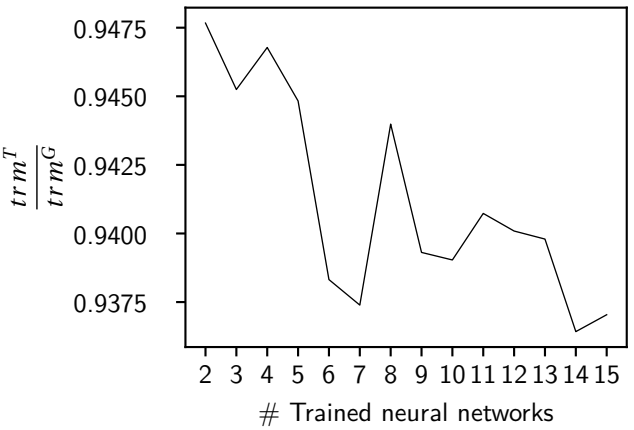


Figure 5.5.3: Value of trm ratio while varying the number of trained neural networks in the estimator.

The summarized results of all three methods are shown in Table 5.5.5. Here, it can be seen that the quality of solutions found by the FM is superior: The mean trm, median trm, standard deviation of the trm and (non-exclusive) number of lowest trms found are better for the FM than for the other methods. However, it also can be seen that this method has an average duration of more than three days, where the GM and TM have average computing times of less than 1 and 11 minutes, 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.

In Figure 5.5.4, the trms 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 trm.

When evaluating the ratio between the trm of the TM and FM, as shown in Figure 5.5.5a, 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 in-

Table 5.5.5: Summarized results for comparison with FM.

	GM	TM	FM
Mean trm	7.861	7.283	6.958
Median trm	7.760	7.051	6.686
Standard deviation trm	1.100	1.245	1.165
Mean computing time (h)	0.014	0.174	52.253
# Lowest trm	11	29	80

stances. This can be seen in Figure 5.5.4, where there is a relatively small improvement for the FM 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 Figure 5.5.5b, 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 $\mathscr{D}$, 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 Figure 5.5.6. Here, a clear increasing trend can be seen between the number of activities and the computing time.

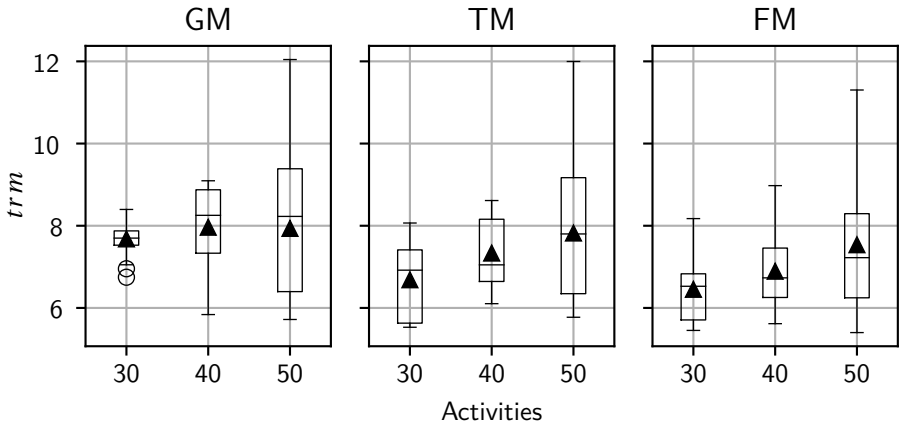
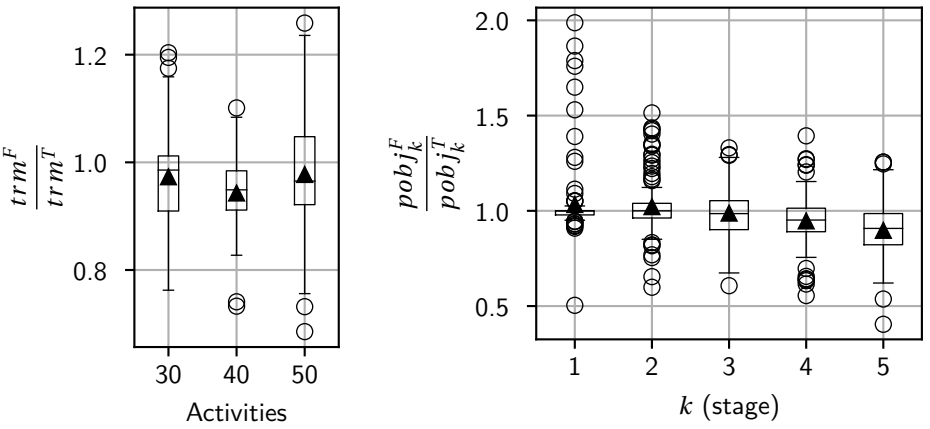


Figure 5.5.4: Value of trm per method.

5.6. CONCLUSIONS

In this chapter, the stochastic optimization problem DRCMPSP/SS is introduced. Furthermore, three solution methods are created: the greedy method, 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



(a) Per number of activities.

(b) Per stage.

Figure 5.5.5: Ratios of partial objectives between FM and TM.

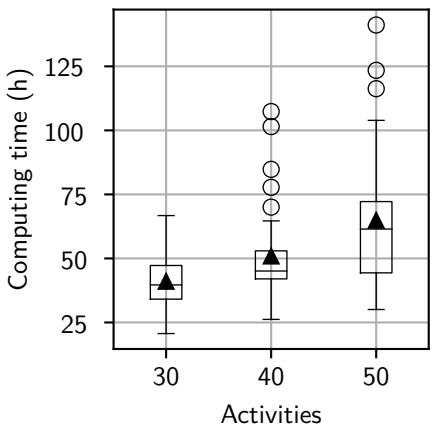


Figure 5.5.6: Computing times per number of activities for the FM.

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.

From a computational point of view, it is shown how to use data from a heuristic optimization algorithm for the RCPSP. This learning process is very versatile. First of all, the data collection process is the same for any population-based search algorithm and, thus, can easily be used in other heuristics. 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.

For future research, one might focus on computational evaluation of the trained method. One possibility is to evaluate the use of profile networks and study the correlation of network parameters, such as density, to the performance of the trained method. Secondly, due to the high computational demands, the number of evaluated instances is limited. Although this already shows certain trends, these can be verified with more computational tests.

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 chapter demonstrates that training from data is possible for the DRCMPSP/SS, and possibly for other variants of the Resource Constrained Project Scheduling Problem.

APPENDIX

5.A. NOTATION

Sets and matrices

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

Variables, parameters and vectors

b^k_{ri}	Resource requirement of resource $r \in R$ for activity $i \in N$ of project $\mathscr{P}^k$.
c	Replacement parameter in Algorithm 11.
$\mathbf{d}^k$	Duration vector of project $\mathscr{P}^k$.
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.
nn^h	Number of hidden layers in neural network.
nn^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 11.
trm	Total relative makespan.
w	Weight parameter in Algorithm 11.
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 $Y_i \in \mathscr{Y}$ 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 $\mathscr{P}^k$.
τ^k_{min}	Earliest arrival time of project $\mathscr{P}^k$.
τ'	Deviation from earliest arrival time.
ω	Iteration treshold parameter.
SP	Serial/parallel parameter.
RC	Resource constrainedness.
RF	Resource factor.

Functions

$cp(\mathscr{P})$	Critical path length of project $\mathscr{P}$.
$ctg^k(X[k-1],\Xi^k)$	Cost to go at decision step $k \in K$, given schedules $X[k-1]$ and distribution Ξ^k, for project $\mathscr{P}^k$.
$F(x,n)$	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 $t \in T$.
$scenario\_objectives$ $(X^k,X[k-1],\xi[k],\mathbb{P})$	Scenario objectives, given solutions $X[k]$, realizations $\xi[k]$ of distributions $\Xi[k]$ for scenario projects $\mathbb{P}$.
$std(x)$	Standard deviation of x.
trm	Total relative makespan.
$\mathscr{X}^k(X[k-1])$	Feasible schedules for project $\mathscr{P}^k$, given solutions $X[k-1]$.

5.B. FULL METHOD ALGORITHM

Algorithm 15 Generate data for project $\mathscr{P}^k$, given previous and current projects $\mathscr{P}[k]$, time steps $\tau[k]$, previous solutions $X[k-1]$ and project generation process Ξ^{k+1}.

$\mathbb{X}^1 \leftarrow [\mathbb{X}^1_1, \cdots, \mathbb{X}^1_\gamma]$ ▷ Initialize population
$\zeta \leftarrow 2$ ▷ Number of scenarios
$\theta \leftarrow 0$ ▷ Termination criterion variable
$\mathbf{q} = [q_1, q_2] \leftarrow [0,0]$
$V_{ij} \leftarrow \emptyset, \quad \forall i \in [1, \cdots, \gamma], j \in \{1,2\}$
$W_{ij} \leftarrow 0, \quad \forall i \in [1, \cdots, \gamma], j \in \{1,2\}$
$m \leftarrow 0$
$\mathscr{D} \leftarrow \emptyset$
$W^* \leftarrow \infty$
while $\theta < \omega$ **do**
  $\mathbb{X}^2 \leftarrow [\mathbb{X}^2_1, \cdots, \mathbb{X}^2_\gamma]$ ▷ Create new population as in Algorithm 11
  $\mathbb{P} \leftarrow$ create ζ projects (realizations of Ξ^{k+1})
  $\mathscr{D}_m \leftarrow \emptyset$
  for $i \in [1, \cdots, \gamma]$ **do**
    for $j \in \{1,2\}$ **do** ▷ For both populations
      $V_{ij} \leftarrow$ scenario_objectives$(\mathbb{X}^j_i, X[k-1], \xi[k], \mathbb{P})$ using Algorithm 12
      $W_{ij} \leftarrow \overline{V_{ij}} +$ makespan of $\mathbb{X}^j_i / cp(\mathscr{P}^k)$
      $\mathscr{D}_m \leftarrow \mathscr{D}_m \cup \{(RP(X[k-1] \cup \mathbb{X}^j_i, \tau^{k+1}_{min}), V_{ij})\}$
    end for
  end for
  $j^* \leftarrow \arg\min_{j \in 1,2} \min_{i \in [1,\cdots,\gamma]} W_{ij}$
  $q_2 \leftarrow \arg\min_{i \in [1,\cdots,\gamma]} W_{ij^*}$
  if $j^* = 2$ and $m > 0$ **then** ▷ Found new best solution
    $p \leftarrow$ Paired t-test between $V_{q_1 1}$ and $V_{q_2 2}$
    if $p < \mu$ **then**
      $\zeta \leftarrow \zeta + 1$
    end if
  end if
  if $\overline{W_2} < W^*$ **then** ▷ Found new lowest average
    $W^* \leftarrow \overline{W_2}$
    $\theta \leftarrow -1$
  end if
  $\theta \leftarrow \theta + 1$
  $m \leftarrow m + 1$
  $\mathbf{q} \leftarrow [q_2, 0]$
  $\mathbb{X}^1_i \leftarrow \mathbb{X}^2_i$ if $W_{i2} < W_{i1}, \quad \forall i \in [1, \cdots, \gamma]$
end while
Save $\mathscr{D}$

6

DISCUSSION AND CONCLUSIONS

In this chapter, we review the work done in this dissertation and discuss it both from an application and an optimization point of view. Subsequently, conclusions of this dissertation are given.

6.1. DISCUSSION

In Chapter 2, a time-indexed Mixed Integer Linear Programming (MILP) model is introduced to model the flexible project structure. The decision for using a time based index was due to models from literature that all use a similar indexation. However, for the standard RCPSP, various formulations are present that do not use time indexing. Depending on the structure of the problem, these alternative formulations can result in better bounds, and better branch-and-bound performance. Therefore, even though the decision for a time-indexed MILP makes it simple to represent the flexible project structure, it is worth evaluating whether other indexations result in improved solver performance.

For this MILP problem, we defined Max One Execution Sets (MOESs) and Non Empty Execution Sets (NEESs) that are used as building blocks for variable reduction. Part of this variable reduction was done by creating cutting planes for each activity. Although this resulted in a netto reduction in computing time, the preprocessing phase has a significant duration. This leaves room for improvement. It should be researched if there are some indications for the effectiveness per cutting plane. Various theoretic approaches for this are listed in Dey and Molinaro (2018). With this, it might be possible to reduce the preprocessing time, while still having a similar effect on the branch-and-bound performance. Furthermore, the cutting planes are only used to remove variables from the MILP, but are not added as constraints for the final branch-and-bound algorithm. Although some preliminary experiments that included these cutting planes did not indicate a reduction in computing time, additional research might find some cases or methods where adding these constraints prove to be effective. This might be done in the root node of the branch-and-bound tree, or in later nodes. For example, this is done for the RCPSP in Araujo et al. (2020) and shows significant improvements.

Besides these cutting planes, it also might be possible to create different types of cutting planes that use NEESs and/or MOESs. In this dissertation, resource usage was not included in the cutting planes. However, for the standard RCPSP, various types of cutting planes are based on resource usage (Baptiste and Demassey, 2004). Therefore, generalizing these cutting planes to the RCPSP-PS/CPR might lead to new solution methods. Since NEESs and MOESs give information on whether an activity is executed, they might be of importance while generalizing these cutting planes.

When comparing the theoretic approach, given in Chapter 2, to the real life case of modular shipping, one might question how similar the instances of the computational tests are to the instances encountered in practice. The approach taken in Chapter 2 considers relatively complicated instances to create a challenge for the optimization algorithm. Due to the lack of data available for modular shipping, this approach was preferred to creating more realistic instances and running the risk of oversimplifying. However, this approach does not give a guarantee that similar performance is expected for realistic instances and, therefore, a case study on computational performance with realistic instances is recommended.

When considering these methods in practice, they should not only be used on the complete design after a ship order comes in. Instead, design should be accompanied by optimization. This allows for designs to be optimized for production, which is one of the aims of a modular product family. However, it should be noted that this requires additional design effort. Similarly, even when using these methods after the design phase, it requires the design to include various production alternatives. This additional investment of effort might result in larger benefits in the production process. However, this requires for the product series to be sufficiently large and for a careful consideration of the amount of detail to be modeled.

In Chapter 3, the model was extended to include consumption and production of nonrenewable resources. This allows for the modeling of resources, such as capital and floor space. In the model, a simplified representation for these resources is used. This was done for two reasons: first, it allows that a single type of resource constraints represents both capital, floor space, and other types of resources. Secondly, the simplification reduced the computational effort compared to full modeling of both resources. Nevertheless, it is important to note the shortcomings of this approach. For capital, it is the assumption of constant cost at different times. In reality, money obtained later in time is worth less due to (missed) interest. For the resource floor space, the two-dimensional problem is simplified to a single dimensional problem. Although both simplifications are valid and still offer useful results, one should be aware of these when implementing solutions obtained by solving the Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources (RCPSP-PS/CPR).

Furthermore, both heuristic algorithms can only solve instances with acyclic group graphs. Although this seems quite a general case for practical purposes, it cannot be guaranteed that there are no use cases without this property. Therefore, if this is encountered, additional methods of solving the selection problem should be explored.

From an optimization point of view, the Hybrid Differential Evolution (HDE) algo-

rithm can be considered relatively simple. It consists of one neighborhood of local improvement and a standard Differential Evolution algorithm. It can be worth researching more complicated algorithms, or additional local neighborhoods. However, as the Ant Colony Optimization (ACO) algorithm shows, more complicated does not necessarily mean better.

One direction of improving the HDE might be to consider the infeasibility of non-renewable resources. Currently, not much effort is put into fixing infeasible solutions: a quite standard approach of penalizing and possibly restarting handles this. Although this does generally result in feasible solutions, this might be because of the algorithm choosing a different selection of activities. If this is the case, it also might be the case that additional effort to make infeasible solutions feasible while keeping the selection of activities fixed will find unexplored parts of the feasible region. This approach was used for the ACO algorithm and was shown to aid the solutions to become feasible.

Finally, we discuss the usage of the heuristic algorithm compared to the exact algorithm. Generally, if the exact algorithm can be used, it should be used. However, if the instances are too large, or the available computing time is too short, the switch should be made to the heuristic algorithms. It can also be the case that both can be used. For example, to create a quick estimate of the schedule with the heuristic algorithm, while taking more time to calculate the definitive schedule. Furthermore, the heuristic solution can be used to give the branch-and-bound algorithm a 'warm start'. Although this improves performance for many optimization problems, additional research should be done to quantify this effect.

In Chapters 4 and 5, scheduling with arriving projects is considered. Both chapters rely on simulations to model the arriving projects. Although the performance of both methods can be seen as an indication of the validity of this research direction, two comments have to be made. First, creating realistic simulations is not a trivial task, as can be deducted from the numerous research in this area (Kim et al., 2005a,b; Park et al., 2016). In this dissertation, we assumed to have simulators to evaluate the possibilities of optimization methods. However, these simulators are not realistic and additional research is needed to create better ones. This has to be done in coorporation with industry experts and/or while using historical data. The second discussion point of the simulators is that it cannot be guaranteed that the presented methods perform similarly for different simulators. Especially for the methods considered in Chapter 5, it might even be the case that the greedy scheduling method provides near-optimal solutions for realistic cases. If this is true, optimization methods are not required in that case.

Furthermore, for the instances considered in Chapter 4, a fixed level of inventory is assumed. This can be seen as a valid consumption for a steady state case, where we expect to have a long horizon of products to be produced. However, at the end of the life cycle of a modular shipping family, additional considerations should be used to determine whether replenishing inventory items is still worth the investment.

Additionally, a comment can be made on the instances considered in Chapter 4. To allow for evaluation of many instances, they were kept relatively small. One assumption for this was to consider a single yard and roughly one simultaneous project being executed. In reality, risk can be mitigated by using a shared product inventory for multiple yards. Although this can be modeled with the RCPSP-MP, it will result in much larger

instances. Therefore, computational tests are required to evaluate the performance. Ideally, this would be done in combination with a case study with realistic instances.

When evaluating the results in Chapter 4 on solution quality, it was shown that the *overshoot-limited method* performed best, even while relying on the *repair function* more often than other methods. This might indicate even more improvement there, as the algorithm often did not converge. Therefore, running the algorithm with a higher computing time limit might result in even better solutions. Furthermore, although combining the two extensions had some positive result, the *combined method* did not create better solutions than the *overshoot-limited method*. Additional research can therefore focus on other ways of combining the extensions.

The scope of Chapter 5 is learning from previous optimization processes. Therefore, the optimization algorithm itself was relatively simple, created by modifying a standard DE algorithm for the RCPSP, which allows delays between activities to be better prepared for future incoming projects. Thus, additional research in the optimization aspect might lead to better solutions. When developing these methods, one should consider that the objective is not only to find the best solution, but also to encounter many different resource profiles to generate training data. Thus, since the goals differ, it might be better to have a different method for generating data then for the final optimization.

Similarly, additional research can be done into the data processing part. As the research in Chapter 5 was done in a relatively new field, the presented method was created based on inspiration from various fields, while the challenge was mostly to find a working method. In future research, this can serve as a benchmark to see if alternative methods can further improve the performance. This also holds for the neural network. Preliminary tests were done with various architectures and the best one was chosen. However, each architecture might be improved based on further parameter tuning and combining several parts. Thus, a potential research topic could be the accuracy of estimating the objective function.

Finally, we consider the data generation process. This is currently done before all project arrivals. However, in practice, there is a lot of time between subsequent project arrivals. Therefore, after the scheduling of one project, significant time can be dedicated to generate data based on simulations with the current resource profiles. This is expected to improve the performance, as the data more closely represents the expected future situation.

When evaluating the link to the application of the proposed optimization methods, two points have to be considered. Firstly, the methods in this thesis were validated computationally by creating virtual instances. Although the structure of these instances is chosen to replicate the structure encountered in practice, or to ensure a computational challenge, multiple differences remain. For example, the resource structure of the instances are created by instance generators from literature. Comparing this to real instances, the instances in this thesis have fewer types of resources that are used more intensively. In real instances, there are usually more types of resources, but many resources are only used in just a few activities. Thus, in order to fully evaluate the computational performance of the proposed methods, instances from practice have to be used.

Secondly, the flexible project structure requires the modeling of *potential* modules. This requires more effort than the current approach, where only the executed project

structure is modeled. To modify this, both the data infrastructure and company processes have to be changed. However, this requires a significant investment from the shipbuilding company. Therefore, it is important to perform case studies to quantify the benefits of optimization in shipbuilding. This gives shipbuilding insight in the potential benefits of scheduling with flexible projects, which can serve as a motivation for using flexible project structures in scheduling.

6.2. CONCLUSION

In this dissertation, scheduling methods are presented that handle the challenges of scheduling for modular production. This was done with the assumption that, in order to evaluate a modular shipbuilding design or production process, the right scheduling decisions are required.

In Chapter 2, a new formulation for the RCPSP-PS was given to capture all aspects of the flexible project structure for modular shipbuilding. It was shown that, even without the presented solution space reduction, this model performed significantly better than a similar, but less general model from literature. Furthermore, this formulation allowed for mathematical analysis, introducing the concepts of NEESs and MOESs. These concepts served as building blocks for the improved solution methods. The method consists of two parts, where the first part is adding valid inequalities and imposing lower bounds based on an adapted critical path method. The second part improved these bounds by creating cutting planes. This second method of preprocessing took significantly more computing time. However, since it allowed for improved performance of the subsequent branch-and-bound algorithm, it proved an effective method of decreasing the computing time. Especially on instances that were computationally more challenging due to the resource constraints, the complete method improved performance significantly compared to the partial method, consisting of only a few building blocks of the solution method.

Both the partial and complete method show that NEESs and MOESs provide suitable building blocks for solution methods for the RCPSP-PS. The resulting method solves instances with up to around a hundred executed activities to optimality. Therefore, this provides a useful method for real world purposes with a small to medium level of detail and multiple hours of available computing time. Furthermore, it allows for the evaluation of heuristic methods, which in turn can be used for larger instances or cases with less available computing time.

These heuristic methods were presented in Chapter 3. Additionally, nonrenewable resources with consumption and production were introduced. With this, resources such as floor space, capital and inventory items can be modeled. Besides the usefulness for scheduling incoming projects, this also allows the shipyard to evaluate various scenarios of yard capacities. By varying resources such as available workers, floor space, machines, inventory and capital, while performing simulations, the effect of varying these resources can be researched. This allows for better insight and justification of investing both time and money to transform to a modular shipbuilding approach.

Furthermore, the heuristic methods presented decreased the computing time from several hours for the exact method to several minutes, while having only a slight loss in

solution quality. This allows for the RCPSP-PS/CPR to be used on instances with high detail and in cases with only several minutes of available computing time. Furthermore, even for cases where plenty of computing time is available, the heuristic methods can be used as an initial solution that in turn can be further improved by exact methods.

From an optimization point of view, it was shown that, although the selection problem is NP-hard, most real world cases are a special case of this problem for which a polynomial time algorithm exists. By restricting ourselves to these cases, a lot of computing time improvement is gained. This shows that the question: "What do we really want to solve?" is important to consider, instead of trying to solve the most general case.

The two algorithms presented were an HDE algorithm and an ACO algorithm. Even though they both performed better on special cases than an existing algorithm from literature, HDE outperformed ACO on nearly every aspect. Furthermore, the HDE algorithm also required less parameters, which in turn resulted in easier tuning.

Subsequently, in Chapter 4, the RCPSP-PS/CPR was expanded with resource allocation, profit maximization and stochastic project arrivals. Two important steps were made here for the shipbuilder. First, instead of creating various scenarios of resource allocation and evaluating them, it now is possible to optimize this and thus automatically find good configurations. Secondly, instead of considering single projects, the step is now made to consider a series of projects. Producing a product family instead of a set of individual products is one of the main concepts of modular production. Therefore, to properly quantify the benefits and make the correct scheduling decisions, it is not sufficient to consider only separate projects. Thus, Chapter 4 bridges an important gap for scheduling in modular shipbuilding.

When evaluating the computational results, it can be seen that although even a basic progressive hedging algorithm performs better than using an MILP solver, extensions are needed to converge to implementable solutions without relying on the *repair function*. *Variable bounding* has the strongest effect on convergence properties. Conversely, the effect on convergence by limiting the overshoot effect of variables is less strong, but ultimately leads to better quality solutions.

Finally, in Chapter 5, production of a series of projects is viewed from a standard RCPSP point of view with stochastic arrivals. In this chapter, the flexible project network was disregarded and the focus was put primarily on scheduling such that the resource profiles were beneficial for future arriving projects. For modular shipbuilding, this can be used in a more developed stage, where the inventory allocation choices, outsourcing choices and modularity choices are already made and only a fixed project structure remains.

From a computational point of view, it is shown that using a solution representation that supports delays between activities can create schedules that allow for improved schedules of projects arriving in the future. The best schedules were obtained by a simulation optimization approach, which has a computing time of several days. However, it is shown that it is possible to learn from the data of this algorithm, by storing the resource profiles and corresponding objective function values. This resulted in the trained method, which only requires a fraction of the computing time of the simulation based method, but still provides significantly better results than without looking ahead.

In conclusion, the models and methods introduced in this dissertation can be used to schedule for modular production. In particular, the methods in Chapters 2 and 3 can be used to handle choices in module definition and usage. Furthermore, Chapter 4 introduces inventory management for a product family, along with operational decisions, such as outsourcing or in-house production. Finally, Chapter 5 handles the challenge of using shared resources for stochastically arriving projects, as encountered in modular production. These methods allow for real-life instances to be created, in order to quantify the results of modular production. This is an important step in the study of modular production in shipbuilding: multiple studies indicate the potential benefits, but few give quantification of these benefits. Furthermore, even if some quantification is given, it is often case specific and not generalizable to other branches of shipbuilding.

Since adopting a modular production approach requires a complete overhaul of data structure, production processes and design methods, quantification of these processes are required to justify the large costs of these changes. Therefore, creating the models and methods that are required for this quantification, as is done in this dissertation, is an essential step for introducing modular production in shipbuilding.

Due to the abstractness of the models and methods introduced in this dissertation, they are also valid in other industries. Naturally, they can be used for modular production of any product, such as airplanes or houses. However, the methods are not restricted to modular projects. Flexible networks were also found in other applications, such as highway construction, aircraft turnaround scheduling and regular housing construction. Here, the improved exact and heuristic methods can result in increased efficiency. Furthermore, the principle of scheduling projects while looking ahead can be of use in many cases. In nearly every industry where the RCPSP is used, there are companies that use the same resources for multiple sequential projects. Therefore, both the resource allocation from Chapter 4 as the resource profile optimization from Chapter 5 can be used in many industries.

BIBLIOGRAPHY

N. Agarwala. Modular Construction and Ihop for Increased Productivity in Shipbuilding. In *International seminar on Nation Building through Ship Building*, 2015.

A. Alliff, A. Mokhtar, M. Abu, U. Amir, B. Abu, A. Hafizah, A. Mokhtar, D. Technology, K. Perdana, S. Besi, and P. History. A Review of Modular Construction Shipbuilding in Malaysian Shipyard. *Proceeding of Ocean, Mechanical and Aerospace*, 3:135–143, 2016.

S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury. Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, 2016. doi:10.1007/s10479-015-2019-x.

I. Ančić, M. Perčić, L. Runko Luttenberger, and N. Vladimir. Potential to reduce environmental pollution from ships through the modular concept approach. *8th International Maritime Science Conference (IMSC 2019)*, pages 371–375, 2019.

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

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

A. Athanasia, G. Anne-Bénédicte, and M. Jacopo. The offshore wind market deployment: Forecasts for 2020, 2030 and impacts on the European supply chain development. *Energy Procedia*, 24(January 2012):2–10, 2012. doi:10.1016/j.egypro.2012.06.080.

H. Bakker, F. Dunke, and S. Nickel. A structuring review on multi-stage optimization under uncertainty: Aligning concepts from theory and practice. *Omega (United Kingdom)*, 96:102080, 2020. doi:10.1016/j.omega.2019.06.006.

P. Baptiste and S. Demassey. Tight LP bounds for resource constrained project scheduling. *OR Spectrum*, 26(2):251–262, 2004. doi:10.1007/s00291-003-0155-1.

R. Barták, O. Čepek, and P. Surynek. Modelling alternatives in temporal networks. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007*, 2007. ISBN 1424407044. doi:10.1109/SCIS.2007.367680.

O. Bellenguez and E. Néron. Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3616 LNCS: 229–243, 2005. doi:10.1007/11593577_14.

J. Bergstra, D. Yamins, and D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 2013. PMLR.

P. Bezerra and S. Scheer. *A Metaheuristic Procedure Combined with 4D Simulation as an Alternative for the Scheduling Process of Housing Complexes*, volume 98. Springer International Publishing, 2021. ISBN 9783030512958. doi:10.1007/978-3-030-51295-8_41.

J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer-Verlag, New York, NY, USA, 1997.

J. Blazewicz, J. K. Lenstra, and A. H. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, January 1983. doi:10.1016/0166-218X(83)90012-4.

D. Braha, A. A. Minai, and Y. Bar-yam. *Complex Engineered Systems*. Understanding Complex Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-32831-5. doi:10.1007/3-540-32834-3.

P. O. Brett, J. J. G. Agis, A. Ebrahimi, S. O. Erikstad, and B. E. Asbjørnslett. a Rational Approach To Handle Uncertainty and Complexity in Marine Systems Design. *SNAME 14th International Marine Design Conference, IMDC 2022*, pages 1–21, 2022. doi:10.5957/IMDC-2022-270.

P. Brucker and S. Knust. Linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research*, 127(2):355–362, 2000. doi:10.1016/S0377-2217(99)00489-0.

J. Buergin, F. Belkadi, C. Hupays, R. K. Gupta, F. Bitte, G. Lanza, and A. Bernard. 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, May 2018. doi:10.1016/j.cirpj.2018.01.003.

C. Capa and G. Ulusoy. Proactive project scheduling in an R&D department a bi-objective genetic algorithm. *IEOM 2015 - 5th International Conference on Industrial Engineering and Operations Management, Proceeding*, pages 1–6, 2015. doi:10.1109/IEOM.2015.7093733.

R. Čapek, P. Šcha, and Z. Hanzálek. Production scheduling with alternative process plans. *European Journal of Operational Research*, 217(2):300–311, 2012. doi:10.1016/j.ejor.2011.09.018.

J. Carlier, A. Moukrim, and H. Xu. The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm. *Discrete Applied Mathematics*, 157(17):3631–3642, 2009. doi:10.1016/j.dam.2009.02.012.

C. C. Carøe and R. Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1):37–45, 1999. doi:10.1016/S0167-6377(98)00050-9.

H. J. Chen, G. Ding, J. Zhang, and S. Qin. Research on priority rules for the stochastic resource constrained multi-project scheduling problem with new project arrival. *Computers and Industrial Engineering*, 137(August):106060, 2019. doi:10.1016/j.cie.2019.106060.

J. Choi, M. J. Realff, and J. H. Lee. 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, September 2007. doi:10.1002/rnc.1164.

A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In *Proceedings of the First European Conference on Artificial Life*, 1991.

E. Commission, D.-G. for Climate Action, D.-G. for Energy, D.-G. for Mobility, Transport, A. De Vita, P. Capros, L. Paroussos, K. Fragkiadakis, P. Karkatsoulis, L. Höglund-Isaksson, W. Winiwarter, P. Purohit, A. Gómez-Sanabria, P. Rafaj, L. Warnecke, A. Deppermann, M. Gusti, S. Frank, P. Lauri, F. Fulvio, A. Florou, M. Kannavou, N. Forsell, T. Fotiou, P. Siskos, P. Havlík, I. Tsiropoulos, S. Evangelopoulou, P. Witzke, M. Kesting, N. Katoufa, I. Mitsios, G. Asimakopoulou, and T. Kalokyris. *EU reference scenario 2020 : energy, transport and GHG emissions : trends to 2050.* Publications Office, 2021. doi:doi/10.2833/35750.

W. Cook, W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. 1997.

T. G. Crainic, X. Fu, M. Gendreau, W. Rei, and S. W. Wallace. Progressive hedging-based metaheuristics for stochastic network design. In *Networks*, volume 58, pages 114–124, September 2011. doi:10.1002/net.20456.

G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation — Proceedings of a Conference*, pages 339–347, New York, 1951. Wiley.

G. B. Dantzig. Linear Programming under Uncertainty. *Management Science*, 1(3-4): 197–206, April 1955. doi:10.1287/mnsc.1.3-4.197.

E. Demeulemeester, M. Vanhoucke, and W. Herroelen. RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, 6(1):17–38, 2003. doi:10.1023/A:1022283403119.

S. S. Dey and M. Molinaro. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170(1):237–266, 2018. doi:10.1007/s10107-018-1302-4.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

W. Ferdous, Y. Bai, T. D. Ngo, A. Manalo, and P. Mendis. New advancements, challenges and opportunities of multi-storey modular buildings – A state-of-the-art review. *Engineering Structures*, 183(January):883–893, 2019. doi:10.1016/j.engstruct.2019.01.061.

P. Fredriksson. Operations and logistics issues in modular assembly processes: Cases from the automotive sector. *Journal of Manufacturing Technology Management*, 17 (2):168–186, 2006. doi:10.1108/17410380610642250.

V. Frigant and Y. Lung. Geographical proximity and supplying relationships in modular production. *International Journal of Urban and Regional Research*, 26(4):742–755, 2002. doi:10.1111/1468-2427.00415.

E. Gagatsi, T. Estrup, and A. Halatsis. Exploring the Potentials of Electrical Waterborne Transport in Europe: The E-ferry Concept. *Transportation Research Procedia*, 14:1571–1580, 2016. doi:10.1016/j.trpro.2016.05.122.

R. Garud, A. Kumaraswamy, and R. Langlois. *Managing in the Modular Age: Architectures, Networks, and Organizations*. Wiley, 2009. ISBN 9781405141949.

M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham, 2019. ISBN 978-3-319-91085-7. doi:10.1007/978-3-319-91086-4.

F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, January 1977. doi:10.1111/j.1540-5915.1977.tb01074.x.

Gunawan, K. Hamada, T. Deguchi, H. Yamamoto, and Y. Morita. Design Optimization of Piping Arrangements in Series Ships based on the Modularization Concept. *International Journal of Technology*, 9(4):675, July 2018. doi:10.14716/ijtech.v9i4.969.

Gunawan, Yanuar, F. A. Waskita, and A. Kurniawan. Modularization of ship engine room using design structure matrix (DSM) based on the genetic algorithm. *Engineering Journal*, 24(4):205–216, 2020. doi:10.4186/ej.2020.24.4.205.

G. Guo, G. Hackebeil, S. M. Ryan, J. P. Watson, and D. L. Woodruff. Integration of progressive hedging and dual decomposition in stochastic integer programs. *Operations Research Letters*, 43(3):311–316, 2015. doi:10.1016/j.orl.2015.03.008.

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

J. H. Hao, M. Liu, J. H. Lin, and C. Wu. A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages. *Computers and Operations Research*, 66:215–224, 2016. doi:10.1016/j.cor.2015.08.005.

M. Haouari, A. Kooli, and E. Néron. Enhanced energetic reasoning-based lower bounds for the resource constrained project scheduling problem. *Computers and Operations Research*, 39(5):1187–1194, 2012. doi:10.1016/j.cor.2011.05.022.

J. R. Hardin, G. L. Nemhauser, and M. W. P. Savelsbergh. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. *Discrete Optimization*, 5(1):19–35, 2008. doi:10.1016/j.disopt.2007.10.003.

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

M. Hasannia Kolaee and S. M. J. Mirzapour Al-e-Hashem. Stochastic medical tourism problem with variable residence time considering gravity function. *RAIRO - Operations Research*, 56(3):1685–1716, May 2022. doi:10.1051/ro/2022082.

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

K. K. Haugen, A. Løkketangen, and D. L. Woodruff. Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132(1):116–122, 2001. doi:10.1016/S0377-2217(00)00116-8.

B. Henriksen and C. C. Røstad. Paths for Modularization. In *Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World.*, pages 272–279. Springer, Berlin, Heidelberg, 2014. ISBN 978-3-662-44733-8. doi:10.1007/978-3-662-44733-8_34.

W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, September 2005. doi:10.1016/j.ejor.2004.04.002.

T. Homem-De-Mello. Variable-sample methods for stochastic optimization. *ACM Transactions on Modeling and Computer Simulation*, 13(2):108–133, April 2003. doi:10.1145/858481.858483.

T. Homem-de Mello and G. Bayraksan. Monte Carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19 (1):56–85, 2014. doi:10.1016/j.sorms.2014.05.001.

S. Hu, Z. Zhang, S. Wang, Y. Kao, and T. Ito. A project scheduling problem with spatial resource constraints and a corresponding guided local search algorithm. *Journal of the Operational Research Society*, 70(8):1349–1361, 2019. doi:10.1080/01605682.2018.1489340.

F. Innella, M. Arashpour, and Y. Bai. Lean Methodologies and Techniques for Modular Construction: Chronological and Critical Review. *Journal of Construction Engineering and Management*, 145(12):04019076, 2019. doi:10.1061/(asce)co.1943-7862.0001712.

X. Jiang, R. Bai, S. W. Wallace, G. Kendall, and D. Landa-Silva. Soft clustering-based scenario bundling for a progressive hedging heuristic in stochastic service network design. *Computers and Operations Research*, 128:105182, 2021. doi:10.1016/j.cor.2020.105182.

A. A. Juan, J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2:62–72, 2015. doi:10.1016/j.orp.2015.03.001.

S. Katoch, S. S. Chauhan, and V. Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, February 2021. doi:10.1007/s11042-020-10139-6.

C. Kellenbrink and S. Helber. Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research*, 246(2):379–391, 2015. doi:10.1016/j.ejor.2015.05.003.

H. Kim, S. S. Lee, J. H. Park, and J. G. Lee. A model for a simulation-based shipbuilding system in a shipyard manufacturing process. *International Journal of Computer Integrated Manufacturing*, 18(6):427–441, 2005a. doi:10.1080/09511920500064789.

H. Kim, S. S. Lee, J. H. Park, and J. G. Lee. A model for a simulation-based shipbuilding system in a shipyard manufacturing process. *International Journal of Computer Integrated Manufacturing*, 18(6):427–441, 2005b. doi:10.1080/09511920500064789.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983. doi:10.1126/science.220.4598.671.

A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, 12(2): 479–502, 2002. doi:10.1137/S1052623499363220.

R. Kolisch and S. Hartmann. Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In *Project Scheduling*, pages 147–178. 1999. doi:10.1007/978-1-4615-5533-9_7.

O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal*, 25(1-2):25–47, 2013. doi:10.1007/s10696-012-9152-5.

M. Kotabe, R. Parente, and J. Y. Murray. Antecedents and outcomes of modular production in the Brazilian automobile industry: A grounded theory approach. *Journal of International Business Studies*, 38(1):84–106, 2007. doi:10.1057/palgrave.jibs.8400244.

A. Kusiak. Integrated product and process design: A modularity perspective. *Journal of Engineering Design*, 13(3):223–231, 2002. doi:10.1080/09544820110108926.

J. Kuster, D. Jannach, and G. Friedrich. Extending the RCPSP for modeling and solving disruption management problems. *Applied Intelligence*, 31(3):234–253, 2009. doi:10.1007/s10489-008-0119-x.

P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, 2003. doi:10.1016/S0004-3702(02)00362-4.

A. Lamghari and R. Dimitrakopoulos. Progressive hedging applied as a metaheuristic to schedule production in open-pit mines accounting for reserve uncertainty. *European Journal of Operational Research*, 253(3):843–855, 2016. doi:10.1016/j.ejor.2016.03.007.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960, URL: http://www.jstor.org/stable/1910129.

S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. pages 1–43, 2020, URL: http://arxiv.org/abs/2005.01643.

B. Lim and S. Zohren. Time-series forecasting with deep learning: A survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194), 2021. doi:10.1098/rsta.2020.0209.

A. Løkketangen and D. L. Woodruff. Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. *Journal of Heuristics*, 2(2):111–128, 1996. doi:10.1007/bf00247208.

M. Lombardi and M. Milano. Optimal methods for resource allocation and scheduling: A cross-disciplinary survey. *Constraints*, 17(1):51–85, 2012. doi:10.1007/s10601-011-9115-6.

D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6 (4):333–346, August 2002. doi:10.1109/TEVC.2002.802450.

J. Neelamkavil. Automation in the prefab and modular construction industry. *2009 26th International Symposium on Automation and Robotics in Construction, ISARC 2009*, pages 299–306, 2009. doi:10.22260/isarc2009/0018.

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

K. Neumann and C. Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533, 2003. doi:10.1007/s001860200251.

J. Olhager. The role of the customer order decoupling point in production and supply chain management. *Computers in Industry*, 61(9):863–868, 2010. doi:10.1016/j.compind.2010.07.011.

R. O'Rourke. Navy littoral combat ship (LCS) program: Background and issues for congress (updated). *Key Congressional Reports for September 2019: Part VI*, pages 187–215, 2020.

M. B. Pamay, K. Bülbül, and G. Ulusoy. Dynamic resource constrained multi-project scheduling problem with weighted earliness/tardiness costs. *International Series in Operations Research and Management Science*, 200:219–247, 2014. doi:10.1007/978-1-4614-9056-2_10.

J. Pandremenos, J. Paralikas, K. Salonitis, and G. Chryssolouris. Modularity concepts for the automotive industry: A critical review. *CIRP Journal of Manufacturing Science and Technology*, 1(3):148–152, 2009. doi:10.1016/j.cirpj.2008.09.012.

J. Park and K. Y. Kim. Meta-modeling using generalized regression neural network and particle swarm optimization. *Applied Soft Computing Journal*, 51:354–369, 2017. doi:10.1016/j.asoc.2016.11.029.

K. P. Park, S. H. Ham, and C. Y. Lee. Application and validation of production planning simulation in shipbuilding. *Ocean Engineering*, 114:154–167, 2016. doi:10.1016/j.oceaneng.2016.01.008.

R. Pellerin, N. Perrier, and F. Berthaut. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 175(2):707–721, January 2019. doi:10.1016/j.ejor.2019.01.063.

Z. Peng, Y. Zhang, Y. Feng, G. Rong, and H. Su. A Progressive Hedging-Based Solution Approach for Integrated Planning and Scheduling Problems under Demand Uncertainty. *Industrial and Engineering Chemistry Research*, 58(32):14880–14896, 2019. doi:10.1021/acs.iecr.9b02620.

M. Pero, M. Stößlein, and R. Cigolini. Linking product modularity to supply chain integration in the construction and shipbuilding industries. *International Journal of Production Economics*, 170:602–615, December 2015. doi:10.1016/j.ijpe.2015.05.011.

N. P. Petersson, S. Tenold, and N. J. White. *Shipping and Globalization in the Post-War Era*. 2019. ISBN 978-3-030-26001-9.

S. Pfeifer, T. Seidenberg, C. Jürgenhake, H. Anacker, and R. Dumitrescu. Towards a modular product architecture for electric ferries using Model-Based Systems Engineering. *Procedia Manufacturing*, 52(2019):228–233, 2020. doi:10.1016/j.promfg.2020.11.039.

F. S. Piran, D. P. Lacerda, M. A. Sellitto, and M. I. W. M. Morandi. Influence of modularity on delivery dependability: analysis in a bus manufacturer. *Production Planning and Control*, 32(8):688–698, 2021. doi:10.1080/09537287.2020.1776411.

S. R. Pires. Managerial implications of the modular consortium model in a Brazilian automotive plant. *International Journal of Operations and Production Management*, 18(3):221–232, March 1998. doi:10.1108/01443579810368290.

T. Poulsen and R. Lema. Is the supply chain ready for the green transformation? The case of offshore wind logistics. *Renewable and Sustainable Energy Reviews*, 73(January): 758–771, 2017. doi:10.1016/j.rser.2017.01.181.

A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, September 1969. doi:10.1287/mnsc.16.1.93.

H. D. Quoc, L. N. The, C. N. Doan, and T. P. Thanh. New Effective Differential Evolution Algorithm for the Project Scheduling Problem. In *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, pages 150–157. IEEE, June 2020. ISBN 978-1-7281-5800-6. doi:10.1109/ICCCI49374.2020.9145982.

V. J. Rayward-Smith. *Project Scheduling: Recent Models, Algorithms and Applications*, volume 52. 2001. ISBN 9781461375296. doi:10.1057/palgrave.jors.2601199.

R. T. Rockafellar and R. J.-B. Wets. Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991. doi:10.1287/moor.16.1.119.

C. Rose. *Automatic Production Planning for the Construction of Complex Ships*. PhD thesis, Delft University of Technology, 2017.

R. Rubeša, N. Fafandjel, and D. Koli. Procedure for Estimating the Effectiveness of. *Engineering Review*, 1(1):55–62, 2011.

A. Sahli, J. Carlier, and A. Moukrim. Comparison of mixed integer linear programming models for the Event Scheduling Problem with Consumption and Production of Resources. *IFAC-PapersOnLine*, 49(12):1044–1049, 2016. doi:10.1016/j.ifacol.2016.07.580.

M. Sako and F. Murray. Modules in Design, Production and Use: Implications for the Global Automotive Industry. *International Motor Vehicle Program Annual Meeting*, pages 1–35, 1999. doi:10.1.1.125.3249.

M. Sako and F. M. Said. Modules in Design , Production and Use : Implications for the Global Automotive Industry 1. In *International Motor Vehicle Program (IMVP) Annual Sponsors Meetin*, 1999.

M. Salemi Parizi, Y. Gocgun, and A. Ghate. Approximate policy iteration for dynamic resource-constrained project scheduling. *Operations Research Letters*, 45(5):442–447, 2017. doi:10.1016/j.orl.2017.06.002.

K. M. Sallam, R. K. Chakrabortty, and M. J. Ryan. A two-stage multi-operator differential evolution algorithm for solving Resource Constrained Project Scheduling problems. *Future Generation Computer Systems*, 108:432–444, 2020. doi:10.1016/j.future.2020.02.074.

U. Satic, P. Jacko, and C. Kirkbride. Performance evaluation of scheduling policies for the dynamic and stochastic resource-constrained multi-project scheduling problem. *International Journal of Production Research*, 0(0):1–13, 2020. doi:10.1080/00207543.2020.1857450.

J. F. Schank, H. Pung, G. T. Lee, M. V. Arena, and J. Birkler. *Outsourcing and Outfitting Practices: Implications for the Ministry of Defence Shipbuilding Programmes*. RAND Corporation, Santa Monica, CA, 2005.

A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency (Algorithms and Combinatorics)*. 2008. ISBN 3540204563.

T. Servranckx and M. Vanhoucke. A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs. *European Journal of Operational Research*, 273(3):841–860, 2019. doi:10.1016/j.ejor.2018.09.005.

A. Shirzadeh Chaleshtarti, S. Shadrokh, M. Khakifirooz, M. Fathi, and P. M. Pardalos. A hybrid genetic and Lagrangian relaxation algorithm for resource-constrained project scheduling under nonrenewable resources. *Applied Soft Computing Journal*, 94:106482, 2020. doi:10.1016/j.asoc.2020.106482.

A. Sprecher, R. Kolisch, and A. Drexl. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1):94–102, 1995. doi:10.1016/0377-2217(93)E0294-8.

M. Stopford. *Maritime Economics 3e*. Routledge, December 2008. ISBN 9781134476534. doi:10.4324/9780203891742.

R. L. Storch. *Ship Production*. Society of Naval Architects and Marine Engineers, 2007. ISBN 9780939773572, URL: https://books.google.nl/books?id=KqJqtgAACAAJ.

R. L. Storch and S. Sukapanpotharam. Common generic block: Mass customization for shipbuilding. *Proceedings of the Institution of Mechanical Engineers Part M: Journal of Engineering for the Maritime Environment*, 217(2):79–94, 2003. doi:10.1243/147509003321921337.

R. Storn and K. Price. Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical Report*, (TR-95-012):1–12, 1995, URL: ftp://ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf.

F. B. Talbot. Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case. *Management Science*, 28(10):1197–1210, October 1982. doi:10.1287/mnsc.28.10.1197.

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

S. Tao and Z. S. Dong. Multi-mode resource-constrained project scheduling problem with alternative project structures. *Computers and Industrial Engineering*, 125 (November 2017):333–347, 2018. doi:10.1016/j.cie.2018.08.027.

J. Torres, C. Li, R. Apap, and I. Grossmann. A Review on the Performance of Linear and Mixed Integer Two-Stage Stochastic Programming Algorithms and Software. *Optimization Online*, pages 1–30, 2019.

A. Türk, S. Gürgen, M. Ozkok, and s. Altin. A comprehensive investigation into the performance of genetic algorithm for effective shipyard topological layout. *Proceedings of the Institution of Mechanical Engineers Part M: Journal of Engineering for the Maritime Environment*, 236(3):726–740, 2022. doi:10.1177/14750902211062057.

V. Valls, F. Ballestín, and S. Quintanilla. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 185(2):495–508, 2008. doi:10.1016/j.ejor.2006.12.033.

T. Van der Beek, D. Souravlias, J. Van Essen, J. Pruyn, and K. Aardal. 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, 2024. doi:https://doi.org/10.1016/j.ejor.2023.07.043.

T. van der Beek. Instances, file format, generator script and results for the Resource Constrained Project Scheduling Problem with a flexible Project Structure and Consumption and Production of Resources. 2021. doi:10.4121/16764205.v1.

T. van der Beek. Instances and file format for the Resource Constrained Project Scheduling Problem with Modular Production. 2022a. doi:10.4121/21774530.V1.

T. van der Beek. Instances and file format for the Dynamic Resource Constrained Project Scheduling Problem with Static Project Schedules. 2022b. doi:10.4121/21565056.V1.

T. van der Beek. Instances and file format for the Resource Constrained Project Scheduling Problem with a flexible Project Structure. 2022c. doi:https://doi.org/10.4121/21106768.v1.

R. M. van Slyke and R. Wets. L -shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4): 638–663, July 1969. doi:10.1137/0117061.

M. Vanhoucke, J. Coelho, D. Debels, B. Maenhout, and L. V. Tavares. An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524, 2008. doi:10.1016/j.ejor.2007.03.032.

I. Vierth, V. Sowa, and K. Cullinane. Evaluating the external costs of trailer transport: a comparison of sea and road. *Maritime Economics and Logistics*, 21(1):61–78, 2019. doi:10.1057/s41278-018-0099-7.

N. Vladimir, I. Ančić, I. Gatin, M. Tošić, and V. Vukčević. Development of design methodology for modular passenger ships for the Mediterranean. In *Simpozij" Teorija i praksa brodogradnje"*, pages 454–461, 2018.

N. Vladimir, A. Bakica, M. Perčić, and I. Jovanović. Modular Approach in the Design of Small Passenger Vessels for Mediterranean. *Journal of Marine Science and Engineering*, 10(1), 2022. doi:10.3390/jmse10010117.

J. Wang and W. Lui. Forward-backward improvement for genetic algorithm based optimization of resource constrained scheduling problem. In *DEStech Transactions on Computer Science and Engineering*, number ameit, pages 349–356, July 2017. ISBN 9781605954578. doi:10.12783/dtcse/ameit2017/12327.

J. P. Watson and D. L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8 (4):355–370, 2011. doi:10.1007/s10287-010-0125-4.

J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra. Project scheduling with finite or infinite number of activity processing modes - A survey. *European Journal of Operational Research*, 208(3):177–205, 2011. doi:10.1016/j.ejor.2010.03.037.

Y. Wei. Automatic Generation of Assembly Sequence for the Planning of Outfitting Processes in Shipbuilding. *Journal of Ship Production and Design*, 28(2), 2012. doi:10.5957/JSPD.28.2.120002.

P. A. Wrigley, P. Wood, S. O'Neill, R. Hall, and D. Robertson. Off-site modular construction and design in nuclear power: A systematic literature review. *Progress in Nuclear Energy*, 134(December 2020):103664, 2021. doi:10.1016/j.pnucene.2021.103664.

I. C. Wu, A. Borrmann, U. Beißert, M. König, and E. Rank. Bridge construction schedule generation with pattern-based construction methods and constraint-based simulation. *Advanced Engineering Informatics*, 24(4):379–388, 2010. doi:10.1016/j.aei.2010.07.002.

F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. Coello Coello. An evolutionary approach for resource constrained project scheduling with uncertain changes. *Computers & Operations Research*, 125, January 2021. doi:10.1016/j.cor.2020.105104.

P. Zheng, P. Zhang, J. Wang, J. Zhang, C. Yang, and Y. Jin. 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, 2020. doi:10.1080/0951192X.2020.1803506.

J. Zhou, P. E. D. Love, X. Wang, K. L. Teo, and Z. Irani. A review of methods and algorithms for optimizing construction scheduling. *Journal of the Operational Research Society*, 64(8):1091–1105, August 2013. doi:10.1057/jors.2012.174.

F. W. Zhu, X. X. Sun, J. Miller, and Z. J. Deng. Innovations in knowledge management: Applying modular design. *International Journal of Innovation Science*, 6(2):83–95, 2014. doi:10.1260/1757-2223.6.2.83.

G. Zhu, J. F. Bard, and G. Yu. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing*, 18(3): 377–390, 2006. doi:10.1287/ijoc.1040.0121.

CURRICULUM VITÆ

Tom VAN DER BEEK

8-11-1991 Born in Bergen op Zoom, the Netherlands.

EDUCATION

2010–2014 Bachelor Mechanical Engineering
Delft University of Technology

2015–2018 Master Offshore Engineering
Delft University of Technology
Thesis: Combinatorial optimisation methods for wind farm installation scheduling
Supervisors: Dr. ir. J.T. van Essen, Dr. ir. S.A. Miedema, Prof. dr. ir. K.I. Aardal, Dr. ir. F. Baart, Ir. T. Damsma

2018-2023 PhD Optimization
Delft University of Technology
Thesis: Scheduling methods for modular shipbuilding
Promotor: Prof. dr. ir. K.I. Aardal
Copromotors: Dr. ir. J. Pruyn
Dr. ir. J.T. van Essen
Diplomas: LNMB (Dutch Network on the Mathematics of Operations Research)

LIST OF PUBLICATIONS

2. **Van der Beek, T.**, van Essen, J.T., Pruyn, J., Aardal, K.I. & Hopman, H. (2019). "Optimizing inventory strategy for modular shipbuilding". *International Conference on Computer Applications in Shipbuilding 2019*

1. **Van der Beek, T.**, Souravlias, D., van Essen, J.T., Pruyn J. & Aardal, K.I. (2024). "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, Vol. 313 (92-111).*